

ARCoSS

LNCS 6346

Mark de Berg  
Ulrich Meyer (Eds.)

# Algorithms – ESA 2010

18th Annual European Symposium  
Liverpool, UK, September 2010  
Proceedings, Part I

1  
Part I



Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison, UK

Takeo Kanade, USA

Josef Kittler, UK

Jon M. Kleinberg, USA

Alfred Kobsa, USA

Friedemann Mattern, Switzerland

John C. Mitchell, USA

Moni Naor, Israel

Oscar Nierstrasz, Switzerland

C. Pandu Rangan, India

Bernhard Steffen, Germany

Madhu Sudan, USA

Demetri Terzopoulos, USA

Doug Tygar, USA

Gerhard Weikum, Germany

## Advanced Research in Computing and Software Science

### Subline of Lectures Notes in Computer Science

## Subline Series Editors

Giorgio Ausiello, *University of Rome ‘La Sapienza’, Italy*

Vladimiro Sassone, *University of Southampton, UK*

## Subline Advisory Board

Susanne Albers, *University of Freiburg, Germany*

Benjamin C. Pierce, *University of Pennsylvania, USA*

Bernhard Steffen, *University of Dortmund, Germany*

Madhu Sudan, *Microsoft Research, Cambridge, MA, USA*

Deng Xiaotie, *City University of Hong Kong*

Jeannette M. Wing, *Carnegie Mellon University, Pittsburgh, PA, USA*

Mark de Berg Ulrich Meyer (Eds.)

# Algorithms – ESA 2010

18th Annual European Symposium  
Liverpool, UK, September 6–8, 2010  
Proceedings, Part I

Volume Editors

Mark de Berg  
Department of Mathematics  
and Computing Science  
TU Eindhoven  
Eindhoven, The Netherlands  
E-mail: mdberg@win.tue.nl

Ulrich Meyer  
Institute for Computer Science  
Goethe University  
Frankfurt/Main, Germany  
E-mail: umeyer@cs.uni-frankfurt.de

Library of Congress Control Number: 2010933821

CR Subject Classification (1998): F.2, I.3.5, C.2, E.1, G.2, D.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743  
ISBN-10 3-642-15774-2 Springer Berlin Heidelberg New York  
ISBN-13 978-3-642-15774-5 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

[springer.com](http://springer.com)

© Springer-Verlag Berlin Heidelberg 2010  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper 06/3180 5 4 3 2 1 0

# Preface

This volume contains the 69 papers presented at the 16th Annual European Symposium on Algorithms (ESA 2010), held in Liverpool during September 6–8, 2010, including three papers by the distinguished invited speakers Artur Czumaj, Herbert Edelsbrunner, and Paolo Ferragina. ESA 2010 was organized as a part of ALGO 2010, which also included the 10th Workshop on Algorithms in Bioinformatics (WABI), the 8th Workshop on Approximation and Online Algorithms (WAOA), and the 10th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS).

The European Symposium on Algorithms covers research in the design, use, and analysis of efficient algorithms and data structures. As in previous years, the symposium had two tracks: the *Design and Analysis Track* and the *Engineering and Applications Track*, each with its own Program Committee. In total 245 papers adhering to the submission guidelines were submitted. Each paper was reviewed by three or four referees. Based on the reviews and the often extensive electronic discussions following them, the committees selected 66 papers in total: 56 (out of 206) to the Design and Analysis Track and 10 (out of 39) to the Engineering and Applications track. We believe that these papers together made up a strong and varied program, showing the depth and breadth of current algorithms research.

Three papers deserve special mentioning: the papers “When LP Is the Cure for Your Matching Woes: Improved Bounds for Stochastic Matchings” by N. Bansal, A. Gupta, J. Li, J. Mestre, V. Nagarajan and A. Rudra and “Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems” by V. Bonifaci and A. Marchetti-Spaccamela, which won the award for the best paper, and the paper “Shortest Paths in Planar Graphs with Real Lengths in  $O(n \log^2 n / \log \log n)$  Time” by S. Mozes and C. Wulff-Nilsen, which won the award for the best student paper. We congratulate the authors on this success.

ESA 2010 was sponsored by the European Association of Theoretical Computer Science, the International Society of Computational Geometry, the London Mathematical Society, Springer, and the University of Liverpool. Besides the sponsors, we also wish to thank the people from the EasyChair Conference System; using their wonderful system saved us an enormous amount of work during the whole process. Finally, we thank all authors who submitted their work to ESA 2010, all Program Committee members for their hard work, and all reviewers who helped the Program Committees in evaluating the submitted papers, and we hope the readers will find the papers in these proceedings instructive and enjoyable.

July 2010

Mark de Berg  
Ulrich Meyer

# Organization

## Program Committee

### *Design and Analysis Track*

Mark de Berg (Chair)

Hans Bodlaender

Peter Bro Miltersen

Sergio Cabello

Kenneth L. Clarkson

Khaled Elbassioni

Leah Epstein

Leszek Gąsieniec

Roberto Grossi

Michael Kaufmann

Samir Khuller

Mikko Koivisto

Sylvain Lazard

Mohammad Mahdian

S. Muthu Muthukrishnan

Petra Mutzel

Leen Stougie

Yusu Wang

Christos Zaroliagis

TU Eindhoven, The Netherlands

Utrecht University, The Netherlands

Aarhus University, Denmark

University of Ljubljana, Slovenia

IBM Almaden, USA

MPI Saarbrücken, Germany

University of Haifa, Israel

University of Liverpool, UK

Università di Pisa, Italy

Universität Tübingen, Germany

University of Maryland, USA

University of Helsinki, Finland

INRIA Nancy Grand Est, France

Yahoo! Research, USA

Rutgers University & Google, USA

TU Dortmund, Germany

VU and CWI Amsterdam,

The Netherlands

Ohio State University,

CTI and University of Patras, Greece

### *Engineering and Applications Track*

András Benczúr

Gerth Brodal

Peter Eades

Lars Engebretsen

Andrew Goldberg

Gunnar Klau

Kishore Kothapalli

Stefano Leonardi

Ulrich Meyer (Chair)

Marina Papatriantafilou

Sylvain Pion

Anita Schöbel

Laura Toma

Prudence Wong

Norbert Zeh

Hungarian Academy of Sciences, Hungary

Aarhus University, Denmark

University of Sydney, Australia

Google Zürich, Switzerland

Microsoft Research, USA

CWI Amsterdam, The Netherlands

IIIT Hyderabad, India

La Sapienza University, Rome, Italy

Goethe University Frankfurt, Germany

Chalmers University, Sweden

INRIA Sophia Antipolis - Méditerranée,  
France

University of Göttingen, Germany

Bowdoin College, USA

University of Liverpool, UK

Dalhousie University, Canada

## Organizing Committee

The Organizing Committee from the University of Liverpool consisted of:

Andrew Collins  
Leszek Gaśieniec (Chair)  
Russell Martin  
Igor Potapov  
Thelma Williams  
Prudence Wong

## Referees

Ittai Abraham	Daniel Cederman	Christoph Dürr
Louigi Addario-Berry	Ho-Leung Chan	Alon Efrat
Isolde Adler	T-H. Hubert Chan	Edith Elkind
Deepak Ajwani	Timothy Chan	Amr Elmasry
Marjan van den Akker	Frédéric Chazal	David Eppstein
Saeed Alaei	Jianer Chen	Funda Ergun
Aris Anagnostopoulos	Ning Chen	Thomas Erlebach
Spyros Angelopoulos	Siu-Wing Cheng	Claus Ernst
Elliot Anshelevich	Otfried Cheong	William Evans
Sunil Arya	Flavio Chierichetti	Hazel Everett
Dominique Attali	Giorgos Christodoulou	Angelo Fanelli
Evripidis Bampis	Ferdinando Cicalese	Mohammad Farshi
Nikhil Bansal	Raphael Clifford	Sandor Fekete
Jérémie Barbay	David Cohen-Steiner	Henning Fernau
Andreas Beckmann	Éric Colin de Verdière	Paolo Ferragina
Anton Belov	Atlas F. Cook IV	Irene Finocchi
Oren Ben-Zwi	José R. Correa	Johannes Fischer
Sergey Bereg	Ovidiu Daescu	Rudolf Fleischer
Hoda Bidkhori	Peter Damaschke	Fedor Fomin
Philip Bille	Atish Das Sarma	Dimitris Fotakis
Vincenzo Bonifaci	Pooya Davoodi	Nikolaos Fountoulakis
Ilaria Bordino	Pedro M. M. de Castro	Kimmo Fredriksson
Prosenjit Bose	Daniel Delling	Tom Friedetzky
David Bremner	Camil Demetrescu	Zachary Friggstad
Patrick Briest	Tamal Dey	Zhang Fu
Andrej Brodnik	Yuanan Diao	Stanley Fung
Costas Busch	Martin Dietzfelbinger	Stefan Funke
Sebastian Böcker	Thomas C. van Dijk	Hal Gabow
Saverio Caminiti	Shahar Dobzinski	Bernd Gärtner
Stefan Canzar	Benjamin Doerr	Frantisek Galcik
Alberto Caprara	Vida Dujmovic	Iftah Gamzu
Ioannis Caragiannis	Laurent Dupont	Jie Gao
Manuel Caroli	Steph Durocher	William Gasarch

Georgios Georgiadis	Steven Kelk	Dániel Marx
Loukas Georgiadis	David Kempe	Nicole Megow
Arpita Ghosh	Elena Kleiman	Julian Mestre
Panos Giannopoulos	Karsten Klein	Aranyak Mehta
Matt Gibson	Christian Knauer	Pauli Miettinen
Anders Gidenstam	Stavros Kolliopoulos	Matus Mihalak
Joan Glaunes	Spyros Kontogiannis	Matthias Mnich
Marc Glisse	Amos Korman	Bojan Mohar
Michael Gnewuch	Guy Kortsarz	Ankur Moitra
Xavier Goaoc	Nitish Korula	Pat Morin
Peter Gottschling	Adrian Kosowski	Gabriel Moruz
Vineet Goyal	Annamária Kovács	David Mount
Fabrizio Grandoni	Richard Kralovic	M. Müller-Hannemann
Alexander Grigoriev	Dieter Kratsch	Tobias Muller
Gaël Guennebaud	Stefan Kratsch	Wolfgang Mulzer
Jiong Guo	Stephan Kreutzer	Veli Mäkinen
Carsten Gutwenger	Nils Kriege	Thomas Mølhave
Nima Haghpanah	Sven Krumke	Seffi Naor
M.T. Hajiaghayi	Piyush Kumar	Giri Narasimhan
Olaf Hall-Holt	Juha Kärkkäinen	Hariharan Narayanan
K. Arnsfelt Hansen	Stefan Langerman	Gonzalo Navarro
T. Dueholm Hansen	Kasper Dalgaard Larsen	Hamid Nazerzadeh
Sariel Har-Peled	Francis Lazarus	Frank Neumann
David Hartvigsen	Lap-Kei Lee	Alantha Newman
Rafael Hassin	Joshua Letchford	Ilan Newman
Pinar Heggernes	Asaf Levin	Hung Ngo
Danny Hermelin	Joshua Levine	Kim Thang Nguyen
John Hershberger	Maarten Löffler	Rolf Niedermeier
Martin Hoefer	Jian Li	Nicolas Nisse
Wing Kai Hon	Christian Liebchen	Marc Noy
Han Hoogeveen	Daniel Lokshantov	Krzysztof Onak
Chien-Chung Huang	Zvi Lotker	Jim Orlin
Thore Husfeldt	Anna Lubiw	Rasmus Pagh
Falk Hüffner	Tamas Lukovszki	K. Panagiotou
Csand Imreh	Meena Mahajan	Gyula Pap
Kazuo Iwama	Kazuhis Makino	Gregor Pardella
Satoru Iwata	Johann Makowsky	Kunsoo Park
Bart Jansen	David Malec	Britta Peis
Bin Jiang	Azarakhsh Malekian	Rudi Pendavingh
Satyen Kale	Sven Mallach	Michal Penn
Marcin Kaminski	David Manlove	Xavier Pennec
Sanjiv Kapoor	Bodo Manthey	Marko Petkovek
Chinmay Karande	A. Marchetti-Spaccamela	Jeff Phillips
Petteri Kaski	Vangelis Markakis	Greg Plaxton
Matthew Katz	Russell Martin	Valentin Polishchuk

Matthias Poloczek  
Laura Poplawski-Ma  
Marc Pouget  
E. Pountourakis  
Kirk Pruhs  
Geppo Pucci  
Yuri Rabinov  
Luis Rademacher  
Tomasz Radzik  
Harald Räcke  
Arash Rafiey  
Balaji Raghavachari  
S. Raghavan  
M. Sohel Rahman  
Rajmohan Rajaraman  
Rajiv Raman  
Jörg Rambau  
Pasi Rastas  
Imran Rauf  
Dror Rawitz  
Saurabh Ray  
Peter Reiter  
Liam Roditty  
Dana Ron  
Johan M. M. van Rooij  
Adi Rosén  
Günter Rote  
Thomas Rothvoß  
Kunihiko Sadakane  
Barna Saha  
Saket Saurabh  
Rahul Savani  
Francesco Scarcello  
Guido Schäfer  
Elad Michael Schiller

Florian Schoppmann  
Anna Schulze  
Celine Scornavacca  
Danny Segev  
C. Seshadhri  
Jiří Sgall  
Hadas Shachnai  
Rahul Shah  
Mordechai Shalom  
Jessica Sherette  
Junghwan Shin  
Somnath Sikdar  
Rodrigo Silveira  
Amitabh Sinha  
René Sitters  
Alexander Skopalik  
Michiel Smid  
Andreas Spillner  
Yannis Stamatiou  
Ulrike Stege  
David Steurer  
Håkan Sundell  
Wing-Kin Sung  
Rob van Stee  
Jukka Suomela  
Zoya Svitkina  
Tami Tamir  
Siamak Tazari  
Orestis Telelis  
Kavitha Telikepalli  
Dimitrios Thilikos  
Mikkel Thorup  
Hans Raj Tiwary  
Kostas Tsichlas  
Elias Tsigaridas

Andy Twigg  
George Tzoumas  
Steve Uhlig  
Takeaki Uno  
Jan Vahrenhold  
Gabriel Valiente  
Sergei Vassilvitskii  
Gert Vegter  
Marinus Veldhorst  
S. Venkatasubramanian  
Angelina Vidali  
Yngve Villanger  
Niko Välimäki  
Uli Wagner  
Tomasz Walen  
Haitao Wang  
Lei Wang  
Volker Weichert  
Oren Weimann  
Renato Werneck  
Matthias Westermann  
Christopher Whidden  
Peter Widmayer  
Ryan Williams  
Gerhard J. Woeginger  
Nicola Wolpert  
Hoi-Ming Wong  
Lirong Xia  
Qiqi Yan  
Neal Young  
Mariette Yvinec  
Bernd Zey  
Lintao Zhang  
Yong Zhang  
Binhai Zhu

# Table of Contents – Part I

## Invited Talk

The Robustness of Level Sets .....	1
<i>Paul Bendich, Herbert Edelsbrunner, Dmitriy Morozov, and Amit Patel</i>	

## Session 1a

Solving an Avionics Real-Time Scheduling Problem by Advanced IP-Methods .....	11
<i>Friedrich Eisenbrand, Karthikeyan Kesavan, Raju S. Mattikalli, Martin Niemeier, Arnold W. Nordsieck, Martin Skutella, José Verschae, and Andreas Wiese</i>	
Non-clairvoyant Speed Scaling for Weighted Flow Time .....	23
<i>Sze-Hang Chan, Tak-Wah Lam, and Lap-Kei Lee</i>	
A Robust PTAS for Machine Covering and Packing .....	36
<i>Martin Skutella and José Verschae</i>	

## Session 1b

Balancing Degree, Diameter and Weight in Euclidean Spanners .....	48
<i>Shay Solomon and Michael Elkin</i>	
Testing Euclidean Spanners .....	60
<i>Frank Hellweg, Melanie Schmidt, and Christian Sohler</i>	
Fast Approximation in Subspaces by Doubling Metric Decomposition ...	72
<i>Marek Cygan, Lukasz Kowalik, Marcin Mucha, Marcin Pilipczuk, and Piotr Sankowski</i>	
$f$ -Sensitivity Distance Oracles and Routing Schemes .....	84
<i>Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty</i>	

## Session 2a

Fast Minor Testing in Planar Graphs .....	97
<i>Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos</i>	
On the Number of Spanning Trees a Planar Graph Can Have .....	110
<i>Kevin Buchin and André Schulz</i>	

Contractions of Planar Graphs in Polynomial Time.....	122
<i>Marcin Kamiński, Daniël Paulusma, and Dimitrios M. Thilikos</i>	

## Session 2b

Communication Complexity of Quasirandom Rumor Spreading .....	134
<i>Petra Berenbrink, Robert Elsässer, and Thomas Sauerwald</i>	
A Complete Characterization of Group-Strategyproof Mechanisms of Cost-Sharing .....	146
<i>Emmanouil Pountourakis and Angelina Vidali</i>	
Contribution Games in Social Networks.....	158
<i>Elliot Anshelevich and Martin Hoefer</i>	

## Session 3a

Improved Bounds for Online Stochastic Matching .....	170
<i>Bahman Bahmani and Michael Kapralov</i>	
Online Stochastic Packing Applied to Display Ad Allocation.....	182
<i>Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Cliff Stein</i>	
Caching Is Hard – Even in the Fault Model .....	195
<i>Marek Chrobak, Gerhard J. Woeginger, Kazuhisa Makino, and Haifeng Xu</i>	

## Session 3b

Superselectors: Efficient Constructions and Applications .....	207
<i>Ferdinando Cicalese and Ugo Vaccaro</i>	
Estimating the Average of a Lipschitz-Continuous Function from One Sample .....	219
<i>Abhimanyu Das and David Kempe</i>	
Streaming Graph Computations with a Helpful Advisor.....	231
<i>Graham Cormode, Michael Mitzenmacher, and Justin Thaler</i>	

## Session 4a

Algorithms for Dominating Set in Disk Graphs: Breaking the $\log n$ Barrier .....	243
<i>Matt Gibson and Imran A. Pirwani</i>	
Minimum Vertex Cover in Rectangle Graphs .....	255
<i>Reuven Bar-Yehuda, Danny Hermelin, and Dror Rawitz</i>	

Feedback Vertex Sets in Tournaments . . . . .	267
<i>Serge Gaspers and Matthias Mnich</i>	

## Session 4b

<i>n</i> -Level Graph Partitioning . . . . .	278
<i>Vitaly Osipov and Peter Sanders</i>	
Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns . . . . .	290
<i>Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger</i>	

Finding the Diameter in Real-World Graphs: Experimentally Turning a Lower Bound into an Upper Bound . . . . .	302
<i>Pierluigi Crescenzi, Roberto Grossi, Claudio Imbrenda, Leonardo Lanzi, and Andrea Marino</i>	

## Session 5a

Budgeted Red-Blue Median and Its Generalizations . . . . .	314
<i>MohammadTaghi Hajiaghayi, Rohit Khandekar, and Guy Kortsarz</i>	
All Ternary Permutation Constraint Satisfaction Problems Parameterized above Average Have Kernels with Quadratic Numbers of Variables . . . . .	326
<i>Gregory Gutin, Leo van Iersel, Matthias Mnich, and Anders Yeo</i>	

Strong Formulations for the Multi-module PESP and a Quadratic Algorithm for Graphical Diophantine Equation Systems . . . . .	338
<i>Laura Galli and Sebastian Stiller</i>	

Robust Algorithms for Sorting Railway Cars . . . . .	350
<i>Christina Büsing and Jens Maue</i>	

## Session 5b

Cloning Voronoi Diagrams via Retroactive Data Structures . . . . .	362
<i>Matthew T. Dickerson, David Eppstein, and Michael T. Goodrich</i>	
A Unified Approach to Approximate Proximity Searching . . . . .	374
<i>Sunil Arya, Guilherme D. da Fonseca, and David M. Mount</i>	
Spatio-temporal Range Searching over Compressed Kinetic Sensor Data . . . . .	386
<i>Sorelle A. Friedler and David M. Mount</i>	

Constructing the Exact Voronoi Diagram of Arbitrary Lines in Three-Dimensional Space: with Fast Point-Location . . . . .	398
<i>Michael Hemmer, Ophir Setter, and Dan Halperin</i>	

## Invited Talk

Local Graph Exploration and Fast Property Testing . . . . .	410
<i>Artur Czumaj</i>	

## Session 6a

A Fully Compressed Algorithm for Computing the Edit Distance of Run-Length Encoded Strings . . . . .	415
<i>Kuan-Yu Chen and Kun-Mao Chao</i>	

Fast Prefix Search in Little Space, with Applications . . . . .	427
<i>Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna</i>	

On the Huffman and Alphabetic Tree Problem with General Cost Functions . . . . .	439
<i>Hiroshi Fujiwara and Tobias Jacobs</i>	

Medium-Space Algorithms for Inverse BWT . . . . .	451
<i>Juha Kärkkäinen and Simon J. Puglisi</i>	

## Session 6b

Median Trajectories . . . . .	463
<i>Kevin Buchin, Maike Buchin, Marc van Kreveld, Maarten Löffler, Rodrigo I. Silveira, Carola Wenk, and Lionov Wiratma</i>	

Optimal Cover of Points by Disks in a Simple Polygon . . . . .	475
<i>Haim Kaplan, Matthew J. Katz, Gila Morgenstern, and Micha Sharir</i>	

Stability of $\varepsilon$ -Kernels . . . . .	487
<i>Pankaj K. Agarwal, Jeff M. Phillips, and Hai Yu</i>	

The Geodesic Diameter of Polygonal Domains . . . . .	500
<i>Sang Won Bae, Matias Korman, and Yoshio Okamoto</i>	

## Session 7a

Polyhedral and Algorithmic Properties of Quantified Linear Programs . . . . .	512
<i>Ulf Lorenz, Alexander Martin, and Jan Wolf</i>	

Approximating Parameterized Convex Optimization Problems . . . . .	524
<i>Joachim Giesen, Martin Jaggi, and Sören Laue</i>	
Approximation Schemes for Multi-Budgeted Independence Systems . . . . .	536
<i>Fabrizio Grandoni and Rico Zenklusen</i>	
 <b>Session 7b</b>	
Algorithmic Meta-theorems for Restrictions of Treewidth . . . . .	549
<i>Michael Lampis</i>	
Determining Edge Expansion and Other Connectivity Measures of Graphs of Bounded Genus . . . . .	561
<i>Vireesh Patel</i>	
Constructing the R* Consensus Tree of Two Trees in Subcubic Time . . .	573
<i>Jesper Jansson and Wing-Kin Sung</i>	
 <b>Author Index</b> . . . . .	585

# Table of Contents – Part II

## Invited Talk

Data Structures: Time, I/Os, Entropy, Joules! . . . . .	1
<i>Paolo Ferragina</i>	

## Session 8a

Weighted Congestion Games: Price of Anarchy, Universal Worst-Case Examples, and Tightness . . . . .	17
<i>Kshipra Bhawalkar, Martin Gairing, and Tim Roughgarden</i>	
Computing Pure Nash and Strong Equilibria in Bottleneck Congestion Games . . . . .	29
<i>Tobias Harks, Martin Hoefer, Max Klimm, and Alexander Skopalik</i>	
Combinatorial Auctions with Verification Are Tractable . . . . .	39
<i>Piotr Krysta and Carmine Ventre</i>	
How to Allocate Goods in an Online Market? . . . . .	51
<i>Yossi Azar, Niv Buchbinder, and Kamal Jain</i>	

## Session 8b

Fréchet Distance of Surfaces: Some Simple Hard Cases . . . . .	63
<i>Kevin Buchin, Maike Buchin, and André Schulz</i>	
Geometric Algorithms for Private-Cache Chip Multiprocessors . . . . .	75
<i>Deepak Ajwani, Nodari Sitchinava, and Norbert Zeh</i>	
Volume in General Metric Spaces . . . . .	87
<i>Ittai Abraham, Yair Bartal, Ofer Neiman, and Leonard J. Schulman</i>	
Shortest Cut Graph of a Surface with Prescribed Vertex Set . . . . .	100
<i>Éric Colin de Verdière</i>	

## Session 9a

Induced Matchings in Subcubic Planar Graphs . . . . .	112
<i>Ross J. Kang, Matthias Mnich, and Tobias Müller</i>	
Robust Matchings and Matroid Intersections . . . . .	123
<i>Ryo Fujita, Yusuke Kobayashi, and Kazuhisa Makino</i>	

## XVIII Table of Contents – Part II

A 25/17-Approximation Algorithm for the Stable Marriage Problem with One-Sided Ties.....	135
Kazuo Iwama, Shuichi Miyazaki, and Hiroki Yanagisawa	
Strongly Stable Assignment .....	147
Ning Chen and Arpita Ghosh	

## Session 9b

Data Structures for Storing Small Sets in the Bitprobe Model .....	159
Jaikumar Radhakrishnan, Smit Shah, and Saswata Shannigrahi	
On Space Efficient Two Dimensional Range Minimum Data Structures .....	171
Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao	
Pairing Heaps with Costless Meld .....	183
Amr Elmasry	
Top- $k$ Ranked Document Search in General Text Databases .....	194
J. Shane Culpepper, Gonzalo Navarro, Simon J. Puglisi, and Andrew Turpin	

## Best-Paper Session

Shortest Paths in Planar Graphs with Real Lengths in $O(n \log^2 n / \log \log n)$ Time .....	206
Shay Mozes and Christian Wulff-Nilsen	
When LP Is the Cure for Your Matching Woes: Improved Bounds for Stochastic Matchings .....	218
Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra	
Feasibility Analysis of Sporadic Real-Time Multiprocessor Task Systems .....	230
Vincenzo Bonifaci and Alberto Marchetti-Spaccamela	
Author Index .....	243

# The Robustness of Level Sets\*

Paul Bendich<sup>1,2,3</sup>, Herbert Edelsbrunner<sup>1,2,3,5</sup>, Dmitriy Morozov<sup>4</sup>, and Amit Patel<sup>1,2</sup>

<sup>1</sup> IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria

<sup>2</sup> Dept. Comput. Sci., Duke Univ., Durham, North Carolina

<sup>3</sup> Dept. Mathematics, Duke Univ. Durham, North Carolina

<sup>4</sup> Depts. Comput. Sci. and Math., Stanford Univ., Stanford, California

<sup>5</sup> Geomagic, Research Triangle Park, North Carolina

**Abstract.** We define the robustness of a level set homology class of a function  $f : \mathbb{X} \rightarrow \mathbb{R}$  as the magnitude of a perturbation necessary to kill the class. Casting this notion into a group theoretic framework, we compute the robustness for each class, using a connection to extended persistent homology. The special case  $\mathbb{X} = \mathbb{R}^3$  has ramifications in medical imaging and scientific visualization.

**Keywords:** Topological spaces, continuous functions, level sets, perturbations, homology, extended persistence, well groups, well diagrams, robustness.

## 1 Introduction

The work reported in this paper has two motivations, one theoretical and the other practical. The former is the recent introduction of *well groups* in the study of mappings between topological spaces. Assuming a metric space of perturbations, we have such a group for each subspace  $\mathbb{A} \subseteq \mathbb{Y}$ , each bound  $r \geq 0$  on the magnitude of the perturbation, and each dimension  $p$ . These groups extend the boolean concept of transversality to a real-valued measure we refer to as *robustness*. Using this measure, we can quantify the robustness of a fixed point of a mapping [8] and prove the stability of the apparent contour of a mapping from an orientable 2-manifold to  $\mathbb{R}^2$  [7]. In this paper, we contribute to the general understanding of well groups by studying the real-valued case. Specifically,

- I. we characterize the well group of  $f : \mathbb{X} \rightarrow \mathbb{R}$  when the space  $\mathbb{A}$  is a single point;
- II. we give an algorithm relating the well diagram of  $f$  and  $\mathbb{A}$  with the extended persistence diagram of  $f$ .

In the full version of this paper, we extend these results to the case when  $\mathbb{A}$  is a finite union of points and intervals. Applications of this theoretical work can be found in scientific visualization, where data in the form of real-valued functions is common. To mention one example, the magnetic resonance image of a person's brain results in a

\* This research is partially supported by the Defense Advanced Research Projects Agency (DARPA), under grants HR0011-05-1-0057 and HR0011-09-0065, as well as the National Science Foundation (NSF), under grant DBI-0820624.

3-dimensional array of intensity values, best viewed as a function from the unit cube to the real numbers. Generically, the preimage of a value  $a \in \mathbb{R}$  is a 2-manifold, referred to as a *contour* or an *isosurface* [10][12]. We contribute to the state of the art by

- III. explaining how the homology of an isosurface can be read off the extended persistence diagram of the function;
- IV. describing how the robustness of features in isosurfaces can be read off the same diagram.

We believe that these results warrant the development of extended persistence diagrams as a new interface tool to efficiently select interesting collections of isosurfaces. We view this tool as complementary to the contour spectra described in [11], which plot continuously varying quantities, such as area and volume, across the sequence of level sets. The most novel aspect of this new tool is the robustness information, which is readily available through subregions of the diagram.

*Outline.* In Section 2 we review necessary background on persistence diagrams and well groups. In Section 3 we present our main result: the characterization of the well groups. In Section 4 we explain the relationship between the points in the persistence diagram and the homology of level sets, extending it to robust homology in Section 5. Finally, Section 6 concludes the paper with a brief discussion of further research directions.

## 2 Background

This section begins with a review of persistence and persistence diagrams. Then we give a description of the 1-parameter family of well groups, and conclude with an example that illustrates the given definitions.

*Persistence.* The persistence of homology classes along a filtration of a topological space can be defined in a quite general context [5]. For this paper, we need only a particular type of filtration, one defined by the sublevel sets of a tame function. Given a real-valued function  $f$  on a compact topological space  $\mathbb{X}$ , we consider the filtration of  $\mathbb{X}$  via the *sublevel sets*  $\mathbb{X}_r(f) = f^{-1}(-\infty, r]$ , for all real values  $r$ . Whenever  $r \leq s$ , the inclusion  $\mathbb{X}_r(f) \hookrightarrow \mathbb{X}_s(f)$  induces maps on the homology groups  $H_p(\mathbb{X}_r(f)) \rightarrow H_p(\mathbb{X}_s(f))$ , for each dimension  $p$ . Here we will use  $\mathbb{Z}/2\mathbb{Z}$  coefficients. Often we will suppress the dimension from our notation, writing  $H(\mathbb{X}_r(f)) = \bigoplus_p H_p(\mathbb{X}_r(f))$ ; in this case, a map  $H(\mathbb{X}_r(f)) \rightarrow H(\mathbb{X}_s(f))$  will of course decompose into maps on each factor. A real value  $r$  is called a *homological regular value* of  $f$  if there exists an  $\varepsilon > 0$  such that the inclusion  $\mathbb{X}_{r-\delta}(f) \hookrightarrow \mathbb{X}_{r+\delta}(f)$  induces an isomorphism between homology groups for all  $\delta < \varepsilon$ . If  $r$  is not a homological regular value, then it is a *homological critical value*.

We say that  $f$  is *tame* if it has finitely many homological critical values and if the homology groups of each sublevel set have finite rank. Assuming that  $f$  is tame, we enumerate its homological critical values  $r_1 < r_2 < \dots < r_n$ . Choosing  $n+1$  homological regular values  $s_i$  such that  $s_0 < r_1 < s_1 < \dots < r_n < s_n$ , we put  $\mathbb{X}_i = \mathbb{X}_{s_i}(f)$ . We

have  $\mathbb{X}_0 = \emptyset$  and  $\mathbb{X}_n = \mathbb{X}$ , by compactness. The inclusions  $\mathbb{X}_i \hookrightarrow \mathbb{X}_j$  induce maps  $f^{i,j} : H(\mathbb{X}_i) \rightarrow H(\mathbb{X}_j)$  for  $0 \leq i \leq j \leq n$  and give the following filtration:

$$0 = H(\mathbb{X}_0) \rightarrow H(\mathbb{X}_1) \rightarrow \dots \rightarrow H(\mathbb{X}_n) = H(\mathbb{X}). \quad (1)$$

Given a class  $\alpha \in H(\mathbb{X}_i)$ , we say that  $\alpha$  is *born* at  $\mathbb{X}_i$  if  $\alpha \notin \text{im } f^{i-1,i}$ . A class  $\alpha$  born at  $\mathbb{X}_i$  is said to *die* entering  $\mathbb{X}_j$  if  $f^{i,j}(\alpha) \in \text{im } f^{i-1,j}$  but  $f^{i,j-1}(\alpha) \notin \text{im } f^{i-1,j-1}$ . We remark that if a class  $\alpha$  is born at  $\mathbb{X}_i$ , then every class in the coset  $[\alpha] = \alpha + \text{im } f^{i-1,i}$  is born at the same time. Of course, whenever such an  $\alpha$  dies entering  $\mathbb{X}_j$ , the entire coset  $[\alpha]$  also dies with it.

*Extended persistence.* Note that the filtration in (1) begins with the zero group but ends with a potentially nonzero group. Hence, it is possible to have classes that are born but never die. We call these *essential* classes, as they represent the actual homology of the space  $\mathbb{X}$ . To measure the persistence of the essential classes, we follow [4] and extend (1) using relative homology groups. More precisely, we consider for each  $i$  the *superlevel set*  $\mathbb{X}^i = f^{-1}[s_{n-i}, \infty)$ . For  $i \leq j$ , the inclusion  $\mathbb{X}^i \hookrightarrow \mathbb{X}^j$  induces a map on relative homology  $H(\mathbb{X}, \mathbb{X}^i) \rightarrow H(\mathbb{X}, \mathbb{X}^j)$ . We have  $\mathbb{X}^0 = \emptyset$  and  $\mathbb{X}^n = \mathbb{X}$  by compactness. These maps therefore lead to the extended filtration:

$$\begin{aligned} 0 &= H(\mathbb{X}_0) \rightarrow H(\mathbb{X}_1) \rightarrow \dots \rightarrow H(\mathbb{X}_n) = H(\mathbb{X}) \\ &= H(\mathbb{X}, \mathbb{X}^0) \rightarrow H(\mathbb{X}, \mathbb{X}^1) \dots \rightarrow H(\mathbb{X}, \mathbb{X}^n) = 0. \end{aligned} \quad (2)$$

We extend the notions of birth and death in the obvious way. Since this filtration begins and ends with the zero group, all classes eventually die.

The information contained within the extended filtration (2) can be compactly represented by *persistence diagrams*  $Dgm_p(f)$ , one for each dimension  $p$  in homology; see Figure 1. Each such diagram is a multiset of points in the plane: it contains one point  $(r_i, r_j)$  for each coset of classes that is born at  $\mathbb{X}_i$  or  $(\mathbb{X}, \mathbb{X}^{n-i+1})$ , and dies entering  $\mathbb{X}_j$  or  $(\mathbb{X}, \mathbb{X}^{n-j+1})$ . In some circumstances, it is convenient to add the points on the diagonal to the diagram, but in this paper, we will refrain from doing so. The persistence diagram contains three important subdiagrams, corresponding to three different combinations of birth and death location. The *ordinary subdiagram*,  $Ord_p(f)$ , represents classes that are born and die during the first half of (2). The *relative subdiagram*,  $Rel_p(f)$ , represents classes that are born and die during the second half. Finally, the *extended subdiagram*,  $Ext_p(f)$ , represents classes that are born during the first half and die during the second half of the extended filtration. Note that points in  $Ord_p(f)$  all lie above the main diagonal while points in  $Rel_p(f)$  all lie below. On the other hand,  $Ext_p(f)$  may contain points on either side of the main diagonal. By  $Dgm(f)$ , we mean the points of all diagrams in all dimensions, overlaid as one multiset of points.

Note that the number of points in  $Ext_p(f)$  is precisely the rank of the  $p$ -th homology group of  $\mathbb{X}$ . A similar formula holds for the sublevel set  $\mathbb{X}_r(f)$ . Using levelset zigzag modules introduced in [3], we will see that this way of reading the rank of homology groups can be extended to level sets and, more generally, to sets of the form  $f^{-1}[a, b]$ .

*Well groups.* Given a continuous function  $f : \mathbb{X} \rightarrow \mathbb{R}$  and a value  $a \in \mathbb{R}$ , we review the definition of the well groups  $U_p(a, r)$  for each radius  $r \geq 0$  and each dimension  $p$ . Since  $a$  will be fixed, we usually drop it from the notation and simply write  $U(r)$ , by which we mean the direct sum of groups  $U_p(a, r)$ , over all homology dimensions  $p$ . We will need the assumption that  $f^{-1}(a)$  has homology groups of finite rank.

To begin, we define the *radius function*  $f_a : \mathbb{X} \rightarrow \mathbb{R}$  by mapping each point  $x$  to  $f_a(x) = |f(x) - a|$ . Using this real-valued function, we filter  $\mathbb{X}$  via sublevel sets:  $\mathbb{X}_r(f_a) = f_a^{-1}[0, r]$ . For  $r \leq s$ , there is a map  $f^{r,s} : H(\mathbb{X}_r(f_a)) \rightarrow H(\mathbb{X}_s(f_a))$ . By an *r-perturbation*  $h$  of  $f$ , we mean a function  $h : \mathbb{X} \rightarrow \mathbb{R}$  such that  $\|h - f\|_\infty = \sup_{x \in \mathbb{X}} |h(x) - f(x)| \leq r$ . The preimage of  $a$  under any such  $h$  will obviously be a subset of  $\mathbb{X}_r(f_a)$ , and hence there is a map on homology,  $j_h : H(h^{-1}(a)) \rightarrow H(\mathbb{X}_r(f_a))$ . Given a class  $\alpha \in H(\mathbb{X}_r(f_a))$ , we say that  $\alpha$  is *supported* by  $h$  if  $\alpha \in \text{im } j_h$ . Equivalently,  $h^{-1}(a)$  carries a chain representative of  $\alpha$ . The *well group*  $U(r) \subseteq H(\mathbb{X}_r(f_a))$  is then defined to consist of the classes that are supported by all  $r$ -perturbations of  $f$ :

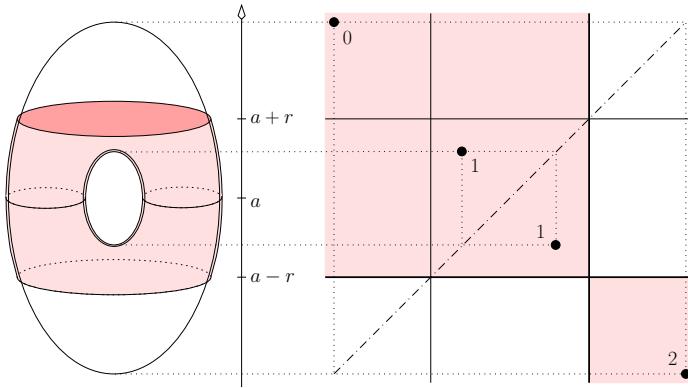
$$U(r) = \bigcap_{\|h-f\|_\infty \leq r} \text{im } j_h.$$

For  $r \leq s$ , the map  $f^{r,s}$  restricts to  $U(r) \rightarrow H(\mathbb{X}_s(f_a))$ . On the other hand,  $H(\mathbb{X}_s(f_a))$  contains  $U(s)$  as a subgroup. It can be shown that  $U(s) \subseteq f^{r,s}(U(r))$  whenever  $r \leq s$ ; see [8]. In other words, the rank of the well group can only decrease as the radius increases.

We call a value of  $r$  at which the rank of the well group decreases a *terminal critical value*. The *well diagram* of  $f$  and  $a$  is then the multiset of terminal critical values of  $f_a$ , taking a value  $k$  times if the rank of the well group drops by  $k$  at the value. Here we note that well groups can be defined in a more general context [8], given a mapping  $f : \mathbb{X} \rightarrow \mathbb{Y}$ , a subspace  $\mathbb{A} \subseteq \mathbb{Y}$ , and a metric space of perturbations. In this general setting, the relationship between the terminal critical values and the homological critical values of  $f_a$  is not completely understood. However, for  $\mathbb{Y} = \mathbb{R}$  and  $\mathbb{A} = \{a\}$ , we will see shortly that the former is a subset of the latter.

*Example.* Consider the torus  $\mathbb{X}$ , as shown in Figure II, along with the vertical height function  $f$  and a value  $a \in \mathbb{R}$ . The preimage of  $a$ ,  $f^{-1}(a) = f_a^{-1}(0)$ , consists of two disjoint circles on the torus; hence there are two components and two independent 1-cycles, all belonging to the well group. For small values of  $r$ ,  $\mathbb{X}_r(f_a)$  consists of two disjoint cylinders. The homology has yet to change; furthermore, although the proof will come later, all classes still belong to the well group.

Now consider the value of  $r$  shown in Figure II. For this  $r$ , the sublevel set  $\mathbb{X}_r(f_a)$  consists of two pair-of-pants glued together along two common circles. We note that  $H_0(\mathbb{X}_r(f_a))$  has dropped in rank by one, while the rank of  $H_1(\mathbb{X}_r(f_a))$  has grown to three. In contrast, the rank of  $U_1(r)$  is less than or equal to one. Indeed, the function  $h : \mathbb{X} \rightarrow \mathbb{R}$ , defined by  $h = f - r$ , is an  $r$ -perturbation of  $f$  and its level set at  $a$ ,  $h^{-1}(a) = f^{-1}(a + r)$ , is a single closed curve. Since the rank of the first homology group of that curve is one, and since the rank of  $\text{im } j_h$  can be no bigger than this rank, the well group  $U_1(r)$  can also have rank at most one. That it does in fact have rank exactly one will follow from our results in the next section.



**Fig. 1.** Left: the torus and the preimage of the interval  $[a - r, a + r]$ . Right: the persistence diagram of the vertical height function. Each point is labeled by the dimension of the corresponding homology class.

### 3 Characterization

In this section, we characterize the well groups. We begin with a consequence of the exactness of the Mayer-Vietoris sequence, see eg. [11], which will provide the main technical ingredient of our proof.

*Mayer-Vietoris sequence.* For convenience, we establish the following notational convention, wherein we reuse the same letter in different fonts. If  $\mathbb{X} \subseteq \mathbb{Y}$  are topological spaces, then inclusion induces a map  $x : H(\mathbb{X}) \rightarrow H(\mathbb{Y})$  on homology groups and we write  $X = \text{im } x$  for the image of this map. Note that  $X$  is always a subgroup of  $H(\mathbb{Y})$ , namely the subgroup of homology classes that have a chain representative carried by  $\mathbb{X}$ . Note also that the rank of  $X$  can never exceed the rank of  $H(\mathbb{X})$ . Suppose that  $\mathbb{W} \subseteq \mathbb{X}$  are two subspaces of  $\mathbb{Y}$ . Then, from the chain of maps  $H(\mathbb{W}) \rightarrow H(\mathbb{X}) \rightarrow H(\mathbb{Y})$ , we see that  $W$  must be a subgroup of  $X$ . The following lemma is a direct consequence of the exactness of the Mayer-Vietoris sequence. However, we will use it often so it seems reasonable to state it formally.

**1 (Mayer-Vietoris Lemma)** *Suppose that we can write a topological space  $\mathbb{Y}$  as  $\mathbb{Y} = \mathbb{C} \cup \mathbb{D}$ , with  $\mathbb{E} = \mathbb{C} \cap \mathbb{D}$ . If a class  $\alpha \in H(\mathbb{Y})$  belongs to  $C$  as well as to  $D$ , then  $\alpha$  also belongs to  $E$ .*

PROOF. Following our convention, we use the notation  $c : H(\mathbb{C}) \rightarrow H(\mathbb{Y})$  for the map on homology induced by the inclusion of  $\mathbb{C}$  in  $\mathbb{Y}$ . Similarly, we write  $d : H(\mathbb{D}) \rightarrow H(\mathbb{Y})$  and  $e : H(\mathbb{E}) \rightarrow H(\mathbb{Y})$ , as well as  $e_c : H(\mathbb{E}) \rightarrow H(\mathbb{C})$  and  $e_d : H(\mathbb{E}) \rightarrow H(\mathbb{D})$ . Note that  $C = \text{im } c$ ,  $D = \text{im } d$ , and  $E = \text{im } e$ . Consider now the relevant portion of the Mayer-Vietoris sequence for the union  $\mathbb{Y} = \mathbb{C} \cup \mathbb{D}$ :

$$H(\mathbb{E}) \xrightarrow{(e_c, e_d)} H(\mathbb{C}) \oplus H(\mathbb{D}) \xrightarrow{c-d} H(\mathbb{Y}).$$

By assumption,  $\alpha \in C$ , so there exists some  $\alpha_c \in H(C)$  such that  $c(\alpha_c) = \alpha$ . Similarly, there exists an  $\alpha_d \in H(D)$  such that  $d(\alpha_d) = \alpha$ . This implies that the pair  $(\alpha_c, \alpha_d)$  belongs to the kernel of  $c - d$ , and thus also, by exactness of the sequence, belongs to the image of  $(e_c, e_d)$ . Hence there exists  $\alpha_e \in H(E)$  with  $e_c(\alpha_e) = \alpha_c$  and  $e_d(\alpha_e) = \alpha_d$ . In particular, since  $e = c \circ e_c$ , we have  $e(\alpha_e) = \alpha$ , and therefore  $\alpha \in E$  as claimed.  $\square$

In the typical application of the Mayer-Vietoris Lemma, we will construct subspaces  $B_0 \subseteq C$  and  $B_1 \subseteq D$  such that  $\alpha \in B_0 \cap B_1$ . From the remark above, we know that  $B_0 \subseteq C$  and  $B_1 \subseteq D$ . The lemma then applies and we can conclude that  $\alpha \in E$ , as before.

*One-point case.* We now suppose that we have a topological space  $\mathbb{X}$  and a function  $f : \mathbb{X} \rightarrow \mathbb{R}$ , and we find the well groups  $U(a, r) = U(r)$ . Recall that  $\mathbb{X}_r(f_a) = f_a^{-1}[0, r] = f^{-1}[a - r, a + r]$ . To state the formula, we distinguish two particular subspaces of  $\mathbb{X}_r(f_a)$ , namely the top level set,  $\mathbb{B}_{0,r} = f^{-1}(a + r)$ , and the bottom level set,  $\mathbb{B}_{1,r} = f^{-1}(a - r)$ . Using the convention from before, we write  $B_{0,r}$  and  $B_{1,r}$  for the images of  $H(\mathbb{B}_{0,r})$  and  $H(\mathbb{B}_{1,r})$  in  $H(\mathbb{X}_r(f_a))$ .

**2 (One-Point Formula)**  $U(r) = B_{0,r} \cap B_{1,r}$ , for every  $r \geq 0$ .

PROOF. We simplify notation by fixing  $r$  and dropping it from our notation. We prove equality by proving the two inclusions in turn. To show  $U \subseteq B_0 \cap B_1$ , consider a class  $\alpha \in U$ . We define  $h_0 = f - r$  and  $h_1 = f + r$  and note that they are  $r$ -perturbations of  $f$ , with  $h_0^{-1}(a) = \mathbb{B}_0$  and  $h_1^{-1}(a) = \mathbb{B}_1$ . By definition of the well group,  $\alpha$  is supported by every  $r$ -perturbation of  $f$ , and therefore by  $h_0$  and by  $h_1$ . It follows that  $\alpha \in B_0 \cap B_1$ .

To show  $B_0 \cap B_1 \subseteq U$ , we consider a class  $\alpha \in B_0 \cap B_1$  and let  $h$  be an arbitrary  $r$ -perturbation of  $f$ . To finish the proof, we just need to show that  $\alpha$  is supported by  $h$ . We define  $C = \{x \in \mathbb{X}_r(f_a) \mid h(x) \geq a\}$  and  $D = \{x \in \mathbb{X}_r(f_a) \mid h(x) \leq a\}$ . Note that  $C \cup D = \mathbb{X}_r(f_a)$  while  $C \cap D = h^{-1}(a)$ . Furthermore, the inequality  $\|h - f\|_\infty \leq r$  implies that  $\mathbb{B}_0 \subseteq C$  and  $\mathbb{B}_1 \subseteq D$ . By the Mayer-Vietoris Lemma,  $\alpha$  is supported by  $h^{-1}(a)$ , as required.  $\square$

We note that the One-Point Formula implies that the well group for a Morse function  $f$  can change only at critical values of the function  $f_a$ . In other words, terminal critical values are, in this simple context, just ordinary critical values. Indeed, if  $[r, s]$  is an interval that contains no critical values of  $f_a$ , then there is a deformation retraction  $\mathbb{X}_s(f_a) \rightarrow \mathbb{X}_r(f_a)$  providing an isomorphism  $H(\mathbb{X}_r(f_a)) \rightarrow H(\mathbb{X}_s(f_a))$ . Furthermore, this retraction maps  $\mathbb{B}_{0,s}$  onto  $\mathbb{B}_{0,r}$ , in such a way that the images of  $H(\mathbb{B}_{0,r})$  and  $H(\mathbb{B}_{0,s})$  in  $H(\mathbb{X}_s(f_a))$  are identical. Similarly, the images of  $H(\mathbb{B}_{1,r})$  and  $H(\mathbb{B}_{1,s})$  in  $H(\mathbb{X}_s(f_a))$  are identical. Hence the well groups  $U(r)$  and  $U(s)$  are isomorphic.

## 4 Combinatorics of Homology

We note that the groups relevant to the One-Point Formula are all groups of a very particular type. Namely, each is the image, under a map induced by inclusion, of the homology of a level set of  $f$ . In this section, we describe a relationship between the

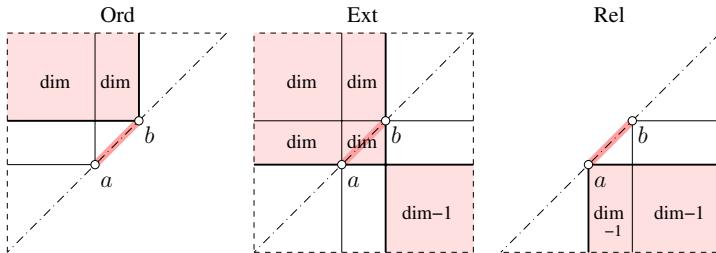
points in the extended persistence diagram of  $f$  and the homology groups of any level set. More generally, we show how the homology of the preimage of any interval can be read from the extended persistence diagram. We also give a similar relationship for the image induced by the inclusion of a smaller interval into a bigger one.

*Common basis.* The main idea here, stated intuitively, is that each point in the persistence diagram corresponds to a unique basis vector of an abstract vector space in such a way that the points in certain subregions of the diagram give crucial information. Slightly more precisely, we choose a persistence module basis  $\mathcal{B}$  for the extended filtration ②, one which results from transforming a basis of the levelset zigzag module in the way described by the Pyramid Basis Theorem; for complete precision, we refer the reader to the full version of this paper. Following ⑥ and ③, we note that these basis vectors correspond bijectively to the points in the extended persistence diagram. We then define an abstract vector space  $V = \langle \mathcal{B} \rangle$ . By  $\mathcal{V}$ , we will mean the collection of those particular vector subspaces of  $V$  that have a basis consisting of vectors chosen from  $\mathcal{B}$ ; in other words,  $\mathcal{V} = \{\langle \mathcal{B}' \rangle \mid \mathcal{B}' \subseteq \mathcal{B}\}$ .

Now suppose that we have a pair of real numbers  $a \leq b$  and consider the homology of  $f^{-1}[a, b]$ , the *interlevel set* defined by  $[a, b]$ . For convenience, we assume that  $a$  and  $b$  are different from all coordinates of points in  $\text{Dgm}(f)$ . We will demonstrate shortly that a basis for  $H(f^{-1}[a, b])$  can be read directly off the extended persistence diagram. To formulate this claim, we define two multisets of points:

$$\begin{aligned} \mathcal{L}_p[a, b] &= \{(x, y) \in \text{Ord}_p(f) \mid x < b, y > b\} \sqcup \{(x, y) \in \text{Ext}_p(f) \mid x < b, y > a\}, \\ \mathcal{R}_p[a, b] &= \{(x, y) \in \text{Ext}_p(f) \mid x > b, y < a\} \sqcup \{(x, y) \in \text{Rel}_p(f) \mid x > a, y < a\}, \end{aligned}$$

for every dimension  $p$ ; see Figure ②. It will be convenient to glue the domains of the three subdiagrams and draw the result as a right-angled triangle, as in Figure ③. In this



**Fig. 2.** From left to right: the shaded regions in the ordinary, extended, and relative subdiagrams in which the points correspond to the basis of the homology of the interlevel set defined by  $[a, b]$

triangle, the birth and death axes go from  $-\infty$  up to  $+\infty$  and then continue on back to  $-\infty$ . In other words, we flip the extended subdiagram upside down and glue its (formerly) upper side to the upper side of the ordinary subdiagram. Similarly, we rotate the relative subdiagram by 180 degrees and glue its (formerly) right side to the right side of the extended subdiagram. After gluing the three domains, we rotate the design by  $-45$  degrees so the triangle rests on its longest side, consisting of the diagonals in the

ordinary and relative subdiagrams. The diagonal of the extended subdiagram is now the vertical symmetry axis passing through the middle of the triangle. We note that there is a straightforward translation of this triangular design to the representation of persistence advocated in [2]. Namely, draw a symmetric right-angled triangle downward from each point in the multiset and call the horizontal lower edge the corresponding *bar*. The *barcode* is the multiset of bars, one for each point in the diagram.

*Reading homology.* The purpose of the multisets  $\mathcal{L}_p[a, b]$  and  $\mathcal{R}_p[a, b]$  is to offer a convenient way to read the homology of a level set or an interlevel set from the persistence diagram. We make this statement precise in the following lemma, which is a corollary of the Pyramid Basis Theorem given in the full version of this paper.

**3 (Interlevel Set Lemma)** *For each dimension  $p$  and each pair of real numbers  $a \leq b$ , there exists an isomorphism taking  $H_p(f^{-1}[a, b])$  onto the vector space  $G \in \mathcal{V}$  spanned by the basis vectors corresponding to the points in  $\mathcal{L}_p[a, b] \cup \mathcal{R}_{p+1}[a, b]$*

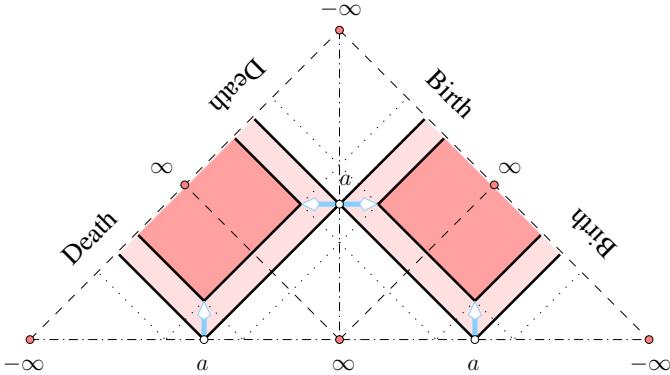
Recall that the points in  $\text{Ext}_p(f)$  determine the homology of  $\mathbb{X}$ . This is a special case of the lemma. To get  $f^{-1}[a, b] = \mathbb{X}$ , we choose  $a$  smaller than the minimum function value and  $b$  larger than the maximum function value. Hence,  $\mathcal{L}_p[a, b] = \text{Ext}_p(f)$  and  $\mathcal{R}_p[a, b] = \emptyset$  for all dimensions  $p$ , as required. Of course the homology of a level set  $f^{-1}(a)$  can also be read off via the Interlevel Set Lemma; one simply sets  $a = b$  and makes the necessary adaptations to the formula.

Now suppose we have a pair of nested intervals  $[a, b] \subseteq [c, d]$ . By the Interlevel Set Lemma, there are isomorphisms that take the homology groups  $H(f^{-1}[a, b])$  and  $H(f^{-1}[c, d])$  onto groups  $G, G' \in \mathcal{V}$ , respectively. The inclusion of the smaller into the larger interval induces a map on homology, which composes with the isomorphisms obtained from the Interlevel Set Lemma to give  $g : G \rightarrow G'$ . Since the two groups are members of  $\mathcal{V}$ , there is a natural map from  $G$  to  $G'$ , namely the one that restricts to the identity on the span of their shared vectors and is zero otherwise. Not surprisingly,  $g$  is exactly that map. We give the proof of this result in the full version of this paper.

**4 (Interval Mapping Lemma)** *Let  $[a, b] \subseteq [c, d]$  and let  $G, G'$  be the corresponding groups in  $\mathcal{V}$ . Then the image of the map  $g : G \rightarrow G'$  is in  $\mathcal{V}$ , with basis  $B(\text{im } g)$  in bijection with the multiset  $(\mathcal{L}_p[a, b] \cap \mathcal{L}_p[c, d]) \cup (\mathcal{R}_{p+1}[a, b] \cap \mathcal{R}_{p+1}[c, d])$ .*

## 5 Combinatorics of Robustness

This section gives a procedure for reading the well diagrams from the persistence diagram for  $f$ . The homology of  $\mathbb{X}_0(f_a) = f^{-1}(a)$  can be read off the persistence diagram of  $f$ , as stated in the Interlevel Set Lemma. Specifically,  $H_p(\mathbb{X}_0(f_a))$  is isomorphic to the vector space whose basis corresponds to  $\mathcal{L}_p[a, a] \cup \mathcal{R}_{p+1}[a, a]$ . Similarly, the homology of  $\mathbb{X}_r(f_a) = f^{-1}[a - r, a + r]$  can be read off the same diagram. By the One-Point Formula, the well group for  $r$  is the intersection of the images of the homology maps induced by the inclusions of  $f^{-1}(a - r)$  and  $f^{-1}(a + r)$  in  $\mathbb{X}_r(f_a)$ . By the Interval Mapping Lemma, this intersection corresponds to a pair of rectangles within the region of  $\mathbb{X}_0(f_a)$ ; see Figure 3.



**Fig. 3.** The triangle design of the extended persistence diagram. The shaded region gives the basis of  $H(f^{-1}(a))$ , while the dark shaded region gives the basis of  $U(a, r)$ .

A point contributes to the well group until  $r$  reaches a value at which the pair of rectangles no longer contains the point. For a point  $(x, y) \in \mathcal{L}_p[a, a]$ , this value of  $r$  is  $\min\{a - x, y - a\}$ , and for  $(x, y) \in \mathcal{R}_{p+1}[a, a]$ , this value is  $\min\{x - a, a - y\}$ . The well diagram is the multiset of the values we get from the points in the persistence diagram.

## 6 Discussion

The main contribution of this paper is a characterization of the well groups of real-valued functions and a recipe for deriving their well diagrams from the extended persistence diagram of the function. These results have ramifications in scientific visualization, in particular in the selection and display of isosurfaces. We conclude this paper by formulating two directions for further research.

The general problem of well group computation remains wide open. One way to think about this is the following. If we have a mapping  $f : \mathbb{X} \rightarrow \mathbb{Y}$  from an  $m$ -dimensional topological space  $\mathbb{X}$  to an  $n$ -dimensional topological space  $\mathbb{Y}$ , and a submanifold  $\mathbb{A} \subseteq \mathbb{Y}$  of dimension  $k$ , we call the computation of the well groups a variant of the  $(m, n, k)$  problem. The full version of this paper provides a complete solution for  $(m, 1, 0)$  and  $(m, 1, 1)$ , when  $\mathbb{Y} = \mathbb{R}$ . In [7], the authors give an algorithm for  $(2, 2, 0)$ , when  $\mathbb{X}$  is an orientable 2-manifold and  $\mathbb{Y} = \mathbb{R}^2$ . Their algorithm extends to  $(m, n, n - m)$ . Everything else is as yet unsolved.

The use of well diagrams to provide local measures of robustness for isosurfaces is a promising research direction in scientific visualization. From the extended persistence diagram drawn as in Figure 3, we obtain a compact representation of all homology groups and the robustness of their classes. Can this rich representation of information be effectively used to design transfer functions [9][13] for highlighting important features in 3-dimensional data sets?

## References

1. Bajaj, C.L., Pascucci, V., Schikore, D.R.: The contour spectrum. In: Proc. 8th IEEE Conf. Visualization, pp. 167–173 (1997)
2. Carlsson, G., Collins, A., Guibas, L.J., Zomorodian, A.: Persistence barcodes for shapes. *Internat. J. Shape Modeling* 11, 149–187 (2005)
3. Carlsson, G., de Silva, V., Morozov, D.: Zigzag persistent homology and real-valued functions. In: Proc. 25th Ann. Sympos. Comput. Geom., pp. 247–256 (2009)
4. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Extending persistence using Poincaré and Lefschetz duality. *Found. Comput. Math.* 9, 79–103 (2009)
5. Edelsbrunner, H., Harer, J.: Persistent homology — a survey. In: Goodman, J.E., Pach, J., Pollack, R. (eds.) *Surveys on Discrete and Computational Geometry. Twenty Years Later*. Contemporary Mathematics, vol. 453, pp. 257–282. Amer. Math. Soc., Providence (2008)
6. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. *Discrete Comput. Geom.* 28, 511–533 (2002)
7. Edelsbrunner, H., Morozov, D., Patel, A.: The stability of the apparent contour of an orientable 2-manifold. In: Pascucci, V., Tierny, J. (eds.) *Topological Methods in Data Analysis and Visualization: Theory, Algorithms, and Applications*. Springer, Heidelberg (to appear)
8. Edelsbrunner, H., Morozov, D., Patel, A.: Quantifying transversality by measuring the robustness of intersections. Dept. Comput. Sci., Duke Univ., Durham, North Carolina (2009) (Manuscript)
9. Fang, S., Biddlecome, T., Tuceryan, M.: Image-based transfer function design for data exploration in volume visualization. In: Proc. 9th IEEE Conf. Visualization, pp. 319–326 (1998)
10. van Krefeld, M., van Oostrum, R., Bajaj, C.L., Pascucci, V., Schikore, D.R.: Contour trees and small seed sets for isosurface traversal. In: Proc. 13th Ann. Sympos. Comput. Geom., pp. 212–220 (1997)
11. Munkres, J.R.: *Elements of Algebraic Topology*. Perseus, Cambridge (1984)
12. Newman, T.S., Yi, H.: A survey of the marching cube algorithm. *Computers and Graphics* 30, 854–879 (2006)
13. Wittenbrink, C.M., Malzbender, T., Goss, M.E.: Opacity-weighted color interpolation for volume sampling. In: Proc. IEEE Proc. Volume Visualization, pp. 135–142 (1998)

# Solving an Avionics Real-Time Scheduling Problem by Advanced IP-Methods<sup>\*</sup>

Friedrich Eisenbrand<sup>1</sup>, Karthikeyan Kesavan<sup>2</sup>, Raju S. Mattikalli<sup>2</sup>,  
Martin Niemeier<sup>1</sup>, Arnold W. Nordsieck<sup>2</sup>, Martin Skutella<sup>3</sup>,  
José Verschae<sup>3</sup>, and Andreas Wiese<sup>3</sup>

<sup>1</sup> EPFL, Lausanne, Switzerland

<sup>2</sup> Boeing, USA

<sup>3</sup> TU Berlin, Germany

**Abstract.** We report on the solution of a real-time scheduling problem that arises in the design of software-based operation control of aircraft. A set of tasks has to be distributed on a minimum number of machines and offsets of the tasks have to be computed. The tasks emit jobs periodically starting at their offset and then need to be executed on the machines without any delay. Also, further constraints in terms of memory usage and redundancy requirements have to be met. Approaches based on standard integer programming formulations fail to solve our real-world instances. By exploiting structural insights of the problem we obtain an IP-formulation and primal heuristics that together solve the real-world instances to optimality and outperform text-book approaches by several orders of magnitude. Our methods lead, for the first time, to an industry strength tool to optimally schedule aircraft sized problems.

## 1 Introduction

Modern aircraft computing systems are employing new architectures that provide significant weight and cost advantages over previous architectures. They include computing, network, and I/O modules that are highly configurable. However these architectures present integration complexities which require automated methods to achieve configuration design centering. One such complexity is the allocation and scheduling of applications on processors. Due to the hard real-time nature of the airplane systems, a static cyclic execution schedule is required. For allocation purposes, applications are characterized by performance, memory, I/O, operational availability, functional separation, and functional grouping requirements. The processors are characterized by performance as implemented in a schedule, limited memory capabilities, and limited I/O capabilities via the network. A significant challenge associated with solving this problem in the context of a commercial aircraft is that of scale – it requires careful consideration of problem formulation and solution efficiency.

\* This work was partially supported by Berlin Mathematical School, by DFG research center MATHEON in Berlin, by DFG Focus Program 1307 within the project “Algorithm Engineering for Real-time Scheduling and Routing”, and by the Swiss National Science Foundation.

We report on integer programming approaches for this problem. It turns out that textbook formulations, like the time-indexed formulation, are not well suited to tackle the problems of complexity arising in real-world applications. By exploiting structural insights of the problem we provide an integer programming model and primal heuristics that outperform the textbook approach significantly. By restricting to a relevant subclass of instances and exploiting a *bin-tree* structure [EHN<sup>+</sup>10], we obtain a model that is tailored to this subclass and outperforms the other formulations drastically. The latter formulation is able to handle industrial size instances of the scheduling problem. Even for real-world instances not belonging to the subclass, one can still use the formulation as a heuristic using a rounding technique. For our real-world instance, this heuristic finally provides optimal solutions.

## 1.1 Problem Definition

Our problem is a variant of the *periodic maintenance problem* (PMP) [WL83]. First, we describe a simplified version called the *basic PMP*. It is the core of the real problem that Boeing is challenged with, the *extended PMP*, which we describe afterwards.

In the basic PMP we are given a set of tasks  $\mathcal{T} = \{\tau_1, \dots, \tau_n\}$  where each task  $\tau_i = (c_i, p_i)$  is characterized by its *execution time*  $c_i \in \mathbb{N}$  and *period*  $p_i \in \mathbb{N}$ . The goal is to assign the tasks to identical machines and to compute offsets  $a_i \in \mathbb{N}_0$ . A task  $\tau_i$  generates one *job* with execution time  $c_i$  at every time unit  $a_i + p_i \cdot k$  for all  $k \in \mathbb{N}_0$ . Each job needs to be processed immediately and non-preemptively after its generation on the task's machine. A *collision* occurs if two jobs are simultaneously active on the same machine. A schedule is feasible if no collision occurs. In the sequel, we denote by  $Q = \{q_1, \dots, q_k\}$  the set of all period lengths arising in the respective instance. We assume that  $q_1 < \dots < q_k$ . An important special case, in particular in real-world instances, is the case of *harmonic periods*. In this case for each pair of tasks  $\tau_i, \tau_{i'}$  we have that either  $p_i | p_{i'}$  or  $p_{i'} | p_i$ .

In the extended PMP, machines have additional resource limitations in terms of memory of different types (RAM, ROM, etc.) and communication links that need to be considered. Each task has a given requirement for each type of memory and needs certain communication links to be open on its machine. Each machine can handle only a limited number of links and a limited bandwidth used by them. Like in the basic PMP, all machines are identical.

Moreover, due to system stability requirements certain tasks need to be assigned to different machines. Also, the machines have to be partitioned into two cabinets (left and right). To this end, we are given sets of tasks which have to be distributed evenly among the cabinets in order to design a fail-safe architecture.

## 1.2 Related Work

There is a large amount of literature on real-time scheduling; see, for example, [BHR93, But04, Leu04] for surveys. The periodic maintenance problem was introduced by Wei and Liu [WL83] in the context of single machine and unit

execution times. Baruah et al. [BRTV90] and independently Korst et al. [KALW91, KAL96] show that the periodic maintenance problem is NP-hard. Moreover, minimizing the number of machines is hard to approximate within a factor of  $n^{1-\varepsilon}$  for any  $\varepsilon > 0$  unless  $P = NP$  [EHN<sup>+</sup>10, Bha98].

The (basic) PMP generalizes BIN-PACKING. For Bin-Packings, First-Fit with decreasing item sizes is a 1.5-approximation algorithm [SL94] which is best possible unless  $P = NP$ . For the harmonic case of the PMP a First-Fit heuristic achieves an approximation factor of 2 and it is  $NP$ -hard to approximate it any better [EHN<sup>+</sup>10]. The general case cannot be approximated non-trivially if  $P \neq NP$  [EHN<sup>+</sup>10].

From a computational point of view, there exists extensive literature on heuristics as well as branch-and-bound and column generations methods for Bin-Packings. In [MT90] polynomial time approximation algorithms as well as lower bounds, and exact algorithms are studied. Lower bounds to the optimal solution (which can be computed fast) are presented in [CPPT07]. Several approaches have been proposed for solving the Bin-Packaging problem with branch-and-price techniques; see, e.g., [Van99, VBJN94, VdC99]. An algorithm called BISON is proposed in [SKJ97], where branch-and-bound techniques and tabu search are combined to design an exact hybrid algorithm. There is also a lot of literature on heuristics for the Bin-Packaging problem. For example, Gupta and Ho propose a heuristic that greedily minimizes the slack of the machines. Fleszar and Hindi [FH02] modify these ideas and combine them with variable neighborhood search to design an hybrid algorithm. Moreover, Loh et al. [LGW08] propose a simple local heuristic based in the concept of *weighted annealing*.

## 2 Structural Insights

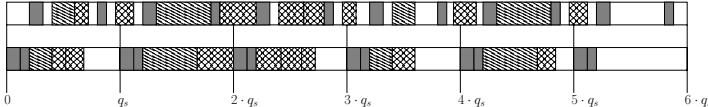
We now review some properties of the (basic) PMP which we will exploit later in our IP-formulations. First, we state a lemma which formulates an algebraic condition for the collision of two tasks, shown by Korst et al.

**Lemma 1 ([KALW91]).** *Let  $\tau_i$  and  $\tau_{i'}$  be two tasks which are scheduled on the same machine with offsets  $a_i \in \mathbb{N}_0$  and  $a_{i'} \in \mathbb{N}_0$ , respectively. They do not collide if and only if*

$$c_{i'} \leq (a_i - a_{i'}) \bmod \gcd(p_i, p_{i'}) \leq \gcd(p_i, p_{i'}) - c_i.$$

We now restrict to the case of harmonic periods and describe some structural properties of this case. First, we sketch the concept of bin-trees which was first introduced in [EHN<sup>+</sup>10].

Assume we have a feasible schedule for an harmonic instance of tasks  $\tau_i = (c_i, p_i)$ ,  $i = 1, \dots, n$  on one machine, given by an offset  $a_i$  for each task. Due to a shifting argument we can show that there exists a feasible schedule in which a task  $\tau_i$  with  $p_i = q_1$  has offset  $a_i = 0$ . This task divides the *hyperperiod*  $[0, q_k]$  (after which the schedule repeats itself) into *bins*  $B_\ell = [\ell \cdot q_1, (\ell + 1) \cdot q_1)$  with  $\ell \in \{0, \dots, q_k/q_1 - 1\}$ . Using an exchange argument we can also show that



**Fig. 1.** A schedule for a single machine and a schedule for the same tasks which is in bin structure. The gray jobs belong to tasks with period length  $q_1$ , the striped jobs to tasks with period length  $q_2 = 3 \cdot q_1$ , and the checkered jobs to tasks with period length  $q_3 = 6 \cdot q_1$ .

w.l.o.g. inside each bin the jobs are ordered by the period length of the tasks which created them. Moreover, the jobs are executed consecutively and all idle time is accumulated at the end of the bin. See [EHN<sup>+</sup>10] for formal proofs of the assumptions made. Figure 1 shows an example schedule with the described adjustments.

An important observation is the following: Consider two bins  $B_\ell = [\ell \cdot q_1, (\ell + 1) \cdot q_1)$  and  $B_{\ell'} = [\ell' \cdot q_1, (\ell' + 1) \cdot q_1)$  such that  $\ell \equiv \ell' \pmod{q_r/q_1}$ . As far as tasks with period length up to  $q_r$  are concerned, these bins look the same. Hence, the whole structure can be represented as a tree. Each node in level  $\ell$  encodes the tasks of period length up to  $q_\ell$  that are scheduled in all its child nodes. We see that if a task with period length  $q_r$  executes a job in a bin  $B_\ell$ , then it executes its other jobs in bins  $B_{\ell'}$  with  $\ell \equiv \ell' \pmod{q_r/q_1}$ .

From any feasible schedule with the structure described above, we can obtain an assignment of tasks to the bins of a machine. We show in the following lemma that the inverse is also true if no bin is overloaded. This allows us to model the basic PMP in terms of assignments of tasks to bins.

**Lemma 2.** Let  $T$  be a set of tasks with period lengths  $q_1 < \dots < q_k$ . Assume that for each task  $\tau_i$  we are given a value  $\ell_i \in \{0, \dots, p_i/q_1 - 1\}$  (assigning the jobs of  $\tau_i$  to bins  $B_{\ell'}$  with  $\ell_i \equiv \ell' \pmod{p_i/q_1}$ ). For each bin  $B_\ell$  with  $\ell \in \{0, \dots, q_k/q_1 - 1\}$  denote by  $T_\ell \subseteq T$  the tasks  $\tau_i$  with  $\ell_i \equiv \ell \pmod{p_i/q_1}$  (i.e., the tasks which run a job in  $B_\ell$ ). If for each bin  $B_\ell$  we have that  $\sum_{\tau_i \in T_\ell} c_i \leq q_1$ , then there is a schedule for the tasks  $T$  on one machine. Moreover, this schedule can be found efficiently.

The proof for this lemma is omitted due to space limitations. To compute a schedule from the assignment of tasks to bins, a greedy type algorithm can be used.

### 3 IP-Formulations

In what follows we describe several formulations for the basic PMP. First we consider a *time indexed formulation*, where we have variables assigning tasks to time slots on each machine. Then, we present a less naïve approach that exploits the algebraic feasibility criterion of Lemma 1. We call this model the *congruence-formulation*. It uses variables that indicate the offset of each task. We also describe a third model, the *bin-formulation*, that is specifically designed for the important case of harmonic periods. This last IP is based on Lemma 2. It uses the concept of bins and directly assigns the tasks to the bins of the machines.

At the end of this section we explain how to model the additional constraints of the extended PMP. For sake of brevity, we define some global variables that are used by all formulations. To this end, let  $\mathcal{M}$  be a set of  $m$  identical machines. Since we are interested in minimizing the number of used machines, we assume that  $m$  is a precomputed upper bound on the total number of needed machines. For example, we can trivially take  $m = |\mathcal{T}|$ . In all our formulations we use variables  $u_j \in \{0, 1\}$  that are equal to one if machine  $M_j \in \mathcal{M}$  is being used by some task. With these variables, the objective function is always to minimize  $\sum_{M_j \in \mathcal{M}} u_j$ .

### 3.1 Time-Indexed-Formulation

Our first formulation is a naïve formulation that uses variables indicating whether a task starts processing at a certain time slot. More precisely, we consider variables  $w_{i,j,t} \in \{0, 1\}$  which have a value of one if machine  $M_j \in \mathcal{M}$  starts processing task  $\tau_i \in \mathcal{T}$  at time  $t$ , and zero otherwise. Linking these variables in a straight forward manner ensures that no two tasks on the same machine collide. The total number of variables is in  $\Theta(|\mathcal{M}| \cdot |\mathcal{T}| \cdot q_k)$  and the number of constraints is in  $\Theta(|\mathcal{T}|^2 \cdot |\mathcal{M}| \cdot q_k^2)$ .

### 3.2 Congruence-Formulation

Now we describe our congruence-formulation for the basic PMP. Its main concept is to introduce integer variables  $a_i$  which model the offset for each task. In order to check whether two tasks collide we derive linear constraints from the feasibility criterion given in Lemma [II](#).

For each task  $\tau_i$  we introduce a variable  $a_i \in \mathbb{N}$  which defines its offset. Additionally, we consider variables  $x_{i,j} \in \{0, 1\}$  that indicate, for each task  $\tau_i \in \mathcal{T}$  and machine  $M_j \in \mathcal{M}$ , whether  $\tau_i$  is assigned to  $M_j$ . To ensure that each task is actually assigned to a machine, we introduce the constraint  $\sum_{M_j \in \mathcal{M}} x_{i,j} = 1$  for each task  $\tau_i \in \mathcal{T}$ .

It remains to ensure that no two tasks  $\tau_i, \tau_{i'}$  on the same machine collide. Lemma [II](#) implies that it suffices to require that there is an integer  $s_{i,i'}$  such that

$$c_{i'} \leq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}) \leq \gcd(p_i, p_{i'}) - c_i.$$

We want to enable the condition above only if two tasks  $\tau_i$  and  $\tau_{i'}$  share a machine. In order to achieve this we introduce variables  $v_{i,i'}$  such that  $v_{i,i'} = 1$  if  $\tau_i$  and  $\tau_{i'}$  are scheduled on the same machine. Hence, for each pair of tasks  $\tau_i$  and  $\tau_{i'}$  we introduce an integral variable  $s_{i,i'}$  and the constraints

$$\begin{aligned} v_{i,i'} \cdot c_{i'} &\leq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}), \\ \gcd(p_i, p_{i'}) - c_i \cdot v_{i,i'} &\geq a_i - a_{i'} + s_{i,i'} \cdot \gcd(p_i, p_{i'}). \end{aligned}$$

Note that if  $v_{i,i'} = 0$  there is always an integral value for  $s_{i,i'}$  such that these constraints are satisfied (independently from the values for  $a_i$  and  $a_{i'}$ ). For the variables  $v_{i,i'}$  we add constraints of the form  $v_{i,i'} \geq x_{i,j} + x_{i',j} - 1$  to ensure that they equal one if two tasks  $\tau_i$  and  $\tau_{i'}$  are scheduled on the same machine.

In total, we have  $\Theta(|\mathcal{T}|^2 + |\mathcal{T}| \cdot |\mathcal{M}|)$  variables and  $\Theta(|\mathcal{T}|^2 \cdot |\mathcal{M}|)$  constraints. The size of the formulation is thus polynomial in the input size.

### 3.3 Bin-Formulation

In this section we consider the case of harmonic period lengths. We make crucial use of the bin-tree concept explained in Section 2 to obtain a strong IP. The main idea is to define variables that model the assignment of tasks to bins on each machine. Then, we add restrictions to ensure that no bin is overloaded. By Lemma 2 this will imply a feasible schedule.

We first describe our model for the single machine case, where we only want to determine whether a set of tasks can be processed on one machine without collisions. In this case we already know the minimum period length of the tasks assigned to the machine, and thus the size of the bins is known to be  $q_1$ . We later generalize the model to the problem of minimizing the number of used machines.

Consider the single machine feasibility problem. Recall that in this case all bins have size  $q_1$ . We introduce a variable  $z_{i,\ell} \in \{0, 1\}$  that determines whether task  $\tau_i \in \mathcal{T}$  is assigned to each bin  $B_\ell$ , with  $\ell \in \{0, \dots, p_i/q_1 - 1\}$ . Notice that we do not need to consider bins with  $\ell \geq p_i/q_1$ , since for these bins the schedule of  $\tau_i$  is repeated periodically. First of all we require that all jobs are assigned to some bin by requiring  $\sum_{\ell=0}^{p_i/q_1-1} z_{i,\ell} = 1$  for all  $\tau_i \in \mathcal{T}$ .

Now we ensure that no bin is overloaded. Notice that if  $\tau_i$  is assigned to bin  $B_\ell$  then this task creates jobs in all bins  $B_{\ell'}$  so that  $\ell' \equiv \ell \pmod{p_i/q_1}$ . Equivalently, a job created by task  $\tau_i$  is processed on bin  $B_\ell$  if and only if  $z_{i,(\ell \pmod{p_i/q_1})} = 1$ . Then, we can guarantee that a bin is not overloaded by imposing a knapsack type constraint

$$\sum_{\tau_i \in \mathcal{T}} c_i \cdot z_{i,(\ell \pmod{p_i/q_1})} \leq q_1 \quad \forall \ell \in \{0, \dots, q_k/q_1 - 1\}.$$

In the multiple machine problem we have the extra difficulty that it is not known a priori which is the smallest period length appearing on each machine. Therefore, the size of the bins on a machine is not determined until all jobs are assigned to the machine.

As before, we consider variables  $z_{i,j,\ell} \in \{0, 1\}$  that indicates whether a task  $\tau_i \in \mathcal{T}$  is assigned to machine  $M_j \in \mathcal{M}$  on bin  $B_\ell$  for  $\ell \in \{0, \dots, q_k/q_1 - 1\}$ . We consider here that bin  $B_\ell$  has size  $q_1$ . Note that in the case that machine  $M_j$  processes no task with period length  $q_1$ , we cannot really consider assignment of jobs to bins of size  $q_1$ , since some job may be partially assigned to more than one bin. We then must “glue” bins of size  $q_1$  together to create new bins of size  $q_2$  or larger. This can be described mathematically by considering the sum of several variables  $z_{i,j,\ell}$ . If, for example,  $q_2 = 2q_1$ , then the variable describing whether a task  $\tau_i$  is assigned to machine  $M_j$  in the first bin of size  $q_2$  is equal to  $z_{i,j,0} + z_{i,j,1}$ .

We generalize the ideas just discussed by introducing dummy variables  $z_{i,j,\ell}^r \in \{0, 1\}$  that are equal to one if  $\tau_i$  is assigned to machine  $j$  on the  $\ell$ -th bin of size  $q_r$ , for  $\ell \in \{0, \dots, q_k/q_r - 1\}$ . Formally, the dummy variables are defined as follows

$$z_{i,j,\ell}^r = \sum_{\ell'= \ell \cdot q_r/q_1}^{(\ell+1) \cdot q_r/q_1 - 1} z_{i,j,\ell'} \quad \forall \tau_i, \forall M_j, \forall r \in \{1, \dots, k\}, \forall \ell \in \{0, \dots, q_r/q_1 - 1\}.$$

In the case that there is a task with period length  $q_r$ , our assignment must satisfy that no bin of size  $q_r$  is overloaded. However, this should not be required for machines that have no job with period length  $q_r$  assigned to it. Therefore, we introduce variables  $d_{j,r} \in \{0, 1\}$  that equal one if there is a tasks with period  $q_r$  assigned to machine  $M_j$ .

$$d_{j,r} \geq \sum_{\ell=0}^{p_i/q_1-1} z_{i,j,\ell} \quad \forall M_j \in \mathcal{M}, \forall r \in \{1, \dots, k\}, \forall \tau_i : p_i = q_r$$

This yields the following inequalities for ensuring that no bin is overloaded.

$$\sum_{\tau_j \in \mathcal{T}} c_i \cdot z_{i,j,t}^r \leq q_r + (1 - d_{j,r})q_k \quad \forall M_j \in \mathcal{M}, \forall \ell \in \{0, \dots, \frac{q_k}{q_r} - 1\}, \forall r = 1, \dots, k$$

Finally, we must link the variables  $u_j$  to the variables  $z_{i,j,\ell}$ . This can be easily done with analogous constraints as in the time-indexed-formulation. Also note that the variables  $x_{i,j}$  can be trivially introduced to our formulation. The bin-formulation needs  $\Theta(|\mathcal{T}| \cdot |\mathcal{M}| \cdot \frac{q_k}{q_1})$  variables and constraints in total (note that the dummy variables do not need to be added explicitly).

### 3.4 Extended Constraints

In order to handle the conditions additionally introduced in the extended PMP we need to add more linear constraints. Due to space limitations we sketch them only briefly. For the memory restrictions we introduce knapsack constraints that model the limited memory on a machine. The constraints that some tasks have to be scheduled on different machines are modeled in a straight-forward manner by suitably linking the variables of the tasks. By introducing variables for the communication links on each machine, we ensure that a machine opens all links which are needed by its tasks. Further knapsack constraints ensure that the total number of links and their total bandwidth does not exceed the resources on each machine. For each available machine we pre-define whether it is in the left or in the right cabinet. We introduce constraints which ensure that sets of tasks are distributed evenly on the cabinets if required.

## 4 Computational Results

In this section we present our computational results. We solved all real-world instances provided by Boeing within minutes, with all constraints of the extended PMP and some additional constraints which we describe later. The most difficult instance has 177 tasks and needs 16 machines. The period lengths are almost harmonic (see details below). Instances of this size are far beyond of what the time-indexed formulation and the congruence formulation can solve in a reasonable amount of time. However, the special design of the bin-formulation allowed to solve the instances within 15 minutes.

For benchmarking purposes we first analyze how our different models perform on random instances. Moreover, we study the quality of solutions obtained by a First-Fit heuristic. The heuristic orders the tasks by period length and execution time and greedily assigns them to the first machine where it can find a suitable start offset. Note that for the basic PMP in the harmonic case this is already a 2-approximation algorithm [EHN<sup>+</sup>10]. In the non-harmonic case one can prove with similar arguments as in [EHN<sup>+</sup>10] that the algorithm uses at most  $2OPT + k - 1$  machines. Reflecting the theoretical results, our benchmarking shows that First-Fit has a good performance in the basic PMP. However, as we will see, it does not cope well with the additional constraints of the extended PMP.

Due to the novelty of our problem, there is no existing standard set of instances for benchmarking. Therefore, we must rely on generating random instances. We consider two ways of generating random instances: pure random instances and random perturbations of real-world instances arising at Boeing. We will call the latter instances the real-world perturbed (RWP) instances. There are four different settings: for the basic PMP, we consider the non-harmonic case with pure random instances, the harmonic case with pure random instances, and the harmonic case with RWP instances. For the extended PMP we benchmark only with RWP instances in the harmonic case.

All computations were done on a two-processor machine with Intel Xeon 2.66 GHz CPUs with 8 GB of RAM, running Linux. We used CPLEX release version 12.1.0.

We remark that additionally we introduce some cuts to the IP formulations. If for two tasks, the sum of their execution times exceeds the greatest common divisor of their periods, Lemma 1 implies that they cannot be assigned to the same machine. Thus we can add separation constraints for these tasks similar to those used in the extended PMP model. Moreover, for any assignment of tasks to a processor, the sum of their total required execution times during the hyperperiod may not exceed the hyperperiod. This is expressed with knapsack type constraints. Notice that in all our IP-formulations we need an upper bound on the number of machines. This was obtained by first running the First-Fit heuristic.

#### 4.1 Non-harmonic Case

In the non-harmonic case we benchmark the following IP-formulations and algorithms: the time-indexed-formulation (TIF), the congruence-formulation (CF), and the First-Fit heuristic (FF). For each pure random instance we drew five different period lengths from the set  $\{2^x \cdot 3^y \cdot 50 \mid x \in \{0, \dots, 4\}, y \in \{0, \dots, 3\}\}$  uniformly at random. This is a typical number of period lengths in real-world instances. For each task  $\tau_i$ , its period length  $p_i$  is chosen uniformly at random from one of the five period lengths. (In our experiments we observed that larger values for the number of period lengths in an instance result in instances which are harder to solve; however, the relation of the running times between the three IP-formulations remains the same.) Its execution time is drawn from an inverse exponential distribution. This results in realistically small execution times in comparison with the period length and hence mimics the real instances from

**Table 1.** The table shows our computational results for the pure random instances in the non-harmonic case

# tasks	IP-formulations		Heuristic	
	CF	TIF	FF	
10	0.11s	98%	–	0%
20	2.52s	92%	–	0%
30	277.41s	42%	–	0%
				1.92%

Boeing. We created 200 random instances each for the case of 10, 20, 30, 40, and 50 tasks. Whenever ten runs in a row did not finish before the timeout of 30 minutes or ran out of memory, we did not consider the respective formulation any further (denoted by dashes in the table).

Table II shows our computational results. In all our tables, for each IP-formulation the left column shows the average running times in seconds. The right column shows the percentage of instances that could be solved to optimality within the time limit. For the First-Fit algorithm, we show the average relative error (in %) of the solutions with respect to the optimal solution. The running time of First-Fit is negligible.

*Discussion.* The First-Fit heuristic apparently performs very well, obtaining the optimal solution most of the time regardless of the number of tasks. This is somewhat surprising given that the problem is theoretically rather difficult (i.e., *NP*-hard to approximate within a factor of  $|\mathcal{T}|^{1-\varepsilon}$ ), see [EHN<sup>+</sup>10]. However, the instances created in that reduction are very special and not likely to arise in our random draws. We notice that TIF is impractical even for small instances due to the huge number of integer variables involved in the formulation. In comparison, CF does much better and is able to solve most instances with up to 30 tasks in reasonable time (less than 30 min.).

## 4.2 Harmonic Case

In the harmonic case we benchmark the following IP-formulations/algorithms: the time-indexed-formulation (TIF), the congruence-formulation (CF), the bin-formulation (BF) and the First-Fit heuristic (FF). In the harmonic case the pure random instances were created by first generating a harmonic sequence of five periods in the following way: We start with period length 50 and successively generate the other periods by multiplying two, three, or six to the previous period. The periods and execution times for the tasks are drawn as in the non-harmonic case. The RWP instances were created by taking tasks uniformly at random from a large harmonic Boeing-instance and perturbing execution time and – for the extended PMP – the memory requirements randomly by up to 25 %. The other extended constraints remain unchanged.

---

<sup>1</sup> We use the shifted geometric mean of running times  $t_i$  calculated by  $(\prod_{i=1}^n (t_i + 1))^{1/n} - 1$ . We use the shift in order to decrease the strong influence of the very easy instances in the mean values.

**Table 2.** Computational results for the pure random instances in the harmonic case (basic PMP)

# tasks	IP-formulations				FF	
	BF	CF	TIF			
10	0.28s (1×)	99%	0.25s (0.9×)	97%	– 0%	0.00%
20	1.8s (1×)	100%	6.54s (3.6×)	90%	– 0%	0.27%
30	8.2s (1×)	97%	369.39s (45.1×)	32%	– 0%	0.06%
40	36.64s (1×)	80%	–	0%	– 0%	0.70%

**Table 3.** Computational results for the RWP instances (harmonic case) for the basic PMP

# tasks	IP-formulations					Heuristic FF
	BF	CF	TIF			
10	0.01s (1×)	100%	0.35s (33.1×)	100%	2.79(265.5×)	98%
20	0.19s (1×)	99%	32.51s (174×)	66%	260.3(1393.4×)	50%
30	0.45s (1×)	99%	487.04s (1072.5×)	3%	–	0%
40	1.17s (1×)	98%	–	0%	–	0%
50	2.96s (1×)	98%	–	0%	–	0%
60	7.25s (1×)	97%	–	0%	–	0%
70	12.76s (1×)	95%	–	0%	–	0%
80	28.47s (1×)	94%	–	0%	–	0%
90	45.58s (1×)	89%	–	0%	–	0%
100	113.89s (1×)	90%	–	0%	–	0%
150	977.97s (1×)	74%	–	0%	–	0%

For the pure random instance we consider the basic PMP only. When running the RWP instances we consider both the basic and the extended PMP. Tables 2, 3, and 4 show our computational results for the harmonic case. For the IP-formulations the value in parenthesis denotes the ratio between the respective running time and the time needed by the bin-formulation.

*Discussion.* In the three settings of the harmonic case the bin-formulation clearly outperforms the two other IP-formulations. While for small instances the congruence formulation is still competitive, as the number of tasks increases the bin formulation becomes superior. The time-indexed formulation failed to find an optimal solution before the timeout even on small instances with ten tasks.

This shows that taking the bin structure into account in the bin-formulation allows a significantly better running time in comparison with the other formulations. In contrast to the congruence formulation no modulo-operation has to be encoded in the IP-model (recall the conditions for a collision derived in Lemma 11). Also, the number of variables is a lot smaller than in the time-indexed formulation.

The First-Fit heuristic performs very well for the basic PMP and finds an optimal solution for most instances. Even though theoretically First-Fit is only

**Table 4.** Computational results for the RWP instances (harmonic case) for the extended PMP

# tasks	IP-formulations					Heuristic FF
	BF	CF	TIF			
10	0.09s (1×)	100%	0.2s (2.4×)	100%	5.03(58.7×)	100%
20	2.16s (1×)	99%	19.96s (9.2×)	85%	29.19(13.5×)	15%
30	19.8s (1×)	99%	119.22s (6×)	20%	—	0%
40	97.02s (1×)	93%	—	0%	—	0%
50	401.75s (1×)	62%	—	0%	—	0%
60	655.06s (1×)	30%	—	0%	—	0%
70	644.54s (1×)	8%	—	0%	—	0%

a 2-approximation algorithm, it performs much better in practice. However, for real-world data we need to consider the extended PMP. In these instances First-Fit mostly missed the optimum by a significant margin. Also, it cannot provide a certificate of optimality and hence, in real settings one has to resort to IP-formulations. Nevertheless, First-Fit can be used as a fast heuristic which computes an upper bound on the number of needed machines.

### 4.3 Original Boeing Instances

We solved each real-world instance from Boeing in less than 15 minutes to optimality. The most challenging one consists of 177 tasks, and an optimal solution uses 16 machines. The arising period lengths were 50, 100, 200, 400, 800, 1000, and 2000. Note that this instance is not harmonic. Nonetheless, the number of jobs having one of the problematic period lengths (that is, 1000 and 2000) were very small (three and six respectively). We transform the instance to be harmonic by taking the 3 tasks with period length 1000 and changing their periods to 200, and changing the 6 tasks with period 2000 to have period length 400. Note that a solution of the modified instance can be easily converted to a solution of the original instance. On the other hand, we could prove that the optimal solution of the restrictive instance is also optimal for the original instance since the separation constraints already contained a set of 16 tasks that had to be assigned to different machines.

The instances from Boeing also have an additional extra constraint not yet discussed: We are given subsets of tasks that must be processed on the same machine. We call these the *cohabitation constraints*. Moreover, for a subset of tasks, a predefined assignment of tasks to machines is already given as part of the input. We have not considered these constraints in the previous experiments for several reasons: For the RWP case it is not clear how to generate meaningful random perturbations of these constraints. Also, the First-Fit algorithms sometimes fail to produce feasible schedules, even though the respective instance has a solution. In particular combinations of cohabitation and cabinet constraints often require a more sophisticated approach than pure greedy. Therefore, the IP formulations are much more appropriate for the real world instances.

## References

- [Bha98] Bhatia, R.: Approximation Algorithms for Scheduling Problems. PhD thesis, University of Maryland (1998)
- [BHR93] Baruah, S.K., Howell, R.R., Rosier, L.E.: Feasibility problems for recurring tasks on one processor. In: Selected papers of the 15th International Symposium on Mathematical Foundations of Computer Science, pp. 3–20. Elsevier, Amsterdam (1993)
- [BRTV90] Baruah, S., Rousier, L., Tulchinsky, I., Varvel, D.: The complexity of periodic maintenance. In: Proceedings of the International Computer Symposium (1990)
- [But04] Buttazzo, G.C.: Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer, Heidelberg (2004)
- [CPPT07] Crainic, T.G., Perboli, G., Pezzuto, M., Tadei, R.: New bin packing fast lower bounds. Computers & Operations Research 34, 3439–3457 (2007)
- [EHN<sup>+</sup>10] Eisenbrand, F., Hähnle, N., Niemeier, M., Skutella, M., Verschae, J., Wiese, A.: Scheduling periodic tasks in a hard real-time environment. In: Proceedings of ICALP 2010. LNCS. Springer, Heidelberg (2010) (to appear)
- [FH02] Fleszar, K., Hindi, K.S.: New heuristics for one-dimensional bin-packing. Computers & Operations Research 29, 821–839 (2002)
- [KAL96] Korst, J., Aarts, E., Lenstra, J.K.: Scheduling periodic tasks. INFORMS Journal on Computing 8, 428–435 (1996)
- [KALW91] Korst, J., Aarts, E., Lenstra, J.K., Wessels, J.: Periodic multiprocessor scheduling. In: Aarts, E.H.L., Rem, M., van Leeuwen, J. (eds.) PARLE 1991. LNCS, vol. 505, pp. 166–178. Springer, Heidelberg (1991)
- [Leu04] Leung, J.Y.-T.: Handbook of Scheduling: Algorithms, Models and Performance Analysis. Chapman & Hall/CRC, Boca Raton (2004)
- [LGW08] Loh, K.-H., Golden, B., Wasil, E.: Solving the one-dimensional bin packing problem with a weight annealing heuristic. Computers & Operations Research 35, 2283–2291 (2008)
- [MT90] Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations, revised edn. John Wiley & Sons, Chichester (November 1990)
- [SKJ97] Scholl, A., Klein, R., Jürgens, C.: BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers & Operations Research 24, 627–645 (1997)
- [SL94] Simchi-Levi, D.: New worst-case results for the bin-packing problem. Naval Research Logistics 41, 579–585 (1994)
- [Van99] Vanderbeck, F.: Computational study of a column generation algorithm for bin packing and cutting stock problems. Mathematical Programming 86, 565–594 (1999)
- [VBJN94] Vance, P.H., Barnhart, C., Johnson, E.L., Nemhauser, G.L.: Solving binary cutting stock problems by column generation and branch-and-bound. Computational Optimization and Applications 3, 111–130 (1994)
- [VdC99] Valério de Carvalho, J.M.: Exact solution of bin packing problems using column generation and branch and bound. Annals of Operations Research 86, 629–659 (1999)
- [WL83] Wei, W.D., Liu, C.L.: On a periodic maintenance problem. Operations Research Letters 2, 90–93 (1983)

# Non-clairvoyant Speed Scaling for Weighted Flow Time

Sze-Hang Chan<sup>1</sup>, Tak-Wah Lam<sup>1</sup>, and Lap-Kei Lee<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Hong Kong, Hong Kong

<sup>2</sup> Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

**Abstract.** We study online job scheduling on a processor that can vary its speed dynamically to manage its power. We attempt to extend the recent success in analyzing total unweighted flow time plus energy to total weighted flow time plus energy. We first consider the non-clairvoyant setting where the size of a job is only known when the job finishes. We show an online algorithm WLAPS that is  $8\alpha^2$ -competitive for weighted flow time plus energy under the traditional power model, which assumes the power  $P(s)$  to run the processor at speed  $s$  to be  $s^\alpha$  for some  $\alpha > 1$ . More interestingly, for any arbitrary power function  $P(s)$ , WLAPS remains competitive when given a more energy-efficient processor; precisely, WLAPS is  $16(1 + \frac{1}{\epsilon})^2$ -competitive when using a processor that, given the power  $P(s)$ , can run at speed  $(1 + \epsilon)s$  for some  $\epsilon > 0$ . Without such speedup, no non-clairvoyant algorithm can be  $O(1)$ -competitive for an arbitrary power function [8]. For the clairvoyant setting (where the size of a job is known at release time), previous results on minimizing weighted flow time plus energy rely on scaling the speed continuously over time [5, 7]. The analysis of WLAPS has inspired us to devise a clairvoyant algorithm LLB which can transform any continuous speed scaling algorithm to one that scales the speed at discrete times only. Under an arbitrary power function, LLB can give an  $4(1 + \frac{1}{\epsilon})$ -competitive algorithm using a processor with  $(1 + \epsilon)$ -speedup.

## 1 Introduction

To reduce energy usage, manufacturers like Intel and IBM are now producing processors that can support *dynamic speed scaling*, which would allow operating systems to manage the power by scaling the processor speed dynamically. The theoretical study of speed scaling was initiated by Yao, Demers and Shenker [18]. They considered a model where a processor can vary the speed  $s$  dynamically, and it consumes energy at the rate  $s^\alpha$ , where  $\alpha > 1$  is a constant (commonly believed to be 2 or 3 [3, 16]). Running jobs slower is more energy-efficient, yet it takes longer time. Taking speed scaling and energy usage into consideration makes job scheduling more complicated than before. A scheduling algorithm needs two components: a job selection policy to determine which job to run, and a speed scaling policy to determine the speed to run the job. The challenge arises from the conflicting objectives of optimizing some quality of service (QoS) of the schedule and minimizing the energy usage.

**Flow and energy.** The past few years have witnessed several interesting results on online scheduling for optimizing the tradeoff between energy usage and flow time (e.g., [1, 5–8, 12–14]). The flow time (or simply the flow) of a job is the time elapsed since the job is released until it is completed. Assuming jobs are equally important, it is natural to find a schedule that minimizes the total flow time (also known as minimizing the total/average response time). In general, jobs have varying importance or weights, and it is more meaningful to minimize the total weighted flow time. In the speed scaling model, minimizing total flow time and energy usage are orthogonal objectives. To understand their tradeoff, Albers and Fujiwara [1] initiated the study of minimizing a linear combination of flow and energy. The intuition is that, from an economic viewpoint, users are willing to pay a certain (say,  $\rho$ ) units of energy to reduce one unit of flow time. By changing the units of time and energy, one can further assume  $\rho = 1$  and thus wants to minimize flow plus energy, or in general, weighted flow plus energy.

**Clairvoyant scheduling for flow plus energy.** Most of the previous work on minimizing flow plus energy focused on the online setting where the size of a job is known at its release time. This is known as the clairvoyant setting. Bansal, Pruhs and Stein [7] were the first to consider jobs with arbitrary weights, and they give an online algorithm BPS that is  $O((\frac{\alpha}{\ln \alpha})^2)$ -competitive for minimizing weighted flow plus energy. Very recently, Bansal, Chan and Pruhs [6] improved the analysis of BPS, which implies that BPS is  $O(\frac{\alpha}{\ln \alpha})$ -competitive.

The BPS algorithm scales the speed as a function of the fraction of unfinished work and thus it keeps changing the speed continuously over time. Practically speaking, it is more desirable to change the speed only at discrete times (say, at job arrival or completion). Focusing on jobs with unit-weight, Lam et al. [14] showed a competitive discrete-time-scaling algorithm called AJC (Active Job Count), which scales the speed as a function of the number of active jobs. When coupled with the job selection policy SRPT (shortest remaining processing time), it is  $O(\frac{\alpha}{\log \alpha})$ -competitive for (unweighted) flow plus energy. Recently, Bansal, Chan and Pruhs [6] gave a tighter analysis of AJC, showing that the competitive ratio is at most 3 (when  $\alpha = 3$ , the  $O(\frac{\alpha}{\log \alpha})$  bound in [14] is equal to 3.25). More importantly, they extend the analysis to a more general model where the speed-to-power function can be an arbitrary function. This makes the results of speed scaling more robust than before. Andrew et al. [2] have further improved the ratio to 2. Extending the work of [2, 6, 14] to jobs with arbitrary weights is non-trivial. It is natural to generalize AJC to AJW (active job weight), which scales the speed according to the total weight of active jobs. However, even assuming the power function in the form  $s^\alpha$ , it has been open whether AJW (plus a job selection policy like HDF) or any discrete-time-scaling algorithm can be competitive for minimizing weighted flow plus energy, let alone for an arbitrary power function.

**Non-clairvoyant scheduling for flow plus energy.** All of the above results assume clairvoyance. In the non-clairvoyant setting, the size of a job is only known when the job is completed. This is a natural assumption from the viewpoint of operating systems. Non-clairvoyant flow time scheduling (on

a fixed-speed processor) has been an interesting problem itself (e.g., [11, 15]). Chan et al. [8] were the first to study non-clairvoyant speed scaling. They focused on unit-weight jobs and considered an algorithm LAPS (Latest Arrival Processor Sharing) which scales the speed as AJC and selects some most recently released jobs to share the processor. LAPS is  $O(\alpha^3)$ -competitive for (unweighted) flow plus energy. Furthermore, it is shown that no algorithm can be  $O(1)$ -competitive for an arbitrary power function. Recently, Chan et al. [9] improved the competitive ratio of LAPS to  $O(\frac{\alpha^2}{\log \alpha})$ . For weighted flow plus energy, Chan et al. [10] showed that a round robin policy plus AJW is  $O(3^\alpha)$ -competitive. Note that all these results assume that the power function is in the form  $s^\alpha$ .

**Our contribution.** The main result of this paper is about a new non-clairvoyant algorithm called WLAPS for minimizing weighted flow plus energy. WLAPS attempts to generalize LAPS [8, 9] to jobs with arbitrary weights. WLAPS uses the speed scaling policy AJW, i.e., the speed is a function of the total weight of active jobs. Like LAPS, WLAPS gives priority to some most recently released jobs. But WLAPS does not schedule a constant fraction of active jobs; instead it ensures that the jobs selected have a total weight equal to a constant fraction of the total weight of all active jobs, and they share the processor according to their job weights. Furthermore, as job weights are arbitrary, they may not make up exactly the required fraction. Thus, we sometimes need to modify the weight of some job to avoid under- or over-scheduling.

Using a slightly complicated potential analysis, we show that for a power function in the form  $s^\alpha$ , WLAPS is  $8\alpha^2$ -competitive for weighted flow plus energy. We also analyze WLAPS with an arbitrary power function  $P(s)$ . In view of the lower bound result in [8], we consider giving WLAPS a more energy-efficient processor which, when given the power  $P(s)$ , can run at speed  $(1 + \epsilon)s$  for some  $\epsilon > 0$ . We call such a processor a  $(1 + \epsilon)$ -speedup processor.<sup>1</sup> We prove that WLAPS is  $16(1 + \frac{1}{\epsilon})^2$ -competitive when given a  $(1 + \epsilon)$ -speedup processor. The main difficulty here is how to lower bound the optimal offline algorithm.

**Table 1.** Results on non-clairvoyant speed scaling for minimizing flow plus energy

	Unit-weight jobs	Arbitrary-weight jobs
Traditional power function ( $s^\alpha$ )	$O(\frac{\alpha^2}{\log \alpha})$ -competitive [9]	$8\alpha^2$ -competitive (new)
Arbitrary power function	unbounded competitive ratio [8] (if no speedup)	$16(1 + \frac{1}{\epsilon})^2$ -competitive (new) with $(1 + \epsilon)$ -speedup

**Implication to clairvoyant speed scaling.** Since the clairvoyant setting is a special case of the non-clairvoyant setting, WLAPS also gives us the first clairvoyant discrete-time-scaling result on weighted flow plus energy, and it is

<sup>1</sup> Note that speed-up processors here are not related to processes (or jobs) with arbitrary speed-up curves in [9]; the former refers to more energy-efficient processors, and the latter is job characteristics, specifying the degrees of parallelizability (i.e., the rate of processing a job as a function of the number of processors assigned to it).

valid for an arbitrary power function. In fact, we can further improve the ratios. In the analysis of WLAPS, when we attempt to lower bound the performance of the optimal offline algorithm, we have devised a novel clairvoyant algorithm LLB (latest lag behind) which can transform any (online/offline) algorithm into one that scales the speed according to AJW, and the cost (weighted flow plus energy) at most doubles. Therefore, we can effectively transform the BPS algorithm [6, 7] to a discrete-time-scaling algorithm using the AJW policy that is  $O(\frac{\alpha}{\ln \alpha})$ -competitive for a power function in the form  $s^\alpha$ , and  $4(1 + \frac{1}{\epsilon})$ -competitive for an arbitrary power function using a processor with  $(1 + \epsilon)$ -speedup for any  $\epsilon > 0$ .

**Bounded maximum speed.** All the above-mentioned results are based on an assumption in [18] that the processor does not have a limit on the speed. Such an unbounded speed model is a convenient model to work with. Among others, it allows an online algorithm to catch up arbitrarily fast and recover from any over-conservative decision on speed. However, this is not a practical model. Recently, there has been a growing interest in algorithms that scale the speed between 0 and a given maximum speed  $T$ . Though not straightforward, most unbounded-speed algorithms have been shown to work in the bounded speed model when the power function is in the form  $s^\alpha$ .

For unit-weight jobs, the clairvoyant results on AJC by Lam et al. [14], Bansal et al. [6], and Andrew et al. [2] all remain valid when the maximum speed  $T$  is bounded. For the clairvoyant-weighted setting and the non-clairvoyant setting, the lower bound results on (fixed-speed) flow time scheduling [4, 15] imply that no online algorithm can be constant competitive (in terms of  $\alpha$ ) for weighted flow plus energy when  $T$  is bounded (if there is no resource augmentation). On the other hand, it has been shown that the algorithms BPS and LAPS when capped at maximum speed  $(1 + \delta)T$  for some  $\delta > 0$ , remains competitive as in the unbounded speed model [5, 10]. Note that relaxing the maximum speed is a less demanding form of resource augmentation than using a more efficient processor allowing speedup, and it is only applicable in the bounded speed model.

In this paper we also extend the results of WLAPS for weighted flow plus energy to the bounded speed model. For the traditional power function, WLAPS is  $\max(8\alpha^2, 8(1 + \frac{1}{\delta})^2)$ -competitive when using a processor with maximum speed  $(1 + \delta)T$  for any  $\delta > 0$ . For an arbitrary power function, WLAPS remains  $16(1 + \frac{1}{\epsilon})^2$ -competitive when using a processor with  $(1 + \epsilon)$ -speedup for any  $\epsilon > 0$ . Note that if we only allow relaxing the maximum speed, no  $O(1)$ -competitive algorithm exists due to the lower bound result in the unbounded speed model [8]. Like [2, 6], our analysis exploits potential function. Yet, our analysis involves a technique of taking the maximum speed  $T$  into the design of potential functions.

## 2 Definitions and Notations

We study job scheduling on a single processor. Jobs with varying sizes arrive over time online; we have no information about a job before it arrives. For any job  $j$ , we use  $r(j)$ ,  $w(j)$  and  $p(j)$  to denote its release time, weight and size

(work requirement), respectively. In the clairvoyant model,  $p(j)$  is known at time  $r(j)$ . In the non-clairvoyant model,  $p(j)$  is only known when the job is completed. The processor can vary its speed between 0 and a maximum speed  $T$  (where  $T$  is some fixed constant or  $\infty$ ). When running at speed  $s$ , the processor processes  $s$  units of work per unit time and consumes energy at the rate  $P(s)$ . Preemption is allowed; a job can be preempted and later resumed at the point of preemption without any penalty. We consider the following two models of power function.

**Traditional power model.** In this model, we assume that the power function is  $P(s) = s^\alpha$  for some  $\alpha > 1$ . Furthermore, we distinguish the two settings: *bounded speed* means that the speed  $s$  cannot exceed a fixed maximum speed  $T$ ; and *unbounded speed* otherwise (i.e.,  $T = \infty$ ).

**Arbitrary power model.** In this model, we consider a power function  $P$  such that  $P(0) = 0$ , and  $P$  is defined, strictly increasing, strictly convex, continuous and differentiable at all speeds in  $[0, T]$ ; if there is no maximum speed  $T$ , the speed range is  $[0, \infty)$  and for any speed  $x$ , there exists  $x'$  such that  $P(x)/x < P(s)/s$  for all  $s > x'$  (otherwise the optimal speed scaling policy is to always run at the infinite speed and an optimal schedule is not well-defined). As shown in [6], it is possible to use any arbitrary power function to emulate such a power function  $P$  with an arbitrarily small increase in the competitive ratio. Thus, we focus on a power function  $P$  satisfying the above assumptions. We use  $Q$  to denote  $P^{-1}$ . Note that  $Q$  is strictly increasing and concave. E.g., if  $P(s) = s^\alpha$ , then  $Q(x) = x^{1/\alpha}$ .

**Flow and energy.** Consider any job set  $I$  and some schedule  $S$  of  $I$ . At any time  $t$ , for any job  $j$ , we let  $q(j, t)$  be the remaining work of  $j$  at  $t$ . A job  $j$  is an *active job* if it has been released but not yet completed, i.e.,  $r(j) \leq t$  and  $q(j, t) > 0$ . The *flow* of a job  $j$  is the time elapsed since  $j$  arrives and until it is completed, and the *weighted flow*  $F(j)$  of  $j$  is  $w(j)$  times its flow. The *total weighted flow* is  $F = \sum_{j \in I} F(j)$ , which is equivalent to  $F = \int_0^\infty w(t) dt$ , where  $w(t)$  is the total weight of active jobs at time  $t$ . The energy usage is  $E = \int_0^\infty P(s(t)) dt$ , where  $s(t)$  is the processor speed at time  $t$ . The objective is to minimize the total weighted flow plus energy usage, denoted by  $G = F + E$ .

**Overview of analysis.** Throughout the paper, we often need to compare an algorithm ALG with another algorithm  $O$  (e.g., the optimal offline algorithm OPT). We will exploit amortization and potential functions. Let  $G_a(t)$  and  $G_o(t)$  denote the weighted flow plus energy incurred up to time  $t$  by ALG and  $O$ , respectively. We will drop the parameter  $t$  when it is clear that  $t$  is the current time. To show that ALG incurs at most  $c$  times the weighted flow plus energy of  $O$ , it suffices to define a potential function  $\Phi(t)$  such that the following conditions hold: (i) *Boundary condition*:  $\Phi = 0$  before any job is released and after all jobs are completed; (ii) *Discrete-event condition*:  $\Phi$  is a continuous function except at some discrete times (e.g., when a job arrives, or when a job is completed by ALG or  $O$ ), and  $\Phi$  does not increase at such times; (iii) *Running condition*: at any other time,  $\frac{dG_a(t)}{dt} + \gamma \frac{d\Phi(t)}{dt} \leq c \cdot \frac{dG_o(t)}{dt}$ , where  $\gamma$  is a positive constant. The correctness of the analysis follows from integrating these conditions over time.

### 3 Non-clairvoyant Speed Scaling

This section considers a non-clairvoyant algorithm WLAPS for minimizing weighted flow plus energy. In Section 3.1, we show that under the traditional power model, WLAPS is  $8\alpha^2$ -competitive if the maximum speed  $T$  is unbounded; if  $T$  is bounded, WLAPS is  $\max(8\alpha^2, 8(1 + \frac{1}{\epsilon})^2)$ -competitive when using a processor with maximum speed relaxed to  $(1 + \epsilon)T$  for any  $\epsilon > 0$ . In Section 3.2, we consider the arbitrary power model. Note that even in the unbounded speed setting, no  $O(1)$ -competitive algorithm exists [8]. Nevertheless, we show that WLAPS is  $16(1 + \frac{1}{\epsilon})^2$ -competitive when using a  $(1 + \epsilon)$ -speedup processor for any  $\epsilon > 0$ . This result holds no matter whether  $T$  is unbounded or bounded.

We now define the algorithm WLAPS. At any time  $t$ , we use  $n_a(t)$  and  $w_a(t)$  to denote respectively the number and total weight of active jobs. The active jobs at time  $t$  are referred to as  $j_1, j_2, \dots, j_{n_a(t)}$ , which are arranged in ascending order of their release times (ties are broken by job ids).

**WLAPS.** Let  $0 < \beta \leq 1$  be any real. Consider any time  $t$ . Let  $\tau$  be the biggest integer such that the total weight of jobs  $j_\tau, j_{\tau+1}, \dots, j_{n_a(t)}$  is at least  $\beta w_a(t)$ . Define  $w'(j_\tau)$ , the adjusted weight of  $j_\tau$ , to be  $\beta w_a(t) - [w(j_{\tau+1}) + \dots + w(j_{n_a(t)})]$ . WLAPS processes  $j_\tau, j_{\tau+1}, \dots, j_{n_a(t)}$  by splitting the processor speed in proportional to the adjusted weight of  $j_\tau$  and the (original) weights of  $j_{\tau+1}, \dots, j_{n_a(t)}$ .

WLAPS basically follows the speed scaling policy AJW, which sets the speed to  $Q(w_a(t))$ . E.g., if  $P(s) = s^\alpha$ , then  $Q(w_a(t)) = w_a(t)^{1/\alpha}$ . In Sections 3.1 and 3.2, we will further refine the policy due to the presence of maximum speed and/or the need of speed-up.

For convenience, at time  $t$ , we define  $w'(j_i)$  for jobs  $j_i \neq j_\tau$  as follows: let  $w'(j_i) = w(j_i)$  if WLAPS is processing  $j_i$  at time  $t$ , and let  $w'(j_i) = 0$  otherwise. Note that  $\sum_{i=1}^{n_a(t)} w'(j_i) = \beta w_a(t)$ . Intuitively,  $w'(j_i)/\beta w_a(t)$  is the fraction of processor speed received by  $j_i$  at time  $t$ . Specifically, job  $j_i$  runs at speed  $\frac{w'(j_i)}{\beta w_a(t)} s_a(t)$ .

**Framework of analysis.** To analyze WLAPS, we exploit amortization and potential functions to compare WLAPS with some offline algorithm OFF. As mentioned in Section 2, we need a potential function  $\Phi(t)$  that satisfies the boundary, discrete-event and running conditions. Below we define a general form of  $\Phi(t)$  that works for the traditional and arbitrary power models.

**Potential function  $\Phi(t)$ .** Consider any time  $t$ . Recall that  $n_a(t)$  and  $w_a(t)$  are respectively the number and total weight of active jobs in WLAPS. We define  $n_o(t)$  and  $w_o(t)$  similarly for OFF. We will drop the parameter  $t$  when  $t$  is clearly the current time. Define the coefficient of  $j_i$ , denoted  $c_i$ , to be  $\sum_{k=1}^i w(j_k)$ . Note that  $c_{n_a} = w_a$ . For any job  $j$ , let  $q_a(j, t)$  and  $q_o(j, t)$  be the remaining work of job  $j$  in WLAPS and OFF, respectively. For each  $j_i$ , let  $x_i = \max\{q_a(j_i, t) - q_o(j_i, t), 0\}$  which is the amount of work of  $j_i$  in WLAPS that is lagging behind OFF. We say a job  $j_i$  is *lagging* if  $x_i > 0$ . Based on the notion of lagging, we define

$$\Phi(t) = \sum_{i=1}^{n_a(t)} f(c_i) \cdot x_i ,$$

where  $f(x)$  is some non-decreasing function of  $x$  (to be defined differently in the traditional and arbitrary power model).

We can check the boundary and discrete-event conditions without the detailed definition of  $f(x)$ . The boundary condition clearly holds due to the definition of  $\Phi$ . Next, we check the discrete-event condition. When a job  $j$  arrives,  $j$  must be non-lagging and the coefficients of all existing jobs of WLAPS remain the same, so  $\Phi$  does not change. When OFF completes a job,  $\Phi$  does not change. When WLAPS completes a job, since  $f(x)$  is non-decreasing and the coefficient of any other job either stays the same or decreases,  $\Phi$  does not increase.

To show the running condition, we need to consider the two power models separately and have a different definition of the function  $f$ . Here we can only discuss some useful properties of  $\Phi$  common to both power models. Consider any time  $t$  when  $\Phi$  does not have discrete change. Let  $s_a$  and  $s_o$  be the current speeds of WLAPS and OFF, respectively. Among the jobs that WLAPS is processing at time  $t$  (i.e.,  $j_\tau, j_{\tau+1}, \dots, j_{n_a}$ ), our analysis will focus on those that are also lagging jobs. Denote the set of such lagging jobs as  $L$ . Note that  $\sum_{i|j_i \in L} w'(j_i) \leq \beta w_a$ . We further define another real number  $\phi$  such that  $\sum_{i|j_i \in L} w'(j_i) = \phi w_a$ . Note that  $\phi \leq \beta$ . WLAPS is processing non-lagging jobs with total weight at least  $(\beta - \phi)w_a$ , and these jobs are also active jobs for OPT. Thus,  $w_o \geq (\beta - \phi)w_a$ .

To bound the rate of change of  $\Phi$ , we consider how  $\Phi$  changes in an infinitesimal amount of time (from  $t$  to  $t + dt$ ), first due to WLAPS only (Lemma 1(i)), and then due to OFF (Lemma 1(ii)). We denote the rate of change of  $\Phi$  due to WLAPS and OFF by  $\frac{d\Phi_a}{dt}$  and  $\frac{d\Phi_o}{dt}$ , respectively. Note that  $\frac{d\Phi}{dt} = \frac{d\Phi_a}{dt} + \frac{d\Phi_o}{dt}$ .

**Lemma 1.** (i)  $\frac{d\Phi_a}{dt} \leq -\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a$ . (ii)  $\frac{d\Phi_o}{dt} \leq f(w_a) \cdot s_o$ .

*Proof.* We first prove (i). It is trivial when  $s_a = 0$ ; it remains to consider  $s_a > 0$ . By the definition of  $\Phi$ , the execution of WLAPS can only decrease the potential. We will show that the rate of decrease is at least  $\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a$ , or equivalently, the rate of change is at most  $-\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a$ .

If we only consider the change due to WLAPS, for any  $j_i \in L$  (WLAPS is processing  $j_i$  and  $j_i$  is lagging), its lagging size  $x_i$  is changing at the rate of  $-\frac{w'(j_i)}{\beta w_a}s_a$ . For other jobs  $j_i \notin L$ ,  $x_i$  does not change. Note that as  $f(x)$  is non-decreasing, we have  $r \cdot f(c) \geq \int_{c-r}^c f(x) dx$  for any constant  $c \geq r$ . Thus,

$$\frac{d\Phi_a}{dt} = \sum_{i|j_i \in L} (f(c_i) \cdot -\frac{w'(j_i)}{\beta w_a}s_a) = -\frac{s_a}{\beta w_a} \sum_{i|j_i \in L} f(c_i)w'(j_i) \leq -\frac{s_a}{\beta w_a} \sum_{i|j_i \in L} \int_{c_i - w'(j_i)}^{c_i} f(x) dx.$$

We view  $\sum_{i|j_i \in L} \int_{c_i - w'(j_i)}^{c_i} f(x) dx$  as the sum of integrations of some non-overlapping ranges. Since  $f(x)$  is non-decreasing, the integration over a specific range must be at least the integration over a range of the same size and with smaller end-points, i.e.,  $\int_{c-r}^c f(x) dx \geq \int_{c'-r}^{c'} f(x) dx$  for  $r \leq c' \leq c$ . By the definition of coefficients, every job in  $L$  has a coefficient at least that of  $j_\tau$  (i.e.,  $c_\tau$ ). Furthermore, the total length of the integration ranges is  $\sum_{i|j_i \in L} w'(j_i)$ . By

“moving” the ranges towards the minimum possible endpoints starting from  $c_\tau - w'(j_\tau)$ , we have

$$\sum_{i|j_i \in L} \int_{c_i - w'(j_i)}^{c_i} f(x) dx \geq \int_{c_\tau - w'(j_\tau)}^{c_\tau - w'(j_\tau) + \sum_{i|j_i \in L} w'(j_i)} f(x) dx ; \text{ and}$$

$$\frac{d\Phi_a}{dt} \leq -\frac{s_a}{\beta w_a} \int_{c_\tau - w'(j_\tau)}^{c_\tau - w'(j_\tau) + \sum_{i|j_i \in L} w'(j_i)} f(x) dx = -\frac{s_a}{\beta w_a} \int_{w_a - \beta w_a}^{w_a - \beta w_a + \phi w_a} f(x) dx.$$

Since  $f(x)$  is non-decreasing, we have

$$\frac{d\Phi_a}{dt} \leq -\frac{s_a}{\beta w_a} \int_{(1-\beta)w_a}^{(1-\beta+\phi)w_a} f(x) dx \leq -\frac{s_a}{\beta w_a} (\phi w_a \cdot f((1-\beta)w_a)) = -\frac{\phi}{\beta} \cdot f((1-\beta)w_a) s_a .$$

For (ii), to upper bound  $\frac{d\Phi_a}{dt}$ , the worst case is that OFF is processing the job  $j_{n_a}$  with the largest coefficient  $c_{n_a} = \sum_{k=1}^{n_a} w(j_k) = w_a$ , so  $\frac{d\Phi_a}{dt} \leq f(w_a) \cdot s_o$ .  $\square$

In Sections 3.1 and 3.2, we will consider the two power models separately, and show the refinement of AJW and the definition of  $f$  that are sufficient to prove the running condition for each model.

### 3.1 Traditional Power Model

We consider the traditional power model, which assumes that the power function  $P(s) = s^\alpha$  for some  $\alpha > 1$ . If the maximum speed  $T$  is unbounded, WLAPS follows exactly AJW and sets its speed at time  $t$  as  $s_a(t) = w_a(t)^{1/\alpha}$ . If  $T$  is bounded, we let WLAPS use a processor with maximum speed  $(1 + \epsilon)T$  for any  $\epsilon > 0$ , and set  $s_a(t) = \min(w_a(t)^{1/\alpha}, (1 + \epsilon)T)$ .

Our main result is the following theorem.

**Theorem 1.** Consider minimizing weighted flow plus energy. (i) If the maximum speed  $T$  is unbounded, WLAPS is  $8\alpha^2$ -competitive. (ii) If  $T$  is bounded, WLAPS is  $\max(8\alpha^2, 8(1 + \frac{1}{\epsilon})^2)$ -competitive, using a processor with maximum speed  $(1 + \epsilon)T$ .

**Unbounded maximum speed.** To prove Theorem 1(i), we set the offline algorithm OFF as the optimal offline algorithm OPT for minimizing weighted flow plus energy, and let  $f(x) = x^{1-1/\alpha}$ , which is clearly a non-decreasing function of  $x$ . Then the theorem follows from the running condition below.

**Lemma 2.** Assume that  $\beta = \frac{1}{2\alpha}$  and  $\gamma = \frac{2}{\beta}$ . At any time when  $\Phi$  does not have discrete change,  $\frac{dG_a}{dt} + \gamma \frac{d\Phi}{dt} \leq 8\alpha^2 \cdot \frac{dG_o}{dt}$ .

*Proof.* Note that  $\frac{dG_a}{dt} = w_a + s_a^\alpha = 2w_a$ , and  $\frac{dG_o}{dt} = w_o + s_o^\alpha \geq (\beta - \phi)w_a + s_o^\alpha$ . By Lemma 1(i),  $\frac{d\Phi_a}{dt} \leq -\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a$ . Since  $s_a = w_a^{1/\alpha}$  and  $f(x) = x^{1-1/\alpha}$ , we have  $f(x)s_a \geq x$  for any  $0 \leq x \leq w_a$ . Thus,  $\frac{d\Phi_a}{dt} \leq -\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a \leq -\frac{\phi}{\beta}(1 - \beta)w_a$ . On the other hand, by Lemma 1(ii),  $\frac{d\Phi_o}{dt} \leq f(w_a) \cdot s_o = w_a^{1-1/\alpha}s_o$ .

We apply the Young's Inequality [17]<sup>2</sup>, by setting  $p = \alpha$ ,  $q = \frac{\alpha}{\alpha-1}$ ,  $x = s_o$  and  $y = w_a^{1-1/\alpha}$ . Then  $\frac{d\Phi_o}{dt} \leq w_a^{1-1/\alpha} s_o \leq (1 - \frac{1}{\alpha})w_a + \frac{1}{\alpha}s_o^\alpha = (1 - 2\beta)w_a + \frac{1}{\alpha}s_o^\alpha$ .

Note that  $\gamma = \frac{2}{\beta}$  and  $\phi \leq \beta$ . Then

$$\begin{aligned} \frac{dG_a}{dt} + \gamma \frac{d\Phi}{dt} &\leq 2w_a + \frac{2}{\beta}(1 - 2\beta)w_a + \frac{2}{\beta\alpha}s_o^\alpha - \frac{2\phi}{\beta^2}(1 - \beta)w_a \\ &= ((\frac{2}{\beta} - \frac{2\phi}{\beta^2}) + (-2 + \frac{2\phi}{\beta}))w_a + \frac{2}{\beta\alpha}s_o^\alpha \\ &\leq \frac{2}{\beta^2}(\beta - \phi)w_a + \frac{2}{\beta\alpha}s_o^\alpha \leq \frac{2}{\beta^2}w_o + \frac{2}{\beta\alpha}s_o^\alpha \leq 8\alpha^2 \cdot \frac{dG_o}{dt}. \end{aligned} \quad \square$$

**Bounded maximum speed.** To prove Theorem 1(ii), we let OFF be the optimal offline algorithm OPT that uses a processor with maximum speed  $T$ , and let  $f(x) = x / \min(x^{1/\alpha}, (1 + \epsilon)T)$ . Again,  $f(x)$  is a non-decreasing function of  $x$ . We prove the running condition below.

**Lemma 3.** *Assume that  $\beta = \min(\frac{1}{2\alpha}, \frac{\epsilon}{2\epsilon+2})$  and  $\gamma = \frac{2}{\beta}$ . At any time when  $\Phi$  does not have discrete change,  $\frac{dG_a}{dt} + \gamma \frac{d\Phi}{dt} \leq \max(8\alpha^2, 8(1 + \frac{1}{\epsilon})^2) \cdot \frac{dG_o}{dt}$ .*

*Proof.* We first show that  $\frac{d\Phi_o}{dt} \leq (1 - 2\beta)w_a + \frac{1}{\alpha}s_o^\alpha$ . By Lemma 1(ii),  $\frac{d\Phi_o}{dt} \leq f(w_a) \cdot s_o$ . If  $w_a^{1/\alpha} \leq (1 + \epsilon)T$ , we have  $f(w_a) = w_a^{1-1/\alpha}$  and hence  $\frac{d\Phi_o}{dt} \leq w_a^{1-1/\alpha} s_o$ . We can show that  $\frac{d\Phi_o}{dt} \leq (1 - 2\beta)w_a + \frac{1}{\alpha}s_o^\alpha$  in the same way as in Lemma 2. If  $w_a^{1/\alpha} > (1 + \epsilon)T$ , we have  $f(w_a) = \frac{w_a}{(1+\epsilon)T}$ . We use the fact that  $s_o \leq T$  and  $\frac{1}{1+\epsilon} = 1 - \frac{\epsilon}{1+\epsilon}$  to conclude that  $\frac{d\Phi_o}{dt} \leq (1 - \frac{\epsilon}{1+\epsilon})\frac{w_a}{T}s_o \leq (1 - \frac{\epsilon}{1+\epsilon})w_a \leq (1 - 2\beta)w_a$ .

Note that  $\frac{dG_a}{dt} = w_a + (\min(w_a^{1/\alpha}, (1 + \epsilon)T))^\alpha \leq 2w_a$ . Using the same argument as in Lemma 2, the lemma follows.  $\square$

### 3.2 Arbitrary Power Model

Consider an arbitrary power function  $P(s)$ . Recall that  $Q$  denotes the inverse of  $P$ . Let  $T$  be the maximum speed; if it does not exist, let  $T = \infty$ . We consider WLAPS being given a  $(1 + \epsilon)$ -speedup processor for any  $\epsilon > 0$ , and define the speed of WLAPS at time  $t$  as  $s_a(t) = (1 + \epsilon) \cdot \min(Q(w_a(t)), T)$ . By definition, the power required by WLAPS is  $P(\min(Q(w_a(t)), T))$ .

We compare WLAPS with offline algorithms using a processor without speedup. Let OPT be an optimal offline algorithm. And let OFF be an optimal algorithm among all offline algorithms that scale its speed as  $s_o(t) = \min(Q(w_o(t)), T)$ , where  $w_o(t)$  is the total weight of active jobs at time  $t$ . Our main result is the following theorem.

**Theorem 2.** *For any  $\epsilon > 0$ , when  $\beta = \frac{\epsilon}{2\epsilon+2}$ , WLAPS with a  $(1 + \epsilon)$ -speedup processor is  $8(1 + \frac{1}{\epsilon})^2$ -competitive for weighted flow plus energy against OFF.*

In Section 4, we will show an offline algorithm which scales its speed as  $s_o(t) = \min(Q(w_o(t)), T)$ , and of which the total weighted flow plus energy incurred is at most two times of that of OPT (Corollary 2(i)). Thus, OFF is a 2-approximation of OPT.

---

<sup>2</sup> Young's Inequality: For positive reals  $p, q, x, y$  where  $\frac{1}{p} + \frac{1}{q} = 1$ ,  $xy \leq \frac{1}{p}x^p + \frac{1}{q}y^q$ .

**Corollary 1.** For any  $\epsilon > 0$ , when  $\beta = \frac{\epsilon}{2\epsilon+2}$ , WLAPS with a  $(1 + \epsilon)$ -speedup processor is  $16(1 + \frac{1}{\epsilon})^2$ -competitive for weighted flow plus energy.

To prove Theorem 2, we set  $f(x) = \frac{x}{\min(Q(x), T)}$ , which is non-decreasing.<sup>3</sup> Then the theorem follows from the running condition below.

**Lemma 4.** Assume that  $\beta = \frac{\epsilon}{2\epsilon+2}$ , and  $\gamma = \frac{2}{\beta(1+\epsilon)}$ . At any time when  $\Phi$  does not have discrete change,  $\frac{dG_a}{dt} + \gamma \frac{d\Phi}{dt} \leq 8(1 + \frac{1}{\epsilon})^2 \cdot \frac{dG_o}{dt}$ .

*Proof.* Note that  $\frac{dG_a}{dt} = w_a + P(\min(Q(w_a), T)) \leq w_a + P(Q(w_a)) = 2w_a$  and  $\frac{dG_o}{dt} \geq w_o \geq (\beta - \phi)w_a$ . By Lemma 1(i),  $\frac{d\Phi_a}{dt} \leq -\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a$ . Recall that  $s_a = (1 + \epsilon) \min(Q(w_a), T)$ . By the definition of  $f(x)$ , we have  $f(x)s_a \geq (1 + \epsilon)x$  for any  $0 \leq x \leq w_a$ . Thus,  $\frac{d\Phi_a}{dt} \leq -\frac{\phi}{\beta} \cdot f((1 - \beta)w_a)s_a \leq -\frac{\phi}{\beta}(1 + \epsilon)(1 - \beta)w_a$ . On the other hand, by Lemma 1(ii),  $\frac{d\Phi_o}{dt} \leq f(w_a) \cdot s_o = \frac{w_a}{\min(Q(w_a), T)} \cdot \min(Q(w_o), T)$ . If  $w_a \geq w_o$ , since  $Q$  is increasing,  $Q(w_a) \geq Q(w_o)$  and hence  $\frac{d\Phi_o}{dt} \leq w_a$ . Otherwise,  $w_a < w_o$ . Since  $f(x)$  is non-decreasing,  $f(w_a) \leq f(w_o) = \frac{w_o}{\min(Q(w_o), T)} = \frac{w_o}{s_o}$ , so  $\frac{d\Phi_o}{dt} \leq f(w_a) \cdot s_o \leq w_o$ . Therefore,  $\frac{d\Phi_o}{dt} \leq \max(w_a, w_o)$  and hence  $\frac{d\Phi}{dt} \leq \max(w_a, w_o) - \frac{\phi}{\beta}(1 + \epsilon)(1 - \beta)w_a$ .

Recall that  $\gamma = \frac{2}{\beta(1+\epsilon)}$ , and note that  $\frac{1}{1+\epsilon} = 1 - 2\beta$  and  $\phi \leq \beta \leq 1$ . Then

$$\begin{aligned} \frac{dG_a}{dt} + \gamma \frac{d\Phi}{dt} &\leq 2w_a + \frac{2(1-2\beta)}{\beta} \max(w_a, w_o) - \frac{2\phi}{\beta^2}(1 - \beta)w_a \\ &\leq (\frac{2}{\beta} + (-2 + \frac{2\phi}{\beta})) \max(w_a, w_o) - \frac{2\phi}{\beta^2}w_a \leq \frac{2}{\beta^2}(\beta \max(w_a, w_o) - \phi w_a) \\ &\leq \frac{2}{\beta^2} \max((\beta - \phi)w_a, \beta w_o) \leq \frac{2}{\beta^2}w_o \leq 8(1 + \frac{1}{\epsilon})^2 \cdot \frac{dG_o}{dt}. \end{aligned} \quad \square$$

## 4 Clairvoyant Transformation to AJW

This section considers clairvoyant speed scaling using AJW under an arbitrary power function  $P(s)$ . Given any (online/offline) algorithm B that is using a  $(1 + \epsilon)$ -speedup processor for  $\epsilon \geq 0$  (if  $\epsilon = 0$ , there is no speedup), we give a clairvoyant algorithm LLB (latest lag behind) which simulates B and gives another schedule using the speed scaling policy AJW. We use LLB(B) to denote the resulting algorithm. LLB(B), when using a  $(1 + \epsilon)$ -speedup processor as B, incurs at most twice the weighted flow plus energy of B (Theorem 3).

If we consider B to be an optimal offline algorithm OPT, we obtain a 2-approximate offline algorithm that scales the speed using AJW (Corollary 2(i)). This offline result is needed in Section 3.2. For the online setting, recall that BPS is a competitive clairvoyant algorithm, and it scales the speed continuously over time. Using LLB, we effectively transform BPS to use discrete-time speed

<sup>3</sup> Since  $P$  is strictly increasing,  $Q$  is strictly increasing. Let  $\rho$  be a real such that  $Q(\rho) = T$ . For any  $x \geq \rho$ ,  $Q(x) \geq Q(\rho) = T$  and  $f(x) = \frac{x}{\min(Q(x), T)} = \frac{x}{T}$  which is non-decreasing. For  $x \in [0, \rho]$ ,  $f(x) = \frac{x}{Q(x)}$ . Note that  $P$  is strictly convex and  $Q$  is concave. For any  $\lambda \in [0, 1]$  and  $x, y \in [0, \rho]$ ,  $(1 - \lambda)Q(x) + \lambda Q(y) \leq Q((1 - \lambda)x + \lambda y)$ . Setting  $x = 0$  gives  $\lambda Q(y) \leq Q(\lambda y)$  and hence  $f(\lambda y) = \frac{\lambda y}{Q(\lambda y)} \leq \frac{y}{Q(y)} = f(y)$ .

scaling. Based on [6], one can show that for any  $\epsilon > 0$ , BPS when using a  $(1+\epsilon)$ -speedup processor is  $2(1+\frac{1}{\epsilon})$ -competitive. Thus, LLB(BPS) with a  $(1+\epsilon)$ -speedup processor is  $4(1+\frac{1}{\epsilon})$ -competitive (Corollary 2(ii)). Note that LLB(BPS) has a better performance than WLAPS, yet the former only works clairvoyantly.

**Algorithm LLB(B).** Let  $T$  denote the maximum speed allowed by the power function  $P(s)$  (if it is unbounded, then  $T = \infty$ ). Consider any time  $t$ . Let  $n_a(t)$  and  $w_a(t)$  be respectively the number and total weight of active jobs in LLB. For any job  $j$ , let  $q_a(j, t)$  and  $q_o(j, t)$  be the remaining work of  $j$  at  $t$  in LLB and B, respectively. For each  $j_i$ , let  $x_i = q_a(j_i, t) - q_o(j_i, t)$  be the difference in remaining work of LLB and B, and let  $y_i$  be the latest time that  $x_i \leq 0$ . We say a job  $j_i$  is *lagging* if  $x_i > 0$ . We denote the active jobs in LLB as  $j_1, j_2, \dots, j_{n_a(t)}$ , which are arranged in increasing order of the time when they have become lagging; i.e.,  $y_1 \leq y_2 \leq \dots \leq y_{n_a(t)}$  (ties are broken by job ids). Furthermore, let  $\ell$  be the number of lagging jobs. Notice that for  $i = 1$  to  $\ell$ ,  $x_i > 0$  and  $y_i < t$ ; and for  $i = \ell + 1$  to  $n_a$ ,  $x_i \leq 0$  and  $y_i = t$ . Assume B is running at speed  $s_o$  at time  $t$ . LLB sets its speed  $s_a = (1 + \epsilon) \cdot \min(Q(w_a(t)), T)$ , and it targets the job  $j_a$  defined to be  $j_\ell$  if  $\ell > 0$ , and  $j_{n_a}$  otherwise.

- If B is processing a job  $j_b$  with  $x_b = 0$ , then LLB runs  $j_b$  with speed  $\min(s_a, s_o)$  and runs  $j_a$  with the remaining speed  $s_a - \min(s_a, s_o)$ .
- Otherwise, (i.e. B is processing a job  $j_b$  with  $x_b \neq 0$  or a job that is not active for LLB), it runs  $j_a$  with speed  $s_a$ .

Our main result is Theorem 3 below, which implies Corollary 2.

**Theorem 3.** *Let B be an (online/offline) algorithm using a  $(1+\epsilon)$ -speedup processor where  $\epsilon \geq 0$ . Under an arbitrary power function, LLB(B) with a  $(1+\epsilon)$ -speedup processor incurs at most two times the total weighted flow plus energy of B.*

**Corollary 2.** *Consider minimizing weighted flow plus energy. Let  $w(t)$  be the total weight of active jobs at time  $t$  and let  $T$  be the maximum speed.*

- (i) *Let OPT be the optimal offline algorithm without using any speedup. LLB (OPT) is a 2-approximate (offline) algorithm that scales the speed at any time  $t$  as  $\min(Q(w(t)), T)$ .*
- (ii) *For any  $\epsilon > 0$ , LLB(BPS) using a  $(1+\epsilon)$ -speedup processor is a  $4(1+\frac{1}{\epsilon})$ -competitive online algorithm that scales the speed at any time  $t$  as  $(1+\epsilon) \cdot \min(Q(w(t)), T)$ .*

Before proving Theorem 3, we have the following observation. Both LLB and B are using a  $(1+\epsilon)$ -speedup processor (note that when  $\epsilon = 0$ , the processor has no speedup). Such a processor, when given power  $P(s)$ , runs at speed  $(1+\epsilon)s$ . In other words, the power function of a  $(1+\epsilon)$ -speedup processor is  $\bar{P}(s) = P(\frac{s}{1+\epsilon})$ . We let  $\bar{T} = (1+\epsilon)T$  which is the maximum speed of the  $(1+\epsilon)$ -speedup processor. Like  $P$ , the power function  $\bar{P}$  satisfies that  $\bar{P}(0) = 0$ , and  $\bar{P}$  is defined, strictly increasing, strictly convex, continuous and differentiable at all speeds in  $[0, \bar{T}]$ ; if  $\bar{T} = \infty$ , the speed range is  $[0, \infty)$ . We let  $\bar{Q}$  denote the inverse of  $\bar{P}$ . At any

time  $t$ , the speed of LLB is  $s_a(t) = (1 + \epsilon) \cdot \min(Q(w_a(t)), T) = \min(\bar{Q}(w_a(t)), \bar{T})$ . Thus, it suffices to analyze LLB based on the power function  $\bar{P}$ .

To show Theorem 3, we again exploit amortization and potential functions as shown in Section 2. We derive a potential function  $\Phi$  that satisfies the three required conditions. Consider any time  $t$ . Recall that the active jobs in LLB are denoted as  $j_1, j_2, \dots, j_{n_a(t)}$ . Define the coefficient  $c_i$  of  $j_i$  to be  $\sum_{k=1}^i w(j_k)$ . The potential function  $\Phi(t)$  is defined as follows.

$$\Phi(t) = \sum_{i=1}^{n_a(t)} f(c_i) \cdot \max(x_i, 0) \quad \text{where } f(x) = \bar{P}'(\bar{Q}(x)).$$

Note that  $\bar{P}'$  is the first derivative of  $\bar{P}$ . Since  $\bar{P}$  is convex,  $\bar{P}'$  is non-increasing, which together with that  $\bar{Q}(x)$  is non-decreasing, implies that  $\bar{P}'(\bar{Q}(x))$  is also non-decreasing. Therefore,  $f(x)$  is a non-decreasing function of  $x$ .

We can show that such potential function  $\Phi$  satisfies the boundary and discrete-event conditions. More interestingly, by the definition of LLB, we can show that at any time,  $\bar{Q}(c_\ell) \leq \bar{T}$ , where  $c_\ell$  is the coefficient of job  $j_\ell$  and also the total weight of lagging jobs in LLB. This allows us to prove the running condition that at any time when  $\Phi$  does not have discrete change,  $\frac{dG_a}{dt} + 2\frac{d\Phi}{dt} \leq 2\frac{dG_a}{dt}$ . The detailed proof is given in the full version of the paper. Then Theorem 3 follows.

## References

1. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms* 3(4), 49 (2007)
2. Andrew, L., Wierman, A., Tang, A.: Optimal speed scaling under arbitrary power functions. *ACM SIGMETRICS Performance Evaluation Review* 37(2), 39–41 (2009)
3. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuksunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro* 20(6), 26–44 (2000)
4. Bansal, N., Chan, H.L.: Weighted flow time does not admit  $O(1)$ -competitive algorithms. In: Proc. SODA, pp. 1238–1244 (2009)
5. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórrsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
6. Bansal, N., Chan, H.L., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proc. SODA, pp. 693–701 (2009)
7. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. *SIAM Journal on Computing* 39(4), 1294–1308 (2009)
8. Chan, H.L., Edmonds, J., Lam, T.W., Lee, L.K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for flow and energy. In: Proc. STACS, pp. 255–264 (2009)
9. Chan, H.L., Edmonds, J., Pruhs, K.: Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In: Proc. SPAA, pp. 1–10 (2009)
10. Chan, S.H., Lam, T.W., Lee, L.K., Ting, H.F., Zhang, P.: Non-clairvoyant scheduling for weighted flow time and energy on speed bounded processors. In: Proc. CATS, pp. 3–10 (2010)

11. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. *Journal of the ACM* 50(4), 551–567 (2003)
12. Lam, T.W., Lee, L.K., Ting, H.F., To, I., Wong, P.: Sleep with guilt and work faster to minimize flow plus energy. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 665–676. Springer, Heidelberg (2009)
13. Lam, T.W., Lee, L.K., To, I., Wong, P.: Competitive non-migratory scheduling for flow time and energy. In: Proc. SPAA, pp. 256–264 (2008)
14. Lam, T.W., Lee, L.K., To, I., Wong, P.: Speed scaling functions for flow time scheduling based on active job count. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 647–659. Springer, Heidelberg (2008)
15. Motwani, R., Phillips, S., Torng, E.: Nonclairvoyant scheduling. *Theoretical Computer Science* 130(1), 17–47 (1994)
16. Mudge, T.: Power: A first-class architectural design constraint. *IEEE Computer* 34(4), 52–58 (2001)
17. Steele, J.M.: The Cauchy-Schwarz master class: An introduction to the art of mathematical inequalities, p. 136. Cambridge University Press, Cambridge (2004)
18. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. FOCS, pp. 374–382 (1995)

# A Robust PTAS for Machine Covering and Packing<sup>∗</sup>

Martin Skutella and José Verschae

Institute of Mathematics, TU Berlin, Germany  
`{skutella,verschae}@math.tu-berlin.de`

**Abstract.** Minimizing the makespan or maximizing the minimum machine load are two of the most important and fundamental parallel machine scheduling problems. In an online scenario, jobs are consecutively added and/or deleted and the goal is to always maintain a (close to) optimal assignment of jobs to machines. The reassignment of a job induces a cost proportional to its size and the total cost for reassigning jobs must preferably be bounded by a constant  $r$  times the total size of added or deleted jobs. Our main result is that, for any  $\varepsilon > 0$ , one can always maintain a  $(1 + \varepsilon)$ -competitive solution for some constant reassignment factor  $r(\varepsilon)$ . For the minimum makespan problem this is the first improvement of the  $(2 + \varepsilon)$ -competitive algorithm with constant reassignment factor published in 1996 by Andrews, Goemans, and Zhang.

## 1 Introduction

We consider two basic scheduling problems where  $n$  jobs need to be assigned to  $m$  identical parallel machines. Each job  $j$  has a processing time  $p_j \geq 0$  and the load of a machine is the total processing time of jobs assigned to it. The *machine covering problem* asks for an assignment of jobs to machines that maximizes the minimum machine load. In the *minimum makespan problem* (or *machine packing problem*), we wish to find a schedule minimizing the maximum machine load.

Both problems are well known to be strongly NP-hard and both allow for a polynomial-time approximation scheme (PTAS); see, e.g., [2, 9, 16]. They have also been studied extensively in the online setting where jobs arrive one by one and must immediately be assigned to a machine at their arrival; see, e.g., [1, 13]. The best known online algorithm for the minimum makespan problem is a 1.9201-competitive algorithm [8]. The best lower bound on the competitive ratio of any deterministic online algorithm currently known is 1.88 [10]. For randomized online algorithms there is a lower bound of  $e/(e - 1) \approx 1.58$ ; see [6, 12].

The online variant of the machine covering problem turns out to be less tractable and there is no online algorithm with constant competitive ratio. The best possible deterministic algorithm greedily assigns jobs to the least loaded machine, and has competitive ratio  $1/m$ ; see [16]. In [5] an upper bound of  $O(1/\sqrt{m})$  is shown for the competitive ratio of any randomized online algorithm, and there is an almost matching  $\tilde{\Omega}(1/\sqrt{m})$ -competitive algorithm.

---

<sup>∗</sup> Supported by Berlin Mathematical School and by DFG research center MATHEON.

There is also a large amount of literature on *online load balancing* problems where jobs are allowed to arrive or depart the system; see, e.g., [4]. Most authors consider a relaxed notion of competitive ratio, where the solution is not compared against the current offline optimum but the largest objective value seen so far.

**Proportional reassignment cost.** We study a relaxed online scenario known as *online load balancing with proportional reassignment cost*. In this setting, jobs may arrive or depart at any time, and when a new job enters the system it must immediately be assigned to a machine. Again, the objective is either to minimize the makespan or to maximize the minimum machine load. Furthermore, upon arrival or departure of a job, one is allowed to reassign other jobs by paying an associated cost: reassigning job  $j$  incurs a cost of  $c \cdot p_j$  for some given constant  $c > 0$ . By scaling we can assume that  $c = 1$ .

The cost due to reassignments is controlled by the *reassignment factor* which is defined as follows. Let  $J$  be the set of jobs that have so far appeared in the system, and let  $J_L \subseteq J$  be the set of jobs that have left the system. We define the reassignment factor  $r$  of an algorithm as the worst case ratio between  $\sum_{j \in J} p_j + \sum_{j \in J_L} p_j$  and the total cost due to reassignments. Alternatively, we can interpret this framework in the following way: given a parameter  $r > 0$ , the arrival or departure of a job  $j$  adds an amount of  $r \cdot p_j$  to the total budget available to spend on reassignments. We call  $r \cdot p_j$  the *reassignment potential* induced by job  $j$ .

Note that  $r = 0$  means that no reassignment is allowed, and thus we are in the classical online setting. On the other hand,  $r = \infty$  implies that we are allowed to reassign all jobs at each arrival/departure, and thus we fall back to the offline case. We are interested in developing  $\alpha$ -competitive algorithms where the migration factor  $r$  is bounded by a constant. Furthermore, we study the trade-off between  $\alpha$  and  $r$ . Arguably, the best that we can hope for under this framework is a *robust PTAS* (also known as *dynamic PTAS*), that is, a family of polynomial-time  $(1 + \varepsilon)$ -competitive algorithms with constant reassignment factor  $r = r(\varepsilon)$ , for all  $\varepsilon > 0$ .

For the minimum makespan problem with proportional reassignment cost, Westbrook [15] gives a 6-competitive algorithm with reassignment factor 1 (according to our definition<sup>1</sup>). Andrews, Goemans, and Zhang [3] improve upon this result, obtaining a 3.5981-competitive algorithm with reassignment factor 1. Furthermore, they give a  $(2 + \varepsilon)$ -competitive algorithm with constant reassignment factor  $r(\varepsilon) \in O(1/\varepsilon)$ .

**Related work.** Sanders, Sivadasan, and Skutella [11] consider a somewhat tighter online model, known as the *bounded migration* framework. This model can be interpreted as the reassignment model with the following modification: after the arrival or departure of a job  $j$ , its reassignment potential  $r \cdot p_j$  must be

---

<sup>1</sup> Our definition differs slightly from the one given in [15]: they do not consider the departure of jobs to add any reassignment potential, and the first assignment of a job also induces cost in their case. However, the concept of constant reassignment factors is the same in both models.

immediately spent or is otherwise lost. In the bounded migration scenario, the value  $r$  is called the *migration factor* of the algorithm, and is a measure of the robustness of the constructed solutions.

Sanders et al. study the bounded migration model for the special case when jobs are not allowed to depart. For the minimum makespan problem, they give a  $3/2$ -competitive algorithm with migration factor  $4/3$ . Moreover, using well known rounding techniques, they formulate the problem as an integer linear programming (ILP) feasibility problem in constant dimension. Combining this with an ILP sensitivity analysis result, they obtain a robust PTAS for the bounded migration model with job arrivals only. An important consequence of their analysis is that no special structure of the solutions is needed to achieve robustness. More precisely, it is possible to take an arbitrary  $(1 + \varepsilon)$ -approximate solution and, at the arrival of a new job, turn it into a  $(1 + \varepsilon)$ -approximate solution to the augmented instance while keeping the migration factor constant. This feature prevents their technique from working in the job departure case.

The machine covering problem is also considered by Sanders et al. [11]. They describe an interesting application of the online version of this problem in the context of storage area networks and describe a 2-competitive algorithm with migration factor 1. Moreover, they give a counterexample showing that it is not possible to start with an arbitrary  $(1/2 + \varepsilon)$ -approximate solution, and then maintain the approximation guarantee while keeping the migration factor constant. This implies that the ideas developed in [11] for the minimum makespan problem cannot be applied directly to derive a robust PTAS for the machine covering problem. Based on ideas in [11], Epstein and Levin [7] develop a robust APTAS for the Bin-Packing problem.

**Our Contribution.** We develop a general framework for obtaining robust PTASes in the reassignment model. Our results can be considered from various different angles and have interesting interpretations in several different contexts:

- (i) We make a significant contribution to the understanding of two fundamental *online scheduling* problems on identical parallel machines that are also relevant building blocks for many more complex real-world problems.
- (ii) We advance the understanding of *robustness* of parallel machine schedules under job arrival and departure, and give valuable insights related to the *sensitivity analysis* of parallel machine schedules.
- (iii) We achieve the best possible performance bound for *machine balancing* with proportional reassignment costs, improving upon earlier work by Westbrook [15] and Andrews, Goemans, and Zhang [3].

Our techniques for deriving the robust PTAS take the ideas in [2] and [11] one step further. We first prove that it is not possible to start with an arbitrary  $(1 - \varepsilon)$ -approximate solution and, at the arrival of a new job, maintain the competitive ratio with constant migration factor. One of our main contributions is to overcome this limitation by giving extra structure to the constructed solutions. Roughly speaking, we do this by asking for solutions such that the sorted vector of machine load values is lexicographically optimal. It turns out that a solution with this property is not only optimal but also robust. In the analysis

$p_7$	$p_4$	
$p_5$	$p_6$	
$p_1$	$p_2$	$p_3$

$p_7$					
$p_6$	$p_1$	$p_4$			
$p_5$	$p_2$	$p_3$			

**Fig. 1.** Left: Unique optimal solution to original instance. Right: Unique optimal solution to instance with new jobs.

we formulate a rounded scheduling problem as an ILP in constant dimension, exploit the structure of the coefficient matrix, and apply sensitivity analysis for ILPs to derive the result.

To keep the presentation short and clear, we mainly focus on the machine covering problem in this extended abstract and present a robust PTAS for the general case of jobs leaving and entering the system. An easy adaptation of the techniques here presented yields a robust PTAS for the minimum makespan problem, improving upon the  $(2 + \varepsilon)$ -competitive algorithm with constant reassignment factor by Andrews, Goemans, and Zhang [3]. Moreover, all our techniques can be extended to a very broad class of problems, where the objective functions solely depend on the load of each machine. Further details on these topics, as well as the complete proofs of our results can be found in [14].

## 2 A Lower Bound on the Best Approximation with Constant Migration Factor

We start by showing that it is not possible to maintain near-optimal solutions to the machine covering problem with constant migration factor in the model of Sanders et al. [11], if arriving jobs are arbitrarily small.

**Lemma 1.** *For any  $\varepsilon > 0$ , there is no  $(19/20 + \varepsilon)$ -competitive algorithm for the machine covering problem with constant migration factor, even for the special case without job departures.*

The proof of the lemma is based on an instance consisting of 3 machines and 7 jobs depicted in Figure 1. Details can be found in [14]. The proof of the lemma can be found in [14].

As mentioned before, this lemma justifies the use of the reassignment cost model instead of the bounded migration framework. Moreover, we see in the proof of the lemma that the limitation of the bounded migration model is caused by arbitrarily small jobs, whose reassignment potential do not allow any other job to be migrated. Nonetheless, in the reassignment model we can deal with small jobs by accumulating them as follows.

Let  $OPT$  denote the value of an optimum solution for the current set of jobs. If a new job  $j$  with  $p_j < \varepsilon \cdot OPT$  arrives, we do not schedule it immediately<sup>2</sup>. Instead,

<sup>2</sup> In order to still satisfy the strict requirements of the considered online scheduling problem, we can assume that job  $j$  is temporarily assigned to an arbitrary machine, say machine 1. Notice that this causes an increase of the reassignment factor by at most 1.

we accumulate several small jobs, until their total processing time surpasses  $\varepsilon \cdot OPT$ . We can then incorporate them as one larger job with processing time at least  $\varepsilon \cdot OPT$ . This can only decrease the value of the solution by a  $1 - \varepsilon$  factor.

The situation for the departure of small jobs is slightly more complicated. We ignore the fact that certain small jobs are gone as long as the following property holds: There is no machine which has lost jobs of total processing time at least  $\varepsilon \cdot OPT$ . Under this condition, the objective function is affected by less than a factor  $1 + \varepsilon$ . If, on the other hand, there is such a machine, we can treat the set of jobs that have left the machine as one single job of size at least  $\varepsilon \cdot OPT$  and act accordingly. Notice that the property above has to be checked dynamically after each reassignment of jobs caused by newly arriving or departing jobs.

**Assumption 1.** *W.l.o.g., all arriving/departing jobs are bigger than  $\varepsilon \cdot OPT$ .*

### 3 A Stable Estimate of the Optimum Value

In this section we describe an upper bound on the optimum solution value of the machine covering problem, also introduced in [2]. However, for it to be useful for the robust PTAS, we need to show that this upper bound is stable. That is, at the arrival/departure of a new job, its value must not change by more than a constant factor.

Let  $\mathcal{I} = (J, M)$  be an instance of our problem, where  $J$  is a set of  $n$  jobs and  $M$  a set of  $m$  machines. Given a subset of jobs  $L$ , we denote by  $p(L)$  the total processing time of jobs in  $L$ , i.e.,  $p(L) := \sum_{j \in L} p_j$ . Instance  $\mathcal{I}$  satisfies property  $(*)$  if  $p_j \leq p(J)/m$ , for all  $j \in J$ . The most natural upper bound to use for our problem is the average load  $p(J)/m$ . Under condition  $(*)$ , the average load is always within a factor 2 of  $OPT$ ; see [21].

**Lemma 2.** *If instance  $\mathcal{I}$  satisfies  $(*)$ , then  $\frac{p(J)}{2m} \leq OPT \leq \frac{p(J)}{m}$ .*

Now we show how to transform arbitrary instances to instances satisfying  $(*)$  without changing the optimal solution value. If  $p_j > p(J)/m \geq OPT$ , then we can assume that  $j$  is being processed on a machine of its own. Thus, removing  $j$  plus its corresponding machine does not change the optimal solution value, but it does reduce the average load. We can iterate this idea until no job is strictly larger than the average load. We call this procedure Algorithm STABLE-AVERAGE. Also, we call  $L$  the set of jobs and  $w$  the number of machines of the corresponding remaining instance. More importantly, we define  $A$  to be the average load of this instance, i.e.,  $A := p(L)/w$ . We call value  $A$  the *stable average* of instance  $\mathcal{I}$ . Also, we obtain that solving the instance with job set  $L$  and  $w$  identical machines is equivalent to solving  $\mathcal{I}$ . Thus Lemma 2 yields:

**Lemma 3.** *The upper bound  $A$  computed by the algorithm above satisfies  $OPT \leq A \leq 2 \cdot OPT$ .*

It is easy to see that, in general, the factor by which the upper bound changes at the arrival/departure of a job is not bounded (consider two machines and two

jobs of sizes 1 and  $K \gg 1$ , respectively; then one job of size  $K - 1$  arrives). However, we can show that if  $A$  is increased by more than a factor 2, then the instance was trivial to solve in the first place. To this end, we show that if the value  $A$  is increased by more than a factor of 2, then a significant amount of jobs must have arrived to the system. The proof of the next lemma is omitted.

**Lemma 4.** *Consider two arbitrary instances  $\mathcal{I} = (J, M)$  and  $\mathcal{I}' = (J', M)$ . Let  $A$ ,  $L$  and  $w$  (resp.  $A'$ ,  $L'$  and  $w'$ ) be the returned values when applying Algorithm STABLE-AVERAGE to  $\mathcal{I}$  (resp.  $\mathcal{I}'$ ). If  $A' > 2A$ , then  $|J \Delta J'| > w/2$  (here  $\Delta$  denotes the symmetric difference between the two sets).*

Moreover, we say that an instance is *trivial* if Algorithm STABLE-AVERAGE returns  $w = 1$ . In this case, the optimal solution to the instance can be constructed by processing the  $m - 1$  largest jobs each on a machine of their own, and the remaining jobs on the remaining machine. Moreover, the optimal value  $OPT$  equals  $A$ . With this definition, we obtain the following easy consequence of Lemma 4.

**Corollary 1.** *Assume that  $\mathcal{I}$  is nontrivial and that instance  $\mathcal{I}'$  is obtained from  $\mathcal{I}$  by adding one job. Then, it must hold that  $A \leq A' \leq 2 \cdot A$ .*

## 4 The Structure of Robust Solutions

In the following, we show a sufficient condition to guarantee that we can achieve near optimal solutions when jobs arrive or depart. For clarity, we first consider a static case: Given an instance  $\mathcal{I}$ , we construct a  $(1 - O(\varepsilon))$ -approximate solution having enough structure so that at the arrival or departure of a job larger than  $\varepsilon \cdot OPT$ , we can maintain the approximation guarantee using constant migration factor. Note that since we are using constant migration factor, we only use the reassignment potential induced by the arriving or departing job. Nonetheless, we do not take care of maintaining the structure so that this procedure can be iterated when further jobs arrive (or depart). We deal with this more complicated scenario in Section 5.

We concentrate on the case of a newly arriving job. But the presented ideas and techniques can be easily adapted to the case of a departing job. Let  $\mathcal{I} = (J, M)$  be an arbitrary instance with optimal value  $OPT$ . If there is no possible confusion, we will also use  $OPT$  to refer to some optimal schedule for  $\mathcal{I}$ . We call  $\mathcal{I}' = (J', M)$  the instance with the additional arriving job  $p_{j^*}$ , and  $OPT'$  the new optimal value.

**Lemma 5.** *Assume that  $\mathcal{I}$  is trivial. Then, starting from an optimal solution, one can construct a  $(1 - \varepsilon)$ -approximate solution to  $\mathcal{I}'$  by using migration factor at most  $2/\varepsilon$ .*

The proof of the lemma can be found in [14]. We devote the rest of this section to the case of nontrivial instances.

## 4.1 Compact Description of a Schedule

As usual in PTASes, we first simplify our instance by rounding. In this section we briefly show how to do this for our problem. The techniques are similar to the ones found, e.g., in [2016]. Nonetheless, we must be careful to ensure that the resulting compact description of schedules is also compatible with schedules containing any new job that may arrive.

It is a well known fact that by only loosing a  $1/(1 + \varepsilon)$  factor in the objective function, we can round down all processing times to the nearest power of  $1 + \varepsilon$ . Thus, in the rest of this paper we assume that, for every job  $j$ , it holds that  $p_j = (1 + \varepsilon)^k$  for some  $k \in \mathbb{Z}$ . Moreover, we need to compute an upper bound,  $\text{UB}$ , which is within a constant factor  $\gamma > 1$  of the optimal value:  $\text{OPT} \leq \text{UB} \leq \gamma \cdot \text{OPT}$ . Throughout this section we use  $\text{UB} = A$ , so that  $\gamma = 2$ . For the general case in Section 5, however, we have to choose this upper bound more carefully.

In what follows, we round our instance such that the number of different processing times is constant. To this end, let  $\sigma, \Sigma \geq 1$  be two constant parameters that will be chosen appropriately later. Our rounding ensures that all processing times belong to the interval  $[\varepsilon \cdot \text{UB}/\sigma, \Sigma \cdot \text{UB}]$ . The value  $\sigma$  will be chosen big enough so that every job that is smaller than  $\varepsilon \cdot \text{UB}/\sigma$  is also smaller than  $\varepsilon \cdot \text{OPT}$ . On the other hand, since  $\Sigma \geq 1$ , every job that is larger than  $\Sigma \cdot \text{UB}$  is also larger than  $\text{OPT}$ , and thus should be processed on a machine of its own. Moreover, since we are assuming that  $\mathcal{I}$  is nontrivial, Corollary 1 implies that  $\text{UB}' := A' \leq 2 \cdot \text{UB}$ . We can therefore choose  $\Sigma \geq 2$  to ensure that a job that is larger than  $\Sigma \cdot \text{UB}$  is also larger than  $\text{OPT}'$ , and thus can also be processed on a machine of its own in optimal solutions to  $\mathcal{I}'$ . This will help to simultaneously round  $\mathcal{I}$  and  $\mathcal{I}'$ , yielding the same approximation guarantee for both instances. More importantly, we note that since the lower and upper bounds are within a constant factor, the rounded instances only have  $O(\log_{1+\varepsilon}(1/\varepsilon)) = O(1/\varepsilon \log(1/\varepsilon))$  different job sizes. Consider the index set

$$I(\text{UB}) := \{i \in \mathbb{Z} : \varepsilon \cdot \text{UB}/\sigma \leq (1 + \varepsilon)^i \leq \Sigma \cdot \text{UB}\} = \{\ell, \dots, u\}.$$

The new rounded instance derived from  $\mathcal{I}$  is described by defining a vector  $N = (n_i)_{i \in I}$ , whose entry  $n_i$  denotes the number of jobs of size  $(1 + \varepsilon)^i$ . More precisely, vector  $N$  is defined as follows. For each  $i = \ell + 1, \dots, u - 1$ , we let

$$n_i := |\{j \in J : p_j = (1 + \varepsilon)^i\}|, \quad (1)$$

i.e.,  $n_i$  is the number of jobs of size  $(1 + \varepsilon)^i$  in the original instance. We call these jobs *big* with respect to  $\text{UB}$ . Bigger jobs are rounded down to  $(1 + \varepsilon)^u$ , i.e., we set

$$n_u := |\{j \in J : p_j \geq (1 + \varepsilon)^u\}|. \quad (2)$$

We call these jobs *huge* with respect to  $\text{UB}$ . Finally, jobs that are smaller than or equal to  $(1 + \varepsilon)^\ell$  are said to be *small* with respect to  $\text{UB}$ . We replace them by jobs of size  $(1 + \varepsilon)^\ell$ , i.e., we set

$$n_\ell := \left\lfloor \frac{1}{(1+\varepsilon)^\ell} \sum_{j: p_j \leq (1+\varepsilon)^\ell} p_j \right\rfloor. \quad (3)$$

By definition (3), the total processing time of small jobs in  $N$  and  $\mathcal{I}$  is roughly equal. By slightly abusing notation, in what follows we also use the symbol  $N$  to refer to the scheduling instance defined by the vector  $N$ . The next lemma is also used in [2]. We omit the proof.

**Lemma 6.** *The value of an optimal solution to  $N$  is within a  $1 - O(\varepsilon)$  factor of  $OPT$ .*

Notice that a solution to the rounded instance can be turned into a schedule for the original instance  $\mathcal{I}$  by simply removing all jobs of size  $(1 + \varepsilon)^\ell$  from the schedule of  $N$ , and then applying a list scheduling algorithm to process the original small jobs. By the same argument as in the proof of Lemma 6, this only costs a factor  $1 - O(\varepsilon)$  in the objective function. We can thus restrict to work with instance  $N$ . To describe a schedule for  $N$  in a compact way, we consider the following definition.

**Definition 1.** *For a given schedule, a machine obeys configuration  $k : I(\text{UB}) \rightarrow \mathbb{N}_0$ , if  $k(i)$  equals the number of jobs of size  $(1 + \varepsilon)^i$  assigned to that machine, for all  $i \in I(\text{UB})$ . The load of configuration  $k$  is defined as  $\text{load}(k) := \sum_{i \in I(\text{UB})} k(i)(1 + \varepsilon)^i$ .*

Let us now consider set  $K := \{k : I(\text{UB}) \rightarrow \mathbb{N}_0 : k(i) \leq \sigma\Sigma/\varepsilon + 1 \text{ for all } i \in I\}$ . Notice that  $|K| \leq (\sigma\Sigma/\varepsilon + 1)^{|I(\text{UB})|} \in 2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$ . The next lemma assures that these are all necessary configurations that we need to consider.

**Lemma 7.** *There is an optimal solution to  $N$  with all machine configurations in  $K$ .*

The proof is given in [14]. Note that the number of jobs per machine in an optimal solution can be upper bounded by  $\sigma/\varepsilon + 1$ . Thus, the set  $K$  contains more configurations than are really needed. Nonetheless, the overestimation of the number of jobs is necessary so that, when a new job arrives and the upper bound increases (by at most a factor of 2), the set  $K$  still contains all necessary configurations.

The optimal schedule for  $N$  found in Lemma 7 can be described by a vector  $(x_k)_{k \in K}$ , where  $x_k$  denotes the number of machines that obey configuration  $k$ . We can see that vector  $x$  satisfies the following set of constraints:

$$\sum_{k \in K} x_k = m, \quad \sum_{k \in K} k(i) \cdot x_k = n_i \quad \text{for all } i \in I(\text{UB}) \quad (4)$$

and  $x_k \in \mathbb{Z}_{\geq 0}$  for all  $k \in K$ . We denote by  $\mathcal{A} = \mathcal{A}(K, I)$  the matrix defining the set of equations (4); the corresponding right-hand-side is denoted by  $b(N, m)$ . Also,  $D = \{x \in \mathbb{Z}_{\geq 0}^K : \mathcal{A} \cdot x = b(N, m)\}$  denotes the set of non-negative integral solutions to (4).

## 4.2 Constructing Stable Solutions

In the following we present the main structural contribution of this paper: We show how to obtain a robust optimal solution to  $N$  such that, upon arrival (or departure) of a new job of size at least  $\varepsilon \cdot OPT$ , we need to migrate jobs of total

processing time at most  $f(\varepsilon) \cdot OPT$  in order to maintain optimality. This implies that the migration factor needed for this case is upper bounded by  $f(\varepsilon)/\varepsilon$ .

Let us order and relabel the set of configurations  $K = \{k_1, \dots, k_{|K|}\}$  in non-decreasing order of their load, i.e.,  $\text{load}(k_1) \leq \text{load}(k_2) \leq \dots \leq \text{load}(k_{|K|})$ .

**Definition 2.** Let  $x, x' \in D$ . We say that  $x'$  is lexicographically smaller than  $x$ , denoted  $x' \prec_{lex} x$ , if  $x_k = x'_k$  for all  $k \in \{k_1, \dots, k_q\}$ , and  $x'_{k_{q+1}} < x_{k_{q+1}}$ , for some  $q$ .

By definition,  $\prec_{lex}$  defines a total order on the solution set  $D$ . Thus there exists a unique lexicographically minimum vector in  $x^* \in D$ . We show that  $x^*$  has the proper structure needed for our purposes. Note that, in particular, it maximizes the minimum machine load. Moreover,  $x^*$  can be computed in polynomial time by solving a sequence of ILPs in constant dimension as follows: for  $q = 1, \dots, |K|$ , set

$$x_{k_q}^* := \min \{x_{k_q} : x \in D \text{ and } x_{k_r} = x_{k_r}^* \text{ for all } r = 1, \dots, q-1\} .$$

Alternatively, we can find  $x^*$  by solving a single ILP in constant dimension, i.e., by minimizing a carefully chosen linear function over  $D$ . Let  $\lambda := (m+1)^{-1}$ , and define  $c_q := \lambda^q$  for  $q = 1, \dots, |K|$ . The following ILP is denoted by [LEX]:

$$\min \left\{ \sum_{q=1}^{|K|} c_q \cdot x_{k_q} : \mathcal{A} \cdot x = b(N, m) \text{ and } x_k \in \mathbb{Z}_{\geq 0} \text{ for all } k \in K \right\} .$$

The proof of the following lemma can be found in [4].

**Lemma 8.** Let  $z$  be an optimal solution to [LEX]. Then,  $z$  is the lexicographically minimal solution in  $D$ .

Let  $S$  be the schedule corresponding to  $z$ . We next show that  $S$  is robust. The new job  $j^*$  can be incorporated into the ILP by only slightly changing the right-hand-side of [LEX]. Indeed, as discussed before, we can assume that  $p_{j^*}$  is a power of  $(1 + \varepsilon)$  and is larger than  $\varepsilon \cdot OPT \geq \varepsilon \cdot \text{UB}/\sigma$  (by choosing  $\sigma \geq \gamma$ ). Then, we can round the new instance  $\mathcal{I}'$  by defining a vector  $N' = (n'_i)_{i \in I}$  as follows: for  $i = \ell, \dots, u-1$  let

$$n'_i = \begin{cases} n_i + 1 & \text{if } p_{j^*} = (1 + \varepsilon)^i, \\ n_i & \text{otherwise,} \end{cases} \quad \text{and} \quad n'_u = \begin{cases} n_u + 1 & \text{if } p_{j^*} \geq (1 + \varepsilon)^u, \\ n_u & \text{otherwise.} \end{cases}$$

In other words, if  $p_{j^*} > \Sigma \text{UB} \geq 2 \text{UB} \geq \text{UB}' \geq OPT'$  then job  $j^*$  is processed on a machine of its own. Therefore we can assume that its size is just  $(1 + \varepsilon)^u$ . Also, note that all jobs whose size was rounded down to  $(1 + \varepsilon)^u$  in the original instance  $\mathcal{I}$  are still larger than  $\Sigma \cdot \text{UB} \geq \text{UB}'$ , and thus get a machine of their own in  $OPT'$ . Moreover, jobs that are smaller than  $\varepsilon \cdot \text{UB}/\sigma$  are also smaller than  $\varepsilon \cdot OPT'$ . Thus, using the same argument as in Lemma 6, solving instance  $N'$  yields a  $(1 - O(\varepsilon))$ -approximate solution to  $\mathcal{I}'$ . Also, analogously to Lemmas 7 and 8,

we can solve this instance by optimizing the following modification of [LEX], which we call [LEX]':

$$\min \left\{ \sum_{q=1}^{|K|} c_q \cdot x_{k_q} : \mathcal{A}x = b(N', m) \text{ and } x_k \in \mathbb{Z}_{\geq 0} \text{ for all } k \in K \right\}.$$

Let  $z'$  be an optimum solution to [LEX]'. Notice that [LEX] and [LEX]' only differ in one entry of the right-hand-side vector by one. Thus, by a result from sensitivity analysis of ILPs, the optimum solutions  $z$  and  $z'$  are relatively close. This yields the next theorem, whose detailed proof is given in [4].

**Theorem 1.** *There exists a static robust PTAS if the arriving job is larger than  $\varepsilon \cdot OPT$ .*

**Running time.** By the proof of Theorem 1, given  $z$ , we can compute  $z'$  by exhaustive search through all vectors whose components differ from  $z$  by at most  $2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$ . Thus, the running time needed to compute the desired solution to  $\mathcal{I}'$  is  $2^{2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}}$ .

**Job departure.** The approach presented in this section also works for the job departure case. Indeed, if instance  $\mathcal{I}'$  contains one job less than  $\mathcal{I}$ , we can assume that  $\mathcal{I}'$  is nontrivial (otherwise see Lemma 5). Therefore  $UB' \geq UB/2$ , and thus choosing  $\sigma \geq 2\gamma$  implies that  $I(UB)$  contains all job sizes between  $\varepsilon \cdot OPT'$  and  $OPT'$ . Thus, rounding instance  $\mathcal{I}'$  within this range decreases the optimum value by at most a factor  $(1 - O(\varepsilon))$ . Again, the right hand sides of [LEX] and [LEX]' differ in only one entry and thus the migration factor needed to construct the solution given by [LEX]' is at most  $2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$ .

## 5 Maintaining Robust Solutions Dynamically

In the previous section we showed how to construct a robust  $(1 - \varepsilon)$ -approximate solution, so that we can maintain the approximation guarantee at the arrival (departure) of an arbitrary new big job and keep the migration factor bounded. Nonetheless, we cannot further iterate this method when more jobs arrive or depart since the optimum values of the new instances may become arbitrarily large (small), and thus the range  $I(UB)$  is not large enough to guarantee the approximation ratios of the rounded instances. On the other hand, we cannot make the index set  $I(UB)$  larger so as to simultaneously round all possible instances and maintain the number of job sizes constant.

We deal with this difficulty by dynamically adjusting the set  $I(UB)$ . In doing so, we must be extremely careful not to destroy the structure of the constructed solutions and maintain the reassignment cost bounded. Whenever the set  $I(UB)$  is shifted to the right (left), we must regroup small jobs into larger (smaller) groups. Therefore,  $I(UB)$  should be shifted only if we can guarantee that there is enough reassignment potential accumulated to regroup all small jobs and simultaneously maintain the structure of optimal solutions. Let  $\mathcal{I}$  be the instance after the  $t$ -th job arrival/departure. For  $t = 1, 2, \dots$ , we run the following algorithm on the current instance  $\mathcal{I}$ .

## ROBUST PTAS

1. Run Algorithm STABLE-AVERAGE on  $\mathcal{I}$  to compute  $A$  and  $w$ .
2. If variable  $A_0$  is undefined or  $A \notin [A_0/2, 2A_0]$ , then set  $\mathcal{I}_0 := \mathcal{I}$  and  $A_0 := A$ .
3. Set  $\text{UB} := 2A_0$  and define sets  $I(\text{UB})$ ,  $K$ , and the ILP [LEX] as in Section 4. Compute the optimum solution  $z$  to [LEX] and the corresponding schedule  $S$ .

Notice that throughout the algorithm,  $OPT \leq A \leq 2A_0 = \text{UB}$ , and thus  $\text{UB}$  is indeed an upper bound on  $OPT$ . Moreover,  $OPT \geq A/2 \geq A_0/4 = \text{UB}/8$ , and thus  $\text{UB}$  is within a factor 8 of  $OPT$  (i.e.,  $\gamma = 8$  with the notation of Section 4). By the discussion in Section 4.2, an appropriate choice of values  $\sigma$  and  $\Sigma$  guarantees that all constructed solutions are  $(1 - O(\varepsilon))$ -approximate. It remains to show that the needed reassignment factor is bounded by a constant.

For the analysis we partition the iterations of the algorithm into blocks. Each block  $B$  consists of a consecutive sequence of iterations where the value of  $A_0$  is kept constant. Thus, for each instance  $\mathcal{I}$  occurring in  $B$ , its stable average  $A$  belongs to the interval  $[A_0/2, 2A_0]$ . Consider two consecutive instances  $\mathcal{I}$  and  $\mathcal{I}'$  that belong to the same block. We add a symbol *prime* to denote the variables corresponding to  $\mathcal{I}'$  (e.g.,  $A'$  is the stable average of  $\mathcal{I}'$ ). Since  $\mathcal{I}$  and  $\mathcal{I}'$  belong to the same block  $B$ , it holds that  $\text{UB} = 2A_0 = \text{UB}'$ , and thus the sets  $I(\text{UB})$  and  $K$  coincide with  $I(\text{UB}')$  and  $K'$ , respectively. Therefore, as already discussed in Section 4.2, the ILPs [LEX] and [LEX]' only differ in one entry of their right-hand-side vectors and the same reasoning as in Theorem 4.2 yields the following lemma.

**Lemma 9.** *If two consecutive instances  $\mathcal{I}$  and  $\mathcal{I}'$  belong to the same block of iterations, then the migration factor used to obtain  $S'$  from  $S$  is at most  $2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$ .*

It remains to consider the case that instance  $\mathcal{I}$  and the next instance  $\mathcal{I}'$  belong to different blocks. For this, we assume that  $\mathcal{I}'$  contains one more job than  $\mathcal{I}$  and that  $A' > 2A_0$ . We can deal with the case  $A' < A_0/2$  in an analogous way. By Lemma 5 we can assume that  $\mathcal{I}$  is nontrivial. Assume that  $\mathcal{I}$  belongs to block  $B$ , and consider the value of  $A_0$  corresponding to this block. It holds that  $A_0 \leq A \leq 2A_0$ . Also, since  $\mathcal{I}$  is nontrivial, Corollary 4 ensures that  $A \leq A' \leq 2A$ , and therefore  $\text{UB} \leq \text{UB}' \leq 4\text{UB}$ .

In order to compare solutions  $z \in \mathbb{N}_0^K$  and  $z' \in \mathbb{N}_0^{K'}$ , we need to interpret them in a common euclidean space containing them. Notice that huge jobs of  $\mathcal{I}$  have processing time larger than  $\Sigma \cdot \text{UB} \geq \text{UB}'$  (assuming  $\Sigma \geq 4$ ). These jobs get a machine of their own in solutions  $OPT$ ,  $OPT'$ ,  $S$ , and  $S'$ ; thus, we do not need to consider them. We can therefore assume that all jobs of  $\mathcal{I}$  and  $\mathcal{I}'$  have processing time at most  $\Sigma \cdot \text{UB}$ . In particular, the entries of vector  $N' = (n'_i)_{i \in I(\text{UB}')}$  are zero if  $(1 + \varepsilon)^i > \Sigma \cdot \text{UB}$ . We can thus interpret  $N'$  as a vector in  $\mathbb{N}_0^{I(\text{UB})}$  by setting to zero the entries  $n'_i$  with  $(1 + \varepsilon)^i < \varepsilon \cdot \text{UB}'/\sigma$ . With this simplification, all feasible solutions to [LEX]' corresponds to solutions to [LEX] where the right hand side has been modified according to  $N'$ . Thus,  $z'$  can be regarded as an optimal solution to this modified version of [LEX].

We bound the difference between  $N$  and  $N'$ , which allows us to bound the difference between  $z$  and  $z'$ . This will imply the result on the reassignment factor.

**Lemma 10.** *Let  $q$  be the number of jobs that have arrived in block  $B$ , including the job that made the algorithm change to the next block. Then,  $\|N - N'\|_1 \in O(q/\varepsilon)$ .*

The proof of the lemma can be found in [14]. Applying Lemma 10 and the same proof technique as in Theorem 1, we obtain the following lemma.

**Lemma 11.** *The reassignment potential used to construct  $S'$  is at most  $q \cdot A_0 \cdot 2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$ .*

**Theorem 2.** *For the machine covering problem with jobs arriving and departing online, there exists a  $(1 - \varepsilon)$ -competitive polynomial algorithm with constant reassignment factor at most  $2^{O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})}$ .*

## References

1. Albers, S.: Online algorithms: a survey. Mathematical Programming 97, 3–26 (2003)
2. Alon, N., Azar, Y., Woeginger, G.J., Yadid, T.: Approximation schemes for scheduling on parallel machines. Journal of Scheduling 1, 55–66 (1998)
3. Andrews, M., Goemans, M., Zhang, L.: Improved bounds for on-line load balancing. Algorithmica 23, 278–301 (1999)
4. Azar, Y.: On-line load balancing. In: Fiat, A., Woeginger, G.J. (eds.) Dagstuhl Seminar 1996. LNCS, vol. 1442, pp. 178–195. Springer, Heidelberg (1998)
5. Azar, Y., Epstein, L.: On-line machine covering. Journal of Scheduling 1, 67–77 (1998)
6. Chen, B., van Vliet, A., Woeginger, G.J.: Lower bounds for randomized online scheduling. Information Processing Letters 51, 219–222 (1994)
7. Epstein, L., Levin, A.: A robust APTAS for the classical bin packing problem. Mathematical Programming 119, 33–49 (2009)
8. Fleischer, R., Wahl, M.: Online scheduling revisited. Journal of Scheduling 3, 343–353 (2000)
9. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM 34, 144–162 (1987)
10. Rudin III, J.F., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM Journal on Computing 32, 717–735 (2003)
11. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Mathematics of Operations Research 34, 481–498 (2009)
12. Sgall, J.: A lower bound for randomized on-line multiprocessor scheduling. Information Processing Letters 63, 51–55 (1997)
13. Sgall, J.: On-line scheduling — a survey. In: Fiat, A., Woeginger, G.J. (eds.) Dagstuhl Seminar 1996. LNCS, vol. 1442, pp. 196–231. Springer, Heidelberg (1998)
14. Skutella, M., Verschae, J.: A robust PTAS for machine covering and packing. Technical Report 011-2010, Technische Universität Berlin (2010), <http://www.math.tu-berlin.de/coga/publications/techreports/2010/Report-011-2010.xhtml>
15. Westbrook, J.: Load balancing for response time. Journal of Algorithms 35, 1–16 (2000)
16. Woeginger, G.J.: A polynomial-time approximation scheme for maximizing the minimum machine completion time. Operations Research Letters 20, 149–154 (1997)

# Balancing Degree, Diameter and Weight in Euclidean Spanners

Shay Solomon\* and Michael Elkin\*\*

Department of Computer Science, Ben-Gurion University of the Negev,  
P.O.B. 653, Beer-Sheva 84105, Israel  
[{shayso,elkinm}@cs.bgu.ac.il](mailto:{shayso,elkinm}@cs.bgu.ac.il)

**Abstract.** In a seminal STOC'95 paper, Arya et al. [4] devised a construction that for any set  $S$  of  $n$  points in  $\mathbb{R}^d$  and any  $\epsilon > 0$ , provides a  $(1 + \epsilon)$ -spanner with diameter  $O(\log n)$ , weight  $O(\log^2 n)w(MST(S))$ , and constant maximum degree. Another construction of [4] provides a  $(1 + \epsilon)$ -spanner with  $O(n)$  edges and diameter  $\alpha(n)$ , where  $\alpha$  stands for the inverse-Ackermann function. Das and Narasimhan [12] devised a construction with constant maximum degree and weight  $O(w(MST(S)))$ , but whose diameter may be arbitrarily large. In another construction by Arya et al. [4] there is diameter  $O(\log n)$  and weight  $O(\log n)w(MST(S))$ , but it may have arbitrarily large maximum degree. These constructions fail to address situations in which we are prepared to compromise on one of the parameters, but cannot afford it to be arbitrarily large.

In this paper we devise a novel *unified* construction that trades between maximum degree, diameter and weight gracefully. For a positive integer  $k$ , our construction provides a  $(1 + \epsilon)$ -spanner with maximum degree  $O(k)$ , diameter  $O(\log_k n + \alpha(k))$ , weight  $O(k \log_k n \log n)w(MST(S))$ , and  $O(n)$  edges. For  $k = O(1)$  this gives rise to maximum degree  $O(1)$ , diameter  $O(\log n)$  and weight  $O(\log^2 n)w(MST(S))$ , which is one of the aforementioned results of [4]. For  $k = n^{1/\alpha(n)}$  this gives rise to diameter  $O(\alpha(n))$ , weight  $O(n^{1/\alpha(n)}(\log n)\alpha(n))w(MST(S))$  and maximum degree  $O(n^{1/\alpha(n)})$ . In the corresponding result from [4] the spanner has the same number of edges and diameter, but its weight and degree may be arbitrarily large. Our construction also provides a similar tradeoff in the complementary range of parameters, i.e., when the weight should be smaller than  $\log^2 n$ , but the diameter is allowed to grow beyond  $\log n$ .

## 1 Introduction

**Euclidean Spanners.** Consider the weighted complete graph  $\mathcal{S} = (S, \binom{S}{2})$  induced by a set  $S$  of  $n$  points in  $\mathbb{R}^d$ ,  $d \geq 2$ . The weight of an edge  $(x, y) \in \binom{S}{2}$ , for a pair of distinct points  $x, y \in S$ , is defined to be the Euclidean distance  $\|x - y\|$  between  $x$  and  $y$ . Let  $G = (S, E)$  be a spanning subgraph of  $\mathcal{S}$ , with  $E \subseteq \binom{S}{2}$ ,

\* This research has been supported by the Clore Fellowship grant No. 81265410.

\*\* This research has been supported by the BSF grant No. 2008430. Both authors are partially supported by the Lynn and William Frankel Center for Computer Science.

and assume that exactly as in  $\mathcal{S}$ , for any edge  $e = (x, y) \in E$ , its weight  $w(e)$  in  $G$  is defined to be  $\|x - y\|$ . For a parameter  $\epsilon > 0$ , the spanning subgraph  $G$  is called a  $(1 + \epsilon)$ -spanner for the point set  $S$  if for every pair  $x, y \in S$  of points, the distance  $dist_G(x, y)$  between  $x$  and  $y$  in  $G$  is at most  $(1 + \epsilon)\|x - y\|$ . Euclidean spanners were introduced more than twenty years ago by Chew [11]. Since then they evolved into an important subarea of Computational Geometry [19, 31, 12, 41, 13, 5, 23, 15, 11, 7, 14]. (See also the recent book by Narasimhan and Smid on Euclidean spanners [21], and the references therein.) Also, Euclidean spanners have numerous applications in geometric approximation algorithms [23, 16, 17], geometric distance oracles [16, 17], Network Design [18, 20] and in other areas.

In many of these applications one is required to construct a  $(1 + \epsilon)$ -spanner  $G = (S, E)$  that satisfies a number of useful properties. First, the spanner should contain  $O(n)$  (or nearly  $O(n)$ ) edges. Second, its *weight*  $w(G) = \sum_{e \in E} w(e)$  should not be much greater than the weight  $w(MST(S))$  of the minimum spanning tree  $MST(S)$  of  $S$ . Third, its *diameter*  $\Lambda = \Lambda(G)$  should be small, i.e., for every pair of points  $x, y \in S$  there should exist a path  $P$  in  $G$  that contains at most  $\Lambda$  edges and has weight  $w(P) = \sum_{e \in E(P)} w(e) \leq (1 + \epsilon)\|x - y\|$ . Fourth, its *maximum degree* (henceforth, *degree*)  $\Delta(G)$  should be small.

In a seminal STOC'95 paper, Arya et al. [4] devised a construction of  $(1 + \epsilon)$ -spanners with lightness<sup>1</sup>  $O(\log^2 n)$ , diameter  $O(\log n)$  and constant degree. They also devised a construction of  $(1 + \epsilon)$ -spanners with  $O(n)$  (respectively,  $O(n \log^* n)$ ) edges and diameter  $O(\alpha(n))$  (resp., at most  $O(1)$ ). However, in the latter construction the resulting spanners may have arbitrarily large (i.e., at least  $\Omega(n)$ ) lightness and degree. There are also a few other known constructions of  $(1 + \epsilon)$ -spanners. Das and Narasimhan [12] devised a construction with constant degree and lightness, but the diameter may be arbitrarily large. (See also [15] for a faster implementation of a spanner construction with constant degree and lightness.) There is also another construction by Arya et al. [4] that guarantees that both the diameter and the lightness are  $O(\log n)$ , but the degree may be arbitrarily large. While these constructions address some important practical scenarios, they certainly do not address all of them. In particular, they fail to address situations in which we are prepared to compromise on one of the parameters, but cannot afford this parameter to be arbitrarily large.

In this paper we devise a novel *unified* construction that trades between degree, diameter and weight gracefully. For a positive integer  $k$ , our construction provides a  $(1 + \epsilon)$ -spanner with degree  $O(k)$ , diameter  $O(\log_k n + \alpha(k))$ , lightness  $O(k \log_k n \log n)$ , and  $O(n)$  edges. Also, we can improve the bound on the diameter from  $O(\log_k n + \alpha(k))$  to  $O(\log_k n)$ , at the expense of increasing the number of edges from  $O(n)$  to  $O(n \log^* n)$ . Note that for  $k = O(1)$  our tradeoff gives rise to degree  $O(1)$ , diameter  $O(\log n)$  and lightness  $O(\log^2 n)$ , which is one of the aforementioned results of [4]. Also, for  $k = n^{1/\alpha(n)}$  it gives rise to a spanner with degree  $O(n^{1/\alpha(n)})$ , diameter  $O(\alpha(n))$  and lightness  $O(n^{1/\alpha(n)}(\log n)\alpha(n))$ .

---

<sup>1</sup> For convenience, we will henceforth refer to the normalized notion of weight  $\Psi(G) = \frac{w(G)}{w(MST(S))}$ , which we call *lightness*.

In the corresponding result from [4] the spanner has the same number of edges and diameter, but its lightness and degree may be arbitrarily large.

In addition, we can achieve lightness  $o(\log^2 n)$  at the expense of increasing the diameter. Specifically, for a parameter  $k$  the second variant of our construction provides a  $(1 + \epsilon)$ -spanner with degree  $O(1)$ , diameter  $O(k \log_k n)$ , and lightness  $O(\log_k n \log n)$ . For example, for  $k = \log^\delta n$ , for an arbitrarily small constant  $\delta > 0$ , we get a  $(1 + \epsilon)$ -spanner with degree  $O(1)$ , diameter  $O(\log^{1+\delta} n)$  and lightness  $O(\frac{\log^2 n}{\log \log n})$ .

Our unified construction can be implemented in  $O(n \log n)$  time in the algebraic computation-tree mode<sup>2</sup>. This matches the state-of-the-art running time of the aforementioned constructions [4][5].

Note that in any construction of spanners with degree  $O(k)$ , the diameter is  $\Omega(\log_k n)$ . Also, Chan and Gupta [7] showed that any  $(1 + \epsilon)$ -spanner with  $O(n)$  edges must have diameter  $\Omega(\alpha(n))$ . Consequently, our upper bound of  $O(\log_k n + \alpha(k))$  on the diameter is tight under the constraints that the degree is  $O(k)$  and the number of edges is  $O(n)$ . If we allow  $O(n \log^* n)$  edges in the spanner, than our bound on the diameter is reduced to  $O(\log_k n)$ , which is again tight under the constraint that the degree is  $O(k)$ .

In addition, Dinitz et al. [14] showed that for any construction of spanners, if the diameter is at most  $O(\log_k n)$ , then the lightness is at least  $\Omega(k \log_k n)$  and vice versa, if the lightness is at most  $O(\log_k n)$ , the diameter is at least  $\Omega(k \log_k n)$ . This lower bound implies that the bound on lightness in both our tradeoffs cannot possibly be improved by more than a factor of  $\log n$ . The same slack of  $\log n$  is present in the result of [4] that guarantees lightness  $O(\log^2 n)$ , diameter  $O(\log n)$  and constant degree.

**Euclidean Spanners for Random Point Sets.** For random point sets in the  $d$ -dimensional unit cube (henceforth, unit cube), we “shave” a factor of  $\log n$  from the lightness bound in both our tradeoffs, and show that the first (respectively, second) variant of our construction achieves maximum degree  $O(k)$  (resp.,  $O(1)$ ), diameter  $O(\log_k n + \alpha(k))$  (resp.,  $O(k \log_k n)$ ) and lightness that is with high probability (henceforth, w.h.p.)  $O(k \log_k n)$  (resp.,  $O(\log_k n)$ ). Note that for  $k = O(1)$  both these tradeoffs give rise to degree  $O(1)$ , diameter  $O(\log n)$  and lightness (w.h.p.)  $O(\log n)$ . In addition to these tradeoffs, we can get a  $(1 + \epsilon)$ -spanner with diameter  $O(\log n)$  and lightness (w.h.p.)  $O(1)$ .

**Spanners for Doubling Metrics.** Spanners for doubling metrics<sup>3</sup> have received much attention in recent years (see, e.g., [8][7]). In particular, Chan et al. [8] showed that for any doubling metric  $(X, \delta)$  there exists a  $(1 + \epsilon)$ -spanner with constant maximum degree. In addition, Chan and Gupta [7] devised a construction of  $(1 + \epsilon)$ -spanners for doubling metrics that achieves the optimal tradeoff

<sup>2</sup> See, e.g., Chapter 3 of [21] for the definition of the algebraic computation-tree model.

<sup>3</sup> The *doubling dimension* of a metric  $(X, \delta)$  is the smallest value  $\zeta$  such that every ball  $B$  in the metric can be covered by at most  $2^\zeta$  balls of half the radius of  $B$ . The metric  $(X, \delta)$  is called *doubling* if its doubling dimension  $\zeta$  is constant.

between the number of edges and the diameter. We present a single construction of  $O(1)$ -spanners for doubling metrics that achieves the optimal tradeoff between the degree, the diameter and the number of edges in the entire range of parameters. Specifically, for a parameter  $k$ , our construction provides an  $O(1)$ -spanner with maximum degree  $O(k)$ , diameter  $O(\log_k n + \alpha(k))$  and  $O(n)$  edges. Also, we can improve the bound on the diameter from  $O(\log_k n + \alpha(k))$  to  $O(\log_k n)$ , at the expense of increasing the number of edges from  $O(n)$  to  $O(n \log^* n)$ . More generally, we can achieve the same optimal tradeoff between the number of edges and the diameter as the spanners of [7] do, while also having the optimal maximum degree. The drawback is, however, that the stretch of our spanners is  $O(1)$  rather than  $1 + \epsilon$ .

**Spanners for Tree Metrics.** We denote by  $\vartheta_n$  the metric induced by  $n$  points  $v_1, v_2, \dots, v_n$  lying on the  $x$ -axis with coordinates  $1, 2, \dots, n$ , respectively. In a classical STOC'82 paper [26], Yao showed that there exists a 1-spanner<sup>4</sup>  $G = (V, E)$  for  $\vartheta_n$  with  $O(n)$  edges and diameter  $O(\alpha(n))$ , and that this is tight. Chazelle [9] extended the result of [26] to arbitrary tree metrics. (See also [2, 6, 25].) The problem is also closely related to the well-studied problem of computing partial-sums [24, 26, 10, 22].

In all constructions [26, 9, 26, 25] of 1-spanners for tree metrics, the degree and lightness of the resulting spanner may be arbitrarily large. Moreover, the constraint that the diameter is  $O(\alpha(n))$  implies that the degree must be  $n^{\Omega(1/\alpha(n))}$ . A similar lower bound on lightness follows from the result of [14].

En route to our tradeoffs for Euclidean spanners, we have extended the results of [26, 9, 26, 25] and devised a construction that achieves the *optimal* (up to constant factors) *tradeoff between all involved parameters*. Specifically, consider an  $n$ -vertex tree  $T$  of degree  $\Delta(T)$ , and let  $k$  be a positive integer. Our construction provides a 1-spanner for the metric  $M_T$  induced by  $T$  with  $O(n)$  edges, degree  $O(k + \Delta(T))$ , diameter  $O(\log_k n + \alpha(k))$ , and lightness  $O(k \log_k n)$ . We can also get a spanner with  $O(n \log^* n)$  edges, diameter  $O(\log_k n)$ , and the same degree and lightness as above. For the complementary range of diameter, another variant of our construction provides a 1-spanner with  $O(n)$  edges, degree  $O(\Delta(T))$ , diameter  $O(k \log_k n)$  and lightness  $O(\log_k n)$ . As was mentioned above, both these tradeoffs are optimal up to constant factors.

We show that this general tradeoff between various parameters of 1-spanners for tree metrics is useful for deriving new results (and improving existing results) in the context of Euclidean spanners and spanners for doubling metrics. We anticipate that this tradeoff will be found useful in the context of partial sums problems, and for other applications.

**Structure of the Paper.** In Sect. 2 we describe our construction of 1-spanners for tree metrics. Therein we start (Sect. 2.1) with outlining our basic scheme. We proceed (Sect. 2.2) with describing our 1-dimensional construction. In Sect.

<sup>4</sup> The graph  $G$  is said to be a *1-spanner* of  $\vartheta_n$  if for every pair of distinct vertices  $v_i, v_j \in V$ , the distance between them in  $G$  is equal to their distance  $\|i - j\|$  in  $\vartheta_n$ . Yao stated this problem in terms of partial sums.

2.3 we extend this construction to general tree metrics. In Sect. 3 we derive our results for Euclidean spanners and spanners for doubling metrics. Due to space constraints, we leave all issues of running time as well as some proofs out of this extended abstract.

**Preliminaries.** Let  $G$  be a spanning subgraph of a metric space  $M = (V, \text{dist})$ . The *stretch* between two vertices  $u, v \in V$  is defined as  $\frac{\text{dist}_G(u,v)}{\text{dist}(u,v)}$ . We say that  $G$  is a *t-spanner* for  $M$  if the maximum stretch taken over all pairs of points in  $V$  is at most  $t$ . Let  $T$  be an arbitrary tree, and denote by  $V(T)$  the vertex set of  $T$ . For any two vertices  $u, v$  in  $T$ , their (weighted) distance in  $T$  is denoted by  $\text{dist}_T(u, v)$ . The tree metric  $M_T$  induced by  $T$  is defined as  $M_T = (V(T), \text{dist}_T)$ . The *size* of  $T$ , denoted  $|T|$ , is the number of vertices in  $T$ . Finally, for a positive integer  $n$ , we denote the set  $\{1, 2, \dots, n\}$  by  $[n]$ .

## 2 1-Spanners for Tree Metrics

### 2.1 The Basic Scheme

Consider an arbitrary  $n$ -vertex (weighted) rooted tree  $(T, rt)$ , and let  $M_T$  be the tree metric induced by  $T$ . Clearly,  $T$  is both a 1-spanner and an MST of  $M_T$ , but its diameter may be huge. We would like to reduce the diameter of this 1-spanner by adding to it some edges. On the other hand, the number of edges of the resulting spanner should still be linear in  $n$ . Moreover, the lightness and the maximum degree of the resulting spanner should also be reasonably small.

Let  $H$  be a spanning subgraph of  $M_T$ . The *monotone distance* between any two points  $u$  and  $v$  in  $H$  is defined as the minimum number of hops in a 1-spanner path in  $H$  connecting them. Two points in  $M_T$  are called *comparable* if one is an ancestor of the other in the underlying tree  $T$ . The *monotone diameter* (respectively, *comparable monotone diameter*) of  $H$ , denoted  $\Lambda(H)$  (resp.,  $\bar{\Lambda}(H)$ ), is defined as the maximum monotone distance in  $H$  between any two points (resp., any two comparable points) in  $M_T$ . Observe that if any two *comparable* points are connected via a 1-spanner path that consists of at most  $h$  hops, then any two *arbitrary* points are connected via a 1-spanner path that consists of at most  $2h$  hops. Consequently,  $\bar{\Lambda}(H) \leq \Lambda(H) \leq 2\bar{\Lambda}(H)$ . We henceforth restrict attention to comparable monotone diameter in the sequel.

Let  $k$  be a fixed parameter. The first ingredient of the algorithm is to select a set of  $O(k)$  *cut-vertices* whose removal from  $T$  partitions it into a collection of subtrees of size  $O(n/k)$  each. As will become clear in the sequel, we also require this set to satisfy several additional properties. Having selected the cut-vertices, the next step of the algorithm is to connect the cut-vertices via  $O(k)$  edges, so that the monotone distance between any pair of comparable cut-vertices will be small. (This phase does not involve a recursive call of the algorithm.) Finally, the algorithm calls itself recursively for each of the subtrees.

We insert all edges of the original tree  $T$  into our final spanner  $H$ . These edges connect between cut-vertices and subtrees in the spanner. We remark that the spanner contains no other edges that connect between cut-vertices and subtrees, or between different subtrees.

## 2.2 1-Dimensional Spaces

In this section we devise an optimal construction of 1-spanners for  $\vartheta_n$ . (See Sect. 1 for its definition.) Our argument extends easily to any 1-dimensional space.

Denote by  $P_n$  the path  $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$  that induces the metric  $\vartheta_n$ . We remark that the edges of  $P_n$  belong to all spanners that we construct.

**Selecting the Cut-Vertices.** The task of selecting the cut-vertices in the 1-dimensional case is trivial. (We assume for simplicity that  $n$  is an integer power of  $k$ .) In addition to the two endpoints  $v_1$  and  $v_n$  of the path, we select the  $k - 1$  points  $r_1, r_2, \dots, r_{k-1}$  to be cut-vertices, where for each  $i \in [k - 1]$ ,  $r_i = v_{i(n/k)}$ . Indeed, by removing the  $k + 1$  cut-vertices  $r_0 = v_1, r_1, \dots, r_{k-1}, r_k = v_n$  from the path (along with their incident edges), we are left with  $k$  intervals  $I_1, I_2, \dots, I_k$  of length at most  $n/k$  each. The two endpoints  $v_1$  and  $v_n$  of the path are called the *sentinels*, and they play a special role in the construction.

**1-Spanners with Low Diameter.** In this section we devise a construction  $H_k(n)$  of 1-spanners for  $\vartheta_n$  with comparable monotone diameter  $\bar{A}(n) = \bar{A}(H_k(n))$  in the range  $\Omega(\alpha(n)) = \bar{A}(n) = O(\log n)$ .

First, the algorithm connects the  $k + 1$  cut-vertices  $r_0 = v_1, r_1, \dots, r_{k-1}, r_k = v_n$  via one of the aforementioned constructions of 1-spanners from [26, 9, 2, 6, 25] (henceforth, *list-spanner*). In other words,  $O(k)$  edges are added between cut-vertices to guarantee that the monotone distance between any two cut-vertices will be at most  $O(\alpha(k))$ . Then the algorithm connects each of the two sentinels to all other  $k$  cut-vertices. Finally, the algorithm calls itself recursively for each of the intervals  $I_1, I_2, \dots, I_k$ . At the bottom level of the recursion, i.e., when  $n \leq k$ , the algorithm uses the list-spanner to connect all points, and also connects each of the two sentinels  $v_1$  and  $v_n$  to all the other  $n - 1$  points.

Denote by  $E(n)$  the number of edges in  $H_k(n)$ , excluding edges of  $P_n$ . Clearly,  $E(n)$  satisfies the recurrence  $E(n) \leq O(k) + kE(n/k)$ , with the base condition  $E(n) = O(n)$ , for  $n \leq k$ , yielding  $E(n) = O(n)$ . Denote by  $\Delta(n)$  the maximum degree of a vertex in  $H_k(n)$ , excluding edges of  $P_n$ . Clearly,  $\Delta(n)$  satisfies the recurrence  $\Delta(n) \leq \max\{k, \Delta(n/k)\}$ , with the base condition  $\Delta(n) \leq n - 1$ , for  $n \leq k$ , yielding  $\Delta(n) \leq k$ . Including edges of  $P_n$ , the number of edges increases by  $n - 1$  units, and the maximum degree increases by at most two units.

To bound the weight  $w(n) = w(H_k(n))$  of  $H_k(n)$ , first note that at most  $O(k)$  edges are added between cut-vertices. Each of these edges has weight at most  $n - 1$ . The total weight of all edges within an interval  $I_i$  is at most  $w(n/k)$ . Observe also that  $w(P_n) = n - 1$ . Hence  $w(n)$  satisfies the recurrence  $w(n) \leq O(nk) + kw(n/k)$ , with the base condition  $w(n) = O(n^2)$ , for  $n \leq k$ . It follows that  $w(n) = O(nk \log_k n) = O(k \log_k n)w(MST(\vartheta_n))$ .

Next, we show that the comparable monotone diameter  $\bar{A}(n)$  of  $H_k(n)$  is  $O(\log_k n + \alpha(k))$ . The *monotone radius*  $R(n)$  of  $H_k(n)$  is defined as the maximum monotone distance in  $H_k(n)$  between one of the sentinels (either  $v_1$  or  $v_n$ ) and some other point in  $\vartheta_n$ . Let  $v_j$  be a point in  $\vartheta_n$ , and let  $i$  be the index such that  $i(n/k) \leq j < (i+1)(n/k)$ . Then the 1-spanner path  $\Pi$  in  $H_k(n)$  connecting the sentinel  $v_1$  and the point  $v_j$  will start with the two edges  $(v_1, v_{i(n/k)})$ ,

$(v_{i(n/k)}, v_{i(n/k)+1})$ . The point  $v_{i(n/k)+1}$  is a sentinel of the  $i$ th interval  $I_i$ , and so the path  $\Pi$  will continue recursively from  $v_{i(n/k)+1}$  to  $v_j$ . Hence, the monotone radius  $R(n)$  satisfies the recurrence  $R(n) \leq 2 + R(n/k)$ , with the base condition  $R(n) = 1$ , for  $n \leq k$ , yielding  $R(n) = O(\log_k n)$ . Finally, it is easy to verify that  $\bar{A}(n)$  satisfies the recurrence  $\bar{A}(n) \leq \max\{\bar{A}(n/k), O(\alpha(k)) + 2R(n/k)\}$ , with the base condition  $\bar{A}(n) = O(\alpha(n))$ , for  $n \leq k$ . Hence  $\bar{A}(n) = O(\log_k n + \alpha(k))$ .

**Theorem 1.** *For any  $n$ -point 1-dimensional space and a parameter  $k$ , there exists a 1-spanner with  $O(n)$  edges, maximum degree at most  $k+2$ , diameter  $O(\log_k n + \alpha(k))$  and lightness  $O(k \log_k n)$ .*

**1-Spanners with High Diameter.** In this section we devise a construction  $H'_k(n)$  of 1-spanners for  $\vartheta_n$  with comparable monotone diameter  $\bar{A}'(n) = \bar{A}(H'_k(n))$  in the range  $\bar{A}'(n) = \Omega(\log n)$ .

The algorithm connects the  $k+1$  cut-vertices  $r_0 = v_1, r_1, \dots, r_{k-1}, r_k = v_n$  via a path of length  $k$ , i.e., it adds the edges  $(r_0, r_1), (r_1, r_2), \dots, (r_{k-1}, r_k)$  into the spanner. In addition, it calls itself recursively for each of the intervals  $I_1, I_2, \dots, I_k$ . At the bottom level of the recursion, i.e., when  $n \leq k$ , the algorithm adds no additional edges to the spanner.

We denote by  $\Delta'(n)$  the maximum degree of a vertex in  $H'_k(n)$ , excluding edges of  $P_n$ . Clearly,  $\Delta'(n)$  satisfies the recurrence  $\Delta'(n) \leq \max\{2, \Delta'(n/k)\}$ , with the base condition  $\Delta'(n) = 0$ , for  $n \leq k$ , yielding  $\Delta'(n) \leq 2$ . Including edges of  $P_n$ , the maximum degree increases by at most two units, and so  $\Delta(H'_k(n)) \leq 4$ . Consequently, the number of edges in  $H'_k(n)$  is no greater than  $2n$ . To bound the weight  $w'(n) = w(H'_k(n))$  of  $H'_k(n)$ , first note that the path connecting all  $k+1$  cut-vertices has weight  $n-1$ . Observe also that  $w(P_n) = n-1$ . Thus  $w'(n)$  satisfies the recurrence  $w'(n) \leq 2(n-1) + kw'(n/k)$ , with the base condition  $w'(n) \leq n-1$ , for  $n \leq k$ , yielding  $w'(n) = O(n \log_k n) = O(\log_k n)w(MST(\vartheta_n))$ .

Note that the monotone radius  $R'(n)$  of  $H'_k(n)$  satisfies the recurrence  $R'(n) \leq k + R'(n/k)$ , with the base condition  $R'(n) \leq n-1$ , for  $n \leq k$ . Hence,  $R'(n) = O(k \log_k n)$ . It is easy to verify that the comparable monotone diameter  $\bar{A}'(n) = \bar{A}(H'_k(n))$  of  $H'_k(n)$  satisfies the recurrence  $\bar{A}'(n) \leq \max\{\bar{A}'(n/k), k+2R'(n/k)\}$ , with the base condition  $\bar{A}'(n) \leq n-1$ , for  $n \leq k$ , and so  $\bar{A}'(n) = O(k \log_k n)$ .

Finally, we remark that the spanner  $H'_k(n)$  is a planar graph.

**Theorem 2.** *For any  $n$ -point 1-dimensional space and a parameter  $k$ , there exists a 1-spanner with maximum degree 4, diameter  $O(k \log_k n)$  and lightness  $O(\log_k n)$ . Moreover, this 1-spanner is a planar graph.*

### 2.3 General Tree Metrics

In this section we extend the constructions of Sect. 2.2 to general tree metrics.

**Selecting the Cut-Vertices.** In this section we present a procedure for selecting, given a tree  $T$ , a subset of  $O(k)$  vertices whose removal from the tree partitions it into subtrees of size  $O(|T|/k)$  each. This subset will also satisfy several additional special properties.

Let  $(T, rt)$  be a rooted tree. For an inner vertex  $v$  in  $T$  with  $ch(v)$  children, we denote its children by  $c_1(v), c_2(v), \dots, c_{ch(v)}(v)$ . Suppose without loss of generality that the size of the subtree  $T_{c_1(v)}$  of  $v$  is no smaller than the size of any other subtree of  $v$ , i.e.,  $|T_{c_1(v)}| \geq |T_{c_2(v)}|, |T_{c_3(v)}|, \dots, |T_{c_{ch(v)}(v)}|$ . We say that the vertex  $c_1(v)$  is the *left-most* child of  $v$ . Also, an edge in  $T$  is called *left-most* if it connects a vertex  $v$  in  $T$  and its left-most child  $c_1(v)$ . We denote by  $P(v) = (v, c_1(v), \dots, l(v))$  the path of left-most edges leading down from  $v$  to the left-most vertex  $l(v)$  in the subtree  $T_v$  of  $T$  rooted at  $v$ . A vertex  $v$  in  $T$  is called *d-balanced*, for  $d \geq 1$ , or simply *balanced* if  $d$  is clear from the context, if  $|T_{c_1(v)}| \leq |T| - d$ . The first balanced vertex along  $P(v)$  is denoted by  $B(v)$ .

Next, we present the Procedure  $CV$  that accepts as input a rooted tree  $(T, rt)$  and a parameter  $d$ , and returns as output a subset of  $V(T)$ . If  $|T| < 2d$ , the procedure returns the empty set  $\emptyset$ . Otherwise, for each child  $c_i(b)$  of the first balanced vertex  $b = B(rt)$  along  $P(rt)$ ,  $i \in [ch(b)]$ , the procedure recursively constructs the subset  $C_i = CV((T_{c_i(b)}, c_i(b)), d)$ , and then returns  $\bigcup_{i=1}^{ch(b)} C_i \cup \{b\}$ .

Let  $(T, rt)$  be an  $n$ -vertex rooted tree, and let  $d$  be a fixed parameter. Next, we analyze the properties of the set  $C = CV((T, rt), d)$  of *cut-vertices*.

For a tree  $\tau$ , the root  $rt(\tau)$  of  $\tau$  and its left-most vertex  $l(\tau)$  are called the *sentinels* of  $\tau$ . Similarly to the 1-dimensional case, we add the two sentinels  $rt(T)$  and  $l(T)$  of the original tree  $T$  to the set  $C$  of cut-vertices. From now on we refer to the appended set  $\tilde{C} = C \cup \{rt(T), l(T)\}$  as the set of *cut-vertices*. Observe that the subset  $\tilde{C}$  induces a tree  $\tilde{\mathcal{Q}} = \mathcal{Q}(T, \tilde{C})$  over  $\tilde{C}$  in the natural way: a vertex  $v \in \tilde{C}$  is defined to be a child of its closest ancestor in  $T$  that belongs to  $\tilde{C}$ . We denote by  $T \setminus \tilde{C}$  the forest obtained from  $T$  by removing all vertices in  $\tilde{C}$ , along with the edges that are incident to them.

**Proposition 1.** 1) For  $n \geq 2d$ ,  $|\tilde{C}| \leq (n/d) + 1$ . 2) The size of any subtree in  $T \setminus \tilde{C}$  is smaller than  $2d$ . 3)  $\tilde{\mathcal{Q}} = \mathcal{Q}(T, \tilde{C})$  is a spanning tree of  $\tilde{C}$  rooted at  $rt(T)$ , with  $\Delta(\tilde{\mathcal{Q}}) \leq \Delta(T)$ . 4) For any subtree  $T'$  in  $T \setminus \tilde{C}$ , only the two sentinels of  $T'$  are incident to a vertex in  $\tilde{C}$ . Also,  $rt(T')$  is incident only to its parent in  $T$  and  $l(T')$  is incident only to its left-most child in  $T$ , unless it is a leaf in  $T$ .

Intuitively, part (4) of this proposition shows that the Procedure  $CV$  “slices” the tree in a “path-like” fashion analogous to the partition of  $\vartheta_n$  into intervals.

**1-Spanners with Low Diameter.** Consider an  $n$ -vertex (weighted) tree  $T$ , and let  $M_T$  be the tree metric induced by  $T$ . In this section we devise a construction  $\mathcal{H}_k(n)$  of 1-spanners for  $M_T$  with comparable monotone diameter  $\bar{\Lambda}(n) = \bar{\Lambda}(\mathcal{H}_k(n))$  in the range  $\Omega(\alpha(n)) = \bar{\Lambda}(n) = O(\log n)$ . Both in this construction and in the one with high diameter presented in the sequel, all edges of the original tree  $T$  are added to the spanner.

Let  $k$  be a fixed parameter such that  $4 \leq k \leq n/2 - 1$ , and set  $d = n/k$ . (We have  $n \geq 2k + 2$  and  $d > 2$ .) To select the set  $\tilde{C}$  of cut-vertices, we invoke the procedure  $CV$  on the input  $(T, rt)$  and  $d$ . Set  $C = CV((T, rt), d)$  and  $\tilde{C} = C \cup \{rt(T), l(T)\}$ . Denote the subtrees in the forest  $T \setminus \tilde{C}$  by  $T_1, T_2, \dots, T_p$ . By Proposition 1,  $|T_i| \leq k + 1$ , and each subtree  $T_i$  in  $T \setminus \tilde{C}$  has size less than  $2n/k$ .

To connect the set  $\tilde{C}$  of cut-vertices, the algorithm first constructs the tree  $\tilde{\mathcal{Q}} = \mathcal{Q}(T, \tilde{C})$ . Note that  $\tilde{\mathcal{Q}}$  inherits the tree structure of  $T$ , i.e., for any two points  $u$  and  $v$  in  $\tilde{C}$ ,  $u$  is an ancestor of  $v$  in  $\tilde{\mathcal{Q}}$  iff it is its ancestor in  $T$ . Consequently, any 1-spanner path in  $\tilde{\mathcal{Q}}$  between two arbitrary comparable<sup>5</sup> points is also a 1-spanner path for them in the original tree  $T$ . The algorithm proceeds by building a 1-spanner for  $\tilde{\mathcal{Q}}$  via one of the aforementioned generalized constructions from [9, 22, 25] (henceforth, *tree-spanner*). In other words,  $O(k)$  edges between cut-vertices are added to the spanner  $\mathcal{H}_k(n)$  to guarantee that the monotone distance in the spanner between any two comparable cut-vertices is  $O(\alpha(k))$ . Then the algorithm adds to the spanner  $\mathcal{H}_k(n)$  edges that connect each of the two sentinels to all other cut-vertices. (In fact, the leaf  $l(T)$  needs not be connected to all cut-vertices, but rather only to those which are its ancestors in  $T$ .) Finally, the algorithm calls itself recursively for each of the subtrees  $T_1, T_2, \dots, T_p$ . At the bottom level of the recursion, i.e., when  $n < 2k + 2$ , the algorithm uses the tree-spanner to connect all points, and, in addition, it adds to the spanner edges that connect each of the two sentinels  $rt(T)$  and  $l(T)$  to all the other  $n - 1$  points.

The properties of the spanner  $\mathcal{H}_k(n)$  are summarized in the next theorem.

**Theorem 3.** *For any tree metric  $M_T$  and a parameter  $k$ , there exists a 1-spanner  $\mathcal{H}_k(n)$  with  $O(n)$  edges, maximum degree at most  $\Delta(T) + 2k$ , diameter  $O(\log_k n + \alpha(k))$  and lightness  $O(k \log_k n)$ .*

We remark that the maximum degree  $\Delta(\mathcal{H})$  of the spanner  $\mathcal{H} = \mathcal{H}_k(n)$  cannot be in general smaller than the maximum degree  $\Delta(T)$  of the original tree. Indeed, consider a unit weight star  $T$  with edge set  $\{(rt, v_1), (rt, v_2), \dots, (rt, v_{n-1})\}$ . Obviously, any spanner  $\mathcal{H}$  for  $M_T$  with  $\Delta(\mathcal{H}) < n - 1$  distorts the distance between the root  $rt$  and some other vertex.

**1-Spanners with High Diameter.** The next theorem gives our construction of 1-spanners for  $M_T$  with comparable monotone diameter in the range  $\Omega(\log n)$ .

**Theorem 4.** *For any tree metric  $M_T$  and a parameter  $k$ , there exists a 1-spanner with  $O(n)$  edges, maximum degree at most  $2\Delta(T)$ , diameter  $O(k \log_k n)$  and lightness  $O(\log_k n)$ . Moreover, this 1-spanner is a planar graph.*

### 3 Euclidean Spanners

In this section we demonstrate that our 1-spanners for tree metrics can be used for constructing Euclidean spanners and spanners for doubling metrics.

We start with employing the Dumbbell Theorem of [4] in conjunction with our 1-spanners for tree metrics to construct Euclidean spanners.

**Theorem 5.** (“Dumbbell Theorem”, Theorem 2 in [4]) *Given a set  $S$  of  $n$  points in  $\mathbb{R}^d$  and a parameter  $\epsilon > 0$ , a forest  $\mathcal{D}$  consisting of  $O(1)$  rooted binary trees of size  $O(n)$  can be built in  $O(n \log n)$  time, having the following properties: 1) For*

---

<sup>5</sup> This may not hold true for two points that are not comparable, as their least common ancestor may not belong to  $\tilde{\mathcal{Q}}$ .

each tree in  $\mathcal{D}$ , there is a 1-1 correspondence between the leaves of this tree and the points of  $S$ . 2) Each internal vertex in the tree has a unique representative point, which can be selected arbitrarily from the points in any of its descendant leaves. 3) Given any two points  $u, v \in S$ , there is a tree in  $\mathcal{D}$ , so that the path formed by walking from representative to representative along the unique path in that tree between these vertices, is a  $(1 + \epsilon)$ -spanner path for  $u$  and  $v$ .

For each dumbbell tree in  $\mathcal{D}$ , we use the following representative assignment from [4]. Leaf labels are propagated up the tree. An internal vertex chooses to itself one of the propagated labels and propagates the other one up the tree. Each label is used at most twice, once at a leaf and once at an internal vertex. Any label assignment induces a weight function over the edges of the dumbbell tree in the obvious way. (The weight of an edge is set to be the Euclidean distance between the representatives corresponding to the two endpoints of that edge.) Arya et al. [4] proved that the lightness of dumbbell trees is always  $O(\log n)$ , regardless of which representative assignment is chosen for the internal vertices.

Next, we describe our construction of Euclidean spanners with diameter in the range  $\Omega(\alpha(n)) = \Lambda = O(\log n)$ . For each dumbbell tree  $DT_i \in \mathcal{D}$ , denote by  $M_i$  the  $O(n)$ -point tree metric induced by  $DT_i$ . To obtain our construction of  $(1 + \epsilon)$ -spanners with low diameter, we set  $k = n^{1/\Lambda}$ , and build the 1-spanner construction  $\mathcal{H}^i = \mathcal{H}_k^i(O(n))$  of Theorem [3] for each of the tree metrics  $M_i$ . Then we translate each  $\mathcal{H}^i$  to be a spanning subgraph  $\hat{\mathcal{H}}^i$  of  $S$  in the obvious way. Let  $\mathcal{E}_k(n)$  be the Euclidean spanner obtained from the union of all the graphs  $\hat{\mathcal{H}}^i$ .

It is easy to see that the number of edges in  $\mathcal{E}_k(n)$  is  $O(n)$ .

Next, we show that  $\Lambda(\mathcal{E}_k(n)) = O(\log_k n + \alpha(k))$ . By the Dumbbell Theorem, for any pair of points  $u, v \in S$ , there exists a dumbbell tree  $DT_i$ , so that the unique path  $P_{u,v}$  between  $u$  and  $v$  in  $DT_i$  is a  $(1 + \epsilon)$ -spanner path. Theorem [3] implies that there is a 1-spanner path  $P$  in  $\mathcal{H}^i$  between  $u$  and  $v$  that consists of  $O(\log_k n + \alpha(k))$  hops. By the triangle inequality, the weight of the corresponding translated path  $\hat{P}$  in  $\hat{\mathcal{H}}^i$  is no greater than the weight of  $P_{u,v}$ . Hence,  $\hat{P}$  is a  $(1 + \epsilon)$ -spanner path for  $u$  and  $v$  that consists of  $O(\log_k n + \alpha(k))$  hops.

We proceed by showing that  $\Delta(\mathcal{E}_k(n)) = O(k)$ . Since each dumbbell tree  $DT_i$  is binary, theorem [3] implies that  $\Delta(\mathcal{H}^i) = O(k)$ . Recall that each label is used at most twice in  $DT_i$ , and so  $\Delta(\hat{\mathcal{H}}^i) \leq 2\Delta(\mathcal{H}^i) = O(k)$ . The union of  $O(1)$  such graphs will also have maximum degree  $O(k)$ .

Finally, we argue that the lightness  $\Psi(\mathcal{E}_k(n))$  of  $\mathcal{E}_k(n)$  is  $O(k \log_k n \log n)$ . Consider a dumbbell tree  $DT_i$ . Recall that the lightness of all dumbbell trees is  $O(\log n)$ , and so  $w(DT_i) = O(\log n)w(MST(S))$ . By Theorem [3], the weight  $w(\mathcal{H}^i)$  of  $\mathcal{H}^i$  is at most  $O(k \log_k n)w(DT_i)$ . By the triangle inequality, the weight of each edge in  $\hat{\mathcal{H}}^i$  is no greater than the corresponding weight in  $\mathcal{H}^i$ , implying that  $w(\hat{\mathcal{H}}^i) \leq w(\mathcal{H}^i) = O(k \log_k n \log n)w(MST(S))$ . The union of  $O(1)$  such graphs will also have weight  $O(k \log_k n \log n)w(MST(S))$ .

To obtain our construction of Euclidean spanners in the range  $\Lambda = \Omega(\log n)$ , we use our 1-spanners for tree metrics from Theorem [4] instead of Theorem [3].

**Corollary 1.** *For any set  $S$  of  $n$  points in  $\mathbb{R}^d$ , any  $\epsilon > 0$  and a parameter  $k$ , there exists a  $(1 + \epsilon)$ -spanner with  $O(n)$  edges, maximum degree  $O(k)$ , diameter*

$O(\log_k n + \alpha(k))$  and lightness  $O(k \log_k n \log n)$ . There also exists a  $(1 + \epsilon)$ -spanner with degree  $O(1)$ , diameter  $O(k \log_k n)$  and lightness  $O(\log_k n \log n)$ .

We show that the lightness of well-separated pair constructions for random point sets in the unit cube is (w.h.p.)  $O(1)$ . Also, the lightness of well-separated pair constructions provides an asymptotic upper bound on the lightness of dumbbell trees. We derive the following result as a corollary.

**Corollary 2.** *For any set  $S$  of  $n$  points that are chosen independently and uniformly at random from the unit cube, any  $\epsilon > 0$  and a parameter  $k$ , there exists a  $(1 + \epsilon)$ -spanner with  $O(n)$  edges, maximum degree  $O(k)$ , diameter  $O(\log_k n + \alpha(k))$  and lightness (w.h.p.)  $O(k \log_k n)$ . There also exists a  $(1 + \epsilon)$ -spanner with maximum degree  $O(1)$ , diameter  $O(k \log_k n)$  and lightness (w.h.p.)  $O(\log_k n)$ .*

Chan et al. [8] showed that for any doubling metric  $(X, \delta)$  there exists a  $(1 + \epsilon)$ -spanner with constant maximum degree. On the way to this result they proved the following lemma, which we employ in conjunction with our 1-spanners for tree metrics to construct our spanners for doubling metrics.

**Lemma 1 (Lemma 3.1 in [8]).** *For any doubling metric  $(X, \delta)$ , there exists a collection  $\mathcal{T}$  of  $m = O(1)$  spanning trees for  $(X, \delta)$ ,  $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_m\}$ , that satisfies the following two properties: 1) For each index  $i \in [m]$ , the maximum degree  $\Delta(\tau_i)$  of the tree  $\tau_i$  is constant, i.e.,  $\Delta(\tau_i) = O(1)$ . 2) For each pair of points  $x, y \in X$  there exists an index  $i \in [m]$  such that  $\text{dist}_{\tau_i}(x, y) = O(1)\delta(x, y)$ .*

To obtain our spanners for doubling metrics we start with constructing the collection  $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_m\}$  of spanning trees with properties listed in Lemma 1. Next, we apply Theorem 3 with some parameter  $k$  to construct a 1-spanner  $\mathcal{Z}^i = \mathcal{Z}_k^i(n)$  for the tree metric induced by the  $i$ th tree  $\tau_i$  in  $\mathcal{T}$ , for each  $i \in [m]$ . Our spanner  $\mathcal{Z}$  is set to be the union of all the 1-spanners  $\mathcal{Z}_i$ , i.e.,  $\mathcal{Z} = \bigcup_{i=1}^m \mathcal{Z}_i$ . We summarize the properties of the resulting spanner  $\mathcal{Z}$  in the next statement.

**Corollary 3.** *For any  $n$ -point doubling metric  $(X, \delta)$  and a parameter  $k$ , there is an  $O(1)$ -spanner  $\mathcal{Z}$  with  $O(n)$  edges, degree  $O(k)$  and diameter  $O(\log_k n + \alpha(k))$ .*

**Acknowledgments.** We are grateful to Sunil Arya, David Mount and Michiel Smid for helpful discussions.

## References

- Agarwal, P.K., Wang, Y., Yin, P.: Lower bound for sparse Euclidean spanners. In: Proc. of 16th SODA, pp. 670–671 (2005)
- Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product queries. Manuscript (1987)
- Althöfer, I., Das, G., Dobkin, D.P., Joseph, D., Soares, J.: On sparse spanners of weighted graphs. Discrete & Computational Geometry 9, 81–100 (1993)
- Arya, S., Das, G., Mount, D.M., Salowe, J.S., Smid, M.H.M.: Euclidean spanners: short, thin, and lanky. In: Proc. of 27th STOC, pp. 489–498 (1995)

5. Arya, S., Smid, M.H.M.: Efficient construction of a bounded degree spanner with low weight. *Algorithmica* 17(1), 33–54 (1997)
6. Bodlaender, H.L., Tel, G., Santoro, N.: Trade-offs in non-reversing diameter. *Nord. J. Comput.* 1(1), 111–134 (1994)
7. Chan, H.T.-H., Gupta, A.: Small hop-diameter sparse spanners for doubling metrics. In: Proc. of 17th SODA, pp. 70–78 (2006)
8. Chan, H.T.-H., Gupta, A., Maggs, B.M., Zhou, S.: On hierarchical routing in doubling metrics. In: Proc. of 16th SODA, pp. 762–771 (2005)
9. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. *Algorithmica* 2, 337–361 (1987)
10. Chazelle, B., Rosenberg, B.: The complexity of computing partial sums off-line. *Int. J. Comput. Geom. Appl.* 1, 33–45 (1991)
11. Chew, L.P.: There is a planar graph almost as good as the complete graph. In: Proc. of 2nd SOCG, pp. 169–177 (1986)
12. Das, G., Narasimhan, G.: A fast algorithm for constructing sparse Euclidean spanners. In: Proc. of 10th SOCG, pp. 132–139 (1994)
13. Das, G., Narasimhan, G., Salowe, J.S.: A new way to weigh malnourished Euclidean graphs. In: Proc. of 6th SODA, pp. 215–222 (1995)
14. Dinitz, Y., Elkin, M., Solomon, S.: Shallow-low-light trees, and tight lower bounds for Euclidean spanners. In: Proc. of 49th FOCS, pp. 519–528 (2008)
15. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.* 31(5), 1479–1500 (2002)
16. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.H.M.: Approximate distance oracles for geometric graphs. In: Proc. of 13th SODA, pp. 828–837 (2002)
17. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.H.M.: Approximate distance oracles for geometric spanners. *ACM Transactions on Algorithms* 4(1) (2008)
18. Hassin, Y., Peleg, D.: Sparse communication networks and efficient routing in the plane. In: Proc. of 19th PODC, pp. 41–50 (2000)
19. Keil, J.M., Gutwin, C.A.: Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry* 7, 13–28 (1992)
20. Mansour, Y., Peleg, D.: An approximation algorithm for min-cost network design. *DIMACS Series in Discr. Math and TCS* 53, 97–106 (2000)
21. Narasimhan, G., Smid, M.: *Geometric Spanner Networks*. Cambridge University Press, Cambridge (2007)
22. Pătrașcu, M., Demaine, E.D.: Tight bounds for the partial-sums problem. In: Proc. of 15th SODA, pp. 20–29 (2004)
23. Rao, S., Smith, W.D.: Approximating geometrical graphs via “spanners” and “banyans”. In: Proc. of 30th STOC, pp. 540–550 (1998)
24. Tarjan, R.E.: Applications of path compression on balanced trees. *J. ACM* 26(4), 690–715 (1979)
25. Thorup, M.: Parallel shortcutting of rooted trees. *J. Algorithms* 23(1), 139–159 (1997)
26. Yao, A.C.: Space-time tradeoff for answering range queries. In: Proc. of 14th STOC, pp. 128–136 (1982)

# Testing Euclidean Spanners

Frank Hellweg, Melanie Schmidt, and Christian Sohler\*

Department of Computer Science, Technische Universität Dortmund

**Abstract.** We develop a *property testing algorithm* with query complexity  $\tilde{O}(\delta^{-5d}\epsilon^{-5}D\log^6\Delta\sqrt{n})$  that tests whether a directed geometric graph  $G = (P, E)$  with maximum degree  $D$  and vertex set  $P \subseteq \{1, \dots, \Delta\}^d$  (for constant  $d$ ) is a Euclidean  $(1 + \delta)$ -spanner. Such a property testing algorithm accepts every  $(1 + \delta)$ -spanner and rejects with high constant probability every graph that is  $\epsilon$ -far from this property, i. e., every graph that differs in more than  $\epsilon|P|$  edges from every  $(1 + \delta)$ -spanner.

## 1 Introduction

Property testing is the computational task of deciding whether a given object has a predetermined property  $\Pi$  or is far away from every object with property  $\Pi$ . Thus, property testing can be viewed as a relaxation of a standard decision problem. The main goal of property testing is to develop randomized algorithms that perform this relaxed decision task by only looking at a small part of the input object, i. e. we want to develop algorithms whose running time is sublinear in the object's description size. Property testing has been introduced by Rubinfeld and Sudan [34] and the study of combinatorial properties has been initiated by Goldreich, Goldwasser, and Ron [26]. Since then, property testing algorithms have been developed for properties of functions [25, 24, 10], properties of distributions [8, 7], algebraic properties [11, 34, 29], graph and hypergraph properties [26, 3, 16, 9], and *geometric properties* which we continue to study in this paper. Previous work on geometric property testing includes testing algorithms for convexity of polygons [21], convexity [33], geometric properties of point sets (for example convex position), the Euclidean minimum spanning tree [18, 17, 19], and clusterability of point sets [11, 7]. The area of sublinear time algorithms is closely related to property testing; geometric properties studied in this field include polyhedron intersection and point location in planar convex subdivisions with bounded face size [13] and approximation algorithms for the weight of the Euclidean minimum spanning tree [15] and metric minimum spanning trees [20].

*Euclidean spanners.* In this paper, we consider Euclidean spanners. A weighted directed geometric graph  $G = (P, E)$  is a directed graph whose vertex set is a set of points in the Euclidean space  $\mathbb{R}^d$  and whose edge weights (lengths) are given by the Euclidean distance of the vertices. Throughout this paper, we will assume that  $d$  is constant. A geometric graph is called a  $(\text{Euclidean}) (1 + \delta)$ -spanner, if for

---

\* Research partly supported by DFG grant SO 514/3-1.

every pair of vertices  $p$  and  $q$  the shortest path distance  $d_G(p, q)$  in  $G$  is at most  $(1+\delta) \cdot \|p-q\|_2$ . Euclidean spanners are a fundamental geometric graph structure as they can be used to approximately solve many geometric proximity problems. Many different constructions of Euclidean spanners are known. For constant  $\delta$ , Euclidian  $(1 + \delta)$ -spanners with a linear number of edges can for example be constructed by using so-called  $\Theta$ -graphs [14,28] or structures based on the well-separated pair decomposition [12,31]. Techniques to construct spanners with bounded degree are also known [5]. For more details we refer to the book [31]. We investigate the question whether a given graph is a Euclidean spanner. The related question of computing the stretch factor  $(1 + \delta)$  of a given graph has recently been studied in [4,22,30]. Additionally, Ahn et al. [2] discuss the problem to find an edge whose removal leads to the smallest possible increase in the stretch factor, and Farshi et al. [23] consider the question which edge should be added to receive the best decrease in the stretch factor (both articles consider very special cases only).

*Our contribution* We develop a property testing algorithm that distinguishes spanners from geometric graphs that are very different from being a spanner. The distance of a given graph to a spanner is measured by the amount of edges that have to be inserted to establish the spanner property, and graphs with a high distance to every spanner are called  $\epsilon$ -far (where ‘high’ is defined depending on the parameter  $\epsilon$ ). A property tester is a sublinear randomized algorithm that has to reject every  $\epsilon$ -far graph with high probability. We develop a property tester with one-sided error, i. e., every spanner is always accepted (where in the general case, this must only happen with high probability). The analysis of our algorithm assumes that the vertices of the input graph lie on a  $d$ -dimensional discrete grid  $\{1, \dots, \Delta\}^d$ . The query complexity and running time of our algorithm is  $\tilde{\mathcal{O}}(\delta^{-5d}\epsilon^{-5}D\log^6\Delta\sqrt{n})$ , so it depends logarithmically on the range of possible coordinates. It is open whether this influence or the exponential dependence on the dimension can be avoided.

## 2 Preliminaries

Let  $G = (P, E)$  be a directed geometric graph with point set  $P := \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ ,  $d$  constant, and edge set  $E \subseteq P \times P$ . We will assume that the algorithm is given the number of vertices  $n$ , but it does not know the positions of the points in  $P$ . It may query the position of the  $i$ -th vertex in  $O(1)$  time for any index  $i$ ,  $1 \leq i \leq n$ . The graph structure is stored in the adjacency list model for general (sparse) graphs proposed in [32]. In this model, it is possible to query the degree  $\deg(p_i)$  of the  $i$ -th vertex in  $O(1)$  time by specifying the index  $i$ ,  $1 \leq i \leq n$  and one can obtain the  $j$ -th edge incident to the  $i$ -th vertex in  $O(1)$  time for any  $1 \leq i \leq n$  and  $1 \leq j \leq \deg(i)$ . The maximum degree of a vertex in  $G$  is denoted by  $D$ . The query complexity of a property testing algorithm is the number of queries of coordinates, degrees and neighbors that it needs to accomplish its task.

We use  $[p, q]$  to denote a directed edge from  $p$  to  $q$  and denote the shortest path distance from  $p$  to  $q$  by  $d_G(p, q)$ , i.e.,  $d_G(p, p) = 0$ ,  $d_G(p, q) = \|p - q\|_2$  for all  $p, q \in P$  with  $[p, q] \in E$  and  $d_G(p, q) := \min_{\text{paths } Q \text{ from } p \text{ to } q} \sum_{[p', q'] \in Q} d_G(p', q')$  for all other pairs of points. A directed geometric graph is a  $(1 + \delta)$ -spanner if it provides a  $(1 + \delta)$ -approximation of the Euclidean distances between all pairs of points as stated in the following definition. Note that we restrict to  $(1 + \delta)$ -spanners with  $0 < \delta < 1$ .

**Definition 1.** Let  $0 < \delta < 1$  be given. A directed geometric graph  $G$  is called a  $(1 + \delta)$ -spanner, if  $d_G(p, q) \leq (1 + \delta)\|q - p\|_2$  for all pairs of points  $[p, q] \in P \times P$  with  $p \neq q$ .

The definition of property testers depends on the definition of  $\epsilon$ -farness. We define that a graph is  $\epsilon$ -far from being a spanner if one has to add, delete or modify more than  $\epsilon n$  edges of the graph to get a spanner. This is slightly different from the definition in [32] where  $\epsilon |E|$  is used instead of  $\epsilon n$ .

**Definition 2.** Let  $G = (P, E)$  be a directed geometric graph and let  $0 < \epsilon < 1$  and  $0 < \delta < 1$  be given.  $G$  is  $\epsilon$ -far from a directed geometric graph  $G' = (P, E')$  if  $|E \setminus E' \cup E' \setminus E| > \epsilon n$ .  $G$  is  $\epsilon$ -far from having the property to be a  $(1 + \delta)$ -spanner if it is  $\epsilon$ -far from every  $(1 + \delta)$ -spanner.

We say that an algorithm  $\mathcal{A}$  is a *property tester with one-sided error* for the property of being a  $(1 + \delta)$ -spanner if it returns

- *true* with a probability of 1 if  $G$  is a  $(1 + \delta)$ -spanner
- *false* with probability at least  $2/3$  if  $G$  is  $\epsilon$ -far from being a  $(1 + \delta)$ -spanner

when given input  $|V|$ ,  $\epsilon$  and  $\delta$  and access to a directed geometric graph  $G = (P, E)$  as described above.

### 3 The Algorithm

Our algorithm for testing geometric spanners depends on two parameters  $s$  and  $s'$  that are specified at the end of this paragraph. It has access to a directed geometric graph in the way described above. First, the algorithm samples a set of  $s$  vertices  $q_1, \dots, q_s$  uniformly at random. Then it independently performs a shortest path computation starting at each vertex  $q_i$ . Dijkstra's algorithm is used until  $s'$  vertices have been reached (in the current iteration). We denote this traversal as *Dijkstra traversal*. After all Dijkstra traversals are finished, the algorithm checks whether a certificate that the graph is not a spanner was found. Let  $W_i$  be the maximum graph distance reached during the Dijkstra traversal started at  $q_i$ . Then we know that all points within graph distance  $W_i$  were reached. Thus, if a point lying within distance  $W_i/(1 + \delta)$  of  $q_i$  was sampled but not reached, then the graph is rejected because in order to satisfy the spanner property for such a point, there has to be a path that is completely included in the searched part of the graph. Thus, the graph cannot be a spanner. If no such point is found for any  $q_i$  then the graph is accepted. We consider two versions

of our algorithm. For the analysis in Section 4 we set  $s' := \tilde{\mathcal{O}}(\delta^{-2d}\epsilon^{-1})$ , for the general case discussed in Section 5 we set  $s' := \tilde{\mathcal{O}}(\delta^{-4d}\epsilon^{-3}\log^6\Delta)$ , where the points are assumed to come from the discrete  $d$ -dimensional space  $\{1, \dots, \Delta\}^d$ . In both cases, the parameter  $s$  is set such that  $s := \tilde{\mathcal{O}}(\delta^{-d}\epsilon^{-2}\sqrt{n})$ .

The maximum degree of a vertex in the graph influences the running time and query complexity of the Dijkstra traversals, because to perform such a computation one needs to collect information about all neighbors of the already searched vertices. If the maximum degree is constant, this is, of course, possible in constant time for each vertex.

**UNIFORMTESTER( $n, G, \delta, \epsilon$ )**

Sample  $s$  points  $q_1, \dots, q_s$  from  $P$  without replacement

**for**  $i \leftarrow 1$  **to**  $s$

    Perform a Dijkstra traversal in  $G$  from  $q_i$  until  
         $s'$  nodes have been visited

    Let  $R$  be the set of vertices visited and let  $W = \max_{q \in R} d_G(q_i, q)$

**forall** sample points  $q_j$  such that  $\|q_i - q_j\|_2 < \frac{1}{1+\delta} \cdot W$

**if**  $q_j \notin R$  **or**  $d_G(q_i, q_j) \geq (1 + \delta)\|q_i - q_j\|_2$   
            **return** false

**return** true

Note that this algorithm only tests small neighborhoods of certain points. At first glance it seems unlikely that the spanner property can be tested by local investigations. Consider a path whose vertices are placed on the boundary of an ellipsoid. Then  $\delta$  can be chosen in a way such that the spanner property is satisfied for all pairs of points except the one with highest distance. This means that the violation cannot be found by sampling only parts of the path. Surprisingly, when distinguishing spanners and graphs that are  $\epsilon$ -far from being a spanner, the situation is different: A graph that locally satisfies the spanner property can be transformed into a spanner by adding only a few edges. We show that a geometric graph cannot be  $\epsilon$ -far from being a spanner if it does only contain ‘global’ violations of the spanner property.

## 4 Testing Graphs with Uniformly Spread Vertices

In this section we show that for input graphs of a certain kind the algorithm UNIFORMTESTER is a property tester with one-sided error, i. e., it accepts every spanner with probability one and rejects every graph far from being a spanner with high probability. The first property holds as the algorithm only rejects a graph if it has found a pair of points that violates the spanner property, so it remains to show the high rejection probability. The special case that we assume is that the points of  $G$  are uniformly spread over the plane in the following sense.

**Definition 3.** Let  $G$  be a directed geometric graph with  $n$  points and let  $n_u$  be a value depending on  $n$  (and  $1/\delta$  and  $1/\epsilon$ ). We say that  $G$  is uniformly  $n_u$ -distributed on grid  $H$  if there exists a  $d$ -dimensional grid  $H$  with a width of  $S = \sqrt[d]{n/n_u}$  cells in each dimension such that every cell of  $H$  contains  $\mathcal{O}(n_u)$  points of  $G$  and such that  $H$  completely contains  $G$ . We denote the cells of  $H$  as base cells and their side length by  $w_0$ .

Now let  $G$  be a uniformly  $n_u$ -distributed graph on grid  $H$ . We say that a pair of points  $(p, q)$  is a *violating pair* if  $d_G(p, q) > (1 + \delta)\|p - q\|_2$ . If  $G$  does not contain a violating pair, then it is a spanner, and if  $G$  is  $\epsilon$ -far from being a spanner, there have to be at least  $\epsilon n$  violating pairs (otherwise adding less than  $\epsilon n$  edges would establish the spanner property). For our algorithm, we need that (some of) these violating pairs can be found by exploring small neighborhoods of certain points. For this purpose we show that if the spanner property holds for all ‘close’ pairs of points in  $G$ , then it can be established for distant pairs of points by inserting no more than  $\epsilon n/2$  edges into  $G$ . This means that if  $G$  is  $\epsilon$ -far from being a spanner, there have to be at least  $\epsilon n/2$  ‘close’ pairs of points violating the spanner property. To realize this, we use exponential grids that are constructed in a similar manner as in [27] where they are used to compute coresets for clustering problems. The basic idea behind this is to connect clouds of points with a small number of edges which is somewhat similar to the use of well-separated pair decompositions for spanner constructions [12, 31]. However, our focus lies on connecting ‘distant’ points with few edges and not on computing a complete spanner.

For our exponential grid it is convenient to triple the side length of the boxes, so we wish to deal with powers of three and for simplicity assume that the side length  $S$  is a power of three.

**Definition 4.** Let  $G$  be a uniformly  $n_u$ -distributed graph on grid  $H$  and let  $c$  be a cell of  $H$ . We set  $i_\delta := \min\{i \mid 3^i \geq (8\sqrt{d}/\delta), i \in \mathbb{N}\}$  and define  $B_{c,i}$  for  $i \geq 0$  as the cube that is centered at the center of  $c$  and has side length  $3^{i+i_\delta} \cdot w_0$ . Let  $p \in P$ . We denote the cell of  $H$  containing  $p$  by  $c(p)$ . Then we define the geometric neighborhood  $\Gamma(p)$  of  $p$  as set of all points that are contained in the area inside  $B_{c(p),0}$ . A point  $q \in P$  is a neighbor of  $p$  if  $q \in \Gamma(p)$ . Additionally we define the extended geometric neighborhood  $\Gamma^e(p)$  of  $p$  as the set of all points that are contained in the area inside  $B_{c(p),1}$  (note that  $\Gamma(p) \subseteq \Gamma^e(p)$ ).

To construct the exponential grid  $H_c$  around  $c$ , we define  $w_i = 3^{i-1} \cdot w_0$ , cover  $B_{c,i}$  for  $i \geq 1$  with  $[(3^{i+i_\delta} \cdot w_0)/3^{i-1} \cdot w_0]^d = (3^{i_\delta+1})^d$  cubes of side length  $w_i$  and include a cube into  $H_c$  if it is not contained in a smaller box  $B_{c,j}, j < i$  but has a non-empty intersection with  $H$  (i. e., we cut off cubes that lie aside of  $G$ ). Note that  $H_c$  does not contain cells lying within  $B_{c,0}$ .

Observe that the number of cells of  $H$  that lie in  $\Gamma(p)$  is at most  $(3^{i_\delta})^d < (3 \cdot 8\sqrt{d}/\delta)^d = O(1/\delta^d)$  for all  $c$  in  $H$  (it may be less if  $c$  lies close to the margin of  $H$ ). Likewise, the number of cells of  $H$  that lie in  $\Gamma^e(p)$  is  $\mathcal{O}(1/\delta^d)$ . For the number of cubes in all exponential grids we state the following. Due to space

limitations, the proof of this statement and of the other statements in this section are omitted.

**Observation 1.** *The value  $n_u$  can be chosen such that  $n_u = \mathcal{O}(\delta^{-d}\epsilon^{-1} \log \delta n)$  and such that for every  $n_u$ -distributed graph on grid  $H$  the total number of cubes in all exponential grids  $H_c$  for all  $c \in H$  is less than  $\epsilon n/2$ .*

Now we use the structure defined by  $H$  to insert  $\epsilon n/2$  edges into  $G$  that ensure that any two points  $p, q \in P$  with  $q \notin \Gamma(p)$  are connected by a path of length  $d_{G'}(p, q) \leq (1 + \delta)\|p - q\|_2$  if this is already true for all pairs consisting of a point and one of its neighbors. This works because the exponential grid grows in such a manner that an arbitrary connection between a cell  $c$  and a cell in the exponential grid  $H_c$  always suffices to establish the spanner property for all points in these cells, if applied recursively.

**Lemma 1.** *Let  $G$  be a uniformly  $n_u$ -distributed graph on grid  $H$  which satisfies  $d_G(p, q) \leq (1 + \delta)\|p - q\|_2$  for all pairs consisting of a point  $p \in P$  and a neighbor  $q \in \Gamma(p)$  of  $p$ . Then  $G$  is  $\epsilon n/2$ -close to being a spanner.*

Lemma 1 implies that our algorithm only has to test whether there are at least  $\epsilon n/2$  violating pairs consisting of a point  $p \in P$  and a neighbor  $q \in \Gamma(p)$  of  $p$ . The next Lemma gives an upper bound on the number of points that have to be explored for a given  $p$  to find all neighbors  $q'$  of  $p$  such that  $p$  and  $q'$  do *not* form a violating pair, i. e., for all neighbors  $q$  of  $p$  that are not found during such a search,  $p$  and  $q$  form a violating pair. This is basically due to the definition of the neighborhoods.

**Lemma 2.** *Let  $G$  be a uniformly  $n_u$ -distributed graph on grid  $H$  and let  $p \in P$  be a point. It suffices to explore  $\mathcal{O}(\delta^{-d}n_u)$  points of  $G$  to find all neighbors  $q \in \Gamma(p)$  with  $d_G(p, q) \leq (1 + \delta)\|p - q\|_2$ .*

Lemma 2 states how many points in the neighborhood have to be explored for each sampled point to test whether a point belongs to a sampled violating pair. Finally we have to ensure that the algorithm samples enough points to get both points of at least one violating pair with high probability.

**Lemma 3.** *Let  $G = (P, E)$  be a directed geometric graph. Assume that it holds that every point  $p \in P$  has at most  $\rho$  neighbors and that every point  $q \in P$  is neighbor of at most  $\rho$  points. Let there be  $\epsilon n/2$  pairs of a point  $p$  and a neighbor  $q \in \Gamma(p)$  such that  $p$  and  $q$  form a violating pair. Then there is an  $s = \mathcal{O}(\rho \cdot \sqrt{n} \cdot \epsilon^{-1})$  such that the probability that  $s$  points sampled uniformly at random contain both points of one of these violating pairs is at least  $5/6$ .*

The proof of Lemma 3 is a standard analysis similar to the birthday problem.

Now we combine the statements to get the main result of this section. Inserting the value of  $n_u$  given in Observation 1 into Lemma 2 shows that exploring  $\mathcal{O}(\delta^{-d} \cdot \frac{1}{\epsilon} \cdot \log(\delta n) \cdot \delta^{-d}) = \tilde{\mathcal{O}}(\delta^{-2d}\epsilon^{-1})$  points suffices to ensure that the algorithm only rejects if it has indeed found a violating pair. By Lemma 3 we gain that sampling  $\mathcal{O}(\delta^{-d}\epsilon^{-2} \log \delta n \sqrt{n}) = \tilde{\mathcal{O}}(\delta^{-d}\epsilon^{-2}\sqrt{n})$  points is sufficient to find

both points of a violating pair with high probability. Combining both facts we get that algorithm **UNIFORMTESTER** is a property tester as defined in Section 2 and that it has a query complexity of  $\tilde{\mathcal{O}}(\delta^{-3d}\epsilon^{-3}D\sqrt{n})$ . When implemented using adequate data structures, the query complexity and running time of the algorithm only differs by logarithmic factors.

**Theorem 1.** *The algorithm **UNIFORMTESTER** is a property tester for the property of being a  $(1 + \delta)$ -spanner under the assumption that the input graphs are uniformly  $n_u$ -distributed on grid  $H$  with  $n_u = \mathcal{O}(\delta^{-d}\epsilon^{-1}\log\delta n)$  chosen as in Observation 1. It has a query complexity and running time of  $\tilde{\mathcal{O}}(\delta^{-3d}\epsilon^{-3}D\sqrt{n})$ .*

## 5 Testing Graphs with Vertices Placed on a Discrete Grid

From now on, we assume that the vertices of  $G$  are coming from the discrete  $d$ -dimensional space  $\{1, \dots, \Delta\}^d$ . We show that under this assumption after adapting the number of vertices to explore for each sampled point the above algorithm is a property tester for the property of  $G$  being a  $(1 + \delta)$ -spanner with a query complexity of  $\tilde{\mathcal{O}}(\delta^{-5d}\epsilon^{-5}D\log^6\Delta\sqrt{n})$ . As above, we state that the algorithm only rejects the input if it finds a witness not satisfying the  $(1 + \delta)$ -spanner property. This ensures that an input graph cannot be rejected if it is a  $(1 + \delta)$ -spanner. Thus, it remains to show that for all input graphs that are  $\epsilon$ -far from being  $(1 + \delta)$ -spanners we find such a witness with high probability. The proof idea is based on the previous proof, but in this case instead of a grid,  $H$  will be defined as a ( $d$ -dimensional) quadtree partitioning, satisfying that no leaf cell of the quadtree exceeds a certain number of vertices; i. e., we build the quadtree by starting with a bounding box including all points and partitioning each box into  $2^d$  subboxes, if it contains more than  $s$  points for some  $s \in \mathcal{O}(\text{poly}(\log\Delta, \frac{1}{\epsilon}, \frac{1}{\delta}))$  recursively.

Similar to the uniform case, we construct an exponential grid around every leaf box  $c$  of  $H$  and insert an edge from  $c$  to every cell of its exponential grid. By bounding the depth of  $H$ , we can bound the number of leaf boxes in  $H$  and thus show that the number of edges we insert in this way does not exceed  $\epsilon n/4$ . The difference to the uniform case is that the neighborhood of a leaf box  $c$  may contain many smaller leaf boxes and thus arbitrarily many points of  $P$ . A neighborhood whose number of points exceeds a certain amount cannot be completely explored by our algorithm; this means, that for the proof we have to ensure that one can fix any violation of the spanner property occurring in such neighbourhoods by inserting at most  $\epsilon n/4$  edges in total. We show that there are only few such neighborhoods, which implies that we can establish the  $(1 + \delta)$ -spanner property for these cells by inserting few edges.

We define the exponential grid around a leaf box of  $H$  similarly to Section 4:

**Definition 5.** *Let  $G = (P, E)$  be a directed geometric graph whose points come from the  $d$ -dimensional grid  $T = \{1, \dots, \Delta\}^d$  and let  $H$  be a  $d$ -dimensional quadtree of the points of  $G$ , satisfying that no leaf box of  $H$  contains more than*

$s = \frac{(3^{i_\delta})^d \cdot 2^{d+3} \cdot \log^2 2\Delta}{\epsilon \log 2} = \mathcal{O}\left(\frac{\log^2 \Delta}{\delta^d \epsilon}\right)$  points of  $G$ . For  $p \in P$ , we denote the leaf box of  $H$  that contains  $p$  by  $c(p)$ . Let  $c$  be a leaf box of  $H$ . We define  $w(c)$  as the side length of  $c$  and  $B_{c,i}$  for  $i \geq 0$  as the cube that is centered at the center of  $c$  and has side length  $3^{i+i_\delta} \cdot w(c)$ . We call the points that are contained in the area inside  $B_{c,0}$  its geometric neighborhood  $\Gamma(c) \subseteq P$  and for two points  $p, q \in P$  we call  $p$  geometric neighbor of  $q$  if  $p$  lies in the geometric neighborhood of  $q$ .

Define  $w_i(c) = 3^{i-1} \cdot w(c)$ ,  $i > 0$ . To construct the exponential grid  $H_c$  around  $c$ , cover  $B_{c,i}$  for  $i \geq 1$  with  $[(3^{i+i_\delta} \cdot w(c))/3^i \cdot w(c)]^d = (3^{i_\delta})^d$  cubes of side length  $w_i(c)$  and include a cube into  $H_c$  if it is not contained in a smaller box  $B_{c,j}$ ,  $j < i$  but it is contained in  $H$ .

Note that since  $\delta < 1$ , the extended neighborhood  $\hat{\Gamma}(c) := B_{c,1} \cap P$  of  $c$  contains all points which can be part of a path from a point  $p \in c$  to a point  $q \in \Gamma(c)$  having a length of at most  $(1 + \delta)\|p - q\|_2$ .

**Definition 6.** We call a leaf box  $c \in H$  samplable if  $\hat{\Gamma}(c)$  contains at most  $k := \frac{2^{d+3} \cdot 3^{d(2i_\delta+1)} s \log^2 \Delta}{\epsilon} = \mathcal{O}\left(\frac{\log^4 \Delta}{\delta^3 d \epsilon^2}\right)$  cells  $c_1, \dots, c_k$  with  $w(c_i) < w(c)$ .

We start the analysis by bounding the total number of cells in the exponential grids around the leaf boxes by  $\frac{\epsilon n}{4}$ . The depth of the quadtree defining  $H$  is bounded by  $\log \Delta$ , since the grid  $T$  has a side length of  $\Delta$  and the smallest possible box has one of  $\sqrt[d]{s}$ . Since a box must contain at least  $s + 1$  vertices to be split into subboxes, the overall number of leaf boxes is at most  $\frac{2^d n \log \Delta}{s}$ . This leads to the following Lemma:

**Lemma 4.** The following statements hold:

- The number of cells in all exponential grids around leaf boxes of  $H$  does not exceed  $\epsilon n / 8$ .
- There are at most  $\frac{\epsilon n}{8 \cdot 3^{di_\delta} s^2}$  cells that are not samplable.

*Proof.* The first statement follows by bounding the maximum number of cells in each of the exponential grids and multiplying with the above bound on the number of quadtree leaf boxes. For the second statement consider an arbitrary leaf box  $c$ . What is the number of larger leaf boxes  $c'$  such that  $c$  is contained in  $\hat{\Gamma}(c')$ ? For each possible size of  $c'$  – there are at most  $\log \Delta$  possible sizes – due to the number of equal-sized boxes contained in an extended neighbourhood there are at most  $(3^{i_\delta+1})^d \log \Delta$  such leaf boxes. Since there are at most  $\frac{2^d n \log \Delta}{s}$  leaf boxes, the total number of *is-contained*-relations is at most  $\frac{2^d \cdot 3^{d(i_\delta+1)} n \log^2 \Delta}{s}$ . Thus, there are at most

$$\frac{2^d \cdot 3^{d(i_\delta+1)} n \log^2 \Delta}{sk} = \frac{\epsilon n}{8 \cdot 3^{di_\delta} s^2}$$

cells that are not samplable.  $\square$

**Lemma 5.** Let  $G$  be  $\epsilon$ -far from being a  $(1 + \delta)$ -spanner. Then there are at least  $\epsilon n / 2$  violating pairs of points in the geometric neighborhoods of samplable leaf boxes of  $H$ .

*Proof.* Assume that there are less than  $\epsilon n/2$  violating points in the geometric neighborhoods of samplable leaf boxes of  $H$ . Then we can insert edges into  $G$  as follows:

- Insert edges from every leaf box  $c \in H$  to all cells of its exponential grid and vice versa; let  $E_1$  be the set of these edges. Due to Lemma 4 it holds that  $|E_1| < \epsilon n/4$ .
- For every leaf box  $c \in H$  that is not samplable, insert edges  $[p, q]$  and  $[q, p]$  for all  $p \in c, q \in \Gamma(c) \setminus (\{p\} \cup X_c)$ , where  $X_c = \{\hat{c} \in H : \hat{c} \cap \Gamma(c) \neq \emptyset \wedge w(\hat{c}) < w(c)\}$ , i.e., excluding those points of  $\Gamma(c)$  that lie in leaf boxes of  $H$  that are smaller than  $c$ . Let  $E_2$  be the set of these edges. Due to Lemma 4 there are at most  $\frac{\epsilon n}{8 \cdot 3^{dis} s^2}$  leaf boxes that we treat in this way. Since we insert at most  $2 \cdot 3^{dis} s^2$  edges for each of them, we insert in total at most  $\epsilon n/4$  edges.
- Insert edges between all violating pairs of nodes in geometric neighborhoods of leaf boxes  $c \in H$  that are samplable; let  $E_3$  be the set of these edges. Due to our assumption,  $|E_3| \leq \epsilon n/2$ .

Let  $G' = (P, E \cup E_1 \cup E_2 \cup E_3)$  be the resulting graph. We have inserted at most  $\epsilon n$  edges into  $G$  to obtain  $G'$ .

*Claim.*  $G'$  is a  $(1 + \delta)$ -spanner.

*Proof.* Let  $p, q \in P$  be arbitrary points. We show  $d_{G'}(p, q) \leq (1 + \delta) \|p - q\|_2$  by induction over  $\pi = \|p - q\|_2$  (note that since all points of  $G'$  lie on a  $\Delta^d$ -grid, the number of possible distances is finite). At first let  $\|p - q\|_2 = m := \min_{p', q' \in P} \|p' - q'\|_2$ . Since this is the smallest possible distance, it holds  $p \in \Gamma(c(q))$  and  $q \in \Gamma(c(p))$ . We consider the following cases:

1.  $c(p)$  is samplable. Since the insertion of  $E_3$  ensures that there are no violating pairs of points inside  $\Gamma(c(p))$ , we have  $d_{G'}(p, q) \leq (1 + \delta) \|p - q\|_2$ .
2.  $c(p)$  is not samplable and  $w(c(p)) \leq w(c(q))$ . Since  $[p, q] \in E_2$ , it holds  $d_{G'}(p, q) \leq (1 + \delta) \|p - q\|_2$ .
3.  $c(p)$  is not samplable and  $w(c(p)) > w(c(q))$ . We consider two subcases:
  - (a)  $c(q)$  is samplable. Thus, the insertion of  $E_3$  ensures that there are no violating pairs of points inside  $\Gamma(c(p))$ , ensuring that  $d_{G'}(p, q) \leq (1 + \delta) \|p - q\|_2$ .
  - (b)  $c(q)$  is not samplable. In this case, it holds  $[p, q] \in E_2$  and therefore  $d_{G'}(p, q) \leq (1 + \delta) \|p - q\|_2$ .

For the inductive step let  $\|p - q\|_2 = \pi$ ; the induction hypothesis is that  $d_{G'}(p', q') \leq (1 + \delta) \|p' - q'\|_2$  holds for all  $p', q' \in P$  such that  $\|p' - q'\|_2 < \pi$ . We consider the following cases:

1.  $p \in \Gamma(c(q))$  and  $q \in \Gamma(c(p))$ . This case follows analogously to the case  $\|p - q\|_2 = m$ .
2.  $p \notin \Gamma(c(q))$ , but  $q \in \Gamma(c(p))$ .
  - (a)  $c(p)$  is samplable. In this case the insertion of  $E_3$  ensures that  $d_{G'}(p, q) \leq (1 + \delta) \|p - q\|_2$ .

- (b)  $c(p)$  is not samplable. If  $w(c(p)) \leq w(c(q))$ , then  $[p, q] \in E_2$ . If  $w(c(p)) > w(c(q))$ , then  $p$  lies in a cell  $c \in H_{c(q)}$  since it is not contained in  $\Gamma(c(q))$ . Therefore  $E_1$  contains an edge  $[r_1, r_2]$ , where  $r_1 \in c(q)$  and  $r_2 \in c$ . Due to the construction of  $H_c(q)$ , it holds  $\|r_2 - p\|_2 < \|p - q\|_2 = \pi$  and  $\|r_1 - q\|_2 < \|p - q\|_2 = \pi$ . Thus, by applying the induction hypothesis we get  $d_{G'}(r_2, p) \leq (1 + \delta)\|r_2 - p\|_2$  and  $d_{G'}(r_1, q) \leq (1 + \delta)\|r_1 - q\|_2$ . Then  $d_{G'}(p, q) \leq (1 + \delta)\|p - q\|_2$  can be proved as in Lemma II.
3.  $q \notin \Gamma(c(p))$ . This case is analogous to the case that  $w(c(p)) < w(c(q))$  in 2(b) except that we consider  $H_{c(p)}$  instead of  $H_{c(q)}$ .  $\square$

We finish the proof of Lemma 5 by stating that Claim 5 is a contradiction to the assumption that  $G$  is  $\epsilon$ -far from being a  $(1 + \delta)$ -spanner.  $\square$

Now reconsider our Algorithm. The largest number of points that can be contained in the extended neighborhood of some samplable leaf box  $c \in H$  is  $(k + (3^{is+1})^d)s = \tilde{\mathcal{O}}(\delta^{-4d}\epsilon^{-3}\log^6\Delta)$ . By setting the number of vertices to explore by a Dijkstra traversal to this number, we can ensure that the extended geometric neighborhood of any sample point  $p_i$  is completely explored, if  $c(p_i)$  is samplable. Since there are at least  $\epsilon n/2$  violating pairs of points in such neighborhoods, we can apply Lemma 3 and state the following theorem.

**Theorem 2.** *Let  $G = (P, E)$  be a geometric graph with  $P \subseteq \{1, \dots, \Delta\}^d$ . Then there is a property testing algorithm with query complexity and running time  $\tilde{\mathcal{O}}(\delta^{-5d}\epsilon^{-5}D\log^6\Delta\sqrt{n})$  that accepts  $G$ , if  $G$  is a  $(1 + \delta)$ -spanner and rejects  $G$  with probability at least  $2/3$ , if  $G$  is  $\epsilon$ -far from being a  $(1 + \delta)$ -spanner.*

## 6 A Lower Bound for Testing Spanners

We give a sketch of the proof of a lower bound of  $\Omega(n^{\frac{1}{3}})$  for the number of queries of property testing algorithms with one-sided error. Our main idea is that it is not possible to distinguish between a line of  $2k$  subsequent points and two coinciding lines of  $k$  subsequent points with only one-sided error and  $o(n^{\frac{1}{3}})$  queries. By permuting the vertices, one can define two classes of graphs such that a tester with one-sided error cannot decide to which of the classes a randomly chosen graph belongs. But all graphs of the first type are 1-spanners, and all graphs of the second type are not spanners (regardless of the stretch factor) as the two lines are not connected at all. They are also  $\epsilon$ -far from being a spanner as there are  $k$  pairs of points with zero distance that have to be connected individually. The following Theorem states the result. Its proof is similar to the proof of Theorem 4.2 in [6].

**Theorem 3.** *Any property tester for the property of being a  $(1 + \delta)$ -spanner with one-sided error has a query complexity of  $\Omega(n^{\frac{1}{3}})$ .*

Note that the construction can be modified such that there are no coinciding points by randomly moving the positions of the points to the left or to the right for the first type of graphs and moving one point of each pair of coinciding points to the left and the other to the right for the second type of graphs.

## Acknowledgements

We thank Petra Berenbrink, Oded Goldreich and Ilan Newman for helpful discussions, anonymous referees for helpful comments and Mohammad Ali Abam for pointing out reference [30].

## References

1. Alon, N., Dar, S., Parnas, M., Ron, D.: Testing of Clustering. *SIAM Journal on Discrete Mathematics* 16(3), 393–417 (2003)
2. Ahn, H.-K., Farshi, M., Knauer, C., Smid, M., Wang, Y.: Dilation-Optimal Edge Deletion in Polygonal Cycles. In: *Algorithms and Computation*, pp. 88–99. Springer, Heidelberg (2007)
3. Alon, N., Fischer, E., Newman, I., Shapira, A.: A combinatorial characterization of the testable graph properties: it's all about regularity. *SIAM Journal on Computing* 39(1), 143–167 (2009)
4. Agarwal, P.K., Klein, R., Knauer, C., Langerman, S., Morin, P., Sharir, M., Soss, M.: Computing the Detour and Spanning Ratio of Paths, Trees, and Cycles in 2D and 3D. *Discret. & Computational Geometry* 39(1-3), 17–37 (2007)
5. Arya, S., Das, G., Mount, M., Salowe, J.S., Smid, M.: Euclidean spanners: short, thin, and lanky. In: *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 489–498 (1995)
6. Ben-Zwi, O., Lachish, O., Newman, I.: Lower bounds for testing Euclidean Minimum Spanning Trees. *Information Processing Letters* 102(6), 219–225 (2007)
7. Batu, T., Fortnow, L., Fischer, E., Kumar, R., Rubinfeld, R., White, P.: Testing Random Variables for Independence and Identity. In: *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 442–451 (2001)
8. Batu, T., Fortnow, L., Rubinfeld, R., Smith, W., White, P.: Testing that distributions are close. In: *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 259–269 (2000)
9. Benjamini, I., Schramm, O., Shapira, A.: Every minor-closed property of sparse graphs is testable. In: *Proceedings of the 40th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 393–402 (2008)
10. Blais, E.: Testing juntas nearly optimally. In: *Proceedings of the 41st Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 151–158 (2009)
11. Blum, M., Luby, M., Rubinfeld, R.: Self-Testing/Correcting with Applications to Numerical Problems. In: *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 73–83 (1990)
12. Callahan, P.B., Kosaraju, S.R.: Faster algorithms for some geometric graph problems in higher dimensions. In: *Proceedings of the 4th Annual ACM-SIAM Symposium on Discr. Algorithms (SODA)*, pp. 291–300 (1993)
13. Chazelle, B., Liu, D., Magen, A.: Sublinear Geometric Algorithms. *SIAM Journal on Computing* 35(3), 627–646 (2006)
14. Clarkson, K.L.: Approximating algorithms for shortest path motion planning. In: *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 56–65 (1987)
15. Čzumaj, A., Ergün, F., Fortnow, L., Magen, A., Newman, I., Rubinfeld, R., Šohler, C.: Approximating the Weight of the Minimum Spanning Tree in Sublinear Time. *SIAM Journal on Comp.* 35(1), 91–109 (2005)

16. Czumaj, A., Shapira, A., Sohler, C.: Testing hereditary properties of nonexpanding bounded-degree graphs. *SIAM Journal on Computing* 38(6), 2499–2510 (2009)
17. Czumaj, A., Sohler, C.: Property Testing with Geometric Queries. In: Meyer auf der Heide, F. (ed.) *ESA 2001. LNCS*, vol. 2161, pp. 266–277. Springer, Heidelberg (2001)
18. Czumaj, A., Sohler, C., Ziegler, M.: Property Testing in Computational Geometry. In: Paterson, M. (ed.) *ESA 2000. LNCS*, vol. 1879, pp. 155–166. Springer, Heidelberg (2000)
19. Czumaj, A., Sohler, C.: Testing Euclidean minimum spanning trees in the plane. *ACM Transactions on Alg.* 4(3) (2008)
20. Czumaj, A., Sohler, C.: Estimating the Weight of Metric Minimum Spanning Trees in Sublinear Time. *SIAM Journal on Computing* 39(3), 904–922 (2009)
21. Ergun, F., Kannan, S., Kumar, R., Rubinfeld, R., Viswanathan, M.: Spot-Checkers. *J. of Computer and System Sciences* 60(3), 717–751 (2000)
22. Eppstein, D., Wortman, K.A.: Minimum dilation stars. *Computational Geometry: Theory and Applications* 37(1), 27–37 (2007)
23. Farshi, M., Giannopoulos, P., Gudmundsson, J.: Finding the best shortcut in a geometric network. In: *Proceedings of the 21th Annunal ACM Symposium on Computational Geometry*, pp. 327–335 (2005)
24. Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., Samorodnitsky, A.: Monotonicity testing over general poset domains. In: *Proceedings of the 34th Annunal ACM Symposium on the Theory of Computing (STOC)*, pp. 474–483 (2002)
25. Goldreich, O., Goldwasser, S., Lehman, E., Ron, D., Samorodnitsky, A.: Testing Monotonicity. *Combinatorica* 20(3), 301–337 (2000)
26. Goldreich, O., Goldwasser, S., Ron, D.: Property Testing and its Connection to Learning and Approximation. *J. of the ACM* 45(4), 653–750 (1998)
27. Har-Peled, S., Mazumdar, S.: On Coresets for k-Means and k-Median Clustering. In: *Proceedings of the 36th Annunal ACM Symposium on the Theory of Computing (STOC)*, pp. 291–300 (2004)
28. Keil, M.: Approximating the complete Euclidean graph. In: Karlsson, R., Lingas, A. (eds.) *SWAT 1988. LNCS*, vol. 318, pp. 208–213. Springer, Heidelberg (1988)
29. Kaufman, T., Sudan, M.: Algebraic property testing: the role of invariance. In: *Proceedings of the 40th Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 403–412 (2008)
30. Narasimhan, G., Smid, M.: Approximating the Stretch Factor of Euclidean Graphs. *SIAM Journal on Computing*, 978–989 (2000)
31. Narasimhan, G., Smid, M.: *Geometric Spanner Networks*. Cambridge University Press, Cambridge (2007)
32. Ron, D., Parnas, M.: Testing the Diameter of Graphs. *Random Structures & Algorithms* 20(2), 165–183 (2002)
33. Rademacher, L., Vempala, S.: Testing Geometric Convexity. In: *Proceedings of the 24th Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pp. 469–480 (2004)
34. Rubinfeld, R., Sudan, M.: Robust Characterizations of Polynomials with Applications to Program Testing. *SIAM Journal on Computing* 25(2), 252–271 (1996)

# Fast Approximation in Subspaces by Doubling Metric Decomposition<sup>\*</sup>

Marek Cygan<sup>1</sup>, Lukasz Kowalik<sup>1</sup>, Marcin Mucha<sup>1</sup>,  
Marcin Pilipczuk<sup>1</sup>, and Piotr Sankowski<sup>1,2</sup>

<sup>1</sup> Institute of Informatics, University of Warsaw, Poland

<sup>2</sup> Dipartimento di Informatica e Sistemistica, Sapienza - University of Rome, Italy  
`{cygan,kowalik,mucha,malcin,sank}@mimuw.edu.pl`

**Abstract.** In this paper we propose and study a new complexity model for approximation algorithms. The main motivation are practical problems over large data sets that need to be solved many times for different scenarios, e.g., many multicast trees that need to be constructed for different groups of users. In our model we allow a preprocessing phase, when some information of the input graph  $G = (V, E)$  is stored in a limited size data structure. Next, the data structure enables processing queries of the form “solve problem A for an input  $S \subseteq V$ ”. We consider problems like STEINER FOREST, FACILITY LOCATION,  $k$ -MEDIAN,  $k$ -CENTER and TSP in the case when the graph induces a doubling metric. Our main results are data structures of near-linear size that are able to answer queries in time close to linear in  $|S|$ . This improves over typical worst case reuniting time of approximation algorithms in the classical setting which is  $\Omega(|E|)$  independently of the query size. In most cases, our approximation guarantees are arbitrarily close to those in the classical setting. Additionally, we present the first fully dynamic algorithm for the Steiner tree problem.

## 1 Introduction

*Motivation.* The complexity and size of the existing communication networks has grown extremely in the recent times. It is now hard to imagine that a group of users willing to communicate sets up a minimum cost communication network or a multicast tree according to an approximate solution to STEINER TREE problem. Instead we are forced to use heuristics that are computationally more efficient but may deliver suboptimal results [IG13]. It is easy to imagine other problems that in principle can be solved with constant approximation factors using state of art algorithms, but due to immense size of the data it is impossible in timely manner. However, in many applications the network is fixed and we need to solve the problem many times for different groups of users.

Here, we propose a completely new approach that exploits this fact to overcome the obstacles stemming from huge data sizes. It is able to efficiently deliver

---

<sup>\*</sup> This work was partially supported by the Polish Ministry of Science grant N206 355636.

results that have good approximation guarantee thanks to the following two assumptions. We assume that the network can be preprocessed beforehand and that the group of users that communicates is substantially smaller than the size of the network. The preprocessing step is independent of the group of users and hence afterwards we can, for example, efficiently compute a Steiner tree for any set of users.

More formally, in the STEINER TREE problem the algorithm is given a weighted graph  $G = (V, E)$  on  $n$  vertices and is allowed some preprocessing. The results of the preprocessing step need to be stored in limited memory. Afterwards, the set  $S \subseteq V$  of terminals is defined and the algorithm should generate as fast as possible a Steiner tree for  $S$ , i.e., a tree in  $G$  of low weight which contains all vertices in  $S$ . Given the query set  $S$  of  $k$  vertices we should compute the Steiner tree  $T$  in time depending only (or, mostly) on  $k$ .

The trivial approach to this problem is to compute the metric closure  $G^*$  of  $G$  and then answer each query by solving the STEINER TREE problem on  $G^*[S]$ . This approach delivers results with constant approximation ratio, but requires  $O(n^2)$  space of the data structure and  $\tilde{O}(k^2)$  query time. Hence it is far from being practical. In this work we aim at solutions that substantially improve both of these bounds; more formally the data structure space should be close to  $O(n)$ , while the query time should be close to  $O(k)$ . Since in a typical situation probably  $k = O(\log n)$ , so even a  $O(k \log n)$  query time is not considered fast enough, as then  $k \log n = \theta(k^2)$ . Note that the  $O(n)$  bound on the structure size is very restrictive: in a way, this bound is sublinear in the sense that we are allowed neither to store the whole distance matrix, nor (if  $G$  is dense) all the edges of  $G$ . This models a situation when during the preprocessing one can use vast resources (e.g., a huge cluster of servers), but the resources are not granted forever and when the system processes the queries the available space is much smaller.

*New Model.* In our model, computations are divided into two stages: the preprocessing stage and the query stage. In the preprocessing stage, the input is a weighted graph  $G = (V, E)$  and we should compute our data structure in polynomial time and space. Apart from the graph  $G$  some additional, problem-specific information may be also provided. In the query stage the algorithm is given the data structure computed in the preprocessing stage, but not  $G$  itself, and a set  $S$  of points of  $V$  (the query — possibly a set of pairs of points from  $V$ , or a weighted set of points from  $V$ , etc.) and computes a solution for the set  $S$ . The definition of “the solution for the set  $S$ ” depends on the specific problem. In this work we consider so-called metric problems, so  $G$  corresponds to a metric space  $(V, d)$  where  $d$  can be represented as the full distance matrix  $M$ . One should keep in mind that the function  $d$  cannot be quickly computed (e.g. in constant time) without the  $\Omega(n^2)$  size matrix  $M$ . In particular, we assume that there is no distance oracle available in the query stage.

Hence, there are three key parameters of an algorithm within our model: the size of the data structure, the query time and the approximation ratio. Less important, but not irrelevant is the preprocessing time. Let us note that though

our model is inspired by large datasets, in this work we ignore streaming effects, external memory issues etc.

Above we have formulated the STEINER TREE problem in our model, now we describe the remaining problems. In STEINER FOREST problem the algorithm is allowed to preprocess a weighted graph  $G = (V, E)$ , whereas the query is composed of the set  $S \subseteq V \times V$  of pairs. The algorithm should generate the Steiner forest for  $S$ , i.e., a subgraph  $H$  of  $G$  of small weight such that each pair in  $S$  is connected in  $H$ . In FACILITY LOCATION problem the algorithm is given in the preprocessing phase a weighted graph with facility opening costs in the nodes. We consider two variants of this problem in our model. In the variant *with unrestricted facilities*, the query is a set  $S \subseteq V$  of clients for which we should open facilities. The goal is to open a subset  $F \subseteq V$  of facilities, and connect each city to an open facility so that the sum of the total opening and connection costs is minimized. In the other variant, one with *restricted facilities*, the facilities that can be opened are given as a part of query (together with their opening costs).

*Our Results.* In this paper we restrict our attention to doubling metric spaces which include growth-restricted metric spaces and constant dimensional Euclidean spaces. In other words we assume that the graph  $G$  induces a doubling metric and the algorithms are given the distance matrix  $G^*$  as an input or compute it at the beginning of the preprocessing phase. This restriction is often assumed in the routing setting [9, 5] and hence it is a natural question to see how it can impact the multicast problems. Using this assumption we show that solutions with nearly optimal bounds are possible. The main result of the paper is the data structure that requires  $O(n \log n)$  memory and can find a constant ratio approximate Steiner tree over a given set of size  $k$  in  $O(k(\log k + \log \log n))$  time. Moreover, we show data structures with essentially the same complexities for solving STEINER FOREST, both versions of FACILITY LOCATION,  $k$ -MEDIAN and TSP. The query bound is optimal, up to  $\log k$  and  $\log \log n$  factors, as no algorithm can answer queries in time less than linear in  $k$  as it needs to read the input. For the exact approximation ratios of our algorithms refer to Section 3.2.

All of these results are based on a new hierarchical data structure for representing a doubling metric that approximates original distances with  $(1 + \epsilon)$ -multiplicative factor. The concept of a hierarchical data structure for representing a doubling metric is not novel – it originates from the work of Clarkson [6] and was then used in a number of papers, in particular our data structure is based on the one due to Jia et al. [10]. Our main technical contribution here is adapting and extending this data structure so that for any subset  $S \subset V$  a substructure corresponding to  $S$  can be retrieved in  $O(k(\log k + \log \log n))$  using only the information in the data structure, without a distance oracle. The substructure is then transformed to a pseudo-spanner described above. Note that our complexity bounds do not depend on the stretch of the metrics, unlike in many previous works (e.g. [11]). Another original concept in our work is an application of spanners (or, more precisely, pseudo-spanners) to improve working time of approximation algorithms for metric problems. As a result, the query times for the metric problems we consider are  $O(k(\text{polylog} k + \log \log n))$ .

Astonishingly, our hierarchical data structure can be used to obtain dynamic algorithms for the STEINER TREE problem. This problem attracted considerable attention [24, 33] in the recent years. However, due to the hardness of the problem none of these papers has given any improvement in the running time over the static algorithms. Here, we give first fully dynamic algorithm for the problem in the case of doubling metric. Our algorithm is given a static graph and then maintains information about the Steiner tree built on a given set  $X$  of nodes. It supports insertion of vertices in  $O(\log^5 k + \log \log n)$  time, and deletion in  $O(\log^5 k)$  time, where  $k = |X|$ .

*Related Work.* The problems considered in this paper are related to several algorithmic topics studied extensively in recent years. Many researchers tried to answer the question whether problems in huge networks can be solved more efficiently than by processing the whole input. Nevertheless, the model proposed in this paper has never been considered before. Moreover, we believe that within the proposed framework it is possible to achieve complexities that are close to being practical. We present such results only in the case of doubling metric, but hope that the further study will extend these results to a more general setting. Our results are related to the following concepts:

- Universal Algorithms — this model does not allow any processing in the query time, we allow it and get much better approximation ratios,
- Spanners and Approximate Distance Oracles — although a spanner of a subspace of a doubling metric can be constructed in  $O(k \log k)$ -time, the construction algorithm requires a distance oracle (i.e. the full  $\Theta(n^2)$ -size distance matrix).
- Sublinear Approximation Algorithms — here we cannot preprocess the data, allowing it we can get much better approximation ratios,
- Dynamic Spanning Trees — most existing results are only applicable to dynamic MST and not dynamic Steiner tree, and the ones concerning the latter work in different models than ours.

## 2 Space Partition Tree

In this section we extend the techniques developed by Jia et al. [10]. Several statements as well as the overall construction are similar to those given by Jia et al. However, our approach is tuned to better suit our needs, in particular to allow for a fast subtree extraction and a spanner construction – techniques introduced in Sections 2 and 3 that are crucial for efficient approximation algorithms.

Let  $(V, d)$  be a finite doubling metric space with  $|V| = n$  and a doubling constant  $\lambda$ , i.e., for every  $r > 0$ , every ball of radius  $2r$  can be covered with at most  $\lambda$  balls of radius  $r$ . By **stretch** we denote the stretch of the metric  $d$ , that is, the largest distance in  $V$  divided by the smallest distance. We use space partition schemes for doubling metrics to create a partition tree. In the next two subsections, we show that this tree can be stored in  $O(n \log n)$  space, and that a subtree induced by any subset  $S \subset V$  can be extracted efficiently.

Let us first briefly introduce the notion of a space partition tree, that is used in the remainder of this paper. Precise definitions and proofs are omitted due to space limitations and will be included in the full version of the paper.

The basic idea is to construct a sequence  $\mathbb{S}_0, \mathbb{S}_1, \dots, \mathbb{S}_M$  of partitions of  $V$ . We require that  $\mathbb{S}_0 = \{\{v\} : v \in V\}$ , and  $\mathbb{S}_M = \{V\}$ , and in general the diameters of the sets in  $\mathbb{S}_k$  are growing exponentially in  $k$ . We also maintain the neighbourhood structure for each  $\mathbb{S}_k$ , i.e., we know which sets in  $\mathbb{S}_k$  are close to each other (this is explained in more detail later on). Notice that the partitions together with the neighbourhood structure are enough to approximate the distance between any two points  $x, y$  — one only needs to find the smallest  $k$ , such that the sets in  $\mathbb{S}_k$  containing  $x$  and  $y$  are close to each other (or are the same set).

There are two natural parameters in this sort of scheme. One of them is how fast the diameters of the sets grow, this is controlled by  $\tau \in \mathbb{R}, \tau \geq 1$  in our constructions. The faster the set diameters grow, the smaller the number of partitions is. The second parameter is how distant can the sets in a partition be to be still considered neighbours, this is controlled by a nonnegative integer  $\eta$  in our constructions. The smaller this parameter is, the smaller the number of neighbours is. Manipulating these parameters allows us to decrease the space required to store the partitions, and consequently also the running time of our algorithms. However, this also comes at a price of lower quality approximation.

In what follows, each  $\mathbb{S}_k$  is a subpartition of  $\mathbb{S}_{k+1}$  for  $k = 0, \dots, M - 1$ . That is, the elements of these partitions form a tree, denoted by  $\mathbb{T}$ , with  $\mathbb{S}_0$  being the set of leaves and  $\mathbb{S}_M$  being the root. We say that  $S \in \mathbb{S}_j$  is a *child* of  $S^* \in \mathbb{S}_{j+1}$  in  $\mathbb{T}$  if  $S \subset S^*$ .

Let  $r_0$  be smaller than the minimal distance between points in  $V$  and let  $r_j = \tau^j r_0$ . We show that  $\mathbb{S}_k$ -s and  $\mathbb{T}$  satisfying the following properties can be constructed in polynomial time:

- (1) **Exponential growth:** Every  $S \in \mathbb{S}_j$  is contained in a ball of radius  $r_j \tau 2^{-\eta} / (\tau - 1)$ .
- (2) **Small neighbourhoods:** For every  $S \in \mathbb{S}_j$ , the union  $\bigcup \{B_{r_j}(v) : v \in S\}$  crosses at most  $\lambda^{3+\eta}$  sets  $S'$  from the partition  $\mathbb{S}_j$  — we say that  $S$  *knows* these  $S'$ . We also extend this notation and say that if  $S$  knows  $S'$ , then every  $v \in S$  knows  $S'$ .
- (3) **Small degrees:** For every  $S^* \in \mathbb{S}_{j+1}$  all children of  $S^*$  know each other and, consequently, there are at most  $\lambda^{\eta+3}$  children of  $S^*$ .
- (4) **Distance approximation:** If  $v, v^* \in V$  are different points such that  $v \in S_1 \in \mathbb{S}_j$ ,  $v \in S_2 \in \mathbb{S}_{j+1}$  and  $v^* \in S_1^* \in \mathbb{S}_j$ ,  $v^* \in S_2^* \in \mathbb{S}_{j+1}$  and  $S_2$  knows  $S_2^*$  but  $S_1$  does not know  $S_1^*$ , then

$$r_j \leq d(v, v^*) < \left(1 + \frac{4\tau 2^{-\eta}}{\tau - 1}\right) \tau r_j;$$

For any  $\varepsilon > 0$ , the  $\tau$  and  $\eta$  constants can be adjusted so that the last condition becomes  $r_j \leq d(v, v^*) \leq (1 + \varepsilon)r_j$ .

*Remark 1.* We note that not all values of  $\tau$  and  $\eta$  make sense for our construction. We omit these additional constraints here.

## 2.1 Compressed Tree $\hat{\mathbb{T}}$ and Additional Information at Nodes

Let us now show how to efficiently compute and store the tree  $\mathbb{T}$ . Recall that the leaves of  $\mathbb{T}$  are one point sets and, while going up in the tree, these sets join into bigger sets. Note that if  $S$  is an inner node of  $\mathbb{T}$  and it has only one child  $S'$  then both nodes  $S$  and  $S'$  represent the same set. Nodes  $S$  and  $S'$  can differ only by their sets of acquaintances, i.e. the sets of nodes known to them. If these sets are equal, there is some sort of redundancy in  $\mathbb{T}$ . To reduce the space usage we store only a compressed version of the tree  $\mathbb{T}$ .

Let us introduce some useful notation. For a node  $v$  of  $\mathbb{T}$  let  $\text{set}(v)$  denote the set corresponding to  $v$  and let  $\text{level}(v)$  denote the level of  $v$ , where leaves are at level zero. Let  $S_a, S_b$  be a pair of sets that know each other at level  $j_{ab}$  and do not know each other at level  $j_{ab} - 1$ . Then the triple  $(S_a, S_b, j_{ab})$  is called a *meeting* of  $S_a$  and  $S_b$  at level  $j_{ab}$ .

**Definition 2 (Compressed tree).** *The compressed version of  $\mathbb{T}$ , denoted  $\hat{\mathbb{T}}$ , is obtained from  $\mathbb{T}$  by replacing all maximal paths such that all inner nodes have exactly one child by a single edge. For each node  $v$  of  $\hat{\mathbb{T}}$  we store  $\text{level}(v)$  (the lowest level of  $\text{set}(v)$  in  $\mathbb{T}$ ) and a list of all meetings of  $\text{set}(v)$ , sorted by level.*

Obviously  $\hat{\mathbb{T}}$  has at most  $2n - 1$  nodes since it has exactly  $n$  leaves and each inner node has at least two children but we also have to ensure that the total number of meetings is reasonable.

Note that the sets at nodes of  $\hat{\mathbb{T}}$  are pairwise distinct. To simplify the presentation we will identify nodes and the corresponding sets. Consider a meeting  $m = (S_a, S_b, j_{ab})$ . Let  $p_a$  (resp.  $p_b$ ) denote the parent of  $S_a$  (resp.  $S_b$ ) in  $\hat{\mathbb{T}}$ . We say that  $S_a$  is *responsible* for the meeting  $m$  when  $\text{level}(p_a) \leq \text{level}(p_b)$  (when  $\text{level}(p_a) = \text{level}(p_b)$ , both  $S_a$  and  $S_b$  are responsible for the meeting  $m$ ). Note that if  $S_a$  is responsible for a meeting  $(S_a, S_b, j_{ab})$ , then  $S_a$  knows  $S_b$  at level  $\text{level}(p_a) - 1$ . From this and Property 2 of the partition tree we get the following.

**Lemma 3.** *Each set in  $\hat{\mathbb{T}}$  is responsible for at most  $\lambda^{3+\eta}$  meetings.*

**Corollary 4.** *There are  $\leq (2n - 1)\lambda^{3+\eta}$  meetings stored in the compressed tree  $\hat{\mathbb{T}}$ , i.e.  $\hat{\mathbb{T}}$  takes  $O(n)$  space.*

**Lemma 5.** *One can augment the tree  $\hat{\mathbb{T}}$  with additional information of size  $O(n\lambda^{3+\eta})$ , so that for any pair of nodes  $x, y$  of  $\hat{\mathbb{T}}$  one can decide if  $x$  and  $y$  know each other, and if that is the case the level of the meeting is returned. The query takes  $O(\eta \log \lambda)$  time.*

*Proof.* For each node  $v$  in  $\hat{\mathbb{T}}$  we store all the meetings it is responsible for, using a dictionary  $D(m)$  — the searches take  $O(\log(\lambda^{3+\eta})) = O(\eta \log \lambda)$  time. To process the query it suffices to check if there is an appropriate meeting in  $D(x)$  or in  $D(y)$ .  $\square$

In order to give a fast subtree extraction algorithm, we need to define the following operation `meet`. Let  $u, v \in \hat{\mathbb{T}}$  be two given nodes. Let  $v(j)$  denote the node in  $\mathbb{T}$  on the path from  $v$  to the root at level  $j$ , similarly define  $u(j)$ . The value of  $\text{meet}(u, v)$  is the lowest level, such that  $v(j)$  and  $u(j)$  know each other. Such level always exists, because in the end all nodes merge into root and nodes know each other at one level before they are merged (see Property 3 of the partition tree). A technical proof of the following lemma is omitted due to space limitations.

**Lemma 6.** *The tree  $\hat{\mathbb{T}}$  can be augmented so that the `meet` operation can be performed in  $O(\eta \log \lambda \log \log n)$  time. The augmented  $\mathbb{T}$  tree can be stored in  $O(\lambda^{3+\eta} n \log n)$  space and computed in polynomial time.*

## 2.2 Fast Subtree Extraction

For any subset  $S \subseteq V$  we are going to define an  $S$ -subtree of  $\hat{\mathbb{T}}$ , denoted  $\hat{\mathbb{T}}(S)$ . Intuitively, this is the subtree of  $\hat{\mathbb{T}}$  induced by the leaves corresponding to  $S$ . Additionally we store all the meetings in  $\hat{\mathbb{T}}$  between the nodes corresponding to the nodes of  $\hat{\mathbb{T}}(S)$ .

More precisely, the set of nodes of  $\hat{\mathbb{T}}(S)$  is defined as  $\{A \cap S : A \subseteq V \text{ and } A \text{ is a node of } \hat{\mathbb{T}}\}$ . A node  $Q$  of  $\hat{\mathbb{T}}(S)$  is an ancestor of a node  $R$  of  $\hat{\mathbb{T}}(S)$  iff  $R \subseteq Q$ . This defines the edges of  $\hat{\mathbb{T}}(S)$ . Moreover, for two nodes  $A, B$  of  $\hat{\mathbb{T}}$  such that both  $A$  and  $B$  intersect  $S$ , if  $A$  knows  $B$  at level  $j$ , we say that  $A \cap S$  knows  $B \cap S$  in  $\hat{\mathbb{T}}(S)$  at level  $j$ . A triple  $(Q, R, j_{QR})$ , where  $j_{QR}$  is a minimal level such that  $Q$  knows  $R$  at level  $j_{QR}$ , is called a *meeting*. The *level* of a node  $Q$  of  $\hat{\mathbb{T}}(S)$  is the lowest level of a node  $A$  of  $\hat{\mathbb{T}}$  such that  $Q = A \cap S$ . Together with each node  $Q$  of  $\hat{\mathbb{T}}(S)$  we store its level and a list of all its meetings  $(Q, R, j_{QR})$ . A node  $Q$  is *responsible* for a meeting  $(Q, R, l)$  when  $\text{level}(\text{parent}(Q)) \leq \text{level}(\text{parent}(R))$ .

*Remark 7.* The subtree  $\hat{\mathbb{T}}(S)$  is not necessarily equal to any compressed tree for the metric space  $(S, d|_{S^2})$ .

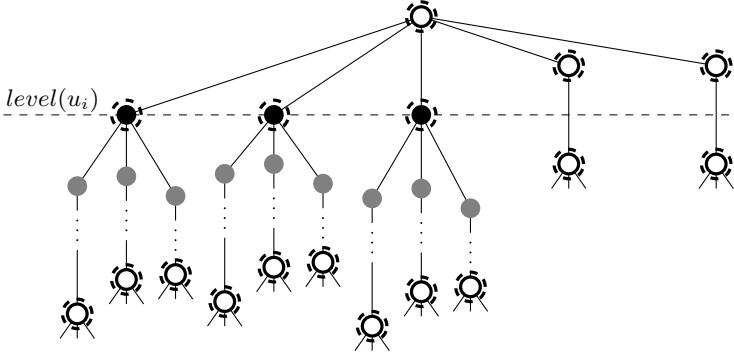
In this subsection we describe how to extract  $\hat{\mathbb{T}}(S)$  from  $\hat{\mathbb{T}}$  efficiently. The extraction runs in two phases. In the first phase we find the nodes and edges of  $\hat{\mathbb{T}}(S)$  and in the second phase we find the meetings.

**Finding the Nodes and Edges of  $\hat{\mathbb{T}}(S)$ .** We construct the extracted tree in a bottom-up fashion. Note that we can not simply go up the tree from the leaves corresponding to  $S$  because we could visit a lot of nodes of  $\hat{\mathbb{T}}$  which are not the nodes of  $\hat{\mathbb{T}}(S)$ . The key observation is that if  $A$  and  $B$  are nodes of  $\hat{\mathbb{T}}$ , such that  $A \cap S$  and  $B \cap S$  are nodes of  $\hat{\mathbb{T}}(S)$  and  $C$  is the lowest common ancestor of  $A$  and  $B$ , then  $C \cap S$  is a node of  $\hat{\mathbb{T}}(S)$  and it has level  $\text{level}(C)$ . Due to space limitations we only state that nodes and edges of  $\hat{\mathbb{T}}(S)$  can be found in  $O(k \log k)$  time.

**Finding the Meetings in  $\hat{\mathbb{T}}(S)$ .** We generate meetings in a top-down fashion. We consider the nodes of  $\hat{\mathbb{T}}(S)$  in groups. Each group corresponds to a single

level. Now assume we consider a group of nodes  $u_1, \dots, u_t$  at some level  $\ell$ . Let  $v_1, \dots, v_{t'}$  be the set of children of all nodes  $u_i$  in  $\hat{\mathbb{T}}(S)$ . For each node  $v_i$ ,  $i = 1, \dots, t'$  we are going to find all the meetings it is responsible for. Any such meeting  $(v_i, x, j)$  is of one of two types:

1.  $\text{parent}(x) \in \{u_1, \dots, u_t\}$ , possibly  $\text{parent}(x) = \text{parent}(v_i)$ , or
2.  $\text{parent}(x) \notin \{u_1, \dots, u_t\}$ , i.e.  $\text{level}(\text{parent}(x)) > \ell$ .



**Fig. 1.** Extracting meetings. The figure contains a part of tree  $\hat{\mathbb{T}}$ . Nodes corresponding to the nodes of  $\hat{\mathbb{T}}(S)$  are surrounded by dashed circles. The currently processed group of nodes  $(u_i, i = 1, \dots, k)$  are filled with black. Nodes from the set  $L$  are filled with gray. The nodes below the gray nodes are the the nodes  $v_j$ , i.e. the children of nodes  $u_i$  in  $\hat{\mathbb{T}}(S)$ .

The meetings of the first kind are generated as follows. Consider the following set of nodes of  $\hat{\mathbb{T}}$  (drawn as grey disks in Figure 1).

$$L = \{x : x \text{ is the first node on the path in } \hat{\mathbb{T}} \text{ from } \text{origin}(u_i) \text{ to } \text{origin}(v_j), \text{ for some } i = 1, \dots, t, j = 1, \dots, t'\}$$

We mark all the nodes of  $L$ . Next, we identify all pairs of nodes of  $L$  that know each other. By Lemma 3 there are at most  $\lambda^{3+\eta} t' = O(t')$  such pairs and these pairs can be easily found by scanning, for each  $x \in L$ , all the meetings  $x$  is responsible for and such that the node  $x$  meets is in  $L$ . In this way we identify all pairs of children  $(v_i, v_j)$  such that  $v_i$  knows  $v_j$ , namely if  $x, y \in L$  and  $x$  knows  $y$  in  $\hat{\mathbb{T}}$ , then  $x \cap S$  knows  $y \cap S$  in  $\hat{\mathbb{T}}(S)$ . Then, if  $v_i$  knows  $v_j$ , the level of their meeting can be found in  $O(\tau \log \lambda \log \log n)$  time using operation  $\text{meet}(\text{origin}(v_i), \text{origin}(v_j))$  from Lemma 6. Hence, finding the meetings of the first type takes  $O(\lambda^{3+\eta} \log \lambda \tau t' \log \log n)$  time for one group of nodes, and  $O(\lambda^{3+\eta} \log \lambda \tau k \log \log n)$  time in total.

Finding the meetings of the second type is easier. Consider any second type meeting  $(v_i, w, l)$ . Let  $u_j$  be the parent of  $v_i$ . Then there is a meeting  $(u_j, w, \text{level}(u_j))$  stored in  $u_j$ . Hence it suffices to consider, for each  $u_j$  all its

meetings at level  $\text{level}(u_j)$ . For every such meeting  $(u_j, w, \text{level}(u_j))$ , and for every child  $v_i$  of  $u_j$  we can apply  $\text{meet}(\text{origin}(v_i), \text{origin}(w))$  from Lemma 6 to find the meeting of  $v_i$  and  $w$ . For the time complexity, note that by Property 2 of the partition tree, a node  $u_j$  meets  $\lambda^{3+\eta} = O(1)$  nodes at level  $\text{level}(u_j)$ . Since we can store the lists of meetings sorted by levels, we can extract all those meetings in  $O(\lambda^{3+\eta})$  time. For each meeting we iterate over the children of  $u_j$  (Property 3 of the partition tree) and apply Lemma 6. This results in  $O(\lambda^{3+\eta} \log \lambda \tau \log \log n)$  time per a child, hence  $O(\lambda^{3+\eta} \log \lambda \tau k \log \log n)$  time in total.

After extracting all the meetings, we sort them by levels in  $O(k \log k)$  time.

We can claim now the following theorem.

**Theorem 8.** *For a given set  $S \subseteq V$  ( $|S| = k$ ) we can extract the  $S$ -subtree of the compressed tree  $\hat{\mathbb{T}}$  in time  $O(\lambda^{3+\eta} \log \lambda \tau k (\log k + \log \log n)) = O(k(\log k + \log \log n))$ .*

### 3 Pseudospanner Construction and Applications in Approximation

In this section we use the subtree extraction procedure described in the previous section, to construct for any set  $S \subseteq V$ , a graph that is essentially a small constant stretch spanner for  $S$ . We then use it to give fast approximations algorithms for several problems.

#### 3.1 Pseudospanner Construction

**Definition 9.** *Let  $G = (V, E_G)$  be an undirected connected graph with a weight function  $w_G : E_G \rightarrow \mathbb{R}_+$ . A graph  $H = (V, E_H)$ ,  $E_H \subseteq E_G$  with a weight function  $w_H : E_H \rightarrow \mathbb{R}_+$  is an  $f$ -pseudospanner for  $G$  if for every pair of vertices  $u, v \in V$  we have  $d_G(u, v) \leq d_H(u, v) \leq f \cdot d_G(u, v)$ , where  $d_G$  and  $d_H$  are shortest path metrics induced by  $w_G$  and  $w_H$ . The number  $f$  in this definition is called the stretch of the pseudospanner. A pseudospanner for a metric space is simply a pseudospanner for the complete weighted graph induced by the metric space.*

*Remark 10.* Note the subtle difference between the above definition and the classical spanner definition. A pseudospanner  $H$  is a subgraph of  $G$  in terms of vertex sets and edge sets but it does not inherit the weight function  $w_G$ . We cannot construct spanners in the usual sense without maintaining the entire distance matrix, which would require prohibitive quadratic space. However, pseudospanners constructed below become classical spanners when provided the original weight function.

Also note, that it immediately follows from the definition of a pseudospanner that for all  $uv \in E_H$  we have  $w_G(u, v) \leq w_H(u, v)$ .

In the remainder of this section we let  $(V, d)$  be a metric space of size  $n$ , where  $d$  is doubling with doubling constant  $\lambda$ . We also use  $\hat{\mathbb{T}}$  to denote the hierarchical

tree data structure corresponding to  $(V, d)$ , and  $\eta$  and  $\tau$  denote the parameters of  $\hat{\mathbb{T}}$ . For any  $S \subseteq V$ , we use  $\hat{\mathbb{T}}(S)$  to denote the subtree of  $\hat{\mathbb{T}}$  corresponding to  $S$ , as described in the previous section. Finally, we define a constant  $C(\eta, \tau) = \left(1 + \left(\frac{\tau}{\tau-1}\right)^2 2^{3-\eta}\right) \tau r_j$ .

**Theorem 11.** *Given  $\hat{\mathbb{T}}$  and set  $S \subseteq V$ , where  $|S| = k$ , one can construct a  $C(\eta, \tau)$ -pseudospanner for  $S$  in time  $O(k(\log k + \log \log n))$ . This spanner has size  $O(k)$ .*

*Remark 12.* Similarly to Property 4 of the partition tree, we can argue that the above theorem gives a  $(1 + \varepsilon)$ -pseudospanner for any  $\varepsilon > 0$ . Here, we need to take  $\tau = 1 + \frac{\varepsilon}{3}$  and  $\eta = O(\frac{1}{\varepsilon^3})$ .

*Remark 13.* It is of course possible to store the whole distance matrix of  $V$  and construct a spanner for any given subspace  $S$  using standard algorithms. However, this approach has a prohibitive  $\Theta(n^2)$  space complexity.

### 3.2 Applications in Approximation

Results of the previous subsection immediately give several interesting approximation algorithms. In all the corollaries below we assume the tree  $\hat{\mathbb{T}}$  is already constructed.

**Corollary 14 (Steiner Forest).** *Given a set of points  $S \subseteq V$ ,  $|S| = k$ , together with a set of requirements  $R$  consisting of pairs of elements of  $S$ , a Steiner forest with total edge-length at most  $2C(\eta, \tau)OPT = (2 + \varepsilon)OPT$ , for any  $\varepsilon > 0$  can be constructed in time  $O(k(\log^2 k + \log \log n))$ .*

*Proof.* We use the  $O(m \log^2 n)$  algorithm of Cole et al. [7] (where  $m$  is the number of edges) on the pseudospanner guaranteed by Theorem 11. This algorithm can give a guarantee  $2 + \varepsilon$  for an arbitrarily small  $\varepsilon$ .  $\square$

Similarly by using the MST approximation for TSP we get

**Corollary 15 (TSP).** *Given a set of points  $S \subseteq V$ ,  $|S| = k$ , a Hamiltonian cycle for  $S$  of total length at most  $2C(\eta, \tau)OPT = (2 + \varepsilon)OPT$  for any  $\varepsilon > 0$  can be constructed in time  $O(k(\log k + \log \log n))$ .*

Currently, the best approximation algorithm for the facility location problem is the 1.52-approximation of Mahdian, Ye and Zhang [12]. A fast implementation using Thorup's ideas [14] runs in deterministic  $O(m \log m)$  time, where  $m = |F| \cdot |C|$ , and if the input is given as a weighted graph of  $n$  vertices and  $m$  edges, in  $\tilde{O}(n + m)$  time, with high probability (i.e. with probability  $\geq 1 - 1/n^{\omega(1)}$ ). In an earlier work, Thorup [15] considers also the  $k$ -center and  $k$ -median problems in the graph model. When the input is given as a weighted graph of  $n$  vertices and  $m$  edges, his algorithms run in  $\tilde{O}(n + m)$  time, w.h.p. and have approximation guarantees of 2 for the  $k$ -center problem and  $12 + o(1)$  for the  $k$ -median problem. By using this latter algorithm with our fast spanner extraction we get the following corollary.

**Corollary 16 (Facility Location with restricted facilities).** *Given two sets of points  $C \subseteq V$  (cities) and  $F \subseteq V$  (facilities) together with opening cost  $f_i$  for each facility  $i \in F$ , for any  $\varepsilon > 0$ , a  $(1.52 + \varepsilon)$ -approximate solution to the facility location problem can be constructed in time  $O((|C| + |F|)(\log^{O(1)}(|C| + |F|) + \log \log |V|))$ , w.h.p.*

The application of our results to the variant of FACILITY LOCATION with unrestricted facilities is not so immediate. We were able to obtain the following.

**Theorem 17 (Facility Location with unrestricted facilities).** *Assume that for each point of  $n$ -point  $V$  there is assigned an opening cost  $f(x)$ . Given a set of  $k$  points  $C \subseteq V$ , for any  $\varepsilon > 0$ , a  $(3.04 + \varepsilon)$ -approximate solution to the facility location problem with cities' set  $C$  and facilities' set  $V$  can be constructed in time  $O(k \log k (\log^{O(1)} k + \log \log n))$ , w.h.p.*

Our approach there is a reduction to the variant with restricted facilities. The general, rough idea is the following: during the preprocessing phase, for every point  $x \in V$  we compute a small set  $F(x)$  of facilities that seem a good choice for  $x$ , and when processing a query for a set of cities  $C$ , we just apply Corollary 16 to cities' set  $C$  and facilities' set  $\bigcup_{c \in C} F(c)$ .

**Corollary 18 ( $k$ -center and  $k$ -median).** *Given a set of points  $C \subseteq V$  and a number  $r \in \mathbb{N}$ , for any  $\varepsilon > 0$ , one can construct:*

- (i) *a  $(2 + \varepsilon)$ -approximate solution to the  $r$ -center problem, or*
- (ii) *a  $(12 + \varepsilon)$ -approximate solution to the  $r$ -median problem*

*in time  $O(|C|(\log |C| + \log \log |V|))$ , w.h.p.*

## 4 Dynamic Minimum Spanning Tree and Steiner Tree

In this section we give one last application of our hierarchical data structure. It has a different flavour from the other applications presented in this paper since it is not based on constructing a spanner, but uses the data structure directly. We solve the Dynamic Minimum Spanning Tree / Steiner Tree (DMST/DST) problem, where we need to maintain a spanning/Steiner tree of a subspace  $X \subseteq V$  throughout a sequence of vertex additions and removals to/from  $X$ .

The quality of our algorithm is measured by the total cost of the tree produced relative to the optimum tree, and time required to add/delete vertices. Let  $|V| = n$ ,  $|X| = k$ . Our goal is to give an algorithm that maintains a constant factor approximation of the optimum tree, while updates are polylogarithmic in  $k$ , and do not depend (or depend only slightly) on  $n$ . It is clear that it is enough to find such an algorithm for DMST. Due to space limitations we only formulate the results.

**Theorem 19.** *Given the compressed tree  $\hat{\mathbb{T}}(V)$ , we can maintain an  $O(1)$ -approximate Minimum Spanning Tree for a subset  $X$  subject to insertions and deletions of vertices. The insert operation works in  $O(\log^5 k + \log \log n)$  time and the delete operation works in  $O(\log^5 k)$  time,  $k = |X|$ . Both times are expected and amortized.*

## References

1. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
2. Bilò, D., Böckenhauer, H.-J., Hromkovič, J., Královíč, R., Mömke, T., Widmayer, P., Zych, A.: Reoptimization of steiner trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 258–269. Springer, Heidelberg (2008)
3. Böckenhauer, H.-J., Hromkovič, J., Královíč, R., Mömke, T., Rossmanith, P.: Reoptimization of steiner trees: Changing the terminal set. *Theor. Comput. Sci.* 410(36), 3428–3435 (2009)
4. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008)
5. Chan, H.T.-H., Gupta, A., Maggs, B.M., Zhou, S.: On hierarchical routing in doubling metrics. In: Proc. SODA 2005, pp. 762–771 (2005)
6. Clarkson, K.L.: Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry* 22(1), 63–93 (1999)
7. Cole, R., Hariharan, R., Lewenstein, M., Porat, E.: A faster implementation of the Goemans-Williamson clustering algorithm. In: Proc. SODA 2001, pp. 17–25 (2001)
8. Escoffier, B., Milanic, M., Paschos, V.T.: Simple and fast reoptimizations for the Steiner tree problem. *Algorithmic Operations Research* 4(2), 86–94 (2009)
9. Har-Peled, S., Mendel, M.: Fast construction of nets in low dimensional metrics, and their applications. In: Proc. SCG 2005, pp. 150–158 (2005)
10. Jia, L., Lin, G., Noubir, G., Rajaraman, R., Sundaram, R.: Universal approximations for TSP, Steiner Tree and Set Cover. In: STOC 2005, pp. 1234–1415 (2005)
11. Krauthgamer, R., Lee, J.R.: Navigating nets: simple algorithms for proximity search. In: Proc. SODA 2004, pp. 798–807 (2004)
12. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. *SIAM Journal on Computing* 36(2), 411–432 (2006)
13. Salama, H.F., Reeves, D.S., Viniotis, Y., Sheu, T.-L.: Evaluation of multicast routing algorithms for real-time communication on high-speed networks. In: Proceedings of the IFIP Sixth International Conference on High Performance Networking VI, pp. 27–42 (1995)
14. Thorup, M.: Quick and good facility location. In: Proc. SODA 2003, pp. 178–185 (2003)
15. Thorup, M.: Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing* 34(2), 405–432 (2005)
16. Winter, P.: Steiner problem in networks: A survey. *Networks* 17(2), 129–167 (1987)

# *f*-Sensitivity Distance Oracles and Routing Schemes

Shiri Chechik<sup>1</sup>, Michael Langberg<sup>2</sup>, David Peleg<sup>1,\*</sup>, and Liam Roditty<sup>3</sup>

<sup>1</sup> Department of Computer Science, The Weizmann Institute, Rehovot, Israel  
`{shiri.chechik,david.peleg}@weizmann.ac.il`

<sup>2</sup> Computer Science Division, Open University of Israel, Raanana, Israel  
`mikel@openu.ac.il`

<sup>3</sup> Department of Computer Science, Bar-Ilan University, Ramat-Gan, Israel  
`liamr@macs.biu.ac.il`

**Abstract.** An *f-sensitivity distance oracle* for a weighted undirected graph  $G(V, E)$  is a data structure capable of answering restricted distance queries between vertex pairs, i.e., calculating distances on a subgraph avoiding some forbidden edges. This paper presents an efficiently constructible *f*-sensitivity distance oracle that given a triplet  $(s, t, F)$ , where  $s$  and  $t$  are vertices and  $F$  is a set of forbidden edges such that  $|F| \leq f$ , returns an estimate of the distance between  $s$  and  $t$  in  $G(V, E \setminus F)$ . For an integer parameter  $k \geq 1$ , the size of the data structure is  $O(fkn^{1+1/k} \log(nW))$ , where  $W$  is the heaviest edge in  $G$ , the stretch (approximation ratio) of the returned distance is  $(8k - 2)(f + 1)$ , and the query time is  $O(|F| \cdot \log^2 n \cdot \log \log n \cdot \log \log d)$ , where  $d$  is the distance between  $s$  and  $t$  in  $G(V, E \setminus F)$ .

The paper also considers *f*-sensitive compact routing schemes, namely, routing schemes that avoid a given set of forbidden (or *failed*) edges. It presents a scheme capable of withstanding up to two edge failures. Given a message  $M$  destined to  $t$  at a source vertex  $s$ , in the presence of a forbidden edge set  $F$  of size  $|F| \leq 2$  (unknown to  $s$ ), our scheme routes  $M$  from  $s$  to  $t$  in a distributed manner, over a path of length at most  $O(k)$  times the length of the optimal path (avoiding  $F$ ). The total amount of information stored in vertices of  $G$  is  $O(kn^{1+1/k} \log(nW) \log n)$ .

## 1 Introduction

*The problems:* This paper considers succinct data structures capable of supporting efficient responses to *distance sensitivity* queries on an undirected graph  $G(V, E)$  with edge weights  $\omega$ . A distance sensitivity query  $(s, t, e)$  requires finding, for a given pair of vertices  $s$  and  $t$  in  $V$  and a *forbidden* edge  $e \in E$ , the distance (namely, the length of the shortest path) between  $s$  and  $t$  in  $G(V, E \setminus \{e\})$ . An *f-sensitivity distance oracle* is a generalized version of the distance sensitivity data structure, in which instead of a single forbidden edge  $e$ , the query may

---

\* Supported in part by a France-Israel cooperation grant (“Mutli-Computing” project) from the Israel Ministry of Science.

include a set  $F$  of size at most  $f$  of forbidden edges. In response to a query  $(s, t, F)$ , the data structure has to return the distance between  $s$  and  $t$  in  $G(V, E \setminus F)$ .

For certain natural applications in communication networks, one may be interested in more than just the distance between  $s$  and  $t$  in  $G(V, E \setminus F)$ . In particular, an  *$f$ -sensitivity routing protocol* is a distributed algorithm that, for any set of forbidden (or *failed*) edges  $F$ , enables the vertex  $s$  to route a message to  $t$  along a shortest or near-shortest path in  $G(V, E \setminus F)$  in an efficient manner (and without knowing  $F$  in advance). In addition to route efficiency, it is desirable to optimize also the amount of memory stored in the routing tables of the nodes, possibly at the cost of lower route efficiency, giving rise to the problem of designing  *$f$ -sensitivity* (or *fault-tolerant*) *compact routing schemes*.

The current paper addresses the design of  $f$ -sensitivity distance oracles and  $f$ -sensitivity compact routing schemes in a relaxed setting in which approximate answers are acceptable. The main results of the paper are summarized by the following theorems. Throughout, our underlying undirected graph  $G$  has edge weights  $\omega \in [1, W]$ ,  $m = |E|$  and  $n = |V|$ . For two vertices  $s$  and  $t$  in  $G$ , denote by  $\text{dist}(s, t, G \setminus F)$  the distance between  $s$  and  $t$  in  $G(V, E \setminus F)$ .

**Theorem 1.** *Let  $f, k \geq 1$  be integer parameters. Let  $F \subset E$  be a set of forbidden edges, where  $|F| \leq f$ . For a pair of vertices  $s, t \in V$  let  $d = \text{dist}(s, t, G \setminus F)$ . There exists a polynomial-time constructible data structure **Sens-Or**( $G, \omega, f, k$ ) of size  $O(fkn^{1+1/k} \log(nW))$ , that returns in time  $O(|F| \log^2 n \cdot \log \log n \cdot \log \log d)$  a distance estimate  $\tilde{d}$  satisfying  $d \leq \tilde{d} \leq (8k - 2)(f + 1) \cdot d$ .*

**Theorem 2.** *There exists a 2-sensitive compact routing scheme that given a message  $M$  at a source vertex  $s$  and a destination  $t$ , in the presence of a forbidden edge set  $F$  of size at most 2 (unknown to  $s$ ), routes  $M$  from  $s$  to  $t$  in a distributed manner over a path of length at most  $O(k \cdot \text{dist}(s, t, G \setminus F))$ . The total amount of information stored in the vertices of  $G$  is  $O(kn^{1+1/k} \log(nW) \log n)$ .*

*Related work:* In [15], Demetrescu et al. showed that it is possible to preprocess a directed weighted graph in time  $\tilde{O}(mn^2)$  to produce a data structure of size  $O(n^2 \log n)$  capable of answering 1-sensitivity distance queries (with a forbidden edge or vertex) in  $O(1)$  time. In two recent papers [89], Karger and Bernstein improved the preprocessing time for 1-sensitivity queries to  $O(n^2 \sqrt{m})$  and then  $\tilde{O}(mn)$ , with unchanged size and query time.

In [17], Duan and Pettie presented an algorithm for 2-sensitivity queries (with 2 forbidden edges or vertices), based on a polynomial time constructible data structure of size  $O(n^2 \log^3 n)$  that is capable of answering 2-sensitivity queries in  $O(\log n)$  time. The authors of [17] comment that their techniques do not seem to extend beyond forbidden sets of size 2, and that even a solution to the 3-sensitivity problem involving a data structure of size  $\tilde{O}(n^2)$  does not seem in reach. In contrast, the current paper dodges the barrier of [17] and handles forbidden sets  $F$  of size greater than 2 by adopting the natural approach of considering *approximate* distances instead of *exact* ones. This approach is used for many “shortest paths” problems. The most notable examples are in efficient compu-

tation of approximate “all pairs” distances [5, 25, 19, 11, 16], spanners [30, 31, 11, 16], distance oracles [39, 32, 7] and compact routing schemes [38, 31].

In the approximate setting, when no forbidden edges are considered, distance oracles were introduced by Thorup and Zwick [39]. They have shown that it is possible to preprocess a weighted undirected graph  $G(V, E)$  into a data structure of size  $O(n^{1+1/k})$  that is capable of answering distance queries in  $O(k)$  time, where the *stretch* (multiplicative approximation factor) of the returned distances is at most  $2k - 1$ . Thus, by considering approximate distances instead of exact ones, our  $f$ -sensitivity distance oracle not only solves a more general sensitivity problem but also does it with considerably lower space requirements. In fact, for constant values of  $k$  that are significantly larger than the fault parameter  $f$ , the stretch of our  $f$ -sensitivity distance oracle is the same as in the distance oracles of [39], while its size remains comparable to that of [39].

Very recently, and independently of our work, Khanna and Baswana [26] presented an approximate distance oracle construction for unweighted graphs with a single vertex failure. More precisely, they have shown how to construct a data structure of size  $O(kn^{1+1/k}/\epsilon^4)$  that answers an approximate distance query in time  $O(k)$  and stretch  $(2k - 1)(1 + \epsilon)$  under a single vertex failure. They have also shown how to find the corresponding approximate shortest path in optimal time. We stress that in this work we consider multiple *edge* faults. Our proof techniques differ significantly from those of [26]. The case of distance oracles that support multiple vertex faults remains open. In [10] we have presented a fault tolerant spanner that supports also vertex failures.

The  $f$ -sensitivity distance oracle is closely related to the fundamental problem of dynamic maintenance of all pairs of shortest paths with worst case update time. Demetrescu and Italiano [14], in a major breakthrough, obtained an algorithm with  $\tilde{O}(n^2)$  amortized update time and  $O(1)$  query time. Their algorithm was slightly improved by Thorup [36]. In the restricted case of unweighted undirected graphs, Roditty and Zwick [34] showed that for any fixed  $\epsilon, \delta > 0$  and every  $t \leq m^{1/2-\delta}$ , there exists a fully dynamic algorithm with an expected amortized update time of  $\tilde{O}(mn/t)$  and worst-case query time of  $O(t)$ . The stretch of the returned distances is at most  $1 + \epsilon$ . Thorup [37] presented the only non-trivial algorithm with a worst case update time. The cost of each update is  $O(n^{2.75})$ .

A large gap between the worst case and amortized update times exists also for the problem of dynamic connectivity of undirected graphs, where the best worst case update time is  $O(\sqrt{n})$  [20] and the best amortized update time is  $O(\log^2 n)$  [21]. Pătrașcu and Thorup [28] considered a restricted model in which all the deleted edges are first deleted in a batch, and queries are answered next. They present a linear size data structure constructible in polynomial time that supports queries in  $O(f \log^2 n \log \log n)$  worst-case time after a batch of  $f$  deletions. Similarly, our result implies that one can preprocess an undirected graph into a data structure that supports a batch of  $f$  deletions followed by a distance query in  $O(f \cdot \log^2 n \cdot \log \log n \cdot \log \log d)$  time.

The design of compact routing schemes has also been studied extensively, focusing on the tradeoffs between the size of the routing tables and the stretch

of the resulting routes. For a general overview of this area see [23][29]. Following a sequence of improvements [31][24][13][18], the best currently known tradeoffs are due to Thorup and Zwick [33], who present a general routing scheme that uses  $\tilde{O}(n^{1/k})$  space at each vertex with a stretch factor of  $2k - 1$  (using handshaking). Corresponding lower bounds were established in [31][21][22][39].

Fault-tolerant label-based distance oracles and routing schemes for graphs of bounded clique-width are presented in [12]. For an  $n$ -vertex graph of tree-width or clique-width  $k$ , the constructed labels are of size  $O(k^2 \log^2 n)$ . We are unaware of previous results in the literature concerning fault tolerant compact routing schemes for general graphs.

*Proof techniques:* Our results on both  $f$ -sensitivity distance oracles and  $f$ -sensitivity routing schemes are based at their core on a well known tree cover paradigm used implicitly in [4] and further developed in [3][11][33] (see [29]). In this paradigm, given an undirected graph  $G$  one constructs a succinct collection of trees that cover the graph  $G$  multiple times, once for every different *scale* of distances. The resulting collection of trees has several properties. Primarily, their union acts as a spanner, and thus preserves distances of the original graph up to a multiplicative factor. More important for our constructions, the collection of trees preserves local neighborhoods in an approximate manner as well. Namely, for every vertex  $v$  and every distance  $\rho$  there is a tree  $T$  in the tree cover that includes the entire  $\rho$ -neighborhood  $B_\rho(v)$  of  $v$ , i.e., all vertices of distance  $\rho$  from  $v$ . Moreover, the path in  $T$  between  $v$  and any neighbor  $u$  in  $B_\rho(v)$  is of length proportional to  $\rho$ . This last property lends itself naturally to our setting.

For distance oracles, given two vertices  $s$  and  $t$ , one needs to find the smallest scale factor  $\rho$  such that both  $s$  and  $t$  are in the tree including  $B_\rho(s)$ . This is done by constructing a *connectivity oracle* for each tree  $T$ , which enables to answer whether  $s$  and  $t$  are indeed connected in  $T$ . For routing schemes, for small values of  $\rho$  and upwards, the scheme attempts to route from  $s$  to  $t$  in the tree containing  $B_\rho(s)$ ; eventually once  $\rho$  is approximately the distance between  $s$  and  $t$ , the routing will succeed. The challenge addressed in this paper is to adapt these ideas to the  $f$ -sensitivity setting.

Our construction of  $f$ -sensitivity distance oracles enhances the tree cover paradigm in two respects. First, in order to answer queries that involves forbidden edges, one must preprocess each tree into an appropriate connectivity oracle, namely, one that takes forbidden edges into account. To this end, we use a slight variation to the  $f$ -sensitivity connectivity oracle proposed recently by Pătrașcu and Thorup [28]. However, this alone does not suffice, as the oracle of [28] requires space which is linear in the number of edges in the graph, whereas we are interested in a data structure that is as small as possible. To reduce the size of the oracle of [28] we use the notion of *connectivity preserver* as will be explained later. Combining these elements with a few additional new ideas leads to the new data structure proposed here.

We now turn to provide a high-level description of our construction of  $f$ -sensitivity routing schemes. The challenge at this point lies in identifying additional suitable information to be stored at the vertices of the tree cover that will

allow successful routing between a given vertex  $s$  and its destination  $t$  even if a forbidden edge is encountered. The case of a single forbidden edge is relatively simple. Each edge  $e$  in each tree  $T$  of the tree cover, if declared as forbidden, disconnects the tree into two connected components. Hence to route from one component to the other, all we need to do is store at the endpoints of  $e$  information concerning an alternate *recovery* edge (if such exists) that connects between the components. However, it is not hard to verify that this will not suffice once two or more edges may be forbidden. For example, an edge acting as backup for the first forbidden edge may indeed be itself forbidden. A naive solution to this point would involve storing for each edge in  $T$  several backup edges, one for any other edge in  $T$ . However, this will increase our storage significantly.

Very roughly speaking, to overcome this we associate with each edge of  $T$  a constant number of backup edges that are chosen carefully to satisfy certain conditions. Then, via case analysis, we show that these backup edges allow successful routing in the presence of two edge faults (that are not known in advance). The properties required of the selected backup edges are dictated by the structure of our case analysis. Our current techniques do not seem to extend to the general case of routing in the presence of  $f$  faults for  $f \geq 3$ .

Due to space limitations, some of our proofs are omitted.

## 2 $f$ -Sensitivity Distance Oracle

Let  $G(V, E)$  be an undirected graph with edge weights  $\omega$ . We assume throughout that  $\omega(e) \in [1, W]$  for every edge  $e$ . For an edge set  $F$  and two vertices  $s, t \in V$ , let  $\text{dist}(s, t, G \setminus F)$  be the distance between  $s$  and  $t$  in  $G \setminus F (= G(V, E \setminus F))$ . In this section we describe the  $f$ -sensitivity distance oracle and prove Theorem 1.

*Construction overview:* Our construction is based on a novel combination of three ingredients. The first ingredient is a *tree cover* for the given graph  $G$ . The tree cover we use is the skeletal representation of undirected graphs presented in [29, 3, 11]. The second ingredient is the *connectivity oracle* recently presented in [28]. Finally, the third is a simple construction of a sparse subgraph that preserves connectivity with up to  $f$  failures. We start by describing the ingredients that we use. We then present our data structure, which is constructed using a suitable combination of the above three ingredients.

*Tree covers:* Let  $G(V, E)$  be an undirected graph with edge weights  $\omega$ , and let  $\rho, k$  be two integers. Let  $B_\rho(v) = \{u \in V \mid \text{dist}(u, v, G) \leq \rho\}$  be the ball of vertices of distance  $\rho$  from  $v$ . A tree cover  $\mathbf{TC}(G, \omega, \rho, k)$  is a collection of rooted trees  $\mathcal{T} = \{T_1, \dots, T_\ell\}$  in  $G$ , with root  $r(T)$  for every  $v \in T$ , such that:

- (i) For every  $v \in V$  there exists a tree  $T \in \mathcal{T}$  such that  $B_\rho(v) \subseteq T$ .
- (ii) For every  $T \in \mathcal{T}$  and every  $v \in T$ ,  $\text{dist}(v, r(T), T) \leq (2k - 1) \cdot \rho$ .
- (iii) For every  $v \in V$ , the number of trees in  $\mathcal{T}$  that contain  $v$  is  $O(k \cdot n^{1/k})$ .

**Proposition 1** ([3, 11, 29]). *For any  $\rho$  and  $k$ , one can construct  $\mathbf{TC}(\rho, k)$  in time  $\tilde{O}(mn^{1/k})$ .*

*Connectivity oracles:* Our second primitive is **Conn.Or**( $G, \omega$ ), a connectivity oracle that given a set  $F \subset E$  of forbidden edges and a pair of nodes  $s$  and  $t$  can

answer efficiently whether  $s$  and  $t$  are connected in  $G \setminus F$ . The properties of the connectivity oracle of [28] are summarized in the following proposition. We use a slight variation of that construction, discussed after the proposition.

**Proposition 2 ([28]).** *There exists a polynomial time constructible data structure  $\text{Conn\_Or}(G, \omega)$  of size  $O(m)$ , that given a set of forbidden edges  $F \subset E$  of size  $f$  and two vertices  $s, t \in V$ , replies in time  $O(f \log^2 n \log \log n)$  whether  $s$  and  $t$  are connected in  $G \setminus F$ .*

Using the data structure presented in [28] as is allows us to answer only a single query. That is, in the process of answering a query  $(s, t, F)$  the connectivity data structure undergoes certain changes, which prevent us from using it to answer a new query  $(s', t', F')$ , asking whether  $s'$  and  $t'$  are connected in  $G \setminus F'$ . However, this is a simple technical limitation, caused by the change of the connectivity data structure, and it can be overcome by employing a rollback mechanism that after each query  $(s, t, F)$  *rewinds* the changes made to the connectivity data structure until it returns to its original form. This rewinding operation will take time proportional to the query time of the data structure on  $(s, t, F)$ , and does not effect the original query time stated in [28]. It is now possible to query the structure again using a different set of forbidden edges.

*Fault tolerant connectivity preserver:* Notice that the size of the data structure **Conn\\_Or** is  $O(m)$ . This is necessary in [28] as the size of the forbidden edge set is not known in advance. However, this is not the case in the sensitivity problem, where the size of the forbidden edge set is known in advance. Hence we would like to get a data structure whose size is independent of the number of edges in the graph. To this end, we need to use a sparse representation of the graph  $G$ , that has the same connectivity as  $G$  itself for any set of  $f$  forbidden edges. This is exactly what our last ingredient is used for. We use an edge fault tolerant connectivity preserver  $H = \text{Conn\_Pres}(G, \omega, f)$ , i.e., a subgraph of  $G(V, E)$  such that  $s$  and  $t$  are connected in  $H \setminus F$  iff they are connected in  $G \setminus F$  for every two vertices  $s, t \in V$  and any subset  $F \subseteq E$  of size at most  $f$ . Our fault tolerant connectivity preserver has the following properties.

**Proposition 3.** *Let  $G(V, E)$  be an undirected graph. There exists a subgraph  $H = \text{Conn\_Pres}(G, \omega, f)$  of  $G$  of size  $O(fn)$  such that for every subset  $F \subseteq E$ ,  $|F| \leq f$  and every two vertices  $s, t \in V$ ,  $s$  and  $t$  are connected in  $H \setminus F$  iff they are connected in  $G \setminus F$ . The subgraph  $H$  can be built in time  $O(fm)$ .*

It was shown in [35][27] how to construct fault-tolerant connectivity preservers with the desired properties. A closely related problem is the  $k$ -edge witness problem studied in [40]. The  $k$ -edge witness problem is to preprocess a given graph  $G$  so that given a set of  $k$  edges  $S$  and two nodes  $u$  and  $v$ , it possible to answer in a short time whether  $S$  is a separator of  $u$  and  $v$  in  $G$ . Roughly speaking, a fault-tolerant connectivity preserver can be constructed by iteratively identifying a spanning forest for  $G$ , adding its edges to  $H$ , and then removing its edges from  $G$ .

As the algorithm collects  $f + 1$  spanning forests, each of at most  $n - 1$  edges, the total number of edges in the resulting subgraph is  $O(fn)$ . We also show:

**Lemma 1.** *For every subset  $F \subseteq E$ , where  $|F| \leq f$ , and every two nodes  $s, t \in V$ , if  $s$  and  $t$  are connected in  $G' = (V, E \setminus F)$  then they are also connected in the subgraph  $H' = (V, E_{PR} \setminus F)$ .*

*Our construction:* We now describe our construction of  $\text{Sens-Or}(G, \omega, f, \mathcal{K})$ , where  $\mathcal{K} = (8k - 2)(f + 1)$  for integers  $k$  and  $f$ .

Our construction involves  $\log(nW)$  iterations, where  $W$  is the weight of the heaviest edge in  $G$  (hence the diameter of  $G$  is at most  $nW$ ). Each iteration deals with a certain *scale* of distances in the graph  $G$ . More specifically, iteration  $i$  addresses distances that are at most  $2^i$  in  $G$ . Each iteration builds a set of connectivity oracles. Each such oracle will allow to answer connectivity queries on a certain subgraph of  $G$ . As we will see shortly, the subgraph for each oracle is specified in two stages, the first defines the vertex set, and the second defines the edge set. We now present our construction for iteration  $i$ .

Let  $H_i$  be the set of *heavy* edges in  $G$  (of weight  $\omega(e) > 2^i$ ). Let  $G_i$  be  $G \setminus H_i$ . It is easy to see that any two vertices that are connected in  $G$  by a shortest path of length at most  $2^i$  are still connected in  $G_i$  by the same path. For reasons that will become clear shortly, we use the graphs  $G_i$  as a base for our construction in iteration  $i$ . We start by defining the vertex set of our connectivity oracles. Namely, let  $\mathbf{TC}_i = \mathbf{TC}(G_i, \omega, 2^i, k)$ . For each tree  $T \in \mathbf{TC}_i$  we build a connectivity oracle on the vertices  $V(T)$  of  $T$ . This completes our first stage.

For the second stage, define the edges to be considered in the connectivity oracle corresponding to  $T \in \mathbf{TC}_i$ . Let  $G_i|_T$  be the subgraph of  $G_i$  induced on the vertices of  $T$ . Constructing the connectivity oracle on the edges of  $G_i|_T$  actually suffices for our construction. However, as the connectivity oracle uses space that is linear in the number of edges, it is too costly to use it directly on  $G_i|_T$ . Thus to save space, we consider a sparse representation of  $G_i|_T$  that still satisfies our needs. This sparse representation is exactly the fault tolerant connectivity preserver discussed above. Namely, let  $\mathbf{Conn-Pres}_T = \mathbf{Conn-Pres}(G_i|_T, \omega, f)$  be a fault tolerant connectivity preserver for  $G_i|_T$ . The subgraph  $\mathbf{Conn-Pres}_T$  is what we use for the connectivity oracle that corresponds to  $T$ .

Our data structure includes a connectivity oracle  $\mathbf{Conn-Or}(\mathbf{Conn-Pres}_T, \omega)$ , or simply  $\mathbf{Conn-Or}_T$ , for each  $T \in \mathbf{TC}_i$ . In addition, for each  $v \in V$  we compute and store a pointer to the tree  $T_i(v) \in \mathbf{TC}_i$  containing  $B_{2^i}(v)$ . This completes our construction for iteration  $i$ .

**Lemma 2.** *The structure  $\text{Sens-Or}(G, \omega, f, \mathcal{K})$  is of size  $O(fkn^{1+1/k} \log(nW))$ .*

*Answering queries:* Given a query  $(s, t, F)$  to our data structure, the oracle operates as follows. For each  $i$  from 1 to  $\log(nW)$ , it checks if  $s$  is connected to  $t$  in the induced graph  $G_i|_{T_i(s)}$  after the set  $F$  of forbidden edges is excluded from it. This is done by querying the connectivity oracle  $\mathbf{Conn-Or}_{T_i(s)}$  of  $T_i(s)$  with  $(s, t, F)$ . If  $s$  and  $t$  are connected, the oracle returns the value  $\mathcal{K} \cdot 2^{i-1}$ , otherwise it proceeds to the next  $i$  value. If no such  $i$  exists, it returns  $\infty$ .

Theorem 1 now follows by the lemmas below.

**Lemma 3 (Correctness).** *The  $f$ -sensitivity distance query algorithm returns an estimate that is within a multiplicative factor of  $\mathcal{K}$  from  $\text{dist}(s, t, G \setminus F)$ .*

**Lemma 4.** *The  $f$ -sensitivity distance query  $(s, t, F)$  can be implemented to return a distance estimate in time  $O(|F| \cdot \log^2 n \cdot \log \log n \cdot \log \log d)$ , where  $d$  is the distance between  $s$  and  $t$  in  $G \setminus F$ .*

### 3 2-Sensitive Compact Routing Schemes

In this section we present an  $f$ -sensitive routing scheme for the case of two forbidden edges (i.e.,  $f = 2$ ) and prove Theorem 2. Let  $s, t \in V$  and assume that a message is to be routed from  $s$  to  $t$ . Loosely speaking, the routing process we suggest is similar in nature to the  $f$ -sensitivity query process described in Section 2. That is, our routing scheme involves at most  $\lceil \log(nW) \rceil$  iterations. In each iteration  $i$ , an attempt is made to route the message from  $s$  to  $t$  in the graph  $G_i|_{T_i(t)} \setminus F$  using the tree  $T_i(t)$  augmented with some additional information to be specified below. (Note that in each iteration  $i$ , the routing attempt is made on the tree  $T_i(t)$  instead of  $T_i(s)$ ; the reason for this will be made clearer later on.) If the routing is unsuccessful, the scheme proceeds to the next iteration. The routing process ends either when the message reaches its destination or after a failure in the final iteration.

Let  $P$  be a shortest path between  $s$  and  $t$  in  $G \setminus F$ , and let  $i = \lceil \log |P| \rceil$ . As argued before,  $P$  is included in  $G_i|_{T_i(t)} \setminus F$ . To prove that our routing scheme succeeds, it suffices to prove that it finds a path of length proportional to  $|P|$  when the routing is done on the augmented tree  $T_i(t)$ . Throughout this section, any standard tree routing operation is preformed by using the tree routing scheme of Thorup and Zwick [38]. That scheme uses  $(1 + o(1)) \log_2 n$ -bit label for each node. These labels are the only information required for their routing scheme and no other data is stored. In addition, the routing decision at each node takes only constant time.

Note that the routing scheme of [38] may assign a node  $t$  different label  $L_T(t)$  for each tree  $T \in TC_i$  it belongs to. In order to enable a node  $s$  to route a message to a node  $t$  over some tree  $T$ , it should be familiar with the label  $L_T(t)$ . Naively, for each node  $t$  we could concatenate all labels assigned to  $t$  in all trees  $T \in TC_i$  it participates in, and use the concatenated string as the new label of  $t$ . However, this could lead to prohibitively large labels. Therefore, for each node  $t$ , we concatenate only the labels given to  $t$  for the trees  $T_i(t)$  for  $1 \leq i \leq \log(nW)$  (and some indication on which tree is  $T_i(t)$ ). Therefore, the attempts to route from  $s$  to  $t$  are made over the trees  $T_i(t)$  instead of  $T_i(s)$ . The size of each node label is  $O(\log(nW) \cdot \log n)$ . In addition, for every tree  $T$  and every node  $v \in T$ ,  $v$  stores the original label  $L_T(v)$ . Each such label is of size  $O(\log n)$ , therefore, we get an additional  $O(\log n)$  factor on the amount of information stored in the vertices. Now consider iteration  $i$  where the node  $s$  tries to route a message to  $t$  over  $T_i(t)$ . Then  $s$  first checks if it belongs to  $T_i(t)$ ; if not, then it proceeds to the next iteration, and so on.

We now turn to describe our 2-sensitivity routing scheme. Let  $T \in \mathbf{TC}_i$ , where  $i \in \{1, \dots, \lceil \log nW \rceil\}$ . Each edge  $e = (u, v) \in T$ , if declared as forbidden, disconnects the tree into two connected components. Let  $T_u(e)$  (respectively,  $T_v(e)$ ) be the component that contains  $u$  (resp.,  $v$ ). As the route may need to cross from one component to the other, our data structure needs to store at each such edge  $e$  some additional information that will allow this task. Specifically, a *recovery edge* of  $e$  is any edge  $e'$  of  $G$  that connects  $T_u(e)$  and  $T_v(e)$ . We define for each edge  $e$  in  $T$  a recovery edge  $rec(e)$ . For the sake of the analysis, and to slightly simplify the routing phase, assume that the edges of the graph are sorted in some order  $\langle e_1, \dots, e_m \rangle$ , and for every edge  $e$ ,  $rec(e)$  is chosen to be a recovery edge  $e_i$  of  $e$  such that  $i$  is minimal. We say that  $e_i < e_j$  when  $i < j$ .

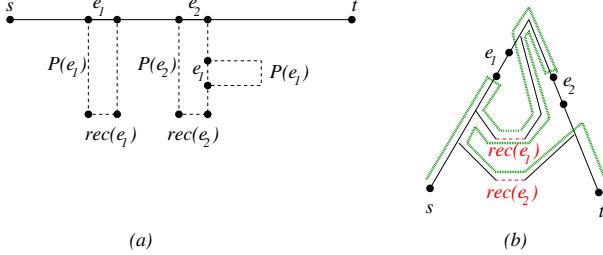
In order to handle two failures, we need to store additional information (i.e., additional recovery edges) in the routing tables of vertices in  $T$ . We show that the total number of recovery edges needed is  $O(|T|)$ .

Consider the recovery edge  $rec(e) = (u', v')$  of the edge  $e$ . The edge  $rec(e)$  connects the subtrees  $T_u(e)$  and  $T_v(e)$  where  $u' \in T_u(e)$  and  $v' \in T_v(e)$ . Denote by  $P(u, u')$  (respectively,  $P(v, v')$ ) the path connecting  $u$  and  $u'$  (respectively,  $v$  and  $v'$ ) in the tree  $T_u(e)$  (respectively,  $T_v(e)$ ), and denote the entire alternative path for  $e = (u, v)$  by  $P(e) = P(u, u') \cdot (u', v') \cdot P(v', v)$ .

Throughout this section, assume the two failed edges are  $e_1 = (u_1, v_1)$  and  $e_2 = (u_2, v_2)$ . Clearly, if both  $e_1$  and  $e_2$  are not in  $T$  then we can just route on  $T$ . Hence, we only have to consider the case when  $T$  contains the failed edges.

We first consider the case that only one of the failed edges is in  $T$ . Assume, w.l.o.g., that  $e_1 \in T$  and that  $e_2 \notin T$ . Notice that  $T \cup \{rec(e_1)\} \setminus \{e_1, e_2\}$  is composed of two connected components only when  $rec(e_1) = e_2$ . To overcome this it suffices to store for each edge  $e \in T$  an additional recovery edge. Then, in the scenario described above, where  $rec(e_1) = e_2$  and  $T \cup \{rec(e_1)\} \setminus \{e_1, e_2\}$  is not connected, we simply use the additional recovery edge of  $e_1$  and we are guaranteed not to encounter additional faulty edges along the rest of the route. Note that if there is only one edge that can serve as a recovery edge for  $e_1$  and this edge is faulty, then it is not possible to route from  $s$  to  $t$  on  $G_i|_T \setminus \{e_1, e_2\}$ . To summarize this case, for each edge  $e$  we store two recovery edges (if exist).

We now consider the case that both  $e_1, e_2 \in T$ . Let  $rec(e_1) = (u'_1, v'_1)$  and  $rec(e_2) = (u'_2, v'_2)$ . If the edge  $e_2$  is not on the alternative path  $P(e_1) = P(u_1, u'_1) \cdot P(u'_1, v'_1) \cdot P(v'_1, v_1)$  of  $e_1$  and  $s$  and  $t$  are connected in  $G_i|_T \setminus \{e_1, e_2\}$  then  $rec(e_1)$  and  $rec(e_2)$  suffice to route from  $s$  to  $t$ . The reason is that it is always possible to bypass  $e_1$  using its alternative path  $P(e_1)$ . Therefore, the routing from  $s$  to  $t$  never gets stuck when reaching  $e_1$ . When the edge  $e_2$  is encountered on the routing path it is bypassed using  $P(e_2) = P(u_2, u'_2) \cdot P(u'_2, v'_2) \cdot P(v'_2, v_2)$ . If the edge  $e_1$  is on  $P(e_2)$ , it is bypassed (again) using  $P(e_1)$ , which does not contain  $e_2$ . This situation is depicted in Figure 11. The situation that  $e_1$  is not on  $P(e_2)$  is symmetric. Therefore, it is only left to consider the situation in which both  $e_1$  is in  $P(e_2)$  and  $e_2$  is in  $P(e_1)$ .



**Fig. 1.** (a) A schematic description of an  $s - t$  route where the failed edge  $e_1$  is encountered twice. (b) The resulting route on the tree. Note that the alternate path  $P(e_1)$  does not always have to be followed blindly; rather, it can be "shortcut" whenever the necessary information is readily available.

This implies that  $rec(e_1) = rec(e_2)$ . To see this, notice that since  $P(e_2)$  contains  $e_1$  the recovery edge  $rec(e_2)$  is also a recovery edge for  $e_1$ , and similarly  $rec(e_1)$  is also a recovery edge for  $e_2$ . Since we have chosen the recovery edges to be minimal with respect to a given ordering, it must be the case that  $rec(e_1) = rec(e_2)$ . Now since  $e_1$  is in  $P(e_2)$ ,  $e_2$  is in  $P(e_1)$  and  $rec(e_1) = rec(e_2)$  it must be that  $P(e_1) \cup \{e_1\} = P(e_2) \cup \{e_2\}$ . To deal with this case, we store for  $e_1$  (and similarly, for each edge  $e \in T$ ) two additional recovery edges  $rec_{u_1}(e_1)$  and  $rec_{v_1}(e_1)$ . The purpose of  $rec_{u_1}(e_1)$  ( $rec_{v_1}(e_1)$ ) is to handle an edge fault on  $P(u_1, u'_1)$  ( $P(v_1, v'_1)$ ). We choose  $rec_{u_1}(e_1)$  such that it will allow to bypass as many edges on  $P(u_1, u'_1)$  as possible. More specifically, consider the path from  $u_1$  to any other recovery edge that differs from  $rec(e_1)$ . This path has some common prefix with  $P(u_1, u'_1)$  (which is possibly empty). For  $rec_{u_1}(e_1)$ , we choose the recovery edge of  $e_1$  that minimizes the length of this common prefix, that is, if  $rec_{u_1}(e_1) = (\hat{u}, \hat{v})$  then the common prefix of  $P(u_1, u'_1)$  and  $P(u_1, \hat{u})$  is the minimal possible. The recovery edge  $rec_{v_1}(e_1)$  is defined analogously with respect to  $P(v_1, v'_1)$ .

We now describe how it is possible to route from  $s$  to  $t$  when  $e_1$  is in  $P(e_2)$ ,  $e_2$  is in  $P(e_1)$  and  $rec(e_1) = rec(e_2) = e' = (u', v')$  using the additional information. The message first encounters the edge  $e_1 = (u_1, v_1)$  on its route. If the recovery edge  $rec(e_1)$  does not exist than  $t$  cannot be reached using  $T$ . If  $rec(e_1) = (u', v')$  exists then the message is routed towards  $u'$  on  $P(u_1, u')$ . It uses  $(u', v')$  and continues to route from  $v'$  toward  $t$  on  $P(v', v_1)$ . Notice that at some stage along the path  $P(u_1, u') \cdot (u', v') \cdot P(v', v_1)$ , the edge  $e_2$  is encountered. Since  $rec(e_1) = rec(e_2) = e'$  it is not possible to bypass  $e_2$  using  $rec(e_2)$ . There are two possible cases to consider here. The first is when the edge  $e_2$  is on the path  $P(u_1, u')$  and the second is when the edge  $e_2$  is on the path  $P(v_1, v')$ . Notice that it is possible to distinguish between the two cases when  $e_2$  is encountered simply by checking whether the edge  $rec(e_1)$  was already traversed.

Consider the first case, where  $e_2$  is on  $P(u_1, u')$  and assume that  $v_2$  is the endpoint of  $e_2$  that is connected to  $u_1$ . There are three subtrees in  $T \setminus \{e_1, e_2\}$ . Let  $T_1$  be the subtree containing  $u_1$  and  $v_2$ . Let  $T_2$  be the subtree containing  $u_2$

and  $u'$  and let  $T_3$  be the subtree containing  $v'$  and  $v_1$ . Note that if  $t \in T$ , then it must be that  $t \in T_3$ , as the routing scheme on  $T$  tries to send the message from  $s$  to  $t$  using  $e_1 = (u_1, v_1)$  which implies that  $t$  is not in the subtree  $T_1 \cup T_2 \cup \{e_2\}$  that contains  $u_1$ . Moreover, as we assume that we first encounter  $e_1$  on the path from  $s$  to  $t$  in  $T$ , it holds that  $s$  is in  $T_1$ .

We first try to use the edge  $rec_{u_1}(e_1)$ . Recall that this edge was chosen such that the path leading to it from  $u_1$  has the minimal possible common prefix with  $P(u_1, u')$ . Therefore, if there is a recovery edge  $\tilde{r} = (\tilde{u}, \tilde{v})$  with endpoint  $\tilde{u}$  in  $T_1$  and  $\tilde{v}$  in  $T_3$ , then clearly  $rec_{u_1}(e_1) = (u'', v'')$  must be such an edge. To see this, assume towards contradiction, that the path  $P(u_1, u'')$  contains the edge  $e_2$ . Note that the path  $P(u_1, \tilde{u})$  contains fewer edges in common with  $P(u_1, u')$  than the path  $P(u_1, u'')$ , in contradiction to the minimality of  $P(u_1, u'')$ . Therefore, if the subtree  $T_1$  contains an edge leading to  $T_3$ , using the edge  $rec_{u_1}(e_1)$  we can reach to the subtree  $T_3$  containing  $t$ .

The more involved subcase is when for every recovery edge  $\tilde{r} = (\tilde{u}, \tilde{v})$  of  $e_1$ , the path  $P(u_1, \tilde{u})$  contains the edge  $e_2$ , or in other words, there is no edge connecting the subtree  $T_1$  with the subtree  $T_3$  (except the faulty edge  $e_1$ ). In this case, our only “chance” to reach  $t$  on the tree  $T$  is by passing through the trees  $T_1$ ,  $T_2$  and finally  $T_3$  in this order. Notice that to connect between  $T_2$  and  $T_3$  we can use the edge  $rec(e_1)$ . Using ideas similar to those presented above, we show that it is possible to reach  $T_2$  from  $T_1$ , and thus the additional information that we have saved will allow us to reach  $t$ . The details, as well as the analysis of the second case, in which the edge  $e_2$  is on the path  $P(v_1, v')$ , are omitted.

**Lemma 5.** *The resulting routing scheme has maximum stretch  $O(k)$ .*

All in all, for each edge  $e = (u, v) \in T$ , both endpoints  $u$  and  $v$  store three additional edges,  $rec(e)$ ,  $rec_u(e)$  and  $rec_v(e)$ . Theorem 2 follows.

**Acknowledgement.** We are grateful to Seth Pettie for his helpful comments.

## References

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.* 28(4), 1167–1181 (1999)
2. Awerbuch, B., Bar-Noy, A., Linial, N., Peleg, D.: Improved routing strategies with succinct tables. *J. Algorithms*, 307–341 (1990)
3. Awerbuch, B., Kutten, S., Peleg, D.: On buffer-economical store-and-forward deadlock prevention. In: *INFOCOM*, pp. 410–414 (1991)
4. Awerbuch, B., Peleg, D.: Sparse partitions. In: *31st FOCS*, pp. 503–513 (1990)
5. Baswana, S., Kavitha, T.: Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In: *FOCS*, pp. 591–602 (2006)
6. Baswana, S., Sen, S.: A simple linear time algorithm for computing sparse spanners in weighted graphs. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) *ICALP 2003. LNCS*, vol. 2719, pp. 384–396. Springer, Heidelberg (2003)
7. Baswana, S., Sen, S.: Approximate distance oracles for unweighted graphs in expected  $O(n^2)$  time. *ACM Trans. Algorithms* 2(4), 557–577 (2006)

8. Bernstein, A., Karger, D.: Improved distance sensitivity oracles via random sampling. In: 19th SODA, pp. 34–43 (2008)
9. Bernstein, A., Karger, D.: A nearly optimal oracle for avoiding failed vertices and edges. In: 41st STOC, pp. 101–110 (2009)
10. Chechik, S., Langberg, M., Peleg, D., Roditty, L.: Fault-tolerant spanners for general graphs. In: 41st STOC, pp. 435–444 (2009)
11. Cohen, E.: Fast algorithms for constructing  $t$ -spanners and paths with stretch  $t$ . In: FOCS, pp. 648–658 (1993)
12. Courcelle, B., Twigg, A.: Compact forbidden-set routing. In: STACS, pp. 37–48 (2007)
13. Cowen, L.J.: Compact routing with minimum stretch. J. Alg. 38, 170–183 (2001)
14. Demetrescu, C., Italiano, G.F.: Experimental analysis of dynamic all pairs shortest path algorithms. In: 15th SODA 2004, pp. 362–371 (2004)
15. Demetrescu, C., Thorup, M., Chowdhury, R., Ramachandran, V.: Oracles for distances avoiding a failed node or link. SIAM J. Comput. 37(5), 1299–1318 (2008)
16. Dor, D., Halperin, S., Zwick, U.: All-pairs almost shortest paths. SIAM J. Comput. 29(5), 1740–1759 (2000)
17. Duan, R., Pettie, S.: Dual-failure distance and connectivity oracles. In: SODA (2009)
18. Eilam, T., Gavoille, C., Peleg, D.: Compact routing schemes with low stretch factor. J. Algorithms 46, 97–114 (2003)
19. Elkin, M.: Computing almost shortest paths. ACM Trans. Alg. 1, 283–323 (2005)
20. Eppstein, D., Galil, Z., Italiano, G.F., Nissenzweig, N.: Sparsification – A technique for speeding up dynamic graph algorithms. J. ACM 44 (1997)
21. Fraigniaud, P., Gavoille, C.: Memory requirement for universal routing schemes. In: 14th PODC, pp. 223–230 (1995)
22. Gavoille, C., Gengler, M.: Space-efficiency for routing schemes of stretch factor three. J. Parallel Distrib. Comput. 61, 679–687 (2001)
23. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distributed Computing 16, 111–120 (2003)
24. 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)
25. Kavitha, T.: Faster algorithms for all-pairs small stretch distances in weighted graphs. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 328–339. Springer, Heidelberg (2007)
26. Khanna, N., Baswana, S.: Approximate shortest paths oracle handling single vertex failure. In: STACS (2010)
27. Nagamochi, H., Ibaraki, T.: Linear time algorithms for finding a sparse  $k$ -connected spanning subgraph of a  $k$ -connected graph. Algorithmica 7, 583–596 (1992)
28. Pătrașcu, M., Thorup, M.: Planning for fast connectivity updates. In: 48th FOCS, pp. 263–271 (2007)
29. Peleg, D.: Distributed computing: a locality-sensitive approach. In: SIAM (2000)
30. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM J. Computing 18(4), 740–747 (1989)
31. Peleg, D., Upfal, E.: A trade-off between space and efficiency for routing tables. J. ACM 36(3), 510–530 (1989)
32. Roditty, L., Thorup, M., Zwick, U.: Deterministic constructions of approximate distance oracles and spanners. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 261–272. Springer, Heidelberg (2005)

33. Roditty, L., Thorup, M., Zwick, U.: Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms* 4(3), 1–17 (2008)
34. Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: 36th STOC, pp. 184–191 (2004)
35. Thurimella, R.: Techniques for the design of parallel graph algorithms. Ph.D. Thesis. Univ. Texas, Austin (1989)
36. Thorup, M.: Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 384–396. Springer, Heidelberg (2004)
37. Thorup, M.: Worst-case update times for fully-dynamic all-pairs shortest paths. In: 37th STOC, pp. 112–119 (2005)
38. Thorup, M., Zwick, U.: Compact routing schemes. In: SPAA, pp. 1–10 (2001)
39. Thorup, M., Zwick, U.: Approximate distance oracles. *J. ACM* 52, 1–24 (2005)
40. Twigg, A.: Compact forbidden-set routing. Ph.D. Thesis. Univ. Cambridge (2006)

# Fast Minor Testing in Planar Graphs

Isolde Adler<sup>1</sup>, Frederic Dorn<sup>2</sup>, Fedor V. Fomin<sup>2</sup>,  
Ignasi Sau<sup>3</sup>, and Dimitrios M. Thilikos<sup>4,\*</sup>

<sup>1</sup> Institut für Informatik, Goethe-Universität, Frankfurt, Germany  
[iadler@informatik.uni-frankfurt.de](mailto:iadler@informatik.uni-frankfurt.de)

<sup>2</sup> Department of Informatics, University of Bergen, Norway  
[{frederic.dorn,fedor.fomin}@ii.uib.no](mailto:{frederic.dorn,fedor.fomin}@ii.uib.no)

<sup>3</sup> Department of Computer Science, Technion, Haifa, Israel  
[ignasi@cs.technion.ac.il](mailto:ignasi@cs.technion.ac.il)

<sup>4</sup> Department of Mathematics, National and Kapodistrian  
University of Athens, Greece  
[sedthilk@math.uoa.gr](mailto:sedthilk@math.uoa.gr)

**Abstract.** Minor containment is a fundamental problem in Algorithmic Graph Theory, as numerous graph algorithms use it as a subroutine. A model of a graph  $H$  in a graph  $G$  is a set of disjoint connected subgraphs of  $G$  indexed by the vertices of  $H$ , such that if  $\{u, v\}$  is an edge of  $H$ , then there is an edge of  $G$  between components  $C_u$  and  $C_v$ . Graph  $H$  is a minor of  $G$  if  $G$  contains a model of  $H$  as a subgraph. We give an algorithm that, given a planar  $n$ -vertex graph  $G$  and an  $h$ -vertex graph  $H$ , either finds in time  $2^{\mathcal{O}(h)} \cdot n + \mathcal{O}(n^2 \cdot \log n)$  a model of  $H$  in  $G$ , or correctly concludes that  $G$  does not contain  $H$  as a minor. Our algorithm is the first *single-exponential* algorithm for this problem and improves all previous minor testing algorithms in planar graphs. Our technique is based on a novel approach called *partially embedded dynamic programming*.

**Keywords:** graph minors, planar graphs, branchwidth, parameterized complexity, dynamic programming.

## 1 Introduction

For two input graphs  $G$  and  $H$ , the MINOR CONTAINMENT problem is to decide whether  $H$  is a minor of  $G$ . This is a classical NP-complete problem [16], and remains NP-complete even when both graphs  $G$  and  $H$  are planar, as it is a generalization of the HAMILTONIAN CYCLE problem. When  $H$  is fixed, by the celebrated result of Robertson and Seymour [26], there is an algorithm to decide if  $H$  is a minor of an input graph  $G$  that runs in time  $f(h) \cdot n^3$ , where  $n$  is the number of vertices of  $G$ ,  $h$  is the number of vertices in  $H$ , and  $f$  is some recursive function. One of the significant algorithmic implications of this result is that, combined with the Graph Minor Theorem of Robertson and Seymour [28], it shows the polynomial-time solvability of many graph problems, some of which

\* Supported by the project “Kapodistrias” (ΑΠΙ 02839/ 28.07.2008) of the National and Kapodistrian University of Athens (project code: 70/ 4/8757.)

were previously not even known to be decidable [15]. However, these algorithmic results are highly *non-practical*. This triggered an ongoing quest in the Theory of Algorithms since then for making Graph Minors constructive and for making its algorithmic proofs practical for a wide range of applications (e.g., [8,20]).

Unfortunately, in the minor testing algorithm of Robertson and Seymour [26], the function  $f(h)$  has an *immense* exponential growth, which makes the algorithm absolutely impractical even for very simple patterns (see [21] for recent theoretical improvements of this function). There were several attempts to improve the running time of the algorithm of Robertson and Seymour. One direction of such improvements is decreasing the degree of the polynomial in  $n$ . For example, Reed and Li gave a linear time algorithm solving  $K_5$ -minor containment [25]. The second direction of improvements is towards reducing the exponential dependency in the function  $f(h)$ , which is a natural direction of study for Parameterized Complexity [14]. A significant step in this direction was done by Hicks [19], who provided in graphs of branchwidth  $k$  and  $m$  edges an  $\mathcal{O}(3^{k^2} \cdot (h+k-1)! \cdot m)$  time algorithm, following the algorithm sketched by Robertson and Seymour [26]. Recently, this was improved to  $\mathcal{O}(2^{(2k+1)\log k} \cdot h^{2k} \cdot 2^{2h^2} \cdot m)$  on general graphs, and in planar, and more generally, in graphs of bounded genus, to  $2^{\mathcal{O}(k)} \cdot h^{2k} \cdot 2^{\mathcal{O}(h)} \cdot n$  [1].

In this paper we focus on the case where the input graph  $G$  is planar.

#### PLANAR $H$ -MINOR CONTAINMENT

**Input:** A planar graph  $G$ .

**Objective:** Either find a model  $M$  of  $H$  in  $G$ , or conclude that  $G$  does not contain such a model.

By arguments inspired by Bidimensionality Theory [5], it can be shown that the  $2^{\mathcal{O}(k)} \cdot h^{2k} \cdot 2^{\mathcal{O}(h)} \cdot n$  time algorithm from [1], combined with the grid minor Theorem of Robertson, Thomas, and Seymour [27], can be used to solve PLANAR  $H$ -MINOR CONTAINMENT in time  $2^{\mathcal{O}(h\log h)} \cdot n + \mathcal{O}(n^2 \cdot \log n)$ . This directly sets up the challenge of designing a *single-exponential* (on the size  $h$  of the pattern  $H$ ) algorithm for this problem. Over the last four decades, many different algorithmic techniques in planar graphs were developed for different type of problems and algorithms, including approximation [4,7], exact [12,22], and parameterized algorithms [3,6,13]. However, it seems that none of these approaches can be used to speed up the algorithm for PLANAR  $H$ -MINOR CONTAINMENT.

**Our results and key ideas.** Our main result is the following theorem.

**Theorem 1.** *Given a planar graph  $G$  on  $n$  vertices and a graph  $H$  on  $h$  vertices, we can solve PLANAR  $H$ -MINOR CONTAINMENT in time  $2^{\mathcal{O}(h)} \cdot n + \mathcal{O}(n^2 \cdot \log n)$ .*

That is, we prove that when  $G$  is planar the behavior of the function  $f(h)$  can be made *single-exponential*, improving over all previous results for this problem [1, 26, 19]. In addition, we can *enumerate* and *count* the number of models within the same time bounds. Let us remark that by our theorem, PLANAR  $H$ -MINOR

CONTAINMENT is solvable in polynomial time when the size of the pattern  $H$  is  $\mathcal{O}(\log n)$ , substantially improving the existing algorithms for small patterns [10].

In order to prove Theorem 1, we introduce a novel approach of dynamic programming in planar graphs of bounded branchwidth, namely *partially embedded dynamic programming*. This approach is extremely helpful in computing graph minors but we believe that this technique can be used in many related problems including PLANAR DISJOINT PATHS. Our technique is inspired by the technique of *embedded dynamic programming* introduced in [11] for solving PLANAR SUBGRAPH ISOMORPHISM for a pattern of size  $h$  and an input graph of size  $n$  in time  $2^{\mathcal{O}(h)} \cdot n$ . There, one controls the partial solutions by the ways the separators of  $G$  can be routed through the pattern. The difference (and difficulty) concerning PLANAR  $H$ -MINOR CONTAINMENT is that we look for a model  $M$  of size  $\mathcal{O}(n)$  out of  $2^{\mathcal{O}(n)}$  possible non-isomorphic models of  $H$  in  $G$ . In partially embedded dynamic programming, we look for potential models of  $H$  in  $G$  with a magnifying glass only at a given separator  $S$  of  $G$ . That is, we consider a collection  $\mathcal{A}$  of graphs  $A$  arising from ‘inflating’ a part of  $H$ , namely the part interacting with  $S$ . Thus, each  $A$  behaves like a subgraph of  $G$  inside the intersection with  $S$ , and outside that intersection  $A$  behaves like a minor of  $G$ ; this is why we call our dynamic programming technique ‘*partially embedded*’.

After giving some preliminaries in Section 2, we first show in Section 3 how PLANAR  $H$ -MINOR CONTAINMENT can be solved in polynomial time for input graphs of large branchwidth (in comparison to the pattern size). If the branchwidth is small, we compute the collection  $\mathcal{A}$  in Section 3.1 and give the partially embedded dynamic programming approach in Section 3.2.

## 2 Preliminaries

**Graphs and graph minors.** We use standard graph terminology, see for instance [9]. All graphs considered in this article are simple and undirected. Given a graph  $G$ , we denote by  $V(G)$  and  $E(G)$  the vertex set and the edge set of  $G$ , respectively. A graph  $H$  is a *subgraph* of a graph  $G$ ,  $H \subseteq G$ , if  $V(H) \subseteq V(G)$  and  $E(H) \subseteq E(G)$ . We define graph operation *contracting* edge  $e = \{x, y\} \in G$  by removing  $e$ , including  $x$  and  $y$ , and making a new vertex  $v_e$  adjacent to the former neighbors of  $x$  and  $y$  (excluding  $x$  and  $y$ ).

Graph  $H$  is a *minor* of graph  $G$  (denoted by  $H \preceq G$ ), if  $H$  can be obtained from a subgraph of  $G$  by a (possibly empty) sequence of edge contractions. In this case we also say that  $G$  is a *major* of  $H$ . Graph  $H$  is a *contraction minor* of graph  $G$  (denoted by  $H \preceq_c G$ ), if  $H$  can be obtained from  $G$  by a (possibly empty) sequence of edge contractions.

A *model* of  $H$  in  $G$  [26] is a mapping  $\phi$ , that assigns to every edge  $e \in E(H)$  an edge  $\phi(e) \in E(G)$ , and to every vertex  $v \in V(H)$  a non-empty connected subgraph  $\phi(v) \subseteq G$ , such that

- (i) the graphs  $\{\phi(v) \mid v \in V(H)\}$  are mutually vertex-disjoint and the edges  $\{\phi(e) \mid e \in E(H)\}$  are pairwise distinct; and
- (ii) for  $e = \{u, v\} \in E(H)$ ,  $\phi(e)$  connects  $\phi(u)$  with  $\phi(v)$ .

Thus,  $H$  is isomorphic to a minor of  $G$  if and only if there exists a model of  $H$  in  $G$ .

With slight abuse of notation, the subgraph  $M \subseteq G$  defined by the union of  $\{\phi(v) \mid v \in V(H)\}$  and  $\{\phi(e) \mid e \in E(H)\}$  is also called a *model* of  $H$  in  $G$ . The edge set of a model  $M$  is partitioned into *c-edges (contraction edges)* and *m-edges (minor edges)*, which are the edges of  $\{\phi(v) \mid v \in V(H)\}$  and  $\{\phi(e) \mid e \in E(H)\}$ , respectively.

**Branchwidth.** A *branch decomposition*  $(T, \mu)$  of a graph  $G$  consists of an unrooted ternary tree  $T$  (i.e., all internal vertices are of degree three) and a bijection  $\mu : L \rightarrow E(G)$  from the set  $L$  of leaves of  $T$  to the edge set of  $G$ . We define for every edge  $e$  of  $T$  the *middle set*  $\text{mid}(e) \subseteq V(G)$  as follows: Let  $T_1$  and  $T_2$  be the two connected components of  $T \setminus \{e\}$ . Then let  $G_i$  be the graph induced by the edge set  $\{\mu(f) : f \in L \cap V(T_i)\}$  for  $i \in \{1, 2\}$ . The *middle set* is the intersection of the vertex sets of  $G_1$  and  $G_2$ , i.e.,  $\text{mid}(e) = V(G_1) \cap V(G_2)$ . The *width* of  $(T, \mu)$  is the maximum order of the middle sets over all edges of  $T$ , i.e.,  $\text{width}(T, \mu) := \max\{|\text{mid}(e)| : e \in E(T)\}$ . The *branchwidth* of  $G$  is defined as  $\text{bw}(G) := \min\{\text{width}(T, \mu) \mid (T, \mu) \text{ branch decomposition of } G\}$ . Note that for each  $e \in E(T)$ ,  $\text{mid}(e)$  is a separator of  $G$ , unless  $\text{mid}(e) = \emptyset$ .

**Remark 1.** For every two edges  $e, f \in E(T)$  with  $e \cap f \neq \emptyset$ , we have  $|\text{mid}(e) \cup \text{mid}(f)| \leq 1.5 \cdot \text{width}(T, \mu)$ .

Intuitively, a graph  $G$  has small branchwidth if  $G$  is close to being a tree. The fundamental grid minor Theorem says that, roughly, a graph has either small branchwidth, or it contains a large grid as a minor. We use the variant for planar graphs.

**Proposition 1 ([27,18]).** Given a planar graph  $G$  on  $n$  vertices with  $\text{bw}(G) \geq k$ , a model of the  $(\lfloor k/3 \rfloor \times \lfloor k/3 \rfloor)$ -grid in  $G$  can be found in time  $\mathcal{O}(n^2 \cdot \log n)$ .

On the other hand, every planar graph is minor of a large enough grid.

**Proposition 2 ([27]).** If  $H$  is a planar graph with  $|V(H)| + 2|E(H)| \leq \ell$ , then  $H$  is isomorphic to a minor of the  $(2\ell \times 2\ell)$ -grid.

**Planar graphs and equivalent drawings.** Let  $\Sigma$  be the unit sphere. A *planar drawing*, or simply *drawing*, of a graph  $G$  with vertex set  $V(G)$  and edge set  $E(G)$  maps vertices to points in the sphere, and edges to simple curves between their end-vertices, such that edges do not cross, except in common end-vertices. A *plane graph* is a graph  $G$  together with a planar drawing. A *planar graph* is a graph that admits a planar drawing. The set of *faces*  $F(G)$  of a plane graph  $G$  is defined as the union of the connected regions of  $\Sigma \setminus G$ . A subgraph of a plane graph  $G$ , induced by the vertices and edges incident to a face  $f \in F(G)$ , is called a *bound* of  $f$  (for further reading, see e.g. [9]). Consider any two drawings  $G_1$  and  $G_2$  of a planar graph  $G$ . A *homeomorphism* of  $G_1$  onto  $G_2$  is a homeomorphism of  $\Sigma$  onto itself which maps vertices, edges, and faces of  $G_1$  onto vertices, edges, and faces of  $G_2$ , respectively. We call two planar drawings *equivalent*, if there is a homeomorphism from one onto the other.

**Proposition 3** (e.g. [24]). *The number of non-equivalent drawings of a planar  $n$ -vertex graph is  $2^{\mathcal{O}(n)}$ . A set of all such drawings can be computed in time  $2^{\mathcal{O}(n)}$ .*

**Proposition 4** ([31]). *The number of non-isomorphic edge-maximal planar  $n$ -vertex graphs is  $2^{\mathcal{O}(n)}$ .*

**Nooses and combinatorial nooses.** A *noose* of a  $\Sigma$ -plane graph  $G$  is a simple closed curve in  $\Sigma$  that meets  $G$  only in vertices. From the Jordan Curve Theorem (e.g. [23]), it follows that nooses separate  $\Sigma$  into two regions.

Let  $V(N) = N \cap V(G)$  be the vertices and  $F(N)$  be the faces intersected by a noose  $N$ . The *length* of  $N$  is  $|V(N)|$ , the number of vertices in  $V(N)$ . The clockwise order in which  $N$  meets the vertices of  $V(N)$  is a cyclic permutation  $\pi$  on the set  $V(N)$ .

A *combinatorial noose*  $N_C = [v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$  in a plane graph  $G$  is an alternating sequence of vertices and faces of  $G$ , such that

- $f_i$  is a face incident to both  $v_i, v_{i+1}$  for all  $i < \ell$ ;
- $v_0 = v_\ell$  and the vertices  $v_1, \dots, v_\ell$  are mutually distinct; and
- if  $f_i = f_j$  for any  $i \neq j$  and  $i, j = 0, \dots, \ell - 1$ , then the vertices  $v_i, v_{i+1}, v_j$ , and  $v_{j+1}$  do not appear in the order  $(v_i, v_j, v_{i+1}, v_{j+1})$  on the bound of face  $f_i = f_j$ .

The *length* of a combinatorial noose  $[v_0, f_0, v_1, f_1, \dots, f_{\ell-1}, v_\ell]$  is  $\ell$ .

**Remark 2.** *The order in which a noose  $N$  intersects the faces  $F(N)$  and the vertices  $V(N)$  of a plane graph  $G$  gives a unique alternating face-vertex sequence of  $F(N) \cup V(N)$  which is a combinatorial noose  $N_C$ . Conversely, for every combinatorial noose  $N_C$  there exists a noose  $N$  with face-vertex sequence  $N_C$ .*

We will refer to combinatorial nooses simply as nooses if it is clear from the context.

**Proposition 5** ([11]). *Every plane graph on  $n$  vertices has  $2^{\mathcal{O}(n)}$  combinatorial nooses.*

**Sphere cut decompositions.** For a plane graph  $G$ , we define a *sphere cut decomposition* (or *sc-decomposition*)  $\langle T, \mu, \pi \rangle$  as a branch decomposition, which for every edge  $e$  of  $T$  has a noose  $N_e$  that divides  $\Sigma$  into two regions  $\Delta_1$  and  $\Delta_2$  such that  $G_i \subseteq \Delta_i \cup N_e$ , where  $G_i$  is the graph induced by the edge set  $\{\mu(f) : f \in L \cap V(T_i)\}$  for  $i \in \{1, 2\}$  and  $T_1 \dot{\cup} T_2 = T \setminus \{e\}$ . Thus  $N_e$  meets  $G$  only in  $V(N_e) = \text{mid}(e)$  and its length is  $|\text{mid}(e)|$ . The vertices of  $\text{mid}(e) = V(G_1) \cap V(G_2)$  are ordered according to a cyclic permutation  $\pi = (\pi_e)_{e \in E(T)}$  on  $\text{mid}(e)$ .

**Theorem 2** ([13, 30]). *Let  $G$  be a planar graph of branchwidth at most  $k$  without vertices of degree one embedded on a sphere. Then there exists a sc-decomposition of  $G$  of width at most  $k$ .*

### 3 Minor Testing in Planar Graphs

For solving PLANAR  $H$ -MINOR CONTAINMENT in single-exponential time  $2^{\mathcal{O}(h)}$ .  $n + \mathcal{O}(n^2 \cdot \log n)$ , we introduce in this section the method of *partially embedded dynamic programming*. We present Algorithm 1 as a roadmap on how we proceed in proving our main Theorem 1.

---

**Algorithm 1.** The main routine for PLANAR  $H$ -MINOR CONTAINMENT.

---

```

Input : A planar graph  $G$ .
Output : A model  $M$  of  $H$  in  $G$ , if it exists.
Compute sc-decomposition  $\langle T, \mu, \pi \rangle$  of  $G$  of width  $\text{bw}(G)$ .
if  $\text{bw}(G) > c \cdot h$  for some constant  $c$  then Compute  $M$ 
else for every plane graph  $A \supseteq H$  produced by PRE-PROC( $H$ ) do
    Run partially embedded dynamic programming on  $\langle T, \mu, \pi \rangle$  to try to find a
    model  $M$  of  $A$  in  $G$ .
```

---

We divide Algorithm 1 into three parts, presented in the following sections.

From the next proposition we can find a model of  $H$  in  $G$  in the case of  $G$  having large branchwidth.

**Proposition 6.** [★<sup>1</sup>] Let  $G$  and  $H$  be planar graphs with  $|V(G)| = n$ ,  $|V(H)| = h$ . There exists a small constant  $c$  such that if  $\text{bw}(G) > c \cdot h$ , then  $G$  contains a model of  $H$ , which can be found in time  $\mathcal{O}(n^2 \cdot \log n + h^4)$ .

Otherwise, let us assume that  $\text{bw}(G) \leq c \cdot h$ . In this case, PRE-PROC( $H$ ) (basically) computes a list of all plane majors  $A$  of  $H$  up to a fixed size linear in  $h$ . This ‘preprocessing step’ is presented in Section 3.1. In the sequel, we will only be interested in a graph  $A$  of our list, if  $A$  is a minor of  $G$  obtained from  $H$  by ‘uncontracting’ some part, such that on a given subset  $S \subseteq V(G)$ , our graph  $A$  looks like a subgraph of  $G$ . Finally, in Section 3.2, we proceed by partially embedded dynamic programming bottom-up along a sphere cut decomposition of  $G$ . Here we make use of the fact that every middle set  $S$  yields a separating noose in an embedding of  $G$ . If  $H$  has a model  $M \subseteq G$  that intersects  $S$ , then the noose comes from a noose in  $M$ , which in turn is present in some major  $A$  of  $H$  of our list. We use this fact to restrict the number of candidates we need to consider.

#### 3.1 Preprocessing

If the branchwidth of  $G$  is at most  $c \cdot h$ , then we compute a sphere cut decomposition of width  $\mathcal{O}(h)$  in time  $\mathcal{O}(n^2)$  by using the algorithm of [17], and we continue with dynamic programming.

In the first step we do preprocessing. Namely, we compute for  $H$  a list of auxiliary graphs  $A$  with  $H \preceq A \preceq M$  and  $|V(A)| = \mathcal{O}(h)$ , such that  $A$  is a

---

<sup>1</sup> The parts marked with [★] can be found in a longer version of our paper [2].

candidate for a model  $M$  in  $G$ . To be precise, we compute a collection  $\mathcal{A}$  of 2-edge-colored plane graphs, each consisting of

- a planar graph  $A$  with  $|V(A)| \leq h + 1.5 \cdot \text{bw}(G)$ , such that  $H$  is a contraction minor of  $A$ ;
- a bipartition of the edge set  $E(A)$  into m-edges and c-edges such that contracting the c-edges of  $A_{m,c}$  creates a graph isomorphic to  $H$ ; and
- a drawing  $\Phi$  of  $A_{m,c}$ .

The simple routine PRE-PROC [★] takes as input  $H$  and outputs the collection  $\mathcal{A}$  of 2-edge-colored plane graphs  $\langle A_{m,c}, \Phi \rangle$ .

When doing dynamic programming in Section 3.2, instead of finding a model  $M$  of  $H$  in  $G$ , we restrict ourselves to finding such a collection  $\mathcal{A}$  consisting of minors of  $M$  which represent both  $H$  and  $M$  in each dynamic programming step.

**Lemma 1.** [★] *For every planar graph  $H$  on  $h$  vertices and every constant  $d$ , the cardinality of the collection  $\mathcal{A}$  of non-isomorphic 2-edge-colored plane graphs on  $d \cdot h$  vertices containing a minor isomorphic to  $H$  is  $2^{\mathcal{O}(h)}$ . Furthermore, we can compute  $\mathcal{A}$  in time  $2^{\mathcal{O}(h)}$ .*

Using Lemma 1, we get the following corollary.

**Corollary 1.** *Algorithm PRE-PROC [★] is correct and runs in time  $2^{\mathcal{O}(\text{bw}(G)+h)}$ .*

### 3.2 Partially Embedded Dynamic Programming

From now on, we will refer to a 2-edge-colored plane graph  $\langle A_{m,c}, \Phi \rangle \in \mathcal{A}$  simply as  $A$ . In this section we present our technique of partially embedded dynamic programming, using it to solve PLANAR  $H$ -MINOR CONTAINMENT in the case of the input graph  $G$  having bounded branchwidth. Before proceeding to a formal description of the dynamic programming, we provide the basic intuition behind our algorithm. Towards this, let us consider graphs  $A \in \mathcal{A}$  satisfying  $H \preceq_c A$  and  $A \preceq G$ .

We define subgraphs PAST, PRESENT, and FUTURE of  $A$  with  $V(A) = V(\text{PAST}) \cup V(\text{PRESENT}) \cup V(\text{FUTURE})$  and  $E(A) = E(\text{PAST}) \dot{\cup} E(\text{PRESENT}) \dot{\cup} E(\text{FUTURE})$ , such that

- PRESENT  $\subseteq G$ , (i.e., we can obtain  $A$  as a minor of  $G$  with PRESENT being subgraph of  $G$ ); and
- $E(\text{PAST}) \subseteq E(H)$ , (i.e., we can obtain  $H$  as a contraction minor of  $A$  without contracting edges in PAST).<sup>2</sup>

Intuitively speaking, in partially embedded dynamic programming, we look for potential models  $M$  of  $H$  with a magnifying glass only in the separators of the sc-decomposition of  $G$ . By decontracting  $H$  at the separators, we obtain the

<sup>2</sup> Here, we slightly abuse notation by assuming that edge sets in different graphs are actually the same, instead of introducing bijective mappings. Note that we make no assumption about the edges in FUTURE.

part PRESENT, which yields a subgraph of  $G$  for which we are enabled to apply embedded dynamic programming. For memorizing the rest of the potential model  $M$ , we contract all necessary edges to PAST in the processed graph and (almost) all edges to FUTURE in the graph remainder. The picture will be made clearer in the sequel.

Given a sc-decomposition of  $G$ , we proceed with dynamic programming: Every edge  $e$  of the sc-decomposition defines a separator  $\text{mid}(e) \subseteq V(G)$  and an associated noose  $N_e$ , which separates the graph  $G_{\text{sub}} \subseteq G$  processed so far from  $G \setminus G_{\text{sub}}$ . At every edge  $e$  of the sc-decomposition, we check for every graph  $A$  of  $\mathcal{A}$  all the ways in which the graph  $G_{\text{sub}}$  can be obtained as a minor of  $A_{\text{sub}} \subseteq A$  with  $A_{\text{sub}} = (V(\text{PAST}) \cup V(\text{PRESENT}), E(\text{PAST}) \cup E(\text{PRESENT}))$ , where  $\text{mid}(e)$  determines  $V(\text{PRESENT})$ . The noose  $N_e$  comes from a noose in  $A$ , and this is controlled by the ways in which  $N_e$  can be routed through the vertices of  $A$ . The number of solutions we get—the *valid partial solutions*—is bounded by the number of combinatorial nooses in  $A$  onto which we can map  $N_e$ . When updating the valid partial solutions at two incident edges of the sc-decomposition, we unite PRESENT and PAST of two solutions and set the graph remainder to FUTURE. In a post-processing step, we contract part of PRESENT, namely those edges with at most one endpoint in the newly obtained separator of the sc-decomposition; this part becomes PAST. We then decontract some edges of FUTURE for the next updating step. This concludes the informal description of the algorithm.

In the remaining part of this section, we will precisely describe and analyze the dynamic programming routine with which we achieve the following result:

**Lemma 2.** *For a plane graph  $G$  with a given sc-decomposition  $\langle T, \mu, \pi \rangle$  of  $G$  of width  $\text{bw}(G)$  and a planar graph  $H$  on  $h$  vertices, we can decide in time  $2^{\mathcal{O}(\text{bw}(G)+h)} \cdot n$  whether  $G$  contains a model  $M$  of  $H$ .*

**Dynamic programming.** We root the sc-decomposition  $\langle T, \mu, \pi \rangle$  at some node  $r \in V(T)$ . For each edge  $e \in T$ , let  $L_e$  be the set of leaves of the subtree rooted at  $e$ . The subgraph  $G_e$  of  $G$  is induced by the edge set  $\{\mu(v) \mid v \in L_e\}$ . The vertices of  $\text{mid}(e)$  form a combinatorial noose  $N$  that separates  $G_e$  from the residual graph.

Let  $A$  be a given plane graph in  $\mathcal{A}$ . If  $A$  is a minor of  $G$ , then there exists a plane model  $M$  of  $A$  in  $G$ . Furthermore, for above noose  $N$  the intersection  $M \cap N$  forms a noose in both model  $M$  and  $A$ . One basic point of partially embedded dynamic programming is to check how the vertices of the combinatorial noose  $N$  are mapped to faces and vertices of  $A$ . For a combinatorial noose  $N_A$  in  $A$ , we can map  $N$  to  $N_A$  bounding (clockwise) a unique subgraph  $A_{\text{sub}}$  of  $A$ .

In each step of the algorithm, we compute the solutions for a sub-problem in  $G_e$ , where each solution consists of three parts, namely

- a plane 2-edge-colored graph  $A \in \mathcal{A}$ ;
- a combinatorial noose  $N_A$  in  $A$ ; and
- a mapping  $\gamma$  from combinatorial noose  $N$  to  $N_A$  (defined below).

$N_A$  has the properties that *a*) it bounds (clockwise) a subgraph  $A_{\text{sub}} \subseteq A$  and *b*) no vertex in  $V(A_{\text{sub}}) \setminus V(N_A)$  is incident to a c-edge. The subgraph  $A_{\text{sub}}$  is

representing the part of model  $M$  already computed, whereas the residual graph of  $A$  represents the part of  $M$  which still has to be verified. For every middle set, we store this information in an array of triples  $\langle A, N_A, \gamma \rangle$ .

We define now valid mappings between combinatorial nooses and describe how partial solutions are stored in the dynamic programming. Then, we give the different DP-steps and finally verify the approach.

**Valid partial solutions.** For middle set  $\text{mid}(e)$  of the rooted sc-decomposition  $\langle T, \mu, \pi \rangle$  of plane graph  $G$ ,  $N = N_e$  is the associated combinatorial noose in  $G$  with face-vertex sequence of  $F(N) \cup V(N)$  separating  $G_e$  from the residual graph. Let  $\mathfrak{N}$  denote the set of all combinatorial nooses of  $A$  whose length is at most the length of  $N$  and which bound (clockwise) a subgraph  $A_{\text{sub}} \subseteq A$  such that no vertex in  $V(A_{\text{sub}}) \setminus V(N_A)$  is an end-vertex of a c-edge. We now map  $N$  to nooses  $N_A \in \mathfrak{N}$ , preserving the order. More precisely, we map vertices of  $N$  to both vertices and faces of  $A$ . Therefore, we consider partitions of  $V(N) = V_1(N) \dot{\cup} V_2(N)$  where vertices in  $V_1(N)$  are mapped to vertices of  $V(A)$  and vertices in  $V_2(N)$  to faces of  $F(A)$ .

We define a mapping  $\gamma : V(N) \cup F(N) \rightarrow V(A) \cup F(A)$  relating  $N$  to the combinatorial nooses in  $\mathfrak{N}$ . For every  $N_A \in \mathfrak{N}$  on faces and vertices of set  $F(N_A) \cup V(N_A)$  and for every partition  $V_1(N) \dot{\cup} V_2(N)$  of  $V(N)$ , mapping  $\gamma$  is valid [ $\star$ ] if

- a)  $\gamma$  restricted to  $V_1(N)$  is a bijection to  $V(N_A)$ ;
- b) every  $v \in V_2(N)$  and  $f \in F(N)$  satisfy  $\gamma(v) \in F(N_A)$  and  $\gamma(f) \in F(N_A)$ ;
- c) for every  $v_i \in V(N)$  and subsequence  $[f_{i-1}, v_i, f_i]$  of  $N$ : if  $v_i \in V_2(N)$ , then face  $\gamma(v_i)$  is equal to both  $\gamma(f_{i-1})$  and  $\gamma(f_i)$ , and if  $v_i \in V_1(N)$ , then vertex  $\gamma(v_i)$  is incident to both  $\gamma(f_{i-1})$  and  $\gamma(f_i)$ ; and
- d)  $A_{\text{sub}}$  is a minor of  $G_e$  with respect to a) – c).

We assign an array  $\Psi_e$  to each  $\text{mid}(e)$  consisting of triples, where each triple  $\langle A, N_A, \gamma \rangle$  represents a minor candidate  $A$  together with a valid mapping  $\gamma$  from a combinatorial noose  $N$  corresponding to  $\text{mid}(e)$  to a combinatorial noose  $N_A \in \mathfrak{N}$ . The vertices and faces of  $N$  are oriented clockwise around the drawing of  $G_e$ . Without loss of generality, we assume for every  $\langle A, N_A, \gamma \rangle$  the orientation of  $N_A$  to be clockwise around the drawing of subgraph  $A_{\text{sub}}$  of  $A$ .

**Step 0: Initializing the leaves.** For every parent edge  $e$  of a leaf  $v$  of  $T$ , we initialize for every  $A \in \mathcal{A}$  the valid mappings from the combinatorial noose bounding the edge  $\mu(v)$  of  $G$  to every combinatorial noose of length at most two in  $A$  (clockwise bounding at most one edge of  $A$ ).

**Step 1a): Update process.** We update the arrays of the middle sets bottom-up in post-order manner from the leaves of  $T$  to root  $r$ . During this updating process it is guaranteed that the ‘local’ solutions for each minor associated with a middle set of the sc-decomposition are combined into a ‘global’ solution for the overall graph  $G$ .

In each dynamic programming step, we compare the arrays of two middle sets  $\text{mid}(e)$  and  $\text{mid}(f)$  in order to create a new array assigned to the middle set

**mid**( $g$ ), where  $e, f$ , and  $g$  have a vertex of  $T$  in common. From [13] we know that the combinatorial noose  $N_g$  is formed by the symmetric difference of the combinatorial nooses  $N_e, N_f$  and that  $G_g = G_e \cup G_f$ . In other words, we are ensured that if two solutions on  $G_e$  and  $G_f$  bounded by  $N_e$  and  $N_f$  fit together, then they form a new solution on  $G_g$  bounded by  $N_g$ . We now determine when two solutions represented as tuples in the arrays  $\Psi_e$  and  $\Psi_f$  fit together. We update two triples  $\langle A^1, N_A^1, \gamma_1 \rangle \in \Psi_e$  and  $\langle A^2, N_A^2, \gamma_2 \rangle \in \Psi_f$  to a new triple in  $\Psi_g$  if

- $A^1 = A^2 =: A \in \mathcal{A}$  and every edge of  $A$  with no endpoint in  $V(N_A^1) \cup V(N_A^2)$  is an m-edge;
- for every  $x \in (V(N_e) \cup F(N_e)) \cap (V(N_f) \cup F(N_f))$ , we have  $\gamma_1(x) = \gamma_2(x)$ ; and
- for the subgraph  $A_{\text{sub}}^1$  of  $A$  separated by  $N_A^1$  and the subgraph  $A_{\text{sub}}^2$  of  $A$  separated by  $N_A^2$ , we have that  $E(A_{\text{sub}}^1) \cap E(A_{\text{sub}}^2) = \emptyset$  and  $V(A_{\text{sub}}^1) \cap V(A_{\text{sub}}^2) \subseteq \{\gamma(v) \mid v \in V(N_e) \cap V(N_f)\}$ .

That is, we only update solutions with the same graph  $A$  and with the two nooses  $N_A^1$  and  $N_A^2$  bounding (clockwise) two edge-disjoint parts of  $A$  and intersecting in a consecutive subsequence of both  $N_A^1$  and  $N_A^2$ . If the two solutions on  $N_e$  and  $N_f$  fit together, we get a valid mapping  $\gamma_3 : N_g \rightarrow N_A^3$  to a noose  $N_A^3$  of  $A$  as follows:

- for every  $x \in (V(N_e) \cup F(N_e)) \cap (V(N_f) \cup F(N_f)) \cap (V(N_g) \cup F(N_g))$  we have  $\gamma_1(x) = \gamma_2(x) = \gamma_3(x)$ ;
- for every  $y \in (V(N_e) \cup F(N_e)) \setminus (V(N_f) \cup F(N_f))$  we have  $\gamma_1(y) = \gamma_3(y)$ ; and
- for every  $z \in (V(N_f) \cup F(N_f)) \setminus (V(N_e) \cup F(N_e))$  we have  $\gamma_2(z) = \gamma_3(z)$ .

We have that  $\gamma_3$  is a valid mapping from  $N_g$  to the combinatorial noose  $N_A^3$  that bounds subgraph  $A_{\text{sub}}^3 = A_{\text{sub}}^1 \cup A_{\text{sub}}^2$ .

**Step 1b): Post-processing.** Before adding a triple  $\langle A, N_A^3, \gamma_3 \rangle$  to array  $\Psi_g$ , we need to manipulate  $A$  so that *a*) it does not grow too big and *b*) it is suitable for future update operations. In  $A$  restricted to subgraph  $A_{\text{sub}}^3$ , we contract all c-edges with at least one end-vertex not in  $N_A^3$  in order to fulfill *a*). Concerning *b*), for every  $B \in \mathcal{A}$  we check for all its nooses  $N_B$  with  $|V(N_B)| = |V(N_A^3)|$  if there is a bijection  $\beta$  from  $N_A^3$  to  $N_B$  such that the following holds. If in a copy of  $B$  we contract those c-edges which *i*) are in the subgraph counter-clockwise bounded by  $N_B$  and *ii*) have at least one end-vertex not in  $N_B$ , then we obtain a 2-edge-colored graph isomorphic to  $A$ . We define  $\delta = \gamma_3 \circ \beta$  and we replace  $\langle A, N_A^3, \gamma_3 \rangle$  in array  $\Psi_g$  by those triples  $\langle B, N_B, \delta \rangle$  which validate properties *i*) and *ii*).

**Step 2: Termination.** If, at some step, we have a solution where the entire minor  $H$  is formed, we terminate the algorithm accepting. That is the case, if for some triple we have that  $H \preceq A_{\text{sub}} \preceq A$  and  $A_{\text{sub}}$  is bounded by  $N_A$ . We output model  $M$  of  $H$  in  $G$  represented by this  $A$  by reconstructing a solution top-down in  $\langle T, \mu, \pi \rangle$ . If at root  $r$  no  $A \in \mathcal{A}$  has been computed, we reject.

We omit here the correctness proof and the running time of the algorithm and refer to the long version of this paper [2]. This completes the proof of Lemma 2.

*Proof of Theorem 1.* We put everything together by verifying Algorithm 1. We produce in time  $\mathcal{O}(n^2 \cdot \log n)$  a sc-decomposition of input graph  $G$  [17]. Next, either we can immediately compute a minor model of  $G$  in time  $\mathcal{O}(n^2 \cdot \log n + h^4)$  (Proposition 6) or we run our 2-step-algorithm: we produce all majors of the minor pattern (Lemma 1) with Algorithm PRE-PROC [★] in time  $2^{\mathcal{O}(h)}$ , and run partially embedded dynamic programming in time  $2^{\mathcal{O}(h)} \cdot n$  (Lemma 2). □

## 4 Conclusions and Further Research

In this paper we showed that PLANAR  $H$ -MINOR CONTAINMENT is solvable in time  $2^{\mathcal{O}(h)} \cdot n + \mathcal{O}(n^2 \cdot \log n)$  for a host graph on  $n$  vertices and a pattern  $H$  on  $h$  vertices. That is, we showed that the problem can be solved in *single-exponential* time in  $h$ , significantly improving all previously known algorithms. Similar to [11], we can enumerate and count the number of models within the same time bounds.

Let us discuss some interesting avenues for further research concerning minor containment problems. First, it seems possible to solve in single-exponential time other variants of planar minor containment using our approach, like looking for a contraction minor, an induced minor, or a topological minor, as it has been recently done in [1] for general host graphs using completely different techniques. Also, it would be interesting to count the number of non-isomorphic models faster than just by enumerating models and removing isomorphic duplicates.

An important question is if, up to some assumption from complexity theory, the running time of our algorithm is tight. In other words, is there a  $2^{\mathcal{O}(h)} \cdot n^{\mathcal{O}(1)}$  algorithm (i.e., a *subexponential* algorithm) solving PLANAR  $H$ -MINOR CONTAINMENT or the existence of such an algorithm would imply the failure of, say, the Exponential Time Hypothesis? A first step could be to study the existence of subexponential algorithms when the pattern is restricted to be a  $k$ -outerplanar graph for some constant  $k$ , or any other subclass of planar graphs.

Conversely, single-exponential algorithms may exist for host graphs more general than planar graphs. The natural candidates are host graphs embeddable in an arbitrary surface. One possible approach could be to use the framework recently introduced in [29] to perform dynamic programming for graphs on surfaces. The main ingredient of this framework is a new type of branch decomposition of graphs on surfaces, called *surface cut decomposition*, which plays the role of sphere cut decompositions for planar graphs.

## References

1. Adler, I., Dorn, F., Fomin, F.V., Sau, I., Thilikos, D.M.: Faster Parameterized Algorithms for Minor Containment. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 322–333. Springer, Heidelberg (2010)

2. Adler, I., Dorn, F., Fomin, F.V., Sau, I., Thilikos, D.M.: Fast Minor Testing in Planar Graphs (2010), <http://users.uoa.gr/~sedthilk/papers/fastminor.pdf>
3. 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)
4. Baker, B.S.: Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM* 41, 153–180 (1994)
5. Demaine, E.D., Fomin, F.V., Hajiaghayi, M.T., Thilikos, D.M.: Subexponential parameterized algorithms on graphs of bounded genus and  $H$ -minor-free graphs. *Journal of the ACM* 52(6), 866–893 (2005)
6. Demaine, E.D., Hajiaghayi, M.: Bidimensionality. In: Kao, M.-Y. (ed.) *Encyclopedia of Algorithms*. Springer, Heidelberg (2008)
7. Demaine, E.D., Hajiaghayi, M.T.: Bidimensionality: new connections between FPT algorithms and PTASs. In: Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 590–601 (2005)
8. Demaine, E.D., Hajiaghayi, M.T., Kawarabayashi, K.i.: Algorithmic Graph Minor Theory: Decomposition, Approximation, and Coloring. In: Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 637–646 (2005)
9. Diestel, R.: *Graph Theory*, vol. 173. Springer, Heidelberg (2005)
10. Dinneen, M., Xiong, L.: The Feasibility and Use of a Minor Containment Algorithm. Computer Science Technical Reports 171, University of Auckland (2000)
11. Dorn, F.: Planar Subgraph Isomorphism Revisited. In: Proc. of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 263–274 (2010)
12. Dorn, F., Fomin, F.V., Thilikos, D.M.: Subexponential parameterized algorithms. *Computer Science Review* 2(1), 29–39 (2008)
13. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica* (2009) (to appear)
14. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
15. Fellows, M.R., Langston, M.A.: On search, decision and the efficiency of polynomial-time algorithms. *J. Comp. Syst. Sc.* 49, 769–779 (1994)
16. 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)
17. Gu, Q.-P., Tamaki, H.: Constant-factor approximations of branch-decomposition and largest grid minor of planar graphs in  $O(n^{1+\varepsilon})$  time. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) *ISAAC 2009. LNCS*, vol. 5878, pp. 984–993. Springer, Heidelberg (2009)
18. Gu, Q.P., Tamaki, H.: Improved bound on the planar branchwidth with respect to the largest grid minor size. Technical Report SFU-CMPT-TR 2009-17, Simon Fraiser University (2009)
19. Hicks, I.V.: Branch decompositions and minor containment. *Networks* 43(1), 1–9 (2004)
20. Kawarabayashi, K.i., Reed, B.A.: Hadwiger’s conjecture is decidable. In: Proc. of the 41st Annual ACM Symposium on Theory of Computing (STOC), pp. 445–454 (2009)
21. Kawarabayashi, K.i., Wollan, P.: A shorter proof of the Graph Minor Algorithm - The Unique Linkage Theorem. In: Proc. of the 42st Annual ACM Symposium on Theory of Computing, STOC (to appear, 2010)

22. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. *SIAM J. Comput.* 9, 615–627 (1980)
23. Mohar, B., Thomassen, C.: *Graphs on surfaces*. John Hopkins University Press (2001)
24. Osthus, D., Prömel, H.J., Taraz, A.: On random planar graphs, the number of planar graphs and their triangulations. *J. Comb. Theory, Ser. B* 88(1), 119–134 (2003)
25. Reed, B.A., Li, Z.: Optimization and Recognition for  $K_5$ -minor Free Graphs in Linear Time. In: Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN), pp. 206–215 (2008)
26. Robertson, N., Seymour, P.: Graph Minors. XIII. The Disjoint Paths Problem. *J. Comb. Theory, Ser. B* 63(1), 65–110 (1995)
27. Robertson, N., Seymour, P., Thomas, R.: Quickly excluding a planar graph. *J. Comb. Theory, Ser. B* 62(2), 323–348 (1994)
28. Robertson, N., Seymour, P.D.: Graph Minors. XX. Wagner’s Conjecture. *J. Comb. Theory, Ser. B* 92(2), 325–357 (2004)
29. Rué, J., Sau, I., Thilikos, D.M.: Dynamic Programming for Graphs on Surfaces. In: Proc. of the 37th International Colloquium on Automata, Languages and Programming, ICALP (to appear, 2010),  
<http://hal.archives-ouvertes.fr/inria-00443582>
30. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. *Combinatorica* 14(2), 217–241 (1994)
31. Tutte, W.T.: A census of planar triangulations. *Canadian Journal of Mathematics* 14, 21–38 (1962)

# On the Number of Spanning Trees a Planar Graph Can Have

Kevin Buchin<sup>1,\*</sup> and André Schulz<sup>2,\*\*</sup>

<sup>1</sup> Department of Mathematics and Computer Science,  
Technical University of Eindhoven  
[k.a.buchin@tue.nl](mailto:k.a.buchin@tue.nl)

<sup>2</sup> Institut für Mathematische Logik und Grundlagenforschung, Universität Münster  
[andre.schulz@uni-muenster.de](mailto:andre.schulz@uni-muenster.de)

**Abstract.** We prove that any planar graph on  $n$  vertices has less than  $O(5.2852^n)$  spanning trees. Under the restriction that the planar graph is 3-connected and contains no triangle and no quadrilateral the number of its spanning trees is less than  $O(2.7156^n)$ . As a consequence of the latter the grid size needed to realize a 3d polytope with integer coordinates can be bounded by  $O(147.7^n)$ . Our observations imply improved upper bounds for related quantities: the number of cycle-free graphs in a planar graph is bounded by  $O(6.4884^n)$ , the number of plane spanning trees on a set of  $n$  points in the plane is bounded by  $O(158.6^n)$ , and the number of plane cycle-free graphs on a set of  $n$  points in the plane is bounded by  $O(194.7^n)$ .

## 1 Introduction

The number of spanning trees of a connected graph, also considered as the complexity of the graph, is an important graph invariant. Its importance largely stems from Kirchhoff's seminal matrix tree theorem: The number of spanning trees equals the absolute value of any cofactor of the Laplacian matrix of the graph. Furthermore, this number is the order of the Jacobian group of the graph, also known as critical group, or as sandpile model in theoretical physics [23]. This group can be represented as a chip firing game on the graph; in this context the number of spanning trees counts the number of the stable and recurrent configurations [4].

Our motivation to study the number of spanning trees of planar graphs comes from an application of Kirchhoff's matrix tree theorem. Instead of computing the number of spanning trees with Kirchhoff's theorem one can use bounds on the number of spanning trees to obtain bounds for the cofactors of the Laplacian matrix. These cofactors appear in various settings. For example, Tutte's famous spring embedding is computed by solving a linear system that is based on the

---

\* Supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707.

\*\* Supported by the German Research Foundation (DFG) under grant SCHU 2458/1-1.

Laplacian matrix [19][20]. As a consequence of Cramer's rule the cofactor of the Laplacian matrix is the denominator of all coordinates in the embedding. Therefore, by multiplying with the number of spanning trees, we can scale to an integer embedding. This idea finds applications in the grid embedding of 3d polytopes [13][14]. Before we describe this application in more detail, we introduce some notation.

Let  $\mathcal{G}_n$  be the set of all planar graphs with  $n$  vertices. For a graph  $G \in \mathcal{G}_n$  we denote the number of its (labeled) spanning trees with  $t(G)$ . For every  $\mathcal{G}_n$  let  $T(n)$  be the maximal number of spanning trees a graph in this class can have, that is  $T(n) = \max_{G \in \mathcal{G}_n} \{t(G)\}$ . We study the growth rate of the function  $T(n)$ . Since it seems intractable to obtain an exact formula for  $T(n)$ , we aim at finding a value  $\alpha$  such that  $T(n) \leq \alpha^n$  for  $n$  large enough. Notice that the graph that realizes the maximum  $T(n)$  has to be a triangulation. Hence, it suffices to look at the subclass of all planar triangulations with  $n$  vertices instead of considering all graphs in  $\mathcal{G}_n$ .

Furthermore, we are interested in the maximal number of spanning trees for planar graphs with special facial structure. In particular, we want to bound

$$T_4(n) = \max_{G \in \mathcal{G}_n} \{t(G) \mid G \text{ is 3-connected and contains no triangle}\},$$

$$T_5(n) = \max_{G \in \mathcal{G}_n} \{t(G) \mid G \text{ is 3-connected and contains no triangle or quadrilateral}\}.$$

Notice that if a graph is planar and 3-connected its facial structure is uniquely determined [21]. Let  $\alpha_4^n$  be an upper bound on  $T_4(n)$  and  $\alpha_5^n$  be an upper bound on  $T_5(n)$ . We refer to the problem of bounding  $\alpha$  as the *general problem*, and to the problems of bounding  $\alpha_4$  and  $\alpha_5$  as *restricted problems*.

For embedding 3d polytopes the necessary grid size (ignoring polynomial factors) can be expressed in terms of  $\alpha$ ,  $\alpha_4$  and  $\alpha_5$ . In this scenario we are dealing with 3-connected planar graphs since  $G$  is the graph of a 3d polytope [17]. If the graph  $G$  contains a triangle the grid size is in  $O(\alpha^{2n})$ , if  $G$  contains a quadrilateral the grid size is in  $O(\alpha_4^{3n})$ . Due to Euler's formula every (3-connected) planar graph contains a pentagon – in this case the grid size in  $O(\alpha_5^{5n})$ . As a consequence better bounds on  $\alpha$ ,  $\alpha_4$  and  $\alpha_5$  directly imply a better bound on the grid size needed to realize a polytope with integer coordinates.

Richter-Gebert used a bound on  $T(n)$  to bound the size of the grid embedding of a 3d polytope [14]. By applying Hadamard's inequality he showed that the cofactors of the Laplacian matrix of a planar graph are less than  $6.5^n$ . This bound can be easily improved to  $6^n$  by noticing that the Laplacian matrix is positive semi-definite, which allows the application of the stronger version of Hadamard's inequality [9, page 477]. Both bounds do not rely on the planarity of  $G$ , but on the fact that the sum of the vertex degrees of  $G$  is below  $6n$ . Ribó and Rote improved Richter-Gebert's analysis and showed that  $5.0295^n \leq T(n) \leq 5.3^n$  [12][15]. The lower bound is realized on a wrapped up triangular grid and was obtained by the transfer-matrix method. For the upper bound they count the number of the spanning trees on the dual graph. This number coincides with the number of spanning trees in the original planar graph. Since the number

of spanning trees is maximized by a triangulation, the dual graph is 3-regular. Applying a result of McKay [11], which bounds the number of spanning trees on  $k$ -regular graphs, yields the bound of  $5.3^n$ . Interestingly, this bound is not directly related to the planarity of the graph. Therefore, Ribó and Rote tried to improve the bound using the *outgoing edge approach*. The approach involves choosing a partial orientation of the graph and estimating the probability of a cycle. To handle dependencies between cycles, Ribó and Rote tried (1) selecting an independent subset of cycles, and (2) using Suen’s inequality [13]. However, they could only prove an upper bound of  $5.5202^n$  for  $T(n)$ , and they showed that their approach is not suitable to break the bound of  $5.3^n$ . For the restricted problems they obtained the bounds  $T_4(n) \leq 3.529^n$  and  $T_5(n) \leq 2.847^n$ .

Bounds for the number of spanning trees of general graphs are often expressed in terms of the vertex degree sequence of the graph. However, the main difficulty in obtaining good values for  $\alpha$  lies in the fact that we do not know the degree sequence of the graph in advance. Therefore, these bounds are not directly applicable. If we would assume that almost every vertex degree is 6, which is true for the best known lower bound example presented in [12], the bound of Grone and Merris [8] gives an upper bound for  $T(n)$  of  $(n/(n-1))^{n-1} 6^{n-1}/n$ , whose asymptotic growth equals the growth rate obtained by Hadamard’s inequality. To apply the more involved bound of Lyons [10] one has to know the probabilities that a simple random walk returns to its start vertex after  $k$  steps (for every start vertex). Even under the assumption that every vertex has degree 6, it is difficult to express the return probabilities in terms of  $k$  to obtain an improvement over  $6^n$ .

Spanning trees are not the only interesting substructures that can be counted in planar graphs. Aichholzer *et al.* [1] list the known upper bounds for other subgraphs contained in a triangulation: Hamiltonian cycles, cycles, perfect matchings, connected graphs and so on. The bounds for Hamiltonian cycles and cycles have been recently improved [5].

**Overview.** In Section 2 we bound the number of spanning trees by the number of *outdegree-one graphs*, i.e., the number of directed graphs obtained by picking for each vertex one outgoing edge. Cycle-free outdegree-one graphs correspond to spanning trees. Therefore we next bound the probability that a random outdegree-one graph has a cycle. For this we analyze the dependencies between cycles. In contrast to Ribó and Rote who showed how to avoid the dependencies in the analysis, we instead make use of the dependencies. Since our method might also find application in analyzing similar dependency structures, we phrase our probabilistic lemma in a more general setting in Section 2.1. More specifically, we develop a framework to analyze a series of events for which dependent events are mutually exclusive. In Section 2.2 we apply this framework to bound the probability of the occurrence of a cycle. From this we derive in Sections 2.3 and 2.4 a linear program whose objective function bounds (the logarithm of) the number of spanning trees. This linear program has infinitely many variables, and we instead consider the dual program with infinitely many constraints.

**Results.** We improve the upper bounds for the number of spanning trees of planar graphs by showing:  $\alpha \leq 5.2852$ ,  $\alpha_4 \leq 3.4162$ , and  $\alpha_5 \leq 2.7156$ . As a consequence the grid size needed to realize a 3d polytope with integer coordinates can now be bounded by  $O(147.7^n)$  instead of  $O(188^n)$ . For grid embeddings of simplicial 3d polytopes our results yield a small improvement to  $O(27.94^n)$  over the old bound of  $O(28.4^n)$ .

The maximal number of cycle-free graphs in a triangulation is another interesting quantity. Aichholzer *et al.* [1] obtained an upper bound of  $6.75^n$  for this number. We show in this paper that the improved bound for  $T(n)$  yields an improved upper bound of  $O(6.4948^n)$ .

Multiplying  $\alpha$  with the number of maximal number of triangulations a point set can have, gives an upper bound for the number of plane spanning trees on a point set. Using  $30^n$  as an upper bound for the number of triangulations of a point set (obtained by Sharir and Sheffer [16]) yields an upper bound of  $O(158.6^n)$  for the number of plane spanning trees on a point set. By the same construction the number of plane cycle-free graphs can be improved to  $O(194.7^n)$ . To our knowledge both bounds are the currently best known bounds.

## 2 Refined Outgoing Edge Approach

Our results are obtained by the *outgoing edge approach* and its refinements. For this we consider each edge  $vw$  of  $G$  as a pair of directed arcs  $v \rightarrow w$  and  $w \rightarrow v$ . Let  $v_1$  be a designated vertex of  $G$ , and let  $v_2, \dots, v_n$  be the remaining vertices. A directed graph is called *outdegree-one*, if  $v_1$  has no outgoing edge, and every remaining vertex is incident to exactly one outgoing edge. A spanning tree can be oriented as outdegree-one graphs by directing its edges towards  $v_1$ . This interpretation associates every spanning tree with exactly one outdegree-one graph. As a consequence the number of outdegree-one graphs contained in  $G$  exceeds  $t(G)$ .

We can obtain all outdegree-one graphs by selecting for every vertex (except  $v_1$ ) an edge as its outgoing edge. Let  $\mathcal{S}$  be such a selection. We denote with  $d_i$  the degree of the vertex  $v_i$ . For every vertex  $v_i$  we have  $d_i$  choices how to select its outgoing edge. This gives us in total  $\prod_{i=2}^n d_i$  different outdegree-one graphs in  $G$ . Due to Euler's formula the average vertex degree is less than 6, and hence we have less than  $6^n$  outdegree-one graphs of  $G$  by the geometric-arithmetic mean inequality. Thus, the outgoing edge approach gives the same bound as the strong Hadamard inequality by a very simple argument.

Outdegree-one graphs without cycles are exactly the (oriented) spanning trees of  $G$ . To improve the bound of  $6^n$  we try to remove all graphs with cycles from our counting scheme. Let us now consider a random selection  $\mathcal{S}$  that picks the outgoing edge for every vertex uniformly at random. This implies that also the selected outdegree-one graph will be picked uniformly at random. Let  $P_{\text{nc}}$  be the probability that the random graph selected by  $\mathcal{S}$  contains no cycle. The exact number of spanning trees for any (not necessary planar) graph  $G$  is given by  $t(G) = (\prod_{i=2}^n d_i) P_{\text{nc}}$ .

## 2.1 The Dependencies of Cycles in a Random Outdegree-One Graph

Assume that the  $t$  cycles contained in  $G$  are enumerated in some order. Notice that in an outdegree-one graph every cycle has to be directed. We consider the two orientations of a cycle with more than two vertices as one cycle. Let  $C_i$  be the event that the  $i$ -th cycle occurs and let  $C_i^c$  be the event that the  $i$ -th cycle does not occur in a random outdegree-one graph. For events  $C_i, C_j$  we denote that they are dependent by  $C_i \leftrightarrow C_j$  and that they are independent by  $C_i \not\leftrightarrow C_j$ . We say that cycles are dependent (independent) if the corresponding events are dependent (independent).

Two cycles are independent if and only if they do not share a vertex. In turn, cycles that share a vertex are not only dependent but *mutually exclusive*, i.e., they cannot occur both in an outdegree-one graph, since this would result in a vertex with two outgoing edges. This gives us the following two properties of the events  $C_i$ . We say events  $E_1, \dots, E_l$  have *mutually exclusive dependencies* if  $E_i \leftrightarrow E_j$  implies  $\Pr[E_i \cap E_j] = 0$ . We say that events  $E_1, \dots, E_l$  have *union-closed independencies* if  $E_i \not\leftrightarrow E_{i_1}, \dots, E_i \not\leftrightarrow E_{i_k}$  implies  $E_i \not\leftrightarrow (E_{i_1} \cup \dots \cup E_{i_k})$ . It is easy to see that the events  $C_i$  have mutually exclusive dependencies and union-closed independencies.

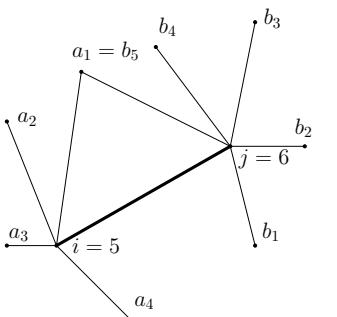
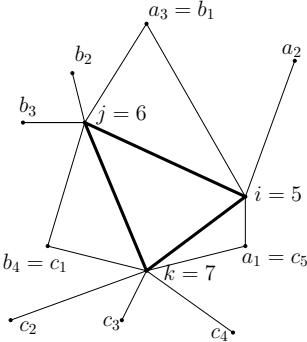
**Lemma 1.** *If events  $E_1, \dots, E_l$  have mutually exclusive dependencies and union-closed independencies then  $1 < k < l$*

$$\Pr[\bigcap_{j=k}^l E_j^c \mid \bigcap_{i=1}^{k-1} E_i^c] \leq \prod_{j=k}^l \left( 1 - \frac{\Pr[E_j]}{\prod_{\substack{1 \leq i < k: \\ E_i \leftrightarrow E_j}} \Pr[E_i^c] \sqrt{\prod_{\substack{k \leq i \leq l: \\ E_i \leftrightarrow E_j}} \Pr[E_i^c]}} \right).$$

The proof of the lemma can be found in the full version of the paper.

## 2.2 Bounding the Probability of the Appearance of Cycles

Before estimating the probability  $P_{nc}$  in terms of the vertex degrees, we introduce some notation. A cycle of length  $k$  is called a  $k$ -cycle. The  $k$ -extension of a cycle is the union of a cycle with all its dependent  $k$ -cycles. We say that the degree of a cycle is the ordered sequence of the degrees of its vertices. Let  $C_{abc}$  a 3 cycle spanned by  $v_a, v_b, v_c$ , and let the degree of  $C_{abc}$  be  $(d_a, d_b, d_c) = (i, j, k)$ . We denote the degrees of the vertices adjacent to  $v_a$  that are not part of  $C_{abc}$  by the sequence  $A$ . In the same fashion we denote the degrees of the vertices around  $v_b$  by  $B$  and the around  $v_c$  by  $C$ . The ordering in  $A, B, C$  respects the counter clockwise ordering of the vertices around  $v_a, v_b, v_c$  in a planar embedding. Since  $G$  is planar and 3-connected the ordering of the sequences is uniquely determined up to a global reflection [21]. Notice that a vertex might occur in two different sequences. We call the tuple  $(i, j, k, A, B, C)$ , the *signature* of the 2-extension


 $(5, 6, (a_1, \dots, a_4), (b_1, \dots, b_5))$ 

 $(5, 6, 7, (a_1, \dots, a_3), (b_1, \dots, b_4), (c_1, \dots, c_5))$ 

**Fig. 1.** Convention for naming the signatures of 2-extensions of 2-cycles (on the left) and 3-cycles (on the right)

of  $C_{abc}$ . Similarly, we define the signature of a 2-extension of a 2-cycle  $C_{ab}$  by the tuple  $(i, j, A, B)$ . The naming convention is depicted in Figure [II](#).

We can express  $P_{nc}$  as  $\Pr[\bigcap_{j=1}^t C_i^c]$ . Our goal is to apply Lemma [II](#) to bound this probability. As a first step we discuss how to express the number of spanning trees  $t(G)$  in the case that the different signatures of  $G$  are known. Instead of  $t(G)$  we bound its logarithm, i.e.,

$$\log t(G) = \sum_{i=2}^n \log d_i + \log \Pr\left[\bigcap_{j=1}^t C_i^c\right]. \quad (1)$$

The probability that an event  $C_i$  occurs can be expressed in terms of vertex degrees. In particular,

$$\begin{aligned} \Pr[C_i] &= 1/(d_a d_b) && \text{the } i\text{-th cycle is a 2-cycle on the vertices } v_a v_b, \\ \Pr[C_j] &= 2 / \prod_{a: v_a \in Z} d_a && \text{the } j\text{-th cycle is at least a 3-cycle on the set } Z. \end{aligned}$$

The way we proceed depends on whether we are addressing the general problem (i.e., we want to bound  $\alpha$ ) or one of the restricted problems (i.e., we want to bound  $\alpha_4$  or  $\alpha_5$ ). In the latter case we limit our analysis to 2-cycles only. In the general case we consider all cycles of length 2 and cycles of length 3 that are triangles in  $G$ .

We start with the general problem. Assume that all cycles  $C$  are enumerated such that the first  $t_3$  cycles are the triangles in  $G$ , and the last  $t_2$  cycles are the 2-cycles of  $G$ . In total we consider  $t := t_2 + t_3$  cycles. All remaining cycles are ignored. Discarding the larger cycles gives an upper bound on  $P_{nc}$  and is therefore applicable. We apply Lemma [II](#) with  $k = 1$  and  $l = t_3$  to bound  $\Pr[\bigcap_{j=1}^{t_3} C_j^c]$ , which is the probability that no 3-cycle occurs. To take also the 2-cycles into account we consider the probability that no 2-cycle occurs under the condition that no triangle occurred as 3-cycle, which is  $\Pr[\bigcap_{j=t_3+1}^t C_j^c | \bigcap_{j=1}^{t_3} C_j^c]$ . Notice

that this probability has the form stated in Lemma 11 for  $l = t$  and  $k = t_3 + 1$ . Thus, we can bound  $\log \Pr[\bigcap_{j=1}^t C_j^c]$  from above by

$$\sum_{j=1}^{t_3} \log \left( 1 - \frac{\Pr[C_j]}{\sqrt{\prod_{\substack{1 \leq i \leq t_3: \\ C_i \leftrightarrow C_j}} \Pr[C_i^c]}} \right) + \sum_{j=t_3+1}^t \log \left( 1 - \frac{\Pr[C_j]}{\prod_{\substack{1 \leq i < t_3+1: \\ C_i \leftrightarrow C_j}} \Pr[C_i^c] \sqrt{\prod_{\substack{t_3 < i \leq t: \\ C_i \leftrightarrow C_j}} \Pr[C_i^c]}} \right). \quad (2)$$

Equation (2) is a sum over cycles. Each summand in this sum depends only on the signature of the 2-extension of such cycle. Hence, we can group the summands in (2) with identical signatures. We denote the number of 2-extensions of 2-cycles with signature  $(i, j, A, B)$  by the variable  $f_{ij}(A, B)$ . Similarly, the number of 2-extensions of 3-cycles with signature  $(i, j, k, A, B, C)$  is denoted by  $f_{ijk}(A, B, C)$ . In order to simplify matters, we refer to  $f_{ij}(A, B)$  and  $f_{ijk}(A, B, C)$  simply as  $f_{ij}$  and  $f_{ijk}$ , or as  $f$  variables.

For better readability we introduce the following notations ( $X$  is used as a placeholder for  $A, B$ , or  $C$ , and  $x$  as a placeholder for  $a, b$ , or  $c$ ):

$$\begin{aligned} P_2(r, X) &:= \prod_{1 \leq p \leq r-1} \left( 1 - \frac{1}{rx_p} \right), \quad P_3(r, X) := \prod_{1 \leq p \leq r-2} \left( 1 - \frac{2}{rx_p x_{p+1}} \right), \\ P_{ij}(A, B) &:= 1 - \frac{1}{ij P_3(i, A) P_3(j, B) \left( 1 - \frac{2}{ija_1} \right) \left( 1 - \frac{2}{ijb_1} \right) \sqrt{P_2(i, A) P_2(j, B)}}, \\ P_{ijk}(A, B, C) &:= 1 - \frac{2}{ijk \sqrt{P_3(i, A) P_3(j, B) P_3(k, C) \left( 1 - \frac{2}{ika_1} \right) \left( 1 - \frac{2}{ijb_1} \right) \left( 1 - \frac{2}{jkc_1} \right)}}. \end{aligned}$$

We rephrase (2) as

$$\log \Pr[\bigcap_{j=1}^t C_j^c] \leq \sum_{i,j,k,A,B,C} f_{ijk}(A, B, C) \log P_{ijk}(A, B, C) + \sum_{i,j,A,B} f_{ij}(A, B) \log P_{ij}(A, B). \quad (3)$$

The sums in the last expression (and following similar sums) range over all feasible signatures. Let us now consider the restricted problems. Both restricted problems are easier to analyze than the general problem, since we consider only 2-cycles. To bound  $\Pr[\bigcap_{j=1}^{t_2} C_j^c]$  we apply Lemma 11 with  $k = 1$  and  $l = t_2$ . Following the presentation of the general problem we define

$$\hat{P}_{ij}(A, B) := 1 - \frac{1}{ij \sqrt{P_2(i, A) P_2(j, B)}},$$

and obtain for the restricted problems

$$\log \Pr[\bigcap_{j=1}^{t_2} C_j^c] \leq \sum_{i,j,A,B} f_{ij}(A, B) \log \hat{P}_{ij}(A, B). \quad (4)$$

### 2.3 A Charging Scheme for the Vertex Degrees

If we insert the bounds (3) or (4) into equation (1) we obtain an upper bound for  $t(G)$  in terms of the signatures of  $G$ . However, we would like to express the first part of equation (1), which is  $D := \sum_{i=1}^n \log d_i$ , also in terms of the  $f$  variables. For convenience we include  $\log d_1$  in the sum for  $D$ , which is applicable since we are looking for an upper bound.

Let us first discuss the general problem. We split  $D$  into four parts:  $D_i := \mu_i D$  for  $i = 1, \dots, 4$ , with  $\sum_{i=1}^4 \mu_i = 1$ . The parameters  $\mu_i$  will be fixed later. We express  $D_1$  and  $D_2$  by the  $f_{ij}$  variables and  $D_3$  and  $D_4$  by the  $f_{ijk}$  variables. Every vertex  $v_a$  contributes  $\mu_1 \log d_a$  to  $D_1$ . On the other hand, every vertex  $v_a$  is part of  $d_a$  2-cycles. We charge the total amount of  $\mu_1 \log d_a$  uniformly to these 2-cycles. Thus, every 2-cycle incident to  $v_a$  gets  $\mu_1 \log d_a / d_a$  from  $v_a$ . In a similar fashion we charge  $D_2$  to the 2-extension of 2-cycles. Let  $v_a v_b$  be an edge in  $G$  and let  $v_r \neq v_b$  be a vertex adjacent to  $v_a$ . Distributing  $\mu_2 \log d_r$  uniformly, assigns every 2-extension with “endpoint”  $v_r$  the fraction of  $\mu_2 \log d_r / (d_r(d_a - 1))$  from  $v_r$ . For  $D_3$  and  $D_4$  we argue analogously. We can therefore express  $D$  by

$$\begin{aligned} D_1 &= \mu_1 \sum_{i,j,A,B} f_{ij}(A, B) \left( \frac{\log i}{i} + \frac{\log j}{j} \right), \\ D_2 &= \mu_2 \sum_{i,j,A,B} f_{ij}(A, B) \left( \sum_{a_r \in A} \frac{\log a_r}{a_r(i-1)} + \sum_{b_r \in B} \frac{\log b_r}{b_r(j-1)} \right), \\ D_3 &= \mu_3 \sum_{i,j,k,A,B,C} f_{ijk}(A, B, C) \left( \frac{\log i}{i} + \frac{\log j}{j} + \frac{\log k}{k} \right), \\ D_4 &= \mu_4 \sum_{i,j,k,A,B,C} f_{ijk}(A, B, C) \left( \sum_{a_r \in A} \frac{\log a_r}{a_r(i-1)} + \sum_{b_r \in B} \frac{\log b_r}{b_r(j-1)} + \sum_{c_r \in C} \frac{\log c_r}{c_r(k-1)} \right). \end{aligned} \tag{5}$$

We can now express  $\log P_{\text{nc}}$  as sum over all signatures. This sum can be subdivided into one part that contains the  $f_{ij}$  variables and one part that contains the  $f_{ijk}$  variables. The part that considers the 2-cycles is given by

$$D1 + D2 + \sum_{i,j,A,B} f_{ij}(A, B) \log P_{ij}(A, B), \tag{G2}$$

and the part that considers the 3-cycles is given by

$$D3 + D4 + \sum_{i,j,k,A,B,C} f_{ijk}(A, B, C) \log P_{ijk}(A, B, C). \tag{G3}$$

For the restricted problems we only have 2-cycles. Using bound (4) and setting  $\mu_3 = \mu_4 = 0$ , we can bound the number of spanning trees by

$$D1 + D2 + \sum_{i,j,A,B} f_{ij}(A, B) \log \hat{P}_{ij}(A, B). \tag{R2}$$

## 2.4 Finding Constraints

In this section we construct *necessary* conditions for the  $f$  variables that have to hold for planar graphs with  $n$  vertices. We reuse the ideas from the charging scheme in Section 2.3. Instead of giving every vertex  $\log d_i$  to distribute, we assign to every vertex an amount of 1. This gives us a total of  $n$  units. Following the construction of the equations of (5) we obtain

$$\sum_{i,j,A,B} f_{ij}(A,B) \left( \frac{1}{i} + \frac{1}{j} \right) = n, \quad (\text{A2})$$

$$\sum_{i,j,k,A,B,C} f_{ijk}(A,B,C) \left( \frac{1}{i} + \frac{1}{j} + \frac{1}{k} \right) = n, \quad (\text{A3})$$

$$\sum_{i,j,A,B} f_{ij}(A,B) \left( \sum_{a_r \in A} \frac{1}{a_r(i-1)} + \sum_{b_r \in B} \frac{1}{b_r(j-1)} \right) = n, \quad (\text{B2})$$

$$\sum_{i,j,k,A,B,C} f_{ijk}(A,B,C) \left( \sum_{a_r \in A} \frac{1}{a_r(i-1)} + \sum_{b_r \in B} \frac{1}{b_r(j-1)} + \sum_{c_r \in C} \frac{1}{c_r(k-1)} \right) = n. \quad (\text{B3})$$

Another set of constraints is given by the number of 2-cycles and 3-cycles a planar graph can have, which is related to the number of edges and faces of  $G$ . Every 2-cycle is counted by some  $f_{ij}$  variable, hence the sum over all  $f_{ij}$  equals the number of edges, which we name  $m$ . Since we consider only 3-cycles of triangles, the sum of the  $f_{ijk}$  variables equals the number of triangles, which for a planar graph is at most  $2n$ . We obtain

$$\sum_{i,j,A,B} f_{ij}(A,B) \leq m, \quad (\text{C2})$$

$$\sum_{i,j,k,A,B,C} f_{ijk}(A,B,C) \leq 2n. \quad (\text{C3})$$

For the general case we have  $m \leq 3n$ , for the restricted case where quadrilaterals are allowed we have  $m \leq 2n$ , and for the remaining case we have  $m \leq 5n/3$ . All these bounds can be obtained by a simple double counting argument using Euler's formula. As trivial condition we restrict the  $f$  variables to be non-negative.

The constraints so far might be fulfilled by a signatures that does not come from a planar graph. In particular, the degree sequence of the graph induced by the cycles might be unrelated to the degree sequence of the graph induced by the 2-extensions. To overcome this ambiguity we consider the number of edges with vertex degree  $i$  at one vertex and degree  $j$  at the other. Let this number be  $n_{ij}$ . Clearly, we have that  $n_{ij} = \sum_{A,B} f_{ij}(A,B)$ , where the sum ranges about all feasible sequences  $A, B$ . On the other hand,  $n_{ij}$  can be counted by its appearances in the 2-extensions of 2-cycles. Every edge with degree  $(i, j)$  will show up in  $(i-1)+(j-1)$  2-extensions. Let  $\chi_i(X)$  denote the number of appearances of  $i$  in the sequence  $X$ . We can express  $((i-1)+(j-1))n_{ij}$  as  $\sum_{k,A,B} f_{ik}(A,B)\chi_j(A) + \sum_{k,A,B} f_{kj}(A,B)\chi_i(B)$ . This leads us to a new constraint of the form

$$(i+j-2) \sum_{A,B} f_{ij}(A,B) = \sum_{k,A,B} f_{ik}(A,B)\chi_j(A) + \sum_{k,A,B} f_{kj}(A,B)\chi_i(B). \quad (\text{Eij})$$

In the case where the smallest face of the graph is a pentagon, we were able to improve the solution of the linear program by adding the constraint (E33). Other constraints of the form (Eij) gave no improvement.

The solutions for the linear programs are included in the full version of this paper. We conclude with the main theorem.

**Theorem 1.** *Let  $G$  be a planar graph with  $n$  vertices. The number of spanning trees of  $G$  is at most  $O(5.28515^n)$ . If  $G$  is 3-connected and contains no triangle, then the number of its spanning trees is bounded by  $O(3.41619^n)$ . If  $G$  is 3-connected and contains no triangle and no quadrilateral, then the number of its spanning trees is bounded by  $O(2.71567^n)$ .*

### 3 Further Bounds and Future Work

The results of Theorem 1 improve several related upper bounds. Using the observations by Ribó *et al.* [13] we obtain the following bounds for grid embeddings of 3d polytopes.

**Corollary 1.** *Let  $G$  be the graph of a 3-polytope  $\mathcal{P}$  with  $n$  vertices.  $\mathcal{P}$  admits a realization as combinatorial equivalent polytope with integer coordinates and*

1. *no coordinate larger than  $O(147.7^n)$ ,*
2. *no coordinate larger than  $O(39.9^n)$ , if  $G$  contains a quadrilateral,*
3. *no coordinate larger than  $O(28.4^n)$ , if  $G$  contains a triangle.*

The number  $F(n)$  of cycle-free graphs in a planar graph with  $n$  vertices is bounded by the number of selections of at most  $n - 1$  edges from the graphs [1]. Thus,  $F(n) \leq \sum_{k=0}^{n-1} \binom{3n-6}{k}$ . For  $0 \leq q \leq 1/2$  we have that  $\sum_{i=0}^{\lfloor qm \rfloor} \binom{m}{qm} < 2^{H(q)m}$ , where  $H(q) := -\log(q)^q - \log(1-q)^{(1-q)}$  is the binary entropy (see for example [7, page 427]). This shows that,  $F(n) < 6.75^n$  by setting  $m = 3n$  and  $q = 1/3$ .

We give a better bound based on the bound for the number of spanning trees. We first bound the number  $F(n, k)$  of forests in  $\mathcal{G}_n$  with  $k$  edges. On one hand, the above argument yields an upper bound of  $F(n, k) \leq f_1(k) := \binom{3n-6}{k}$ . On the other hand, every forest with  $k$  edges can be constructed by selecting  $k$  edges from a spanning tree of  $\mathcal{G}_n$ . This gives as alternative bound  $F(n, k) \leq f_2(k) = \binom{n-1}{k} T(n)$ . Now, the number of cycle-free graphs is bounded by

$$F(n) = \sum_{k=0}^{n-1} F(n, k) \leq n \max_{0 \leq k \leq n} F(n, k) \leq n \max_{0 \leq q \leq 1} \min(f_1(qn), f_2(qn)).$$

We use  $\binom{n}{qn} \leq 2^{nH(q)}$  as upper bound for the binomial coefficients (see for example [6, page 1097]) to obtain

$$f_1(qn) < \hat{f}_1(qn) := 2^{3nH(q/3)} \quad \text{and} \quad f_2(qn) < \hat{f}_2(qn) := T(n)2^{nH(q)}.$$

The computed maximal value for the minimum of  $\hat{f}_1$  and  $\hat{f}_2$  is realized at  $qn = 0.94741 n$ . This yields a bound of  $n \cdot 6.4948^n$  for the number of cycle-free graphs. For the computation of these values we used numerical methods. Observe that  $\hat{f}_1(qn)$  realizes  $6.4948^n$  at a larger value  $q$  than  $\hat{f}_2(qn)$ . The correctness of the numerical computations follows from the monotonicity of  $\hat{f}_1$  and  $\hat{f}_2$  in  $(n/2, n]$ . For the number of plane spanning trees and cycle-free graphs on a planar point set, we obtain improved upper bounds by multiplying our bounds with the bound of  $O(30^n)$  on the maximum number of triangulations on a planar point set [16].

**Theorem 2.** *The number of cycle-free graphs in a planar graph with  $n$  vertices is bounded by  $n \cdot 6.4948^n$ . The number of plane spanning trees on  $n$  planar points is in  $O(158.6^n)$ , the number of cycle-free graphs is in  $O(194.7^n)$ .*

We expect better bounds for the number of cycle-free graphs in a planar graph from a more direct application of the outgoing edge approach. By adding a new vertex that is linked to a subset of the other vertices, every cycle-free graph can be turned into a spanning tree of the augmented graph. Without excluding any cycles we get a bound of  $7^n$ . Under the assumption that almost every vertex has degree 6, the refined outgoing edge method would yield a bound of  $6.5027^n$  when excluding 2-cycles and of  $6.4244$  when excluding 2 and 3-cycles. So far we were not able to check all constraints of the corresponding linear programs.

We finish our presentation with a discussion on how one could improve our results further. Since we consider only 2-cycles and 3-cycles from triangles, one would obtain a better bound for  $P_{nc}$  by taking also larger cycles into account. We do not expect to win anything by considering 3-cycles that are not triangles, because in the lower bound example (the wrapped up triangular grid) all 3-cycles are triangles. The analysis using larger cycles is more complicated and needs an extensive case distinction. Furthermore, we expect that there would be too many cases left for the brute force check. From our perspective, the following refinement seems tractable: Beside the 2-cycles, and 3-cycles on triangles, we also analyze 4-cycles that belong to two triangles sharing an edge (the diagonal). The 4-cycles can be analyzed by extending the events  $C_i$  for the 2-cycles to the following event: the  $i$ -th 2-cycle occurs, or the corresponding 4-cycle, whose diagonal is associated with the  $i$ -th cycle occurs. Assuming that the solution of the corresponding linear program is given by having almost every vertex degree 6, this would lead to  $\alpha = 5.25603$ . Since the resulting linear program is more complicated, the verification of the dual solutions is tedious.

Notice that Lemma 11 uses two enumerations of the events  $C_i$  to avoid the influence of the ordering. An elaborated enumeration scheme of the events  $C_i$  might give better bounds. Furthermore, we could consider “extension of extensions” to analyze larger locally connected pieces of the graph at once. This results in a powerful but very complicated incarnation of the outgoing edge approach.

The reader might think, that additional constraints in the linear programs might improve the outcome of our analysis. However, we expect that the solutions of the dual programs give the correct distribution of signatures. In particular, the solutions the dual programs match the candidates for the lower bound examples that were presented in [12].

*Acknowledgements.* We thank Günter Rote for suggesting this problem to us and for many inspiring and fruitful discussions on this subject.

## References

1. Aichholzer, O., Hackl, T., Huemer, C., Hurtado, F., Krasser, H., Vogtenhuber, B.: On the number of plane geometric graphs. *Graph. Comb.* 23(1), 67–84 (2007)
2. Bacher, R., de la Harpe, P., Nagnibeda, T.: The lattice of integral flows and the lattice of integral cuts on a finite graph. *Bull. Soc. Math. de France* 125, 167–198 (1997)
3. Biggs, N.: Algebraic potential theory on graphs. *Bull. London Math. Soc.* 29, 641–682 (1997)
4. Biggs, N.: Chip-firing and the critical group of a graph. *J. Algebraic Combin.* 9, 25–46 (1999)
5. Buchin, K., Knauer, C., Kriegel, K., Schulz, A., Seidel, R.: On the number of cycles in planar graphs. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 97–107. Springer, Heidelberg (2007)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
8. Grone, R., Merris, R.: A bound for the complexity of a simple graph. *Discrete Mathematics* 69(1), 97–99 (1988)
9. Horn, R.A., Johnson, C.R.: Matrix Analysis. Cambridge University Press, Cambridge (1985)
10. Lyons, R.: Asymptotic enumeration of spanning trees. *Combinatorics, Probability & Computing* 14(4), 491–522 (2005)
11. McKay, B.D.: Spanning trees in regular graphs. *Euro. J. Combinatorics* 4, 149–160 (1983)
12. Ribó Mor, A.: Realization and Counting Problems for Planar Structures: Trees and Linkages, Polytopes and Polyominoes. PhD thesis, Freie Universität Berlin (2006)
13. Ribó Mor, A., Rote, G., Schulz, A.: Embedding 3-polytopes on a small grid. In: Erickson, J. (ed.) Symposium on Computational Geometry, pp. 112–118. ACM, New York (2007)
14. Richter-Gebert, J.: Realization Spaces of Polytopes. Lecture Notes in Mathematics, vol. 1643. Springer, Heidelberg (1996)
15. Rote, G.: The number of spanning trees in a planar graph. In: Oberwolfach Reports, vol. 2, European Mathematical Society, Publishing House (2005)
16. Sharir, M., Sheffer, A.: Counting triangulations of planar point sets (2010) (manuscript)
17. Steinitz, E.: Encyclopädie der mathematischen Wissenschaften. In: Polyeder und Raumteilungen, pp. 1–139 (1922)
18. Suen, S.: A correlation inequality and a poisson limit theorem for nonoverlapping balanced subgraphs of a random graph. *Random Struct. Algorithms* 1(2), 231–242 (1990)
19. Tutte, W.T.: Convex representations of graphs. *Proceedings London Mathematical Society* 10(38), 304–320 (1960)
20. Tutte, W.T.: How to draw a graph. *Proceedings London Mathematical Society* 13(52), 743–768 (1963)
21. Whitney, H.: A set of topological invariants for graphs. *Amer. J. Math.* 55, 235–321 (1933)

# Contractions of Planar Graphs in Polynomial Time<sup>\*</sup>

Marcin Kamiński<sup>1, \*\*</sup>, Daniël Paulusma<sup>2</sup>, and Dimitrios M. Thilikos<sup>3, \*\*\*</sup>

<sup>1</sup> Département d’Informatique  
Université Libre de Bruxelles  
[Marcin.Kaminski@ulb.ac.be](mailto:Marcin.Kaminski@ulb.ac.be)

<sup>2</sup> Department of Computer Science  
University of Durham

[Daniel.Paulusma@durham.ac.uk](mailto:Daniel.Paulusma@durham.ac.uk)

<sup>3</sup> Department of Mathematics  
National and Kapodistrian University of Athens  
[sedthilk@math.uoa.gr](mailto:sedthilk@math.uoa.gr)

**Abstract.** We prove that for every graph  $H$ , there exists a polynomial-time algorithm deciding if a planar graph can be contracted to  $H$ . We introduce contractions and topological minors of embedded (plane) graphs and show that a plane graph  $H$  is an embedded contraction of a plane graph  $G$ , if and only if, the dual of  $H$  is an embedded topological minor of the dual of  $G$ . We show how to reduce finding embedded topological minors in plane graphs to solving an instance of the disjoint paths problem. Finally, we extend the result to graphs embeddable in an arbitrary surface.

**Keywords:** planar graph, dual graph, contraction, topological minor.

## 1 Introduction

An *edge contraction* of an edge  $e$  in a graph is the graph obtained by removing  $e$ , identifying its two endpoints, and eliminating parallel edges that may appear. Some basic properties of contractions are collected in [21]. Formally, for an edge  $e$  with endpoints  $u$  and  $w$ , the contraction of  $e$ , denoted by  $G/e$ , is the graph with vertex set  $V(G/e) = V(G) \setminus \{u, w\} \cup \{v_{uw}\}$  and edge set

$$\begin{aligned} E(G/e) = & E \setminus \{ \{x, y\} \in E : x \in \{u, w\}, y \in V \} \\ & \cup \{ \{v_{uw}, x\} : \{x, u\} \in E \vee \{x, w\} \in E \}. \end{aligned}$$

A graph  $H$  is a *contraction* of a graph  $G$  (or  $G$  is *contractible* to  $H$ ) if  $H$  can be obtained from  $G$  by a sequence of edge contractions.

---

\* This research was done while the two last authors were visiting the Département d’Informatique Université Libre de Bruxelles in January 2010.

\*\* Chargé de Recherches du FRS-FNRS.

\*\*\* Supported by the project “Kapodistrias” (AII 02839/28.07.2008) of the National and Kapodistrian University of Athens.

## 1.1 Previous Work

The problem of checking whether a graph is a contraction of another has already attracted some attention. In this subsection we briefly survey known results.

*Stars and triangle-free patterns.* Perhaps the first systematic study of contractions was undertaken by Brouwer and Veldman [2]. Here are two main theorems from that paper.

**Theorem 1 (Theorem 3 in [2]).** *A graph  $G$  is contractible to  $K_{1,m}$  if and only if  $G$  is connected and contains an independent set  $S$  of  $m$  vertices such that  $G - S$  is connected.*

In particular, a graph is contractible to  $P_3$  if and only if it is connected and is neither a cycle nor a complete graph. The theorem also allows to detect, in polynomial time, if a graph is contractible to  $K_{1,m}$ . It suffices to enumerate over all sets  $S$  with  $m$  independent vertices and check if the graph  $G - S$  is connected. This gives an  $|V(G)|^{\mathcal{O}(m)}$  algorithm, which is polynomial for every fixed  $m$ .

**Theorem 2 (Theorem 9 in [2]).** *If  $H$  is a connected triangle-free graph other than a star, then contractibility to  $H$  is NP-complete.*

Hence, checking if a graph is contractible to  $P_4$  or  $C_4$  is NP-complete. More generally, it is NP-complete for every bipartite graph with at least one connected component that is not a star.

*Patterns up to 5 vertices.* The research direction initiated by Brouwer and Veldman was continued by Levin, Paulusma, and Woeginger [10], [11]. Here is the main result established in these two papers.

**Theorem 3 (Theorem 3 in [10]).** *Let  $H$  be a connected graph on at most 5 vertices. If  $H$  has a dominating vertex, then contractibility to  $H$  can be decided in polynomial time. If  $H$  does not have a dominating vertex, then contractibility to  $H$  is NP-complete.*

However, the existence of a dominating vertex in the pattern  $H$  is not enough to ensure that contractibility to  $H$  can be decided in polynomial time. A pattern on 69 vertices for which contractibility to  $H$  is NP-complete was exhibited in [9].

*When the pattern is part of input.* Looking at contractions to fixed pattern graphs is justified by the following theorems proved by Matoušek and Thomas in [14].

**Theorem 4 (Theorem 4.1 in [14]).** *The problem of deciding, given two input graphs  $G$  and  $H$ , whether  $G$  is contractible to  $H$  is NP-complete even if we impose one of the following restrictions on  $G$  and  $H$ :*

- (i)  $H$  and  $G$  are trees of bounded diameter,
- (ii)  $H$  and  $G$  are trees all whose vertices but one have degree at most 5.

**Theorem 5 (Theorem 4.3 in [14]).** *For every fixed  $k$ , the problem of deciding, given two input graphs  $G$  and  $H$ , whether  $G$  is contractible to  $H$  is NP-complete even if we restrict  $G$  to partial  $k$ -trees and  $H$  to  $k$ -connected graphs.*

The authors also proved a positive result.

**Theorem 6 (Theorem 5.14 in [14]).** *For every fixed  $\Delta, k$ , there exists an  $\mathcal{O}(|V(H)|^{k+1} \cdot |V(G)|)$  algorithm to decide, given two input graphs  $G$  and  $H$ , whether  $G$  is contractible to  $H$ , when the maximum degree of  $H$  is at most  $\Delta$  and  $G$  is a partial  $k$ -tree.*

*Cyclicity.* The *cyclicity* of a graph  $G$  is defined as the largest integer  $k$  for which  $G$  is contractible to a cycle on  $k$  vertices. Exact values for some graphs and lower and upper bounds for some classes of graphs are given by Hammack in [8]. He also presented a polynomial-time algorithm for computing cyclicity of a planar graph.

*Non-recursive classes closed under taking of contractions.* Another type of containment relation close to contractions – induced minors – was studied by Matoušek, Nešetřil, and Thomas in [12]. A graph is an *induced minor* of another if the first is a contraction of an induced subgraph of the latter. The authors of [12] prove (Theorem 1.8) that there exists a class closed under taking of induced minors which is non-recursive (i.e. there is no algorithm to test the membership in this class). Clearly, a class of graphs closed under taking of induced minors is also closed under taking of contractions (and induced subgraphs) so we restate their result in the following way.

**Theorem 7 (Theorem 1.18 in [12]).** *There exists a non-recursive class of graphs closed under taking of contractions (and induced subgraphs).*

*Wagner’s Conjecture for contractions.* The statement usually referred to as *Wagner’s Conjecture* (although Klaus Wagner insisted he had never posed it, as explained in [4], p.355) is that for any infinite sequence  $G_0, G_1, \dots$  of graphs, there is a pair  $i, j$  such that  $i < j$  and  $G_i$  is a minor of  $G_j$ . The proof of Wagner’s Conjecture is one of the highlights of the Graph Minors project [20].

A contraction version of Wagner’s Conjecture was considered by Demaine, Hajiaghayi, and Kawarabayashi in [3]. They disproved this version showing the following.

**Theorem 8 (Theorem 31 in [3]).** *There is an infinite sequence  $G_0, G_1, \dots$  of connected graphs such that, for every pair  $i, j$  ( $i \neq j$ ),  $G_i$  is not a contraction of  $G_j$ .*

However, the authors also proved that the conjecture holds when the graphs in the sequence are required to be trees, or triangulated planar graphs, or 2-connected embedded outerplanar graphs.

## 1.2 Our Contribution

A graph is a *minor* of another if the first is a contraction of a subgraph of the latter. Graph Minors is a celebrated project by Robertson and Seymour

that is considered to be an important part of modern Graph Theory. One of the algorithmic consequences of Graph Minors is that, for every graph  $H$ , there exists a cubic-time algorithm deciding whether the input graph contains  $H$  as a minor [18]; and another, for every class of graphs closed under taking minors, there exists a cubic-time algorithm deciding whether the input graph belongs to this class [20].

While graph minors are well-studied both from combinatorial and algorithmic point of view, relatively little is known about graph contractions which are rather close to graph minors. Algorithmically, they are much less tractable compared to minors. As mentioned in the previous subsection, there are graphs for which it is NP-complete to decide if the input graph is contractible to them; and there are non-recursive classes of graphs, that are closed under taking of contractions, where is there is no algorithm deciding whether an input graph belongs to this class.

In this work we show, for a large class of inputs – graphs embeddable on surfaces, how to decide in polynomial time if a fixed graph is a contraction of the input. We focus on the case of planar graphs – graphs embeddable in the plane (or, equivalently, on the sphere). All the essential ingredients of the solution are already present when the input is constrained to be planar. In Section 5 we show how to extend the algorithm from graphs embeddable in the plane to graphs embeddable in an arbitrary surface.

The key idea is to introduce embedded versions of contractions and topological minors for plane graphs. Those embedded containment relations differ from usual contractions and topological minors in respecting the embedding. We show that a plane graph  $H$  is an embedded contraction of a plane graph  $G$ , if the dual of  $H$  is an embedded topological minor of the dual of  $G$ .

To use this duality algorithmically, we need to show that embedded topological minors can also be found in polynomial time. This is done by reducing the problem of finding an embedded topological minor to solving an instance of the disjoint path problem that, in turn, can be solved in cubic time due to the main algorithmic result of Graph Minors [19].

## 2 Definitions

*Basics.* We consider both simple graphs and multigraphs. We do not allow any of them to have loops. When there is no ambiguity, we say “a graph” and mean a simple graph or multigraph. We say “a multigraph” when we want to stress that multiple edges are allowed and “a simple graph” if they are not allowed. For a (multi)graph  $G$ , let  $V(G)$  be its vertex set and  $E(G)$  its edge (multi)set. Plane graphs are always assumed to be drawn on the unit sphere and their edges are arbitrary polygonal arcs (not necessarily straight line segments). For notation not defined here, we refer the reader to the monograph [1].

The dual of a plane graph  $G$  will be denoted by  $G^*$ . Notice that there is a one-to-one correspondence between the edges of  $G$  and the edges of  $G^*$ . We keep the convention that  $e^*$  is the edge of  $G^*$  corresponding to edge  $e$  of  $G$ .

A graph  $H$  is a *subdivision* of a graph  $G$ , when  $H$  can be obtained from  $G$  by subdividing its edges (i.e., replacing edges by paths). A graph  $H$  is a *topological minor* of a graph  $G$  if  $H$  is a subdivision of a subgraph of  $G$ . Vertices of degree  $\geq 3$  in a subdivision are called *branch vertices*.

In this paper we consider the algorithmic problem of contracting an input graph  $G$  to a fixed graph  $H$ . Below we will assume that both  $H$  and  $G$  are connected. This can be done without loss of generality. If  $G$  and  $H$  are not connected, we consider contracting different connected components of  $G$  to different connected components of  $H$ . Since  $H$  is fixed, this will only contribute to a constant (in  $|V(G)|$ ) factor in the computational complexity of the algorithm.

*Embeddings.* In this work, we only need to distinguish between essentially different embeddings of a planar graph. This motivates the following definition.

Two plane graphs  $G$  and  $H$  are *combinatorially equivalent* ( $G \simeq H$ ) if there exists a homeomorphism of the unit sphere (in which they are embedded) which transforms one into the other. The relation of being combinatorially equivalent is reflexive, symmetric and transitive, and thus an equivalence relation. Let  $\mathcal{G}$  be the class of all plane graphs isomorphic to a planar graph  $G$  and let us consider the quotient set  $\mathcal{G}/\simeq$ . The equivalence classes (i.e., the elements of the quotient set) can be thought of as *embeddings*. In fact, we will work with embeddings but for simplicity, we will pick a plane graph representative for each embedding.

*Homotopic edges and thin graphs.* Two edges of a plane graph are *homotopic edges* if they together bound a 2-face. Following [1], a *thin graph* is a plane multigraph without homotopic pairs of edges. In other words, if there are two parallel edges  $e, f$  between a pair of vertices in a thin graph, each of the two open regions defined by the union of  $e$  and  $f$  must contain at least one vertex. It turns out that thin plane multigraphs cannot have more edges than simple plane graphs.

**Lemma 1 (Lemma 5 in [1]).** *If  $G$  is a thin graph, then  $|E(G)| \leq 3|V(G)| - 6$ .*

*Embedded containment relations.* An *embedded contraction* of an edge  $e$  of a plane graph  $G$  is a plane graph  $G'$  that is obtained by homeomorphically mapping the endpoints of  $e$  in  $G$  to a single vertex without any edge crossings and recursively removing one of two homotopic edges, if a graph has such a pair. Notice that there are many embedded contractions of an edge of a plane graph  $G$  but they are all combinatorially equivalent.

An *embedded dissolution* of a vertex  $v$  of degree 2 in a plane graph  $G$  is an embedded contraction of one of the two edges  $v$  is incident with in  $G$ .

Let  $G$  and  $H$  be two plane graphs. We say that  $H$  is an *embedded contraction* of  $G$  ( $H \leq_{ec} G$ ), if  $H$  is combinatorially equivalent to a graph that can be obtained from  $G$  by a series of embedded contractions. We say that  $H$  is an *embedded topological minor* of  $G$  ( $H \leq_{etm} G$ ), if  $H$  is combinatorially equivalent to a graph that can be obtained from  $G$  by a series of vertex and edge deletions, and embedded dissolution of vertices of degree 2.

### 3 Contractions vs. Topological Minors

**Lemma 2.** *Let  $H$  and  $G$  be two thin graphs and  $H^*$ ,  $G^*$  their respective duals.*

$$H \leq_{ec} G \iff H^* \leq_{etm} G^*$$

*Proof.* Let  $G$  be a thin graph and  $e$  an edge of  $G$ . Let  $G_{/e}$  be an embedded contraction of  $e$  in  $G$ . Notice that  $G_{/e}^*$  is isomorphic to a plane graph obtained from  $G^*$  by deleting  $e^*$  and recursively applying embedded dissolutions of vertices of degree 2. (Homotopic faces in a plane graph correspond to vertices of degree 2 in its dual.) Let us also note that  $G_{/e}$  is a plane graph with no homotopic edges.

If  $H$  can be obtained from  $G$  by a series of embedded contractions, then  $H^*$  can be obtained from  $G^*$  by a series of edge deletions and embedded dissolutions of vertices of degree 2. Hence, if  $H$  is an embedded contraction of  $G$ , then  $H^*$  is an embedded topological minor of  $G^*$ . This proves the forward implication.

For the backward implication, suppose that  $H^*$  is an embedded topological minor of  $G^*$ ; that is  $H^*$  can be obtained from  $G^*$  by a sequence of vertex deletions, edge deletions, and embedded dissolutions of vertices of degree 2.

Let us notice that removing a vertex  $v$  in a thin plane graph can be simulated by removing all but two edges incident to  $v$ , then applying an embedded dissolution to  $v$  and removing the new edge. (No vertices of degree 1 will be created since the graph was thin and can be made thin after recursively applying embedded dissolution to vertices of degree 2.) Hence, a sequence of vertex deletions, edge deletions, and embedded dissolutions of vertices of degree 2 can be replaced by a sequence of edge deletions and embedded dissolutions of vertices of degree 2. The sequence can be rearranged and split into groups – every group consists of an edge removal and appropriate embedded dissolutions of vertices of degree 2. (When a graph has no homotopic edges, all vertices of its dual are of degree  $\geq 3$ .)

Each group of operations in a plane graph corresponds to an embedded edge contraction in its dual. The sequence of operations that transform  $G^*$  into  $H^*$  corresponds to a sequence of embedded edge contractions that brings  $G$  into  $H$ . Hence, the backward implication.  $\square$

A simple planar graph  $H$  is a *pattern* of a planar multigraph  $H'$ , if  $V(H) = V(H')$  and two vertices are adjacent in  $H$  if and only if they are adjacent in  $H'$ . In other words, a pattern of the multigraph is the graph obtained by replacing multiple with single edges. Let  $\mathcal{C}(H)$  be a maximal set of thin plane multigraphs whose pattern is  $H$  such that they are all combinatorially different.

**Lemma 3.** *For every planar graph  $H$ , the set  $\mathcal{C}(H)$  is finite.*

*Proof.* Combinatorially different embeddings of a planar multigraph  $H$  are determined by cyclic orders of neighbors on vertices. There might be infinitely many embeddings of a planar multigraph. However, we are confined to thin plane graphs only and each will have at most  $3|V(H)| - 6$  edges by Lemma 11. Hence, the number of possible different cyclic orderings is finite.  $\square$

**Theorem 9.** Let  $H$  and  $G$  be simple planar graphs and  $\tilde{G}$  a plane graph isomorphic to  $G$ . Then,

$$H <_c G \iff \exists \tilde{H} \in \mathcal{C}(H) \text{ such that } \tilde{H} <_{ec} \tilde{G}.$$

*Proof.* For the backward implication, let  $H$  be the pattern of some  $\tilde{H} \in \mathcal{C}(H)$ . ( $H$  is a simple graph.) Let us notice that if  $\tilde{H}$  is combinatorially equivalent to an embedded contraction of  $\tilde{G}$ , then  $\tilde{G}$  (and its abstract graph  $G$ ) are isomorphic to a contraction of  $H$ .

For the forward implication, let us assume that  $H <_c G$ . There exists a sequence of edge contractions that brings  $G$  into  $H$ . Let us apply the same sequence as a sequence of embedded contractions to  $\tilde{G}$  and call the resulting graph  $\tilde{T}$ . From the definition of embedded contraction,  $\tilde{T}$  is thin. Notice that its pattern is  $H$ . From the choice of  $\mathcal{C}(H)$ , there exists  $\tilde{H} \in \mathcal{C}(H)$  that is combinatorially equivalent to  $\tilde{T}$ .  $\square$

A direct consequence of Lemma 2 and Theorem 9 is the following corollary.

**Corollary 1.** Let  $H$  and  $G$  be planar graphs and  $\tilde{G}$  a plane graph isomorphic to  $G$ . Then,

$$H <_c G \iff \exists \tilde{H} \in \mathcal{C}(H) \text{ such that } \tilde{H}^* <_{etm} \tilde{G}^*.$$

## 4 Embedded Topological Minors and the Algorithm

In this section, we reduce the problem of finding an embedded topological minor to the the problem of finding a collection of disjoint paths in a graph. Here is a result from Graph Minors we will need later.

**Theorem 10 (19).** *There exists an algorithm that given a graph  $G$  and  $k$  pairs  $(s_1, t_1), \dots, (s_k, t_k)$  of vertices of  $G$  decides whether there are  $k$  vertex-disjoint paths  $P_1, \dots, P_k$  in  $G$  such that  $P_i$  joins  $s_i$  and  $t_i$ , for all  $i = 1, \dots, k$ , and if so, finds them. The algorithm runs in time  $\mathcal{O}(|V(G)|^3)$ .*

This result can be used to determine whether a graph  $H$  is a topological minor of the input graph. The idea is to choose a set of  $|V(H)|$  branch vertices in  $G$  and turn it into an instance of disjoint paths problem with  $|E(H)|$  paths, each edge of  $H$  should correspond to one path. The disjoint path algorithm from Theorem 10 needs to be run for every choice of branch vertices. The running time of the algorithm deciding whether a graph  $H$  is a topological minor of the input graph  $G$  is then  $\mathcal{O}(|V(G)|^{|V(H)|})$ . Note that the running time is indeed polynomial as we assume  $H$  to be a fixed graph. Whether there is an algorithm for this problem that runs in time  $f(|V(H)|) \cdot |V(G)|^{\mathcal{O}(1)}$  is one of the major open problems in the theory of parameterized complexity, even when  $G$  is assumed to be planar. The reduction from embedded topological minors to disjoint paths is more complicated since we have to take into account the cyclic order of paths incident with the vertex. Below we show how this can be done.

**Theorem 11.** *For every plane graph  $H$ , there exists a polynomial-time algorithm that given a plane graph  $G$  decides if  $H$  is an embedded topological minor of  $G$ , and if so, finds the subgraph which is a subdivision of  $H$ .*

*Proof.* A  $k$ -star is a connected bipartite graph whose one part has one vertex (the *center*) and the other part has  $k$  vertices (the *leaves*). A star is a graph that is a  $k$ -star for some  $k$ . A *labelled star* is a subgraph of  $G$  that is a star and whose center is labelled with a vertex from  $V(H)$  and whose leaves are labelled with different edges from  $E(H)$  that are incident with  $v$  in  $H$ . A labelled star  $Q$  is said to be *compatible with*  $v \in V(H)$  if it is a  $\deg(v)$ -star, its center is labelled with  $v$ , and the cyclic ordering of the labels on the leaves of  $Q$  is the same as the cyclic ordering of the edges incident with  $v$  in  $H$ .

Let us fix an ordering  $v_1, \dots, v_{|V(H)|}$  of  $V(H)$  and an ordering  $e_1, \dots, e_{|E(H)|}$  of  $E(H)$ . A *branching* is a  $|V(H)|$ -tuple  $(Q_{v_i} : i = 1, \dots, |V(H)|)$  such that  $Q_{v_i}$  for  $i = 1, \dots, |V(H)|$  is a labelled star compatible with  $v_i$ . A *good branching* is one in which no two centers of stars coincide. Let  $\mathcal{Q}$  be the set of all different good branchings. Notice that  $|\mathcal{Q}|$  is bounded by  $|V(G)|^{\mathcal{O}(|V(H)|)}$ .

For a branching from  $\mathcal{Q}$  we define an instance of the disjoint path problem. We start with  $|E(H)|$  pairs of terminals and later will be possibly removing some. For every  $j = 1, \dots, |E(H)|$ , let  $\{s_j, t_j\}$  be the two vertices of  $G$  that are labelled with  $e_j$  in the branching. We then remove from the set of pairs such  $\{s_j, t_j\}$  that  $s_j$  and  $t_j$  are adjacent. We then remove all centers of stars from  $G$ .

**Claim.**  $H$  is an embedded topological minor of  $G$  if and only if there exists a branching from  $\mathcal{Q}$  that defines a feasible disjoint path instance.

If  $H$  is an embedded topological minor of  $G$ , then the branching is given by the set of stars centered at the branch vertices of  $H$  whose edges incident with the center are those that belong to the model of  $H$  in  $G$ .

If there is a branching in  $\mathcal{Q}$  that defines a feasible disjoint path instance, then the union of the disjoint paths and the stars in the branching give a model of an embedded topological minor of  $H$  in  $G$ .

Now we are ready to present an algorithm that for a fixed graph  $H$  decides if a plane graph  $G$  contains  $H$  as an embedded topological minor of  $G$ . First the algorithm constructs the set  $\mathcal{Q}$ . Then, for every branching from  $\mathcal{Q}$  the algorithm constructs an instance of the disjoint paths problem and tests its feasibility.

The correctness of the algorithm is a direct consequence of the Claim. To see that the running time of the algorithm is polynomial in  $|V(G)|$ , notice that – as mentioned before – the cardinality of  $\mathcal{Q}$  is bounded by  $|V(G)|^{\mathcal{O}(|V(H)|)}$ ; building an instance of the disjoint paths problem out of a branching can be done in polynomial time; and testing feasibility of those instances can also be done in polynomial time by Theorem 10.  $\square$

**Theorem 12.** *For every graph  $H$ , there exists a polynomial-time algorithm that given a planar graph  $G$  decides whether  $H$  is a contraction of  $G$ , and if so finds a series of contractions transforming  $G$  into  $H$ .*

*Proof.* We can assume that both  $G$  and  $H$  are connected; otherwise, we consider contractions of different connected components of  $G$  to different connected

components of  $H$ . We can also assume that  $H$  is planar since  $G$  can never be contracted to a non-planar graph.

First we embed  $G$  in the plane using the linear-time algorithm from [15]. Let  $\tilde{G}$  be this plane graph isomorphic to  $G$  and  $\tilde{G}^*$  its dual. For every graph  $H$  from  $\mathcal{C}(H)$ , test if  $\tilde{H}^*$  is an embedded topological minor of  $\tilde{G}^*$ , using the algorithm from Theorem [11].

The correctness of the algorithm follows from Corollary [1] and Lemma [3]. The fact that the algorithm runs in polynomial time follows from Lemma [3] and Theorem [11].  $\square$

## 5 Bounded Genus Graphs

In this section, we show how to extend our result from Theorem [2] to graphs on surfaces other than the plane. For terminology and notions related to graphs on surfaces, we refer the reader to the standard monograph [13].

*Thin graphs on surfaces.* We fix a surface  $\Sigma$  of Euler genus  $g$  and consider graphs embeddable in this surface. First, we notice that it is possible to extend the definition of thin graphs to graphs embedded in other surfaces. A *thin graph* is a multigraph embeddable in  $\Sigma$  without homotopic pairs of edges. Then, we observe that the proof of Lemma [1] in [4] uses the Euler's formula only. Since all faces of a thin graph are incident with at least 3 edges, one can derive the following counterpart of Lemma [1] for graphs of genus  $g$ .

**Lemma 4.** *If  $G$  is a thin graph embeddable on a surface of Euler genus  $g$ , then*

$$|E(G)| \leq 3 \cdot (|V(G)| + g) - 6.$$

It is not difficult to see that this leads to the following equivalent of Lemma [3].

**Lemma 5.** *For every graph  $H$  embeddable on a surface of Euler genus  $g$ , the set  $\mathcal{C}(H)$  is finite.*

*Containment relations on surfaces.* The same definitions of embedded contractions and embedded topological minors that we provided for the plane stay valid for graphs embeddable in  $\Sigma$ . The main reason is that surfaces are locally homeomorphic to the plane and our definitions are also local. Also the dual graph of a graph embedded in  $\Sigma$  is well defined. It is easy to check that the proofs of Lemma [2] and Theorem [9] hold in case of any surface, not only the plane. A direct consequence of this is that Corollary [1] has a version for surfaces of higher genus.

**Corollary 2.** *Let  $H$  and  $G$  be graphs embeddable on a surface of Euler genus  $g$  and  $\tilde{G}$  a graph embedded on a surface of genus  $g$  and isomorphic to  $G$ . Then,*

$$H <_c G \iff \exists \tilde{H} \in \mathcal{C}(H) \text{ such that } \tilde{H}^* <_{etm} \tilde{G}^*.$$

*Algorithm.* It is also possible to adapt the proof of Theorem [11] to graphs embeddable in  $\Sigma$ . Combining these together, we can prove the following theorem.

**Theorem 13.** *For every integer  $g \geq 0$  and a graph  $H$ , there exists a polynomial-time algorithm that given a graph embeddable on a surface of Euler genus  $g$  decides whether  $H$  is a contraction of  $G$ , and if so finds a series of contractions transforming  $G$  into  $H$ .*

## 6 Discussion

We conclude with a number of remarks and a conjecture.

*Solution via dual.* We prove our result by investigating what operation in the dual graph  $G^*$  corresponds to contractions in  $G$ . We want to mention that the same approach proved to be successful in studying maximum cuts in planar graphs. A maximum cut of a graph is its maximum bipartite subgraph. Orlova and Dorfman [17] and independently Hadlock [7] noticed that a (maximum) bipartite subgraph in  $G$  is an (maximum) Eulerian subgraph in  $G^*$ . Maximum Eulerian subgraphs can be found in polynomial time, therefore they proved that the maximum cut problem can be solved in polynomial-time in planar graphs.

*Non-recursive classes of planar graphs.* We prove in this paper that for every graph  $H$ , there exists an algorithm that given a planar input graph  $G$  decides whether  $H$  is a contraction of  $G$ . To complement this result we would like to note that there are classes of planar graphs closed under taking of contractions that are non-recursive. A closer look at the proof of Theorem 7 in [12] reveals that the graphs in the non-recursive class from the theorem are in fact planar (and even have no  $K_5^-$  minor). We state it more formally.

**Corollary 3.** *There exists a non-recursive class of planar graphs closed under taking contractions.*

*Cyclicity in bounded genus graphs.* Cycles have a unique embedding into the plane (up to combinatorial equivalence) and the dual of a cycle on  $k$  vertices is the multigraph with two vertices and  $k$  parallel edges. Therefore computing cyclicity of a plane graph is equivalent to solving the maximum flow problem in the dual for every pair of vertices (as the source and sink).

Hammack, also using the maximum flow problem, showed how to compute cyclicity of a planar graph [8]. Our result for bounded genus graphs allow to extend this result to graphs embeddable in an arbitrary surface.

*Complexity of topological minor checking.* The total running time of our algorithm is  $\mathcal{O}(|V(G)|^{|V(H)|})$ . (We will focus on planar graphs but the discussion also holds for classes of graphs of bounded genus.) The computational complexity heavily depends on the complexity of computing topological minors. As mentioned before, the best algorithm for deciding whether a fixed graph  $H$  is a topological minor of the input graph  $G$  runs in time  $\mathcal{O}(|V(G)|^{|V(H)|})$ . If we consider the problem from the parameterized complexity point of view, that is asking for an  $f(|V(H)|) \cdot |V(G)|^{O(1)}$  step algorithm (i.e. classify it in

the complexity class FPT when parameterized by the size of  $H$ ), this is not satisfying. (We refer to [5][16] for more information on parameterized complexity.)

Whether topological minor checking belongs to the class FPT is not known even when the input graph is restricted to be planar. It is conceivable that an FPT algorithm for this problem would also give an FPT algorithm for embedded topological minor, and consequently, for contractions in surface embeddable graphs.

However, we believe that the existence of such an algorithm is rather unlikely and the problem is W[1]-hard. (W-hardness is a technical notion in the theory of parameterized complexity that makes it rather impossible that a problem belongs in FPT.) We state this as a conjecture.

**Conjecture.** *For a graph  $H$ , the problem of deciding whether  $H$  is a topological minor of a (planar) input graph  $G$  is W[1]-hard, when parameterized by  $|V(H)|$ .*

## Acknowledgements

We thank Samuel Fiorini for his kind support and Gwenaël Joret for stimulating discussions. We also gratefully acknowledge support from the Actions de Recherche Concertées (ARC) fund of the Communauté française de Belgique.

## References

1. Alber, J., Fellows, M.R., Niedermeier, R.: Polynomial-time data reduction for dominating set. *J. ACM* 51(3), 363–384 (2004)
2. Brouwer, A.E., Veldman, H.J.: Contractibility and NP-completeness. *Journal of Graph Theory* 11(1), 71–79 (1987)
3. Demaine, E.D., Hajaghayi, M., Kawarabayashi, K.-i.: Algorithmic graph minor theory: Improved grid minor bounds and Wagner’s contraction. *Algorithmica* 54(2), 142–180 (2009)
4. Diestel, R.: *Graph Theory*, Electronic edn. Springer, Heidelberg (2005)
5. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
6. Flum, J., Grohe, M.: Parameterized complexity theory. In: *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin (2006)
7. Hadlock, F.: Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.* 4(3), 221–225 (1975)
8. Hammack, R.: Cyclicity of graphs. *J. Graph Theory* 32(2), 160–170 (1999)
9. van ’t Hof, P., Kamiński, M., Paulusma, D., Szeider, S., Thilikos, D.M.: On contracting graphs to fixed pattern graphs. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) *SOFSEM 2010. LNCS*, vol. 5901, pp. 503–514. Springer, Heidelberg (2010)
10. Levin, A., Paulusma, D., Woeginger, G.J.: The computational complexity of graph contractions I: Polynomially solvable and NP-complete cases. *Networks* 51(3), 178–189 (2008)
11. Levin, A., Paulusma, D., Woeginger, G.J.: The computational complexity of graph contractions II: Two tough polynomially solvable cases. *Networks* 52(1), 32–56 (2008)

12. Matoušek, J., Nešetřil, J., Thomas, R.: On polynomial-time decidability of induced-minor-closed classes. *Comment. Math. Univ. Carolin.* 29(4), 703–710 (1988)
13. Mohar, B., Thomassen, C.: *Graphs on Surfaces*. The Johns Hopkins University Press, Baltimore (2001)
14. Matousek, J., Thomas, R.: On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics* 108(1-3), 343–364 (1992)
15. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.* 12(1), 6–26 (1999)
16. Niedermeier, R.: *Invitation to fixed-parameter algorithms*. Oxford Lecture Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2006)
17. Orlova, G., Dorfman, Y.: Finding the maximum cut in a graph. *Tekhnicheskaya Kibernetika (Engineering Cybernetics)* 10, 502–506 (1972)
18. Robertson, N., Seymour, P.D.: Graph minors XII. Distance on a surface. *J. Comb. Theory, Ser. B* 64(2), 240–272 (1995)
19. Robertson, N., Seymour, P.D.: Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B* 63(1), 65–110 (1995)
20. Robertson, N., Seymour, P.D.: Graph minors XX. Wagner’s conjecture. *J. Comb. Theory, Ser. B* 92(2), 325–357 (2004)
21. Wolle, T., Bodlaender, H.L.: A note on edge contraction. Technical Report UU-CS-2004-028, Department of Information and Computing Sciences, Utrecht University (2004)

# Communication Complexity of Quasirandom Rumor Spreading\*

Petra Berenbrink<sup>1</sup>, Robert Elsässer<sup>2</sup>, and Thomas Sauerwald<sup>1</sup>

<sup>1</sup> Simon Fraser University, Burnaby, Canada

{petra,tsauerwa}@cs.sfu.ca

<sup>2</sup> University of Freiburg, Germany

elsa@informatik.uni-freiburg.de

**Abstract.** We consider rumor spreading on random graphs and hypercubes in the quasirandom phone call model. In this model, every node has a list of neighbors whose order is specified by an adversary. In step  $i$  every node opens a channel to its  $i$ th neighbor (modulo degree) on that list, beginning from a randomly chosen starting position. Then, the channels can be used for bi-directional communication in that step. The goal is to spread a message efficiently to all nodes of the graph.

We show three results. For random graphs (with sufficiently many edges) we present an address-oblivious algorithm with runtime  $O(\log n)$  that uses at most  $O(n \log \log n)$  message transmissions. For hypercubes of dimension  $\log n$  we present an address-oblivious algorithm with runtime  $O(\log n)$  that uses at most  $O(n(\log \log n)^2)$  message transmissions. For hypercubes we also show a lower bound of  $\Omega(n \log n / \log \log n)$  on the total number of message transmissions required by any  $O(\log n)$  time address-oblivious algorithm in the standard random phone call model. Together with a result of [8], our results imply that for random graphs and hypercubes the communication complexity of the quasirandom phone call model is significantly smaller than that of the standard phone call model. This seems to be surprising given the small amount of randomness used in our model.

## 1 Introduction

In this paper we consider rumor spreading (a.k.a. randomized broadcast) in random graphs and hypercubes. This problem is motivated by overlay topologies in peer to peer (P2P) systems, in which each node possesses a list of neighboring peers. Our goal is to develop time-efficient rumor spreading algorithms which produce a minimal number of message transmissions and use a small amount of randomness. Since P2P networks are decentralized platforms for sharing data and computing resources, it is very important to provide efficient, simple, and robust rumor spreading algorithms for P2P overlays. Minimization of the number of transmission (communication complexity) is important for applications

---

\* The second author was partially supported by the German Research Foundation (DFG) under contract EL 399/2-1.

such as the maintenance of replicated databases in which often huge amounts of broadcasts are necessary to deal with frequent updates in the system.

We assume the *quasirandom phone call model*, a variant of the *standard phone call model*. Let us first introduce the standard phone call model (also known as random phone call model, see [5]). In this model, each node  $v$  may perform the following actions in every step: 1) create a new rumor to be spread, 2) establish a communication channel between itself and one randomly chosen neighbor, 3) transmit a message over incident channels (opened by  $v$  or by some neighbor of  $v$ ) and 4) close the channel opened in the current step. Note that open channels can be used for bi-directional communications. *Calling nodes* (i.e., the nodes that opened the channels) can send their messages to their neighbors. These are called **push** transmissions. *Called nodes* can also perform so called **pull** transmissions, i.e., they send the message to the calling nodes. These transmissions are simply called **pull** transmissions. In the phone call model (both standard and quasirandom) it is assumed that nodes can combine several rumors to one larger message. Nodes can send messages over all their open channels in one time step.

The major challenge for rumor spreading algorithms in the phone call model is to decide whether or not a node should forward the rumor over an open communication channel. An algorithm is called *address-oblivious* (see [15]) if the decision of node  $v$  to send a rumor over an open channel  $(v, w)$  or  $(w, v)$  does not depend on  $w$ . However, this decision can depend on the communication partners chosen in earlier rounds or on decisions made so far. Hence, according to such an algorithm a node has to decide whether to use a channel without knowing if the rumor is already known by the neighbor in question. If there are only very few rumors in the network, then many communication channels may be established without ever being used for transmissions. Thus, the phone call model is especially of interest in situations where rumors are frequently generated. Then, the cost of establishing communication channels amortizes over all message transmissions.

In the case of the *quasirandom* phone call model it is assumed that every node has a cyclic list of all its neighbors, whose order is specified by an adversary. At the beginning, each node  $v$  chooses a random position in the list, independently of the other nodes. Assume that  $1 \leq \ell \leq d$  is the random choice of node  $v$ , where  $d$  is the degree of  $v$ . Then  $v$  communicates in step  $i$  with the neighbor  $(i + \ell) \bmod d + 1$  from the list. To create the list we assume that the adversary has total knowledge about the topology of the network, but cannot foresee any node's random choice w.r.t. the position selected at the beginning (cf. [6]).

In this paper we show three results. For random graphs (with sufficiently many edges) we present an algorithm with runtime  $O(\log n)$  that uses at most  $O(n \log \log n)$  message transmissions. For  $\log n$ -dimensional hypercubes we devise an algorithm with runtime  $O(\log n)$  that uses at most  $O(n \cdot (\log \log n)^2)$  message transmissions. Both algorithms are oblivious and complete rumor spreading in the quasirandom phone call model with probability  $1 - n^{-\Omega(1)}$ . For hypercubes we show a lower bound of  $\Omega(n \log n / \log \log n)$  on the number of message

transmissions required by any  $O(\log n)$  time oblivious algorithm in the standard phone call model. In [8] Elsässer shows a similar lower bound for oblivious rumor spreading algorithms on random graphs in the standard phone call model. Hence our results imply that for random graphs and hypercubes the communication complexity of the quasirandom phone call model is substantially smaller than of the standard phone call model. This seems to be surprising given the small amount of randomness used in our model.

## 1.1 Related Work

Due to space constraints, we consider only results which focus on the analytical study of **push** and **push & pull** algorithms.

*Runtime.* Most rumor spreading studies analyze the runtime of the **push** algorithm in the standard phone call model for different graph classes. For complete graphs of size  $n$ , Pittel [17] shows that (with probability  $1 - o(1)$ ) it is possible to spread a rumor in time  $\log_2(n) + \ln(n) + f(n)$ , where  $f(n)$  is a slow growing function, improving a result of Frieze and Grimmett [12]. In [11], Feige et al. determine asymptotically optimal upper bounds for the runtime on  $G(n, p)$  graphs (i.e., traditional Erdős-Rényi random graphs [10]), bounded degree graphs, and hypercubes, which all hold w.h.p.<sup>1</sup>. Recently, Fountoulakis et al. [13] prove a tighter bound for the runtime on sufficiently dense  $G(n, p)$  graphs, similar to the result of [17] for complete graphs. Very recently, Chierichetti et al. [4] show that the runtime of the combined **push & pull** model is  $O(\Phi^{-1} \cdot \log n \cdot \text{polylog}(\Phi^{-1}))$  w.h.p. for any graph  $G$ , where  $\Phi$  denotes the conductance of  $G$ .

In [6], Doerr et al. analyze the so called quasirandom rumor spreading. They show that for hypercubes and  $G(n, p)$  graphs  $O(\log n)$  steps suffice to inform every node, w.h.p. These bounds are similar to the ones in the standard phone call model (**push** model). The results of [6] are extended to further graph classes with good expansion properties in [7]. Note that in [6, 7] the authors mainly concentrate on the runtime efficiency, and the best known algorithms there require  $\Theta(n \log n)$  message transmissions in hypercubes and  $G(n, p)$  graphs.

*Number of Message Transmissions.* Karp et al. [15] note that in complete networks the **pull** approach is inferior to the **push** approach until roughly  $n/2$  nodes receive the rumor. Then the **pull** approach becomes superior. They present a **push & pull** algorithm, together with a termination mechanism, which bounds the number of total transmissions to  $O(n \log \log n)$  (w.h.p.), and show that this result is asymptotically optimal.

For sparser graphs and the standard phone call model it is not possible to get an oblivious algorithm that uses  $O(n \log \log n)$  message transmissions, together with a runtime of  $O(\log n)$ . In [8] Elsässer considers random  $G(n, p)$  graphs and shows a lower bound of  $\Omega(n \log n / \log(pn))$  message transmissions for oblivious rumor spreading algorithms with a runtime of  $O(\log n)$ . For  $p > \log^2 n / n$  he

---

<sup>1</sup> W.h.p. or “with high probability” means with probability at least  $1 - n^{-c}$  for some constant  $c > 0$ .

develops an oblivious algorithm that spreads a rumor in time  $O(\log n)$  using  $O(n \cdot (\log \log n + \log n / \log(pn)))$  transmissions, w.h.p.

In [9] the authors consider a simple modification of the standard phone call model, called RANDOM[4], where every node is allowed to open a channel to *four different* randomly chosen neighbors in every time step. For  $G(n, p)$  graphs with  $p > \log^2 n/n$ , they show that this modification results in a reduction of the number of message transmissions down to  $O(n \log \log n)$ . Similar results are shown for random  $d$ -regular graphs in [1].

The authors of [2] present an extension of RANDOM[4] which they call RR model. In their model each node has a randomly ordered cyclic list with all its neighbors. In step  $i$ , the node opens a communication channel to the  $i$ th neighbor in its list. The RR model is the same as the quasirandom model except that the adversarial order is replaced by the random order. The authors present an oblivious algorithm for graphs with very good edge and node expansion properties which has a runtime  $O(\log n)$  and which uses  $O(n\sqrt{\log n})$  message transmissions, w.h.p. The authors establish a lower bound of  $\Omega(n\sqrt{\log n / \log d})$  on the number of message transmissions for oblivious rumor spreading algorithms (assuming a runtime of  $O(\log n)$ ), showing that their upper bound is tight up to a  $\sqrt{\log \log n}$  factor if  $d$  is polylogarithmic in  $n$ . Since on these graphs all time efficient algorithms known so far may lead to a communication overhead of  $\Theta(n \log n / \log \log n)$ , this result shows that avoiding the re-opening of previously used channels makes it possible to reduce the number of message transmissions per node by almost a quadratic factor.

The algorithms of [2, 8, 9, 15] spread the rumor using push transmissions until a constant fraction of the nodes receives the rumor (we call these nodes *informed* in the following). Then the algorithms spread the rumor via pull transmissions until every node is informed. To save on communications, the algorithms of [1, 2, 8, 9] only allow each node  $v$  a certain number of transmissions which depends on the age the rumor had at the time  $v$  received it for the first time.

## 1.2 Model

In this paper we consider random graphs  $G(n, p) = (V, E)$  and hypercubes  $H_d$  of dimension  $d$ . A random graph  $G(n, p)$  consists of  $n$  nodes. The probability that any pair of nodes is connected is  $p$ . For simplicity, we assume  $(\log^2 n)/n \leq p \leq 2^{o(\sqrt{\log n})}/n$  in this extended abstract, although our results can be generalized to a larger regime of  $p$ . The expected number of edges for  $G(n, p)$  is  $pn \cdot (n - 1)$ . Let  $d(v)$  be the degree of node  $v$  and  $N(v)$  be the set of neighbors in  $V$ . For  $S \subset V$ , let  $N(S)$  be the set of neighbors of nodes in  $S$ . Let  $\alpha$  be the node expansion value of  $G(n, p)$ . Then  $\alpha = \min_{S \in V, |S| \leq n/4} N(S)/|S|$ . It is known that for our choice of  $p$ ,  $\alpha$  is a constant close to 1 w.h.p. (3).

The  $d$ -dimensional hypercube  $H_d$  consists of  $n = 2^d$  many nodes. A binary string of length 2 is assigned to every node and two nodes are connected if their binary strings differ in *exactly* one bit. Hence, the degree of any node of  $H_d$  is  $d$ . Note that hypercubes have much smaller expansion than random graphs.

We assume that every node has an estimation of  $n$  which is accurate to within a constant factor. We also assume that all nodes have access to a global clock, and that they work synchronously. As communication model we assume a variant of the phone call model. In the standard phone call model (see [5]) in each step  $t$  every node can create an arbitrary amount of rumors to be spread. To measure the communication cost we only count the number of message transmissions, i.e., opening a channel is not counted. Following [1, 2, 8, 15], we assume here that new pieces of information are generated frequently in the network, and then the cost of establishing communication channels amortizes over all message transmissions. However, we only concentrate on the distribution and lifetime of a single rumor.

The quasirandom variant of the phone call model considered in this paper was introduced in [6]. In the quasirandom phone call model every node  $v$  has a list  $\tilde{L}_v = \tilde{L}_v[0], \tilde{L}_v[1], \dots, \tilde{L}_v[d(v) - 1]$  of length  $d(v)$  with all its neighbors. The order of that list is arbitrary, i.e., it may be determined by an adversary. For spreading the rumor, every node  $v$  chooses a random position  $i_v$  in the list, independently of the other nodes. For its  $j$ -th communication  $v$  will open a channel to node  $L_v[(i_v + j - 1) \bmod d(v)]$ . We define  $L_v = L_v[0], L_v[1], \dots, L_v[d(v)]$  as the list beginning at neighbor  $i_v$ .

Nodes that received the rumor will be called *informed*. By  $I_t$  ( $H_t$ ) we denote the set of informed (uninformed) nodes in step  $t$ . Furthermore, let  $I_t^+$  be the set of nodes that receive the rumor *for the first time* in step  $t$ . These nodes will also be called *newly informed* nodes.

### 1.3 Our Contribution

In this paper we show the following results. For random graphs we present an oblivious algorithm (in the quasirandom model) that spreads a rumor in time  $O(\log n)$  using  $O(n \log \log n)$  message transmissions, w.h.p. Compared to [6], we reduce the number of message transmissions by a factor of  $\log n / \log \log n$ . Moreover, our upper bound in the quasirandom model is significantly smaller than the lower bound for the standard phone call model (cf. [8]).

For the hypercube we show a result that is slightly weaker than our result for random graphs. We present an oblivious algorithm (which is similar to the algorithm for random graphs) that spreads a rumor in time  $O(\log n)$  using  $O(n \cdot (\log \log n)^2)$  message transmissions, w.h.p. We also show that in the standard phone call model, any oblivious algorithm with runtime  $O(\log n)$  requires  $\Omega(n \log n / \log \log n)$  message transmissions. The communication complexity of this problem has not been analyzed before, neither in the standard nor in the quasirandom phone call model. Therefore the best known algorithms require  $O(\log n)$  time, but produce  $\Omega(n \log n)$  message transmissions. In comparison to that, we reduce the number of message transmissions by a factor of  $\log n / (\log \log n)^2$ . Again, our algorithm outperforms the lower bound on the communication complexity in the standard phone call model.

Our two results demonstrate that on two important networks, rumor spreading can be done much more efficiently in the quasirandom phone model than in the standard phone call model. From a higher level, the results provide evidence that avoiding previously chosen communication partners is more important than choosing all communication partners independently and uniformly at random.

## 2 Random Graphs

In this section we present an algorithm with runtime  $\mathcal{O}(\log n)$  and communication complexity  $\mathcal{O}(n \log \log n)$  for random graphs. Note that all the results presented in this section can be generalized to expanders in which the girth is large ( $\Omega(\log \log n)$ ). The details are omitted in this extended abstract.

### 2.1 Our Algorithm

We assume that the rumor we want to spread is generated at time 0, i.e., at time  $t$  the age of the rumor equals  $t$ . The algorithm describes the behavior of the nodes w.r.t. one specific rumor. Each node is, depending on the age of the rumor, in one of the following phases:

In the following algorithm,  $\rho$  is a sufficiently large constant.

*Phase 0:  $[age \leq \lceil \rho \log n \rceil]$*  The node which generates the rumor performs **push** in each step of this phase. No other node transmits the rumor in this phase.

*Phase 1:  $[\lceil \rho \log n \rceil + 1 \leq age \leq 2 \cdot \lceil \rho(\log n + 320) \rceil]$*  Nodes that received the rumor in Phase 0 use the first 320 steps of this phase to perform **push** in each of these steps. If a node receives a rumor for the *first* time in step  $t \in \{\lceil \rho \log n \rceil + 1, \dots, 2 \cdot \lceil \rho \log n \rceil\}$ , then the node performs **push** in the *exactly* 320 next steps.

*Phase 2:  $[\lceil 2 \lceil \rho(\log n + 320) \rceil + 1 \leq age \leq 2 \cdot \lceil \rho \log n + \rho \log \log n \rceil]$*  Every informed node performs **push** in every step of this phase.

*Phase 3:  $[\lceil 2 \lceil \rho \log n + \rho \log \log n \rceil + 1 \leq age \leq 3 \cdot \lceil \rho \log n \rceil]$*  Every node which becomes informed in this phase performs **pull**, i.e., it sends the message over all incoming channels. All other informed nodes perform **pull** over all incoming channels with a probability of  $1/\log n$ .

*Phase 4:  $[\lceil 3 \lceil \rho \log n \rceil + 1 \leq age \leq 3 \cdot \lceil \rho \log n + \rho \log \log n \rceil]$*  All informed nodes perform **pull** transmissions.

It is easy to see that at the end of Phase 0, exactly  $\rho \log n + 1$  nodes are informed (Observation 1). In Phase 1 we inform half of the nodes (see Lemma 1). At the end of Phase 2 we have  $n \cdot (1 - 2 \log \log n / \log n)$  informed nodes, w.h.p. (Lemma 2). Phase 3 and Phase 4 are analyzed in Lemma 3. There we show that w.h.p. at the end of Phase 4 all nodes are informed.

## 2.2 Analysis of the Algorithm

For a graph  $G(n, p)$  and our choice of  $p$  the degree of each node is  $np \cdot (1 \pm o(1))$ , with probability  $1 - n^{-3}$ . For simplicity we ignore the  $1 \pm o(1)$  factor in our analysis and assume  $d = pn$ .

**Theorem 1.** *Assume that  $G = G(n, p)$  with  $(\log^2 n)/n \leq p \leq 2^{o(\sqrt{\log n})}/n$ . The algorithm above spreads a rumor in  $G$  in time  $O(\log n)$  using  $O(n \log \log n)$  message transmissions, w.h.p.*

In the rest of this section we will prove the above theorem. The proof is split into several lemmata. It is easy to see that in Phase 0 the node that generated the rumor informs  $\rho \log n$  different neighbors, which results in the following observation.

**Observation 1.** *At the end of Phase 0 there are  $\rho \log n$  informed nodes.*

Now we concentrate on Phase 1 and show the following lemma.

**Lemma 1.** *With probability  $1 - n^{-2}$ , at least  $n/2$  nodes are informed at the end of Phase 1.*

*Proof.* Assuming that the nodes all have a degree  $d$  we show that

1. After the first  $\rho \cdot (\log n)/2$  steps at least  $6n/d$  nodes are informed, where  $\rho > 8$ .
2. After  $\rho \cdot ((\log n)/2 - 320)$  additional steps we have at least  $n/40$  informed nodes.
3. After the last  $320 \cdot \rho$  steps we have  $n/2$  informed nodes for  $\rho$  large enough.

*Part 1).* This follows from Claim A.1 of [2].

*Part 2).* In this case the number of informed nodes lies in the range  $[6n/d, n/40]$ . We show inductively that with a very high probability the number of informed nodes grows by a factor of 2.1 every 160 steps. To do so we divide the time into  $\ell = (\rho \cdot ((\log n)/2 - 320))/160$  subphases. For  $0 \leq i \leq \ell$ , subphase  $\tau_i$  starts in step  $\rho \cdot (\log n)/2 + 160i + 1$  and ends in step  $\rho \cdot (\log n)/2 + 160(i+1)$ . Let  $I_{\tau_i}^+$  be the newly informed nodes in Subphase  $\tau_i$ , and  $I_{\tau_i}$  be the informed nodes at the beginning of Subphase  $\tau_i$ . Note that all nodes in  $I_{\tau_i}^+$  perform a push transmissions in Subphase  $\tau_{i+1}$ .

We show by induction that for  $0 \leq i \leq \ell$  we have  $|I_{\tau_i}^+| \geq 2.1 \cdot |I_{\tau_i}|$ , which then implies that  $|I_{\tau_i}^+| \geq |I_{\tau_{i+1}}|/2$ .

Fix a subphase  $\tau_{i+1}$ . One can show that there are  $n/6$  uninformed nodes at the beginning of the subphase such that, with probability  $1 - \varepsilon^n$ , all of these nodes have at least  $|I_{\tau_i}^+| \cdot d/(2n)$  neighbors in the set of nodes  $I_{\tau_i}^+$ . (Due to space limitations, we do not prove this claim.) Hence, such an uninformed node remains uninformed in the time interval  $\tau_{i+1}$  with probability at most  $(1 - 160/d)^{|I_{\tau_i}^+| \cdot d/(2n)}$ . This holds since the first positions are chosen independently

and uniformly at random, and a neighbor misses a specific node in 160 steps with probability  $1 - 160/d$ . Thus,

$$\begin{aligned}\mathbf{E} [|I_{\tau_{i+1}}^+|] &\geq \left(1 - \left(1 - \frac{160}{d}\right)^{|I_{\tau_i}^+| \cdot d / (2n)}\right) \cdot \frac{n}{6} \\ &\geq \left(1 - \left(\frac{1}{e}\right)^{80|I_{\tau_i}^+|/n}\right) \cdot \frac{n}{6} \geq \left(1 - \left(\frac{1}{e}\right)^{40|I_{\tau_{i+1}}|/n}\right) \cdot \frac{n}{6} \\ &\geq \left(1 - \left(1 - \frac{1}{n/(40 \cdot |I_{\tau_{i+1}}|) + 1}\right)\right) \cdot \frac{n}{6} > 2.2 \cdot |I_{\tau_{i+1}}|\end{aligned}$$

Here, the third equation uses the induction hypothesis. Using Azuma-Hoeffding ([16]) we obtain with probability  $1 - o(n^{-3})$  that  $|I_{\tau_{i+1}}^+| \geq 2.1 \cdot |I_{\tau_{i+1}}|$ .

*Part 3).* Now the number of informed nodes lies in the range  $[n/40, n/2]$ . We divide the time into  $\ell = 2\rho$  subphases. For  $0 \leq i \leq \ell$ , subphase  $\tau_i$  starts in step  $\rho \cdot (\log n/2 - 320) + 160i + 1$  and ends in step  $\rho \cdot (\log n/2 - 320) + 160(i + 1)$ . Our goal is to show inductively that for all but the last phase  $|I_{\tau_i}^+| \geq 2.1 \cdot |I_{\tau_i}|$ . In the last phase we inform enough nodes so that half of the nodes are informed at the end of this phase.

Similar to Part 2) we fix a subphase  $\tau_{i+1}$  and define  $H_{\tau_{i+1}}$  as the number of uninformed nodes at the *beginning* of Subphase  $\tau_{i+1}$ . One can show that  $|H_{\tau_{i+1}}|/2$  of the uninformed nodes have at least  $|I_{\tau_i}^+|d/(2n)$  neighbors in the set of nodes  $I_{\tau_i}^+$ , with probability  $1 - \varepsilon^n$  (again, we omit the proof of this claim due to space limitations). Such an uninformed node remains uninformed in  $\tau_{i+1}$  with probability at most  $(1 - 160/d)^{|I_{\tau_i}^+|d/(2n)}$ . Thus,

$$\begin{aligned}\mathbf{E} [|I_{\tau_{i+1}}^+|] &\geq \left(1 - \left(1 - \frac{160}{d}\right)^{|I_{\tau_i}^+|d/(2n)}\right) \cdot \frac{|H_{\tau_{i+1}}|}{2} \\ &\geq \left(1 - \left(\frac{1}{e}\right)^{80|I_{\tau_i}^+|/n}\right) \cdot \frac{|H_{\tau_{i+1}}|}{2} \geq \left(1 - \left(\frac{1}{e}\right)^{40|I_{\tau_{i+1}}|/n}\right) \cdot \frac{|H_{\tau_{i+1}}|}{2}.\end{aligned}$$

The remainder of the proof is a case analysis depending on  $|I_{\tau_{i+1}}|$ . If  $n/40 \leq |I_{\tau_{i+1}}| \leq n/10$ , then

$$\left(1 - \left(\frac{1}{e}\right)^{40|I_{\tau_{i+1}}|/n}\right) \cdot \frac{|H_{\tau_{i+1}}|}{2} \geq \left(1 - \left(\frac{1}{e}\right)\right) \cdot \frac{9n}{20} \geq \frac{2.2 \cdot n}{10}.$$

Using the method of bounded independent differences [16] one can show that with probability  $1 - o(n^{-3})$  it holds that  $|I_{\tau_{i+1}}^+| \geq 2.1 \cdot |I(\tau_{i+1})|$ . For  $n/10 < |I_{\tau_{i+1}}| \leq n/6$

$$\left(1 - \left(\frac{1}{e}\right)^{40|I_{\tau_{i+1}}|/n}\right) \cdot \frac{|H_{\tau_{i+1}}|}{2} \geq \left(1 - \left(\frac{1}{e}\right)^4\right) \cdot \frac{5n}{12} \geq \frac{2.2 \cdot n}{6}.$$

Then, with probability  $1 - o(n^{-3})$  we have  $|I_{\tau_{i+1}}^+| \geq 2.1 \cdot |I_{\tau_{i+1}}|$  [16].

For  $|I_{\tau_{i+1}}| \geq n/6$  we get

$$\begin{aligned} & |I_{\tau_{i+1}}| + \left(1 - \left(\frac{1}{e}\right)^{40|I_{\tau_{i+1}}|/n}\right) \cdot \frac{|H_{\tau_{i+1}}|}{2} \\ & \geq |I_{\tau_{i+1}}| + \left(1 - \left(\frac{1}{e}\right)^{40/6}\right) \cdot \left(\frac{n - |I_{\tau_{i+1}}|}{2}\right) \geq \frac{41n}{80}. \end{aligned}$$

Again, we obtain with probability  $1 - o(n^{-3})$  that  $|I_{\tau_{i+2}}| > n/2$ .  $\square$

**Lemma 2.** *Assume  $\rho \geq 30$ . With probability  $1 - n^{-2}$ , there are at most  $(n \cdot 2 \log \log n / \log n)$  uninformed nodes at the end of Phase 2.*

*Proof.* Note that in this phase every informed node performs a **push** transmission in every step. Let  $T$  be a random variable defined as the time step in which  $|I_T| > n/2$  for the first time (this happens w.h.p. in Phase 1). Let  $\tau$  be the time interval  $[T - 160, T]$ . For the sake of this proof we assume that only the nodes of  $I_\tau^+$  perform **push** transmissions in this phase. Due to Lemma 1,  $|I_\tau^+| > n/5$ .

One can show that, with probability  $1 - \varepsilon^n$  ( $\varepsilon > 0$  is a constant) there are at most  $n \cdot \log \log n / \log n$  nodes in  $H_T$  which have fewer than  $d/10$  neighbors in  $I_\tau^+$ . After  $\rho \log \log n$  additional steps each of the other (uninformed) nodes remains uninformed with probability at least

$$\left(1 - \frac{\rho \log \log n}{d}\right)^{d/10} \leq e^{-\rho \log \log n / 10} < \log^{-3} n,$$

for  $\rho \geq 30$ . Thus, if there are at most  $n \cdot \log \log n / \log n$  nodes in  $H_T$  which have fewer than  $d/10$  neighbors in  $I_\tau^+$ , the expected number of new informed nodes in Phase 2 is at least

$$\left(|H_T| - \frac{n \log \log n}{\log n}\right) \cdot (1 - \log^{-3} n).$$

Then, using [16] one can show that with probability at least  $1 - n^{-2}$ , the number of newly informed nodes in this phase is at least

$$\left(|H_T| - \frac{2n \log \log n}{\log n}\right).$$

Hence with probability at least  $1 - n^{-2}$ , the number of uninformed nodes after this phase is at most  $n \cdot 2 \log \log n / \log n$ .  $\square$

Finally, we concentrate on Phases 3 and 4.

**Lemma 3.** *Assume  $\rho \geq 30$ . With probability  $1 - n^{-2}$  all nodes are informed at the end of Phase 4.*

*Proof.* For a node  $u$  and time interval  $\tau = [t, t']$ , let  $L_u(\tau)$  be the set of nodes chosen by  $u$  in steps  $\tau = t, t+1, \dots, t'$ . Define  $t_2 = 3\rho \cdot (\log n + \log \log n)$  as the end of Phase 4,  $t_1 = 3 \cdot \rho \log n$  as the beginning of Phase 4, and  $t_0 = 2\rho(\log n + \log \log n)$  as the beginning of Phase 3.

First we consider Phase 4 and divide the time interval  $[t_1 + 1, t_2]$  into  $k' = (t_2 - t_1)/320$  subintervals of length 320. For any  $0 \leq i \leq k' - 1$  we define

$$\tilde{\tau}_i = [t_2 - 320i, t_2 - 320 \cdot (i + 1) + 1].$$

For a node  $v$ , let

$$U_0(v) = L_v[\tilde{\tau}_0] \text{ and } U_i(v) = \cup_{w \in U_{i-1}} L_w[\tilde{\tau}_i].$$

We can visualize  $\cup_{i \leq k' - 1} U_i(v)$  as tree of depth  $k' - 1$  rooted in  $v$ . The level  $i$  nodes are the nodes in  $U_i(v)$ . Then, one can show that  $|U_{k'-1}(v) \cap H_{t_0}| = \Omega(\log^3 n)$  with probability  $1 - o(n^{-3})$ .

In the following we consider two cases. In the first case, we assume that  $\cup_{i \leq k' - 1} U_i(v) \cap I_{t_0} \neq \emptyset$  for some node  $v$ . Then  $v$  is informed in Phase 4 since all informed nodes perform pull transmissions in that phase. In the second case, let  $U_{k'-1}(v) \cap I_{t_0} = \emptyset$ . For this case we show that in Phase 4  $v$  will be the root of a communication tree consisting of nodes which are still all uninformed in step  $t_0$ . Then we will show that w.h.p. at least one of the leaves of the tree will be informed in Phase 3. The rumor will be propagated to  $v$  via the path between  $v$  and the informed leaf.

Now we need some additional definitions. We divide the time interval  $[t_0 + 1, t_1]$  into  $k'' = (t_1 - t_0)/160$  rounds of length 160. For any  $0 \leq i \leq k'' - 1$

$$\tilde{\tau}'_i = [t_1 - 160i, t_1 - 160 \cdot (i + 1) + 1].$$

For  $0 \leq i \leq \rho \log n$ , let

$$\begin{aligned} \tilde{U}_{-1}^H(v) &= U_{k'-1}(v) \\ \tilde{U}_i^H(v) &= \cup_{w \in \tilde{U}_{i-1}^H(v)} L_w[\tilde{\tau}'_i] \cap H_{t_0} \\ \tilde{U}_i^I(v) &= \cup_{w \in \tilde{U}_{i-1}^H(v)} L_w[\tilde{\tau}'_i] \cap I_{t_0}. \end{aligned}$$

A node  $\tilde{w}_i \in \tilde{U}_i^I(v)$  is connected to a node  $\tilde{w}_{-1} \in \tilde{U}_{-1}^H(v)$  by a path  $P = (\tilde{w}_i, \dots, \tilde{w}_0, \tilde{w}_{-1})$ , where  $\tilde{w}_{i-1}, \dots, \tilde{w}_0, \tilde{w}_{-1} \in H_{t_0}$ , and  $\tilde{w}_{j+1} \in L_{\tilde{w}_j}(\tilde{\tau}'_{j+1})$ . Now define

$$\tilde{U}_{0 \rightarrow i}^H(v) = \cup_{j=0}^i \tilde{U}_j^H(v) \quad \text{and} \quad \tilde{U}_{0 \rightarrow i}^I(v) = \cup_{j=0}^i \tilde{U}_j^I(v).$$

Since  $|\tilde{U}_{-1}^H(v)| = \Omega(\log^3 n)$ , we can apply the same techniques as in Lemma 11 and obtain that

$$|\tilde{U}_i^H(v) \cup \tilde{U}_i^I(v)| \leq 2.1 \cdot |\tilde{U}_{i-1}^H(v)|$$

for any  $i \geq 1$  as long as  $|\tilde{U}_{i-1}^I(v)| = O(\log^2 n)$  and  $|\tilde{U}_i^H(v)| < n/40$ . However, since  $|H_{t_0}| \leq 2n \log \log n / \log n$ , there exists some  $i < k''$  such that  $|\tilde{U}_{0 \rightarrow i}^I(v)| >$

$\rho \log^2 n$ . Then, we can argue that every node  $u \in \tilde{U}_{0 \rightarrow i}^I(v)$  performs pull transmissions with probability  $1/\log n$ . Since for every  $u$  a path  $(u, \tilde{w}_s, \dots, \tilde{w}_0, \dots, v)$  with  $s < k''$  exists, which consists of nodes of  $H_{t_0}$ , all the nodes on this path will perform pull transmisions. Hence, there is a node  $u \in L_{\tilde{w}_s}(\tilde{\tau}'_{s+1})$  which transmits the rumor at the right time, with probability

$$1 - \left(1 - \frac{1}{\log n}\right)^{\rho \log^2 n} = 1 - o(n^{-3}),$$

if  $\rho$  is large enough.  $\square$

Let us now prove Theorem 1. The correctness (every node gets informed w.h.p.) follows from the lemmata above. It remains to analyze the total number of message transmissions. In Phase 0, the algorithm uses  $\mathcal{O}(\log n)$  message transmissions. In Phases 1, 2 and 4, the algorithm uses  $\mathcal{O}(n \log \log n)$  message transmissions. By Lemma 2, we know that after Phase 2 at most  $\mathcal{O}(n \log \log n / (\log n))$  uninformed nodes remain. These nodes generate at most  $\mathcal{O}(n \log \log n)$  message transmissions in Phase 3. Using a Chernoff bound, we can show that the nodes that are informed at the end of Phase 2 use at most  $\mathcal{O}(n)$  message transmissions. Hence the total number of message transmissions is  $\mathcal{O}(n \log \log n)$ .  $\square$

### 3 Hypercubes

In this section we first present a lower bound on the communication complexity for rumor spreading on hypercubes in the standard phone call model and then an upper bound (which is much smaller) in the quasirandom phone call model.

**Theorem 2.** *Let  $G = (V, E)$  be a  $d$ -dimensional hypercube with  $n = 2^d$  nodes. Any algorithm in the standard phone call model with runtime  $O(\log n)$  requires  $\Omega(n \log n / \log \log n)$  message transmissions, with probability at least  $1 - n^{-\omega(1)}$ .*

The proof of Theorem 2 is omitted due to space limitations.

#### 3.1 Our Algorithm

In this section we present our upper bound on the communication complexity for rumor spreading in the quasirandom phone call model. In the algorithm below, the total number of message transmissions is  $\mathcal{O}(n(\log \log n)^2)$ , which can be shown as in the proof of Theorem 1 above.

*Phase 1:*  $[1 \leqslant \text{age} \leqslant \lceil \rho \log n \rceil]$  If a node receives a rumor for the *first* time in step  $t \in \{\lceil \rho \log n \rceil + 1, \dots, 2 \cdot \lceil \rho \log n \rceil\}$ , then the node performs **push** in the *exactly*  $C \log \log n$  next steps.

*Phase 2:*  $\lceil \rho \log n \rceil + 1 \leqslant \text{age} \leqslant 2 \cdot \lceil \rho \log n \rceil$  Every node which becomes informed in this phase performs **pull** over each incoming channel. All other informed nodes perform **pull** with a probability of  $1/\log n$  over each incoming channel.

*Phase 3:*  $2 \lceil \rho \log n \rceil + 1 \leqslant \text{age} \leqslant 2 \cdot \lceil \rho \log n + \rho(\log \log n)^2 \rceil$  All informed nodes perform **pull** transmissions in every step of this phase.

### 3.2 Analysis of the Algorithm

**Theorem 3.** Assume that  $H_d$  is a hypercube of dimension  $\log n$ . The algorithm above spreads a rumor in  $H_d$  in time  $O(\log n)$  using  $O(n(\log \log n)^2)$  message transmissions, w.h.p.

The proof of Theorem 3 is omitted due to space limitations. The analysis is similar to the one for random graphs. However, the lack of strong expansion properties makes it more difficult and one has to resort to the special structure and the symmetries of hypercubes.

## References

- [1] Berenbrink, P., Elsässer, R., Friedetzky, T.: Efficient Randomised Broadcasting in Random Regular Networks with Applications in Peer-to-Peer Systems. In: Proc. of PODC 2008, pp. 155–164 (2008)
- [2] Berenbrink, P., Elsässer, R., Sauerwald, T.: Randomised Broadcasting: Memory vs. Randomness. In: Proc. of LATIN 2010 (to appear, 2010)
- [3] Bollobás, B.: Random Graphs. Academic Press, London (1985)
- [4] Chierichetti, F., Lattanzi, S., Panconesi, A.: Almost Tight Bounds for Rumour Spreading with Conductance. In: Proc. of STOC 2010 (to appear, 2010)
- [5] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proc. of PODC 1987, pp. 1–12 (1987)
- [6] Doerr, B., Friedrich, T., Sauerwald, T.: Quasirandom Rumor Spreading. In: Proc. of SODA 2008, pp. 773–781 (2008)
- [7] Doerr, B., Friedrich, T., Sauerwald, T.: Quasirandom rumor spreading: expanders, push vs. pull, and robustness. In: Proc. of ICALP 2009, pp. 366–377 (2009)
- [8] Elsässer, R.: On the communication complexity of randomized broadcasting in random-like graphs. In: Proc. of SPAA 2006, pp. 148–157 (2006)
- [9] Elsässer, R., Sauerwald, T.: The power of memory in randomized broadcasting. In: Proc. of SODA 2008, pp. 218–227 (2008)
- [10] Erdős, P., Rényi, A.: On random graphs I. Publ. Math. Debrecen 6, 290–297 (1959)
- [11] Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized broadcast in networks. Random Structures and Algorithms 1(4), 447–460 (1990)
- [12] Frieze, A.M., Grimmett, G.R.: The shortest-path problem for graphs with random arc-lengths. Discrete Applied Mathematics 10, 57–77 (1985)
- [13] Fountoulakis, N., Huber, A., Panagiotou, K.: Reliable broadcasting in random networks and the effect of density. In: Proc. of INFOCOM 2010 (to appear, 2010)
- [14] Hagerup, T., Rüb, C.: A guided tour of Chernoff bounds. Information Processing Letters 36(6), 305–308 (1990)
- [15] Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumor spreading. In: Proc. of FOCS 2000, pp. 565–574 (2000)
- [16] McDiarmid, C.: On the method of bounded differences. In: Surveys in Combinatorics. London Math. Soc. Lectures Notes, vol. 141, pp. 148–188. Cambridge Univ. Press, Cambridge (1989)
- [17] Pittel, B.: On spreading rumor. SIAM Journal on Applied Mathematics 47(1), 213–223 (1987)

# A Complete Characterization of Group-Strategyproof Mechanisms of Cost-Sharing<sup>\*</sup>

Emmanouil Pountourakis<sup>1, \*\*</sup> and Angelina Vidali<sup>2, \*\*\*</sup>

<sup>1</sup> Department of EECS, Northwestern University, Evanston IL, USA  
[manolis@u.northwestern.edu](mailto:manolis@u.northwestern.edu)

<sup>2</sup> Max-Planck-Institut für Informatik, Saarbrücken, Germany  
[angelina@mpi-inf.mpg.de](mailto:angelina@mpi-inf.mpg.de)

**Abstract.** We study the problem of designing group-strategyproof cost-sharing mechanisms. The players report their bids for getting serviced and the mechanism decides a set of players that are going to be serviced and how much each one of them is going to pay. We determine three conditions: *Fence Monotonicity*, *Stability* of the allocation and *Validity* of the tie-breaking rule that are necessary and sufficient for group-strategyproofness, regardless of the cost function. Consequently, Fence Monotonicity characterizes group-strategyproof cost-sharing schemes closing an important open problem. Finally, we use our results to prove that there exist families of cost functions, where any group-strategyproof mechanism has arbitrarily poor budget balance.

## 1 Introduction

Algorithmic Mechanism Design [1] is a field of Game Theory, that tries to construct algorithms for allocating resources that give to the players incentives to report their true interest in receiving a good, a service, or in participating in a given collective activity. The pivotal constraint when designing a mechanism for any problem is that it is truthful. Truthfulness also known as strategyproofness requires that no player can strictly improve her utility by lying. In many settings this single requirement for an algorithm to be truthful restricts the repertoire of possible algorithms dramatically [2–5].

In settings, where truthfulness does not impose such severe limitations, like for example in Cost-sharing problems, it is desirable to construct mechanisms that are also resistant to manipulation by groups of players. A group of players forms a *successful coalition* when the utility of *each* player in the group does not decrease and the utility of *at least one* player *strictly* increases. Group-strategyproofness naturally generalizes truthfulness by requiring that no group

---

\* Supported in part by IST-2009-215270.

\*\* This work was done while the author was studying at the University of Athens.

\*\*\* Supported by a research scholarship from the Alexander von Humboldt Foundation.

of players can form a successful coalition by lying, when the values of the other players are fixed.

In this paper we study the following problem: A set of  $n$  customers/players are interested in receiving a service. Players report their valuation of the service and the mechanism decides the players that are going to be serviced and the price that each one of them will pay, i.e., we consider direct revelation mechanisms. We want to characterize all possible mechanisms that satisfy group-strategyproofness via identifying necessary and sufficient conditions for the payment functions that give rise to these mechanisms.

We provide a complete characterization of group-strategyproof mechanisms closing an open question posed in [6], [7, Chapter 15]. Immorlica et. al. [6] identified the property of semi-cross-montonicity, a local property that should be satisfied by the payment part (cost-sharing scheme) of every group-strategyproof mechanism. We introduce a generalization of semi-cross-monotonicity, Fence Monotonicity, which still refers only to the payment part of the mechanism, and is not only *necessary* but also *sufficient* for group-strategyproofness. Given any payment rule that satisfies Fence Monotonicity, we show that every allocation that satisfies *Stability* and *Validity* of the tie-breaking rule, yields a group-strategyproof mechanism. Our characterization of group-strategyproofness is regardless of the cost function and without any additional constraints (like fixed tie-breaking rules [6,8]).

Our results provide a new, as general as it can be, framework for designing group-strategyproof mechanisms. Thus, it opens new perspectives to the study of the very important problem of Cost-Sharing, the study of which was initiated by Moulin [9], where we additionally have a cost function  $C$  such that for each subset of players  $S$  the cost for providing service to all the players in  $S$  is  $C(S)$ . However, the strength of our results is that they apply to any cost function, since throughout our proof we do not make any assumptions at all about this cost function. We believe that our work here can be the starting point for constructing new interesting classes of mechanisms for specific cost-sharing problems.

As our characterization is the first characterization of group-strategyproofness, it is interesting to make a parallelization between the many known characterizations of strategyproofness [2,5,10,11] and our characterization of group-strategyproofness, not however, in the technical level, but in order to determine the quality and usefulness of our characterization. Fence Monotonicity, is a condition rather similar to Cycle Monotonicity [10], in the sense that both are conditions that should be satisfied by all possible restrictions of the output space of the mechanism. A great virtue for a characterization of a mechanism that uses money is to be able to separate the payment from the allocation part. Cycle Monotonicity is a necessary property for the allocation. If it is satisfied, we know a way to define truthful payments [10]. Fence Monotonicity refers only to the payments of a mechanism and if it is satisfied, Stability and Validity of the tie-breaking rule can be used to determine a group-strategyproof allocation. Managing to separate the payments from the allocation and avoiding to add any additional restrictions in the characterization we propose are undoubtedly its great virtues.

We believe that Fence Monotonicity, can also be a starting point in the quest for alternative characterizations, as Cycle monotonicity has done so far [2–5][11]. Unfortunately, even though Fence Monotonicity is succinct in its description, it is much more complicated than Cycle Monotonicity. Nonetheless, group-strategyproofness is a notion much more complicated than strategyproofness and we believe that Fence Monotonicity is not only important and unavoidable but also useful. We demonstrate the latter by proving the first of its kind very simple lower bound on the budget balance of any group-strategyproof mechanism.

**Our results and related work.** The design of group-strategyproof mechanisms for cost-sharing was first discussed by Moulin and Shenker [9][12]. Moulin [9] defined a condition on the payments called cross-monotonicity, which states that the payment of a serviced player should not increase as the set of serviced players grows. Any mechanism with payments satisfying cross-monotonicity can be easily turned to a simple mechanism called after Moulin. A Moulin mechanism first checks if it can provide service to all players, so that each one has non-negative utility and if not, it gradually diminishes the set of players that are candidates to be serviced, by throwing away at each step a player that cannot pay to get serviced (and who because of cross-monotonicity still cannot pay if the set of candidates becomes smaller). In fact, if the cost function is sub-modular and 1-budget balanced, then the only possible group-strategyproof mechanisms are Moulin mechanisms [12]. The great majority of cost-sharing mechanisms proposed are Moulin mechanisms.

Nevertheless, recent results showed that for several important cost-sharing games Moulin mechanisms can only achieve a very bad budget balance factor [6][13][14]. Some alternative, very interesting, and much more complicated in their description mechanisms that are group-strategyproof but not Moulin have been proposed [6][15], however, these do not exhaust the class of group-strategyproof mechanisms. In this work we introduce *Fencing Mechanisms*, a new general framework for designing group-strategyproof mechanisms, that generalizes Moulin mechanisms [9].

Recently, Mehta et. al. [16] proposed the notion of weak group-strategyproofness, that relaxes group-strategyproofness. It regards a formation of a coalition, as successful, when *each* player who participates in the coalition *strictly* increases her personal utility. They also introduce acyclic mechanisms, a general framework for designing weakly group-strategyproof mechanisms, however, the question of determining all possible weakly group-strategyproof mechanisms is an important question that remains open. Another alternative notion was proposed by Bleischwitz et. al. in [17].

The problem we solve here, i.e., finding a complete characterization of the cost-sharing schemes that give rise to a group-strategyproof mechanism was a major open problem posed in [6][7, Chapter 15]. Many interesting results arose in the attempt to find such a characterization [8][15]. In contrast to previous characterization attempts that characterized mechanisms satisfying some additional boundary constraints [6][8], our characterization is complete and succinct.

The only previously known complete characterization was for the case of two players [8, 15]. It remains open how our characterization can help for constructing new efficient mechanisms for specific cost-sharing problems. We believe that it can significantly enrich the repertoire of mechanisms with good budget balance guarantees for specific problems.

In the notion of group-strategyproofness it is important to understand that ties play a very important role. This is in contrast to strategyproofness, where ties can be in most cases broken arbitrarily (see for example [18]). An intuitive way to understand this is that a mechanism designer of a group-strategyproof mechanism expects a player to tell a lie in order to help the other players increase their utility, even when she would not gain any profit for herself. This player is at a tie but decides strategically if she should lie or not. Consequently, a characterization that assumes *a priori* a tie-breaking rule, and hence, greatly restricts the repertoire of possible mechanisms [6, 19] might be useful for specific problems and easier in its statement, but can never capture the very notion group-strategyproofness.

Our proofs are involved and based on set-theoretic arguments and the repeated use of induction. The main difficulty of our work was to identify some necessary and sufficient conditions for group-strategyproof payments that are also succinct to describe and add to our understanding of the notion of group-strategyproofness. In proving the necessity of Fence Monotonicity, we first have to prove lemmas that also reveal interesting properties of the allocation part of the mechanism. A novel tool that we introduce is the *harm relation* that generalizes the notion of negative elements defined in [6]. Proving that Fencing Mechanisms, i.e., mechanisms with payments satisfying Fence Monotonicity, and allocation satisfying Stability and Validity of the tie-breaking rule, are group-strategyproof turns out to be rather complicated. Due to space limitations, the full details of our proofs are omitted and we refer the reader to the full version of the paper. It is likely that our techniques can be extended for characterizing weakly group-strategyproof mechanisms.

## 2 Defining the Model

**The Mechanism.** Suppose that  $\mathcal{A} = \{1, 2, \dots, n\}$  is a set of players interested in receiving a service. Each of the players has a private type  $v_i$ , which is her valuation for receiving the service.

A cost sharing mechanism  $(O, p)$  consists of a pair of functions,  $O : \mathbb{R}^n \rightarrow 2^{\mathcal{A}}$  that associates with each bid vector  $b$  the set of serviced players and  $p : \mathbb{R}^n \rightarrow \mathbb{R}^n$  that associates with each bid vector  $b$  a vector  $p(b) = (p_1(b), \dots, p_n(b))$ , where the  $i$ -th coordinate is the payment of player  $i$ . Assuming quasi-linear utilities, each player wants to maximize the quantity  $v_i a_i - p_i(b)$  where  $a_i = 1$  if  $i \in O(b)$  and  $a_i = 0$  if  $i \notin O(b)$ .

As it is common in the literature and in order to avoid absurd mechanisms, we concentrate on mechanisms that satisfy the following very simple conditions [6, 9, 12]: *Voluntary Participation (VP)*: A player that is not serviced is not charged

( $i \notin O(b) \Rightarrow p_i(b) = 0$ ) and a serviced player is never charged more than her bid ( $i \in O(b) \Rightarrow p_i(b) \leq b_i$ ). *No Positive Transfer (NPT)*: The payment of each player  $i$  is non-negative ( $p_i(b) \geq 0$  for all  $i$ ). *Consumer Sovereignty (CS)*: For each player  $i$  there exists a value  $b_i^* \in \mathbb{R}$  such that if she bids  $b_i^*$ , then it is guaranteed that player  $i$  will receive the service no matter what the other players bid.<sup>1</sup>

In accordance with [6,9] we will concentrate on mechanisms, where the players are additionally able to deny receiving service, no matter what the other players bid. To ensure this we just assume that players can report negative bids. Then VP and NPT imply that if a player announces a negative amount, she will not receive the service. Even though negative bids may not seem realistic, they can model the denial of revealing any information to the mechanism. To put things simply, if a player  $i$  bids  $-1$ , then she does not receive service and if she bids  $b_i^*$ , then she receives service, no matter what the other players bid.

We are interested in mechanisms that are *group-strategyproof (GSP)*. A mechanism is GSP if for every two valuation vectors  $v, v'$  and every coalition of players  $S \subseteq \mathcal{A}$ , satisfying  $v_i = v'_i$  for all  $i \notin S$ , one of the following is true: (a) There is some  $i \in S$ , such that  $v_i a'_i - p_i(v') < v_i a_i - p_i(v)$  or (b) for all  $i \in S$ , it holds that  $v_i a'_i - p_i(v') = v_i a_i - p_i(v)$ .

**The cost function, cost sharing scheme, and budget balance.** The cost of providing service is given by a cost function  $C : 2^{\mathcal{A}} \rightarrow \mathbb{R}^+ \cup \{0\}$ , where  $C(S)$  specifies the cost of providing service to all players in  $S$ .

A *cost-sharing scheme* is a function  $\xi : \mathcal{A} \times 2^{\mathcal{A}} \rightarrow \mathbb{R}^+ \cup \{0\}$  such that for every  $S \subset \mathcal{A}$  and every  $i \notin S$ , we have  $\xi(i, S) = 0$ . Immorlica et. al. [6] showed that in any GSP mechanism of our setting the payment of a player depends only on the allocation produced by the mechanism for a particular bid vector and not directly on the bids of the players. As a consequence, the payments of any GSP mechanism are given by a cost-sharing scheme  $\xi$ , i.e.,  $p_i(b) = \xi(i, O(b))$ .

A desirable property cost-sharing schemes with respect to some cost function is budget balance. We say that a cost-sharing scheme  $\xi$  is  $\alpha$ -*budget balanced*, where  $0 \leq \alpha \leq 1$ , if for  $S \subseteq \mathcal{A}$  it holds that  $\alpha \cdot C(S) \leq \sum_{i \in S} \xi(i, S) \leq C(S)$ . Correspondingly, we say that a mechanism is  $\alpha$ -budget balanced if its cost sharing scheme is  $\alpha$ -budget balanced. We chose to define the cost function last in order to stress that our results are completely independent of the cost function and any budget-balance assumption, thus they apply to any cost-sharing problem.

An important and well-studied property of cost-sharing schemes is cross-monotonicity, which is sufficient for group-strategyproofness [9]. A cost-sharing scheme is *cross-monotonic* if  $\xi(i, S) \geq \xi(i, T)$  for every  $S \subset T \subseteq \mathcal{A}$  and every player  $i \in S$ . This means that the payment of a player cannot increase as the number of players that receive service increases.

In the attempt to provide a characterization of GSP mechanisms Immorlica et. al. [6] provided a partial characterization and identified semi-cross-monotonicity,

<sup>1</sup> From VP it holds that  $b_i^* \geq \max_{b \in \mathbb{R}^n} p_i(b)$ . It is easy to verify that strategyproofness implies that any value greater than  $b_i^*$  satisfies CS for player  $i$ . Thus, when we refer to this crucial value, we will w.l.o.g. assume that this inequality is strict.

an important condition that should be satisfied by the cost-sharing scheme of any GSP mechanism. A cost sharing scheme  $\xi$  is *semi-cross-monotonic* if for every  $S \subseteq \mathcal{A}$ , and every player  $i \in S$ , either for all  $j \in S \setminus \{i\}$ ,  $\xi(j, S \setminus \{i\}) \leq \xi(j, S)$  or for all  $j \in S \setminus \{i\}$ ,  $\xi(j, S \setminus \{i\}) \geq \xi(j, S)$ .

As we later show in Proposition 1(i), semi-cross-monotonicity can be almost directly derived by Fence Monotonicity, the property we introduce in this work.

### 3 Our Characterization

#### 3.1 Fence Monotonicity, and Fencing Mechanisms

Fence Monotonicity, considers each time a restriction of the mechanism that can only output as the serviced set, subsets of  $U$  that contain all players in  $L$ . To be more formal, consider all possible subsets of the players  $L, U$  such that  $L \subseteq U \subseteq \mathcal{A}$ . Given a pair  $L \subseteq U$ , Fence Monotonicity, considers only sets of players  $S$  with  $L \subseteq S \subseteq U$ . Each Fence Monotonicity, condition should be satisfied in any such restriction of the mechanism.

We denote by  $\xi^*(i, L, U)$  the minimum payment of player  $i$  for getting serviced when the output of the mechanism is restricted by  $L$  and  $U$ , i.e.,  $\xi^*(i, L, U) := \min_{\{L \subseteq S \subseteq U, i \in S\}} \xi(i, S)$ .

**Definition 1 (Fence Monotonicity).** We will say that a cost-sharing scheme satisfies Fence Monotonicity, if for every  $L \subseteq U \subseteq \mathcal{A}$ , it satisfies the following three conditions:

- (a) There exists at least one set  $S$  with  $L \subseteq S \subseteq U$ , such that for all  $i \in S$ , we have  $\xi(i, S) = \xi^*(i, L, U)$ .
- (b) For each player  $i \in U \setminus L$  there exists at least one set  $S_i$  with  $i \in S_i$  and  $L \subseteq S_i \subseteq U$ , such that for all  $j \in S_i \setminus L$ , we have  $\xi(j, S_i) = \xi^*(j, L, U)$ .  
(Since  $i \in S_i \setminus L$ , it holds that  $\xi(i, S_i) = \xi^*(i, L, U)$ .)
- (c) If for some  $C \subset U$  there is a player  $j \in C$  with  $\xi(j, C) < \xi^*(j, L, U)$  (obviously  $L \not\subseteq C$ ), then there exists at least one set  $T \neq \emptyset$  with  $T \subseteq L \setminus C$ , such that for all  $i \in T$ ,  $\xi(i, C \cup T) = \xi^*(i, L, U)$ .

An alternative way of expressing conditions (a) and (b) of Fence Monotonicity, at some  $L, U$  is the following: We say that a player  $i$  is serviced optimally in some set  $S \ni i$  if she is serviced with her minimum possible payment in this restriction, i.e.,  $\xi^*(i, L, U) = \xi(i, S)$ . A set  $S$  is optimal if all of its players are serviced optimally. Condition (a) requires the existence of an optimal set. Next, consider the weaker notion of a semi-optimal set  $S$ , where at least the players in  $S \setminus L$  are serviced optimally. Condition (b) implies that each agent  $i \in U \setminus L$  belongs to some semi-optimal set. Notice that two players may not belong to a common semi-optimal set.

Condition (c) is the most important and involved condition and cannot be expressed using the notions above. It compares the minimum possible payment of a player in this restriction of the mechanism with her payment when the outcome can be any subset of  $U$ , i.e., it should not necessarily contain the players

in  $L$ . Assume that player  $j$  is better off when the outcome is some set  $C$ , where  $L \not\subseteq C$ , i.e., loosely speaking some of the players in  $L \setminus C$  “harm” player  $j$  by their presence in the outcome. Then condition (c) requires the existence of some non-empty set  $T$  with  $T \subseteq L \setminus C$ , such that each player in  $T$  is indifferent between her payment in  $C \cup T$  and her minimum payment in the original restriction.

Observe that if  $\xi$  is cross-monotonic, then the set  $U$  is optimal (and consequently semi-optimal). Hence, conditions (a) and (b) are always satisfied. Moreover, condition (c) is trivially satisfied, as for every  $C \subset U$ , it holds that for all  $i \in C$ ,  $\xi(i, C) \leq \xi(i, U) = \xi^*(i, L, U)$ . As a result, cross-monotonicity implies Fence Monotonicity.

**Proposition 1.** *Let  $\xi$  be a cost sharing scheme*

(i) *If it satisfies condition (a) of Fence Monotonicity, for all  $L, U$  with  $|U \setminus L| = 1$ , then it satisfies semi-cross-monotonicity.*

(ii) *If it satisfies conditions (b) of Fence Monotonicity, for all  $L, U$  with  $|U \setminus L| = 2$  and (c) of Fence Monotonicity, for all  $L, U$  with  $|U \setminus L| = 1$ , then it satisfies the following property: For all  $S \subseteq \mathcal{A}$  and all distinct  $i, j \in S$ , if  $\xi(j, S \setminus \{i\}) < \xi(j, S)$  then  $\xi(i, S \setminus \{j\}) = \xi(i, S)$ .*

Proposition II shows exactly how one can derive the two necessary conditions for group-strategyproofness, which were identified by Immorlica et. al. [6]: Part (i) of this Proposition shows how we can derive semi-cross-monotonicity and part (ii) how we can derive the condition identified in Remark B.1 from [6] (Remark B.1 is a special case of part (ii), namely they showed that if  $\xi(i, S \setminus \{j\}) \leq \xi(i, S)$  and  $\xi(j, S \setminus \{i\}) \leq \xi(j, S)$ , then at most one equality can be strict). The two conditions they identified are obtained if apply Fence Monotonicity, for cases when  $U \setminus L$  contains either one or two players, i.e., when we restrict the outcome space to only two or four different possible outcomes.

**Theorem 1.** *A cost sharing scheme gives rise to a GSP mechanism if and only if it satisfies Fence Monotonicity.*

If you are given a cost sharing scheme that satisfies Fence Monotonicity, it is not straightforward how to construct a GSP mechanism. Fence Monotonicity, can yield a GSP mechanism if and only if it is coupled with an allocation rule that satisfies two properties, which we refer to as *Stability* and *Validity* of the tie-breaking rule respectively. We call the underlying mechanisms of this framework Fencing mechanisms.

The mechanisms we design can be put in the following general framework: Given a bid vector as input, we search for a certain pair of sets  $L, U$ , where  $L \subseteq U \subseteq \mathcal{A}$  that meets the criteria of Stability we define below and then we choose one of the allocations in this restriction according to a valid tie-breaking rule. If the search for the stable pair is exhaustive, then the resulting algorithm runs in exponential time. Given an arbitrary cost sharing scheme that satisfies Fence Monotonicity, we do not know of any polynomial time algorithm for computing the stable pair at every input. However, if we restrict our attention to payments that satisfy stronger conditions like, for example, cross-monotonicity we can come up with a polynomial-time algorithm for finding a stable pair.

**Definition 2 (Stability).** A pair  $L, U$  is stable at  $b$  if the following conditions are true: 1. For all  $i \in L$ ,  $b_i > \xi^*(i, L, U)$ , 2. for all  $i \in U \setminus L$ ,  $b_i = \xi^*(i, L, U)$ , and 3. for all  $R \subseteq A \setminus U$ , there is some  $i \in R$ , such that  $b_i < \xi^*(i, L, U \cup R)$ .

The first Stability condition ensures that each player in  $L$  can be serviced with strictly positive utility. The second Stability condition implies that every player in  $U \setminus L$  can be serviced in at least one outcome but with zero utility. The last property requires that if we enlarge  $U$ , then at least one of the newly added players cannot pay in any possible outcome.

After identifying a stable pair these mechanisms output a set  $S$ , where  $L \subseteq S \subseteq U$  given by a tie-breaking function. The mapping  $\sigma : 2^{\mathcal{A}} \times 2^{\mathcal{A}} \times \mathbb{R}^n \rightarrow 2^{\mathcal{A}}$  is a valid tie-breaking rule for  $\xi$ , if for all  $L \subseteq U \subseteq A$ , the set  $S = \sigma(L, U, b)$  satisfies  $L \subseteq S \subseteq U$  and for all  $i \in S$ ,  $\xi(i, S) = \xi^*(i, L, U)$ .

The dependence on the bid vector allows the mechanism of our framework to change its tie-breaking rule between two bid vectors that share a common stable pair. Obviously, condition (a) of Fence Monotonicity, guarantees that a valid tie-breaking rule always exists.

**Definition 3.** We will say that a mechanism  $(O, \xi)$  is a Fencing Mechanism if and only if

1.  $\xi$  satisfies Fence Monotonicity, and
2. for any bid vector  $b$ ,  $O(b) = \sigma(L, U, b)$ , where  $L, U$  is a stable pair at  $b$  and  $\sigma$  is a valid tie-breaking rule.

It is easy to verify that every Fencing Mechanism satisfies VP from Stability and valid tie-breaking. Moreover, it satisfies CS, because if a player bids higher than any of her payments, then again by Stability she belongs to the set  $L$  and gets serviced.<sup>2</sup>

**Theorem 2.** A mechanism is GSP if and only if it is a Fencing Mechanism.

### 3.2 Every GSP Mechanism Is a Fencing Mechanism

**Necessity of Fence Monotonicity.** Here, we prove that the cost-sharing scheme of any GSP mechanism satisfies Fence Monotonicity. Let  $(O, \xi)$  be an arbitrary GSP mechanism and consider some  $U \subseteq \mathcal{A}$ . We show that for every  $L \subseteq U$ ,  $\xi$  satisfies each one of the Fence Monotonicity, conditions using induction on  $|U \setminus L|$ . We first define the notion of a *harm relation* and we prove that it is a strict partial order.

**Lemma 1.** If  $U \subseteq U_1$  and  $L_1 \subseteq L$ , then for all  $i \in U$ ,  $\xi^*(i, L, U) \geq \xi^*(i, L_1, U_1)$ .

---

<sup>2</sup> Assume that  $\xi$  is cross-monotonic and let  $S$  be the output of Moulin mechanism for some bid vector  $b$ . Then, the pair  $L, U$ , where  $L = \{i \in S \mid b_i > \xi(i, S)\}$  and  $U = S$ , is the unique stable pair at  $b$ . Moreover, the tie breaking rule  $\sigma(L, U, b) = U$  is always valid. Therefore, Moulin mechanisms can be viewed as a special case of Fencing mechanisms.

**Proposition 2 (Harm relation).** Fix two sets  $L \subseteq U$  and suppose that  $i, j \in U$ . We say that  $i$  harms  $j$ , if  $\xi^*(j, L, U) < \xi^*(j, L \cup \{i\}, U)$ . If  $i$  does not harm  $j$ , then  $\xi^*(j, L, U) = \xi^*(j, L \cup \{i\}, U)$  (we get this equality applying Lemma 1).

(i) The harm relation satisfies anti-symmetry and transitivity and consequently it is a strict partial order and the induced sub-graph  $G[U \setminus L]$  is a directed acyclic graph.

(ii) For every  $i \in L$ , one of the following holds: either every  $j \in U \setminus L$  harms  $i$  or there exists some sink  $k$  of the subgraph  $G[U \setminus L]$  that does not harm  $i$ .

(iii) For every  $i \in U \setminus L$ , one of the following holds: either  $i$  is a sink of the sub-graph  $G[U \setminus L]$  or there exists some a sink  $k$  of the sub-graph  $G[U \setminus L]$  that is harmed by  $i$ .

We continue by revealing several important allocation properties of GSP mechanisms and use them to prove the induction step for each property of Fence Monotonicity.

**Conditions (a) and (b) of Fence Monotonicity.** For the proof of conditions (a) and (b) we consider bid vectors where all the players in  $L$  have bidden  $b_i^*$ , all players in  $A \setminus U$  have bidden  $-1$ , and the players in  $U \setminus L$  have bidden exactly  $\xi^*(i, L, U)$  (with the minor exception of one or two players for condition (b)).

**Lemma 2.** At the bid vector  $b$ , where for all  $i \in L$ ,  $b_i = b_i^*$ , for all  $i \in U \setminus L$ ,  $b_i = \xi^*(i, L, U)$ , and for all  $i \notin U$ ,  $b_i = -1$ , it holds that  $L \subseteq O(b) \subseteq U$  and for all  $i \in O(b)$ ,  $\xi(i, O(b)) = \xi^*(i, L, U)$ .

Setting  $S = O(b)$ , condition (a) of Fence Monotonicity, is satisfied at  $L, U$ .

**Lemma 3.** Let  $b$  be the bid vector as defined in Lemma 2

(i) Consider any sink  $k$  of  $G[U \setminus L]$ . At any bid vector  $b^k$ , where  $b_k^k > \xi^*(k, L, U)$  and the rest of the players bid according to  $b$ , it holds that  $L \subseteq O(b^k) \subseteq U$ ,  $k \in O(b^k)$ , and for all  $j \in O(b^k) \setminus L$ ,  $\xi(j, O(b^k)) = \xi^*(j, L, U)$ .

(ii) Consider any player  $i$  that harms a sink  $k$  of  $G[U \setminus L]$ . At any bid vector  $b^i$ , where  $b_i^i > \xi^*(i, L, U)$ ,  $\xi^*(k, L, U) < b_k^i < \xi^*(k, L \cup \{i\}, U)$ , and the rest of the players bid according to  $b$ , it holds that  $L \subseteq O(b^i) \subseteq U$ ,  $k \notin O(b^i)$ ,  $i \in O(b^i)$ , and for all  $j \in O(b^i) \setminus L$ ,  $\xi(j, O(b^i)) = \xi^*(j, L, U)$ .

Thus, condition (b) at  $L, U$  for any sink  $k$  is satisfied by setting  $S_k = O(b^k)$ . By Proposition 2 (ii), every non-sink  $i$  harms a sink  $k$ , hence, condition (b) is also satisfied for  $i$  at  $L, U$  by setting  $S_i = O(b^i)$ .

**Condition (c) of Fence Monotonicity.** To show that the cost-sharing scheme satisfies the third property of Fence Monotonicity, at  $L, U$ , we need the induction hypothesis only for showing (as we have already done) that condition (a) of Fence Monotonicity, is satisfied at this pair and specifically only the allocation properties of Lemma 2. Now we consider inputs for which we do not get anymore directly from CS, that the players in  $L$  surely receive service. The idea is to gradually generalize the bid vectors and characterize the allocation of the

**Table 1.** Allocation properties of GSP mechanisms. Every family of inputs we consider is a subset of the previous one. Last property sheds light on an important property of the allocation of (non-Moulin) GSP mechanisms. It reveals the following fact: If the bids of all players in a set  $L \subseteq U$  have surpassed their respective minimum payments at  $L, U$ , (and the players in  $\mathcal{A} \setminus U$  do not want to participate), then a GSP mechanism never excludes a subset of  $L$  from the outcome in favor of another serviced player, who would pay less if they were not present in the outcome. Loosely speaking the players in  $L$  rule out any outcome  $C \subset U$  such that there is some  $j \in C$  such that  $\xi(j, C) < \xi^*(j, L, U)$ .

$L$	$U \setminus L$	$\notin U$	allocation of a GSP mechanism
$b_i^*$	$\xi^*(i, L, U)$	-1	$\forall i \in O(b), \xi(i, O(b)) = \xi^*(i, L, U) \text{ & } L \subseteq O(b) \subseteq U$
$> \xi^*(i, L, U)$	$\xi^*(i, L, U)$	-1	$\forall i \in O(b), \xi(i, O(b)) = \xi^*(i, L, U) \text{ & } L \subseteq O(b) \subseteq U$
$> \xi^*(i, L, U)$	$\in \mathbb{R}$	-1	$\forall i \in O(b), \xi(i, O(b)) \geq \xi^*(i, L, U)$

mechanism. The following table contains allocation properties that any GSP mechanism satisfies.

Using the last property in Table 1, it is easy to show that any GSP mechanism satisfies condition (c) of Fence Monotonicity. Let  $\epsilon > 0$  be a small quantity satisfying that for all  $S', S \subseteq U$  and  $i \in S \cap S'$ ,  $\xi(i, S') - \xi(i, S) > 0 \Rightarrow \xi(i, S') - \xi(i, S) > \epsilon$ .

**Lemma 4.** Consider any  $C \subset U$  such that there is some  $j \in C$  with  $\xi(j, C) < \xi^*(j, L, U)$ . At the bid vector  $b^c$ , where for all  $i \in C$ ,  $b_i^c = b_i^*$ , for all  $i \in L \setminus C$ ,  $b_i^c = \xi^*(i, L, U) + \epsilon$ , and for all  $i \notin C \cup L$ ,  $b_i^c = -1$ , it holds that  $C \subset O(b^c) \subseteq L \cup C$  and for all  $i \in O(b^c) \setminus C$ ,  $\xi(i, O(b^c)) = \xi^*(i, L, U)$ .

Setting  $T = O(b^c) \setminus C$  we can conclude that every GSP mechanism satisfies condition (c) of Fence Monotonicity.

**Necessity of Stability and Valid tie-breaking.** Next, we show that the allocation of every GSP mechanism satisfies Stability and uses a Valid tie-breaking rule. Since its cost-sharing scheme must satisfy Fence Monotonicity, we can prove the following generalization of the second allocation property in Table 1.

**Lemma 5.** Let  $L \subseteq U \subseteq \mathcal{A}$ . For every bid vector  $b$  such that  $L, U$  is stable, it holds that  $L \subseteq O(b) \subseteq U$  and for all  $i \in O(b)$ ,  $\xi(i, O(b)) = \xi^*(i, L, U)$ .

Lemma 5 implies that an allocation should satisfy Stability and Validity of the tie-breaking rule, whenever a stable pair exists. Lemma 5 guarantees the existence of a unique stable pair at every input, which completes our proof.

### 3.3 The Classes of GSP and Fencing Mechanisms Coincide

We complete our characterization by proving that Fencing Mechanisms are GSP.

**Lemma 6.** For every bid vector  $b$  and set  $L$  with  $L \subseteq \mathcal{A}$ , there exists a unique set  $U$  with  $U \supseteq L$ , such that for all  $i \in U \setminus L$  we have  $b_i \geq \xi^*(i, L, U)$  and any other set with the same property is a subset of  $U$ . Moreover, the pair  $L, U$  satisfies condition 3. of Stability.

The next two Lemmas prove our statement.

**Lemma 7.** *For the inputs where there exists a stable pair it, it is unique and every Fencing Mechanism is GSP.*

**Lemma 8.** *For every bid vector  $b$  there exists a unique stable pair. Consequently, Fencing Mechanisms are meaningful and GSP.*

There is a specific reason behind this orderings. We prove Lemma 8 by induction, thus, for showing the induction step we deal with bid vectors where a stable pair exists. Applying Lemma 7 at these inputs turns out to simplify our proof substantially.

## 4 Conclusion and Future Directions

We demonstrate the use of our characterization by showing that even in the case of three players GSP mechanisms fail in having a constant budget balance.

**Theorem 3.** *There are cost function families where GSP mechanism has arbitrarily poor budget balance ratio.*

We believe that the most interesting future directions are the following: How can our characterization be applied for obtaining cost-sharing mechanisms with better budget-balance guarantees or lower bounds for specific problems? Does there exist a polynomial-time algorithm for finding the allocation of a cost-sharing scheme that satisfies Fence Monotonicity, or maybe can we show that the problem of finding a stable pair is computationally hard? A natural question that arises in this context is if it is computationally more efficient to find the appropriate outcome than identifying the stable pair.

**Theorem 4.** *Suppose that we are given the outcome of a GSP mechanism at  $b$ . Given polynomial access to  $\xi^*(i, L, U)$  for all  $L \subseteq U \subseteq \mathcal{A}$  and all  $i \in U$ , we can identify the stable pair in polynomial time.*

**Acknowledgements.** We would like to thank Elias Koutsoupias for suggesting the problem, as well as for many very helpful insights and discussions. We would also like to thank Janina Brenner, Nicole Immorlica, Evangelos Markakis, Tim Roughgarden, and Florian Schoppmann for helpful discussions and some pointers in the bibliography.

## References

1. Nisan, N., Ronen, A.: Algorithmic mechanism design (extended abstract). In: Proceedings of the 31st annual ACM symposium on Theory of computing (STOC 1999), pp. 129–140. ACM, New York (1999)
2. Roberts, K.: The characterization of implementable choice rules. In: Laffont, J.J. (ed.) Aggregation and Revelation of Preferences. Papers presented at the first European Summer Workshop of the Econometric Society, pp. 321–348. North-Holland, Amsterdam (1979)

3. Lavi, R., Mu’alem, A., Nisan, N.: Towards a characterization of truthful combinatorial auctions. In: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003), Washington, DC, USA, p. 574. IEEE Computer Society, Los Alamitos (2003)
4. Dobzinski, S., Sundararajan, M.: On characterizations of truthful mechanisms for combinatorial auctions and scheduling. In: Proceedings of the 9th ACM conference on Electronic commerce (EC 2008), pp. 38–47. ACM, New York (2008)
5. Christodoulou, G., Koutsoupias, E., Vidali, A.: A characterization of 2-player mechanisms for scheduling. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 297–307. Springer, Heidelberg (2008)
6. Immorlica, N., Mahdian, M., Mirrokni, V.S.: Limitations of cross-monotonic cost-sharing schemes. ACM Trans. Algorithms 4, 1–25 (2008)
7. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
8. Juarez, R.: Group-strategy proof cost sharing. Working paper (2008)
9. Moulin, H.: Incremental cost sharing: Characterization by coalition strategy-proofness. In: Social Choice and Welfare, pp. 279–320 (1999)
10. Rochet, J.C.: A necessary and sufficient condition for rationalizability in a quasi-linear context. Journal of Mathematical Economics 16, 191–200 (1987)
11. Saks, M., Yu, L.: Weak monotonicity suffices for truthfulness on convex domains. In: Proceedings of the 6th ACM conference on Electronic commerce (EC 2005), pp. 286–293. ACM, New York (2005)
12. Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs: budget balance versus efficiency. Economic Theory 18, 511–533 (2001)
13. Brenner, J.A., Schäfer, G.: Cost sharing methods for makespan and completion time scheduling. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 670–681. Springer, Heidelberg (2007)
14. Roughgarden, T., Sundararajan, M.: New trade-offs in cost-sharing mechanisms. In: Proceedings of the 38th annual ACM symposium on Theory of computing (STOC 2006), pp. 79–88. ACM, New York (2006)
15. Penna, P., Ventre, C.: The algorithmic structure of group strategyproof budget-balanced cost-sharing mechanisms. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 337–348. Springer, Heidelberg (2006)
16. Mehta, A., Roughgarden, T., Sundararajan, M.: Beyond moulin mechanisms. In: Proceedings of the 8th ACM conference on Electronic commerce (EC 2007), pp. 1–10. ACM, New York (2007)
17. Bleischwitz, Y., Monien, B., Schoppmann, F.: To be or not to be (served). In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 515–528. Springer, Heidelberg (2007)
18. Nisan, N., Ronen, A.: Algorithmic mechanism design. Games and Economic Behavior 35, 166–196 (2001)
19. Juarez, R.: Prior-free cost sharing design: group strategyproofness and the worst absolute loss. Social Computing and Behavioral Modeling, 1–7 (2009)

# Contribution Games in Social Networks\*

Elliot Anshelevich<sup>1</sup> and Martin Hoefer<sup>2</sup>

<sup>1</sup> Dept. of Computer Science, Rensselaer Polytechnic Institute, Troy, NY

<sup>2</sup> Dept. of Computer Science, RWTH Aachen University, Germany

[mhoefer@cs.rwth-aachen.de](mailto:mhoefer@cs.rwth-aachen.de)

**Abstract.** We consider *network contribution games*, where each agent in a social network has a budget of effort that he can contribute to different collaborative projects. Depending on the contribution of the involved agents a project will be successful to a different degree, and to measure the success we use a reward function for each project. Every agent is trying to maximize the reward from all projects that it is involved in. We consider pairwise equilibria of this game and characterize the existence, computational complexity, and quality of equilibrium based on the types of reward functions involved. For example, when all reward functions are concave, we prove that the price of anarchy is at most 2. For convex functions the same only holds under some special but very natural conditions. A special focus of the paper are minimum effort games, where the success of a project depends only on the minimum effort of any of the participants. Finally, we briefly discuss additional aspects like approximate equilibria and convergence of dynamics.

## 1 Introduction

Understanding the degree to which rational agents will participate in and contribute to joint projects is critical in many areas of society. With the advent of the Internet and the consideration of rationality in the design of multi-agent and peer-to-peer systems, these aspects are becoming of interest to computer scientists and subject to analytical computer science research. Not surprisingly, the study of contribution incentives has been an area of vital research interest in economics and related areas with seminal contributions to the topic over the last decades. A prominent example from experimental economics is the *minimum effort coordination game* [24], in which a number of participants contribute to a joint project, and the outcome depends solely on the minimum contribution of any agent. While the Nash equilibria in this game exhibit a quite simple structure, behavior in laboratory experiments led to sometimes surprising patterns see, e.g., [15, 11] for recent examples. On the analytical side this game was studied, for instance, with respect to logit-response dynamics and stochastic potential in [4].

---

\* This work was supported in part by NSF CCF-0914782, by DFG through UMIC Research Center at RWTH Aachen University, and by DFG grant Ho 3831/3-1.

In this paper we propose and study a simple framework of *network contribution games* for contribution, collaboration, and coordination of actors embedded in social networks. The game contains the minimum effort coordination game as a special case and is closely related to many other games from the economics literature. In such a game each player is a vertex in a graph, and the edges represent bilateral projects that he can engage in. Each player has a budget of effort that he can contribute to different edges. Budgets and contributions are non-negative numbers, and we use them as an abstraction for the different ways and degrees by which actors can contribute to a bilateral project, e.g., by allocating time, money, and personal energy to maintaining a friendship, a relationship, or a collaboration, the development of a new product, or the installation or standardization of a new technology – to name a few. Depending on the contribution of the involved actors a project will exhibit different levels of success, and to measure the success we use a reward function for each project. Finally, each player strives to maximize the total success of all projects he is involved in.

A major issue that we address in our games is the impact of collaboration. An incentive for collaboration evolves naturally when agents are embedded in social networks and involved in joint projects. We are interested in the way that a limited collaboration between agents influences properties of equilibria in contribution games like existence, computational complexity, the convergence of natural dynamics, as well as measures of inefficiency. In particular, in addition to unilateral strategy changes we will allow pairs of players to change their strategies in a coordinated manner. States that are resilient against such bilateral deviations are termed *2-strong* [3] or *pairwise equilibria* [20].

**Network Contribution Games.** We consider *network contribution games* as models for the contribution to relationships and projects in social networks. In our games we are given a simple and undirected graph  $G = (V, E)$  with  $n$  nodes and  $m$  edges. Every node  $v \in V$  is a *player*, and every edge  $e \in E$  represents a *project* (collaboration, relationship, etc.). A player  $v$  has a given *budget*  $B_v \geq 0$  of the total amount of *effort* that it is able to apply to all of its projects (i.e., edges incident to  $v$ ). Budgets are called *uniform* if  $B_u = B_v$  for any  $u, v \in V$ . In this case, unless stated otherwise, we assume that  $B_v = 1$  for all  $v \in V$  and scale reward functions accordingly.

We denote by  $E_v$  the set of edges incident to  $v$ . A *strategy* for player  $v$  is a function  $s_v : E_v \rightarrow \mathbb{R}_{\geq 0}$  that satisfies  $\sum_{e=(v,u)} s_v(e) \leq B_v$  and specifies the amount of effort  $s_v(e)$  that  $v$  puts into his project  $e \in E_v$ . A *state* of the game is simply a vector  $s = (s_1, \dots, s_n)$ . The success of a project  $e$  is measured by a *reward function*  $f_e : \mathbb{R}_{\geq 0}^2 \rightarrow \mathbb{R}$ , for which  $f_e(x, y) \geq 0$  and non-decreasing in  $x, y \geq 0$ . The *utility* or *welfare* of a player  $v$  is simply the total success of all its projects, i.e.,  $w_v(s) = \sum_{e=(v,u)} f_e(s_v(e), s_u(e))$ , so both endpoints benefit equally from their relationship. In addition, we will assume that reward functions  $f_e$  are symmetric, so  $f_e(x, y) = f_e(y, x)$  for all  $x, y \geq 0$ , and for ease of presentation we will assume they are continuous and differentiable, although most of our results can be obtained without these assumptions.

We are interested in the existence and computational complexity of stable states, their performance in terms of social welfare, and the convergence of natural dynamics to equilibrium. The central concept of stability in strategic games is the (*pure*) *Nash equilibrium*, which is resilient against unilateral deviations, i.e., a state  $s$  such that  $w_v(s_v, s_{-v}) \geq w_v(s'_v, s_{-v})$  for each  $v \in V$  and all possible strategies  $s'_v$ . For the *social welfare*  $w(s)$  of a state  $s$  we use the natural utilitarian approach and define  $w(s) = \sum_{v \in V} w_v(s)$ . A *social optimum*  $s^*$  is a state with  $w(s^*) \geq w(s)$  for every possible state  $s$  of the game.

In games such as ours, it makes sense to consider bilateral deviations, as well as unilateral ones. Nash equilibria have shortcomings in this context, for instance for a pair of adjacent nodes who would – although being unilaterally unable to increase their utility – benefit from cooperating and increasing the effort on their mutual project. The prediction of Nash equilibrium that such a state is stable is quite unreasonable. In fact, it is easy to show that when considering pure Nash equilibria, the function  $\Phi(s) = w(s)/2$  is an exact potential function for our games. Hence,  $s^*$  is an optimal Nash equilibrium, the price of stability is 1, and iterative better response dynamics converge to an equilibrium. Additionally, for many natural reward functions  $f_e$ , the price of anarchy remains unbounded.<sup>1</sup>

Following the reasoning in, e.g., [20, 19], we instead consider pairwise equilibria and focus on the more interesting case of *bilateral deviations*. An improving bilateral deviation in a state  $s$  is a pair of strategies  $(s'_u, s'_v)$  such that  $w_v(s'_u, s'_v, s_{-u,v}) > w_v(s)$  and  $w_u(s'_u, s'_v, s_{-u,v}) > w_u(s)$ . A *pairwise equilibrium* is a Nash equilibrium, for which there are no improving bilateral deviations. Notice that we are actually using a stronger notion of pairwise stability than described in [19], since any pair of players can change their strategies in an arbitrary manner, instead of changing their contributions on just a single edge. Our notion of pairwise equilibrium is exactly the notion of *2-strong equilibrium* [3].

We evaluate the performance of stable states using prices of anarchy and stability, respectively. The *price of anarchy (stability)* for our game is the worst-case ratio of  $w(s^*)/w(s)$  for the worst (best) pairwise equilibrium  $s$ . The prices for a class of games (e.g., with convex or concave reward functions) are simply the worst-case ratios of any game in the class.

An obvious extension to our model is the case of *general contribution games*, in which projects are arbitrary subsets of players, and *setwise equilibria*, where no set of players involved in a project can all profit from a joint deviation. Some of our results (as detailed below) directly extend to this case. For instance, the price of anarchy often depends on  $k$ , the size of the largest project. However, apart from some preliminary results specified below, a general study of this and various other interesting aspects remains as an open problem.

**Our Results.** We study the properties of pairwise equilibrium in network contribution games. Consider the effort  $s_v(e)$  expended by player  $v$  on an edge  $e = (u, v)$ . The fact that  $f_e$  is monotonic nondecreasing tells us that  $w_v$  increases in  $s_v(e)$ . Depending on the application being considered, however, the

---

<sup>1</sup> Consider, for instance, a path of length 3 with  $\{e_1, e_2, e_3\}$  and  $f_{e_1}(x, y) = f_{e_3}(x, y) = \min(x, y)$  and  $f_{e_2}(x, y) = M \cdot \min(x, y)$ , for some large number  $M$ .

**Table 1.** Summary of some of our results. For the cases where equilibrium always exists, we can also give algorithms to compute it and convergence results. All of our PoA upper bounds are tight. (\*) If  $\forall e, f_e(x, 0) = 0$ , NP-hard otherwise. (\*\*) When  $\frac{\partial^2 f}{\partial x \partial y} \geq 0$  (see Section 2) (\*\*\*) If budgets are uniform, NP-hard otherwise.

Reward Functions	Existence	Price of Anarchy
General convex	Yes (*)	2 (**)
General concave	Not always	2
$c_e \cdot (x + y)$	Decision in P	1
Minimum effort convex	Yes (***)	2 (***)
Minimum effort concave	Yes	2
Maximum effort	Yes	2
Approximate Equilibrium	OPT is a 2-apx. Equilibrium	

utility could possess the property of “diminishing returns”, or on the contrary, could increase at a faster rate as  $v$  puts more effort on  $e$ . In other words, for a fixed effort amount  $s_u(e)$ ,  $f_e$  as a function of  $s_v(e)$  could be concave or convex, and we will distinguish the treatment of the framework based on these properties.

In Section 2 we consider the case of convex reward functions. For a large class of convex functions we can show a tight bound for the price of anarchy of 2. However, for games in this class pairwise equilibria might not exist. In fact, we show that it is NP-hard to decide their existence, even when the edges have reward functions of either the form  $f_e(x, y) = c_e \cdot (xy)$  or  $f_e(x, y) = c_e \cdot (x + y)$  for constants  $c_e > 0$ . If, however, *all* functions are of the form  $f_e(x, y) = c_e \cdot (xy)$ , then existence and polynomial time computability is guaranteed. We show this existence result for a substantially larger class of functions that may not even be convex, although it includes the class of all convex functions  $f_e$  with  $f_e(x, 0) = 0$ . As an interesting special case, we prove that if all functions are  $f_e(x, y) = c_e \cdot (x + y)$ , it is possible to determine efficiently if pairwise equilibria exist and to compute them in polynomial time when they exist.

In Section 2.2 we consider pairwise equilibria for concave reward functions. In this case, pairwise equilibria may also not exist, for example in the triangle graph with uniform budgets and reward functions  $f_e(x, y) = \sqrt{xy}$ . Nevertheless, when they exist we can show tight bounds of 2 on prices of anarchy and stability.

Section 3 treats different special cases of particular interest. First we study the important case of minimum effort games with reward functions  $f_e(x, y) = h_e(\min(x, y))$ . If functions  $h_e$  are convex, pairwise equilibria do not necessarily exist, and it is NP-hard to decide the existence for a given game. Perhaps surprisingly, if budgets are uniform, i.e., if  $B_v = B_u$  for all  $u, v \in V$ , then pairwise equilibria exist for all convex functions  $h_e$ , and the prices of anarchy and stability are exactly 2. If functions  $h_e$  are concave, we can always guarantee existence. Our bounds for concave functions in Section 2.2 imply tight bounds on the prices of anarchy and stability of 2.

Section 4 treats additional aspects of pairwise equilibria. Here we briefly mention our results on maximum effort games, for which we can show existence of

pairwise equilibria and tight bounds on prices of anarchy and stability. In addition, we treat approximate equilibria and show that a social optimum  $s^*$  is always a 2-approximate equilibrium. In addition, we mention our results regarding convergence of dynamics to pairwise equilibria when they exist.

Due to lack of space, most of our proofs are deferred to the full version of the paper. [6]

**Related Work.** The model most related to ours is the co-author model [20, 19]. The motivation is very similar to ours, although there are many important differences. For example, in the usual co-author model, the nodes cannot choose how to split their effort between their projects, only which projects to participate in. Moreover, we consider general reward functions, and our notion of pairwise stability is stronger than in [20, 19].

In [9], Bramoullé and Kranton consider an extremely general model of network games designed to model public goods. Nevertheless, our game is not a special case of this model, since in [9] the strategy of a node is simply a level of effort it contributes, not how much effort it contributes *to each project*. There are many extensions to this model, e.g., Corbo et al. [12] consider similar models in the context of peer-to-peer networks. Their work closely connects to the seminal paper on contribution games by Ballester et al. [8], which has prompted numerous similar follow-up studies.

The literature on games played in networks is too much to survey here: we will address only the most relevant lines of research. In the last few years, there have been several fascinating papers on network bargaining games (e.g., [22, 10]), and in general on games played in networks where every edge represents a two-player game (e.g., [13, 18]). All these games either require that every node plays the same strategy on all neighboring edges, or leaves the node free to play any strategy on any edge. While every edge in our game can be considered to be a (very simple) two-player game, the strategies (i.e., contributions) that a node puts on every edge are neither the same nor arbitrarily different: specifically they are constrained by a budget on the total effort that a node can contribute to all neighboring edges in total. To the best of our knowledge, such games have not been studied before (except as part of the works mentioned below).

Our game bears some resemblance to network formation games where players attempt to maximize different forms of “centrality” [16, 21, 23], although our utility functions and equilibrium structure are extremely different. Minimum effort coordination games as proposed by van Huyck et al. [24] represent a special case of our general model. They are a vital research topic in experimental economics, see the papers mentioned above and [14] for a recent survey. We study a generalized and networked variant in Section 3. A slightly different adjustment to networks has recently appeared in [2]. Our work complements this body of work with provable guarantees on the efficiency of equilibria and the convergence times of dynamics.

Some of the special cases we consider bear a resemblance to stable matching [17], and in fact correlated variants of stable matching can be considered an “integral” version of our game. Our results generalize existence and convergence

results for correlated stable matching (as, e.g., in [1]), and our price of anarchy results greatly generalize the results of [5].

It is worth mentioning the connection of our reward functions with the “Combinatorial Agency” framework (see, e.g., [7]). In this framework, many people work together on one project, and the success of this project depends in a complex (usually probabilistic) manner on whether the people involved choose a high level of effort. It is an interesting open problem to extend our results to the case in which every project of a game is an instance of the combinatorial agency problem.

## 2 Polynomials, Convex, and Concave Reward Functions

In this section we initially consider a class of reward functions that guarantee a good price of anarchy. The class will be denoted by  $\mathcal{C}$  and contains all convex functions for which all mixed second partial derivatives are non-negative. We start by introducing the notions of *coordinate-convex* and *coordinate-concave* functions.

**Definition 1.** A function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is coordinate-convex if for all of its arguments  $x_i$ , we have that  $\frac{\partial^2 f}{\partial x_i^2} \geq 0$ ; coordinate-concave if for all of its arguments  $x_i$ , we have that  $\frac{\partial^2 f}{\partial x_i^2} \leq 0$ .

Note that *every convex function is coordinate-convex*, and similarly every concave function is coordinate-concave. However, coordinate-convexity/concavity is necessary but not sufficient for convexity/concavity. For instance, the function  $\log(1 + xy)$  is coordinate-concave, but not concave – indeed, it is convex as long as  $x = y \in [0, 1]$ .

**Definition 2.** Define a symmetric nondecreasing function  $f : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  to be in class  $\mathcal{C}$  iff  $f$  is coordinate-convex and  $\frac{\partial^2 f}{\partial x \partial y} \geq 0$ .

The class  $\mathcal{C}$  is of particular interest to us, because we can show the following result. It can be extended to general contribution games and setwise equilibria, where the price of anarchy is equal to  $k$ .

**Theorem 1.** The price of anarchy in network contribution games is at most 2 when all reward functions belong to class  $\mathcal{C}$ .

A large class of functions that belong to  $\mathcal{C}$  are based on polynomials. Consider a polynomial  $p(x, y)$  in two variables with non-negative coefficients that is symmetric (i.e.,  $p(x, y) = p(y, x)$ ) and non-negative for  $x, y \geq 0$ . For every such  $p$  we consider all possible extensions to a function  $f(x, y) = h(p(x, y))$  with  $h : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  being nondecreasing and convex. We call the union of all these extensions the class  $\mathcal{P}$ . Clearly, every  $p(x, y) \in \mathcal{P}$  since  $h(x) = x$  is convex. In particular,  $\mathcal{P}$  contains a large variety of functions such as  $xy$ ,  $(x + y)^2$ ,  $e^{x+y}$ ,  $x^3 + y^3 + 2xy$ , etc. It is simple to observe that  $\mathcal{P} \subset \mathcal{C}$ , and thus the price of anarchy result for  $\mathcal{C}$  will hold for every game with arbitrary functions from  $\mathcal{P}$ .

## 2.1 Existence and Complexity of Pairwise Equilibria

While Theorem 1 shows that the price of anarchy is 2, this result says nothing about the existence and complexity of computing pairwise equilibria. In fact, even for simple games with reward functions  $f_e(x, y) = c_e \cdot (x + y)$  and small constants  $c_e$ , pairwise equilibria can be absent.

*Example 1.* In our example there is a triangle graph with nodes  $u_1$ ,  $u_2$ , and  $u_3$ , edges  $e_1 = (u_1, u_2)$ ,  $e_2 = (u_2, u_3)$ , and  $e_3 = (u_3, u_1)$ , and uniform budgets. Edge  $e_i$  has reward function  $f_i$  with  $f_1(x, y) = f_2(x, y) = 3(x + y)$ , and  $f_3(x, y) = 2(x + y)$ . A pairwise equilibrium must not allow profitable unilateral deviations. Thus,  $s_1(e_1) = s_3(e_2) = 1$ . Player 2 can assign his budget arbitrarily. This yields  $w_1(s) = 3 + 3s_2(e_1)$  and  $w_3(s) = 3 + 3s_2(e_2)$ . Changing to a state  $s'$  where  $u_1$  and  $u_3$  bilaterally deviate by moving all their budget to  $e_3$  yields  $w_1(s') = 3s_2(e_1) + 4 > w_1(s)$  and  $w_3(s') = 3s_2(e_2) + 4 > w_3(s)$ . Hence, no pairwise equilibrium exists.

Although there are games without pairwise equilibria, there is a large class of functions slightly orthogonal to  $\mathcal{C}$  for which we can show existence and an efficient algorithm for computation. Note that the existence result can be extended to general contribution games and setwise equilibria.

**Theorem 2.** *A pairwise equilibrium always exists and can be computed efficiently when all reward functions  $f_e$  are coordinate-convex and  $f_e(x, 0) = 0$  for all  $e \in E$  and  $x \geq 0$ .*

Theorem 2 establishes existence and efficient computation of equilibria for many functions from class  $\mathcal{C}$ , as well as other ones that do not belong to it. In particular, it shows existence for all convex functions  $f_e$  that are 0-valued when one of its arguments is 0, as well as for many non-convex ones, such as the weighted product function  $f_e(x, y) = c_e \cdot (xy)$ . In general, however, we can show that deciding existence for pairwise equilibria for a given game is NP-hard, even for very simple reward functions from  $\mathcal{C}$  such as  $f_e(x, y) = c_e \cdot (x + y)$  and  $f(x, y) = c_e \cdot (xy)$  with constants  $c_e > 0$ .

**Theorem 3.** *It is NP-hard to decide if a network contribution game admits a pairwise equilibrium even if all functions are either  $f_e(x, y) = c_e \cdot (x + y)$  or  $f_e(x, y) = c_e \cdot (xy)$ .*

Finally, let us focus on an interesting special case. The hardness in the previous theorem comes from the interplay of reward functions  $xy$  that tend to a clustering of effort and  $x + y$  that create cycles. We observed above that if all functions are  $c_e \cdot (xy)$ , then equilibria exist and can be computed in polynomial time. Here we show that for the case that  $f_e(x, y) = c_e \cdot (x + y)$  for all  $e \in E$ , we can decide efficiently if a pairwise equilibrium exists. Furthermore, if an equilibrium exists, we can compute it in polynomial time.

**Theorem 4.** *There is an efficient algorithm to decide the existence of a pairwise equilibrium, and to compute one if one exists, when all reward functions are of the form  $f_e(x, y) = c_e \cdot (x + y)$  for arbitrary constants  $c_e > 0$ . Moreover, the price of anarchy is 1 in this case.*

## 2.2 Concave Reward Functions

In this section we consider the case when reward functions  $f_e(x, y)$  are concave. In the introduction we observed that a pairwise equilibrium may not exist. However, when it does exist we obtain the following general result.

**Theorem 5.** *The price of anarchy in network contribution games is at most 2 when all reward functions are coordinate-concave.*

## 3 Minimum Effort Games

In this section we consider the interesting case (studied for example in [24, 4, 15, 11]) when all reward functions are of the form  $f_e(x, y) = h_e(\min(x, y))$ . In other words, the reward of an edge depends only on the minimum effort of its two endpoints. In our treatment we again distinguish between the case of increasing marginal returns (convex functions  $h_e$ ) and diminishing marginal returns (concave functions  $h_e$ ). Note that in this case bilateral deviations are in many ways essential to make the game meaningful, as there are an infinite number of Nash equilibria. In addition, we can assume w.l.o.g. that in every pairwise equilibrium  $s$  there is a unique value  $s_e$  for each  $e = (u, v) \in E$  such that  $s_v(e) = s_u(e) = s_e$ . The same can be assumed for optima  $s^*$ .

### 3.1 Convex Functions in Minimum Effort Games

In this section we consider reward functions  $f_e(x, y) = h_e(\min(x, y))$  with convex functions  $h_e(x)$ . This case bears some similarities with our treatment of the class  $\mathcal{C}$  in Section 2. In fact, we can show existence of pairwise equilibria in games with uniform budgets. We call an equilibrium  $s$  *integral* if  $s_e \in \{0, 1\}$  for all  $e \in E$ . Note that all results in this section can be extended to general contribution games and setwise equilibria. The price of anarchy then becomes  $k$ .

**Theorem 6.** *A pairwise equilibrium always exists in games with uniform budgets and  $f_e(x, y) = h_e(\min(x, y))$  when all  $h_e$  are convex. If all  $h_e$  are strictly convex, then all pairwise equilibria are integral.*

**Theorem 7.** *The price of anarchy in network contribution games is at most 2 when all reward functions  $f_e(x, y) = h_e(\min(x, y))$  with convex  $h_e$ , and budgets are uniform.*

For the case of arbitrary budgets and convex functions, however, we can again find a game without a pairwise equilibrium.

*Example 2.* Consider a path of length 4. We denote the vertices along this path by  $u, v, w, z$ . All players have budget 2, except for player  $z$  that has budget 1. The profit functions are  $h_{u,v}(x) = 2x^2$ ,  $h_{v,w}(x) = 5x$ , and  $h_{w,z}(x) = 6x$ . Observe that this game allows no pairwise equilibrium: If  $2 \geq s_{v,w} > 1$ , then player  $w$  has an incentive to increase the effort toward  $z$ . If  $1 \geq s_{v,w} > 0$ , then player  $v$  has an incentive to increase effort toward  $u$ . If  $s_{v,w} = 0$ ,  $v$  and  $w$  can jointly increase their profits by contributing 2 on  $(v, w)$ .

Using this example we can construct games in which deciding existence of pairwise equilibria is hard. We remark that the following theorem can be extended to show hardness also for games with uniform budgets in which functions are either concave or convex.

**Theorem 8.** *It is NP-hard to decide if a network contribution game admits a pairwise equilibrium if budgets are arbitrary and all functions are  $f_e(x, y) = h_e(\min(x, y))$  with convex  $h_e$ .*

### 3.2 Concave Functions in Minimum Effort Games

In this section we consider the case of diminishing returns, i.e., when all  $h_e$  are concave functions. Note that in this case the function  $f_e = h_e(\min(x, y))$  is coordinate-concave. Therefore, the results from Section 2.2 show that the price of anarchy is at most 2. However, for general coordinate-concave functions it is not possible to establish the existence of pairwise equilibria, which we do for concave  $h_e$  below. In fact, if the functions  $h_e$  are strictly concave, we can show that the equilibrium is unique. Below we sketch most parts of the proof, which requires very different techniques than most of our other arguments.

**Theorem 9.** *A pairwise equilibrium always exists in games with  $f_e(x, y) = h_e(\min(x, y))$  when all  $h_e$  are continuous, piecewise differentiable, and concave, and we can compute it efficiently. Moreover, if all  $h_e$  are strictly concave, then this equilibrium is unique.*

*Proof.* (sketch) We create a pairwise equilibrium in an iterative manner. For any solution and set of nodes  $S$ , define  $BR_v(S)$  as the set of best responses for node  $v$  if it can control the strategies of nodes  $S$ . We begin by computing  $BR_v(V)$  independently for each player  $v$  ( $V$  is the set of all nodes). In particular, this simulates that  $v$  is the player that always creates the minimum of every edge, and we pick  $s_v$  such that it maximizes  $\sum_{e=(u,v)} h_e(s_v(e))$ . This is a concave maximization problem (or equivalently a convex minimization problem), which can be solved by standard methods in polynomial time. Let  $h_e^+(x)$  be the derivative of  $h_e(x)$  in the positive direction, and  $h_e^-(x)$  be the derivative of  $h_e(x)$  in the negative direction. We have the property that for  $s_v$  calculated as above, for every edge  $e$  with  $s_v(e) > 0$  it holds that  $h_e^-(s_v(e)) \geq h_{e'}^+(s_v(e'))$  for every edge  $e'$  incident to  $v$ . Define  $h'_v$  as the minimum value of  $h_e^-(s_v(e))$  for all edges  $e$  incident to  $v$  with  $s_v(e) > 0$ .

Our algorithm proceeds as follows. At the start all players are asleep; we will call edges with both endpoints asleep *sleeping edges*, and all other edges *awake edges*. Let  $S_i$  denote the set of sleeping players in iteration  $i$ , and  $\overline{S}_i$  the set of awake players; in the beginning  $S_1 = V$ . In each iteration  $i$ , we pick one player to wake up, and fix its contributions on all of its adjacent edges. In particular, we choose a node  $v \in S_i$  with the currently highest derivative value  $h'_v$  (see below for tie-breaking rule). We set  $v$ 's contribution to an edge  $e = (u, v)$  to  $s_v(e)$ , where  $s_v \in BR_v(S_i)$ . Define  $\overline{BR}_v(S_i)$  as the set of best responses in  $BR_v(S_i)$  for which  $s_v(e) = s_u(e)$  for all awake edges  $e = (u, v)$ . For  $s_v \in \overline{BR}_v(S_i)$  player

$v$  exactly matches the contributions of the awake nodes  $\overline{S_i}$  on all awake edges between  $v$  and  $\overline{S_i}$ . It is possible to show that  $\overline{BR}_v(S_i)$  is non-empty, and our algorithm sets the contributions of  $v$  to  $s_v \in \overline{BR}_v(S_i)$ . Moreover, we set the contribution of other sleeping players  $u \in S_i$  to be  $s_u(e) = s_v(e)$  on the sleeping edges, so we assume  $u$  fully matches  $v$ 's contribution on edge  $e$ .  $u$  will not change its contributions on these edges when it is woken up, and  $v$  receives exactly the reward of  $\overline{BR}_v(S_i)$ . Now that node  $v$  is awake, we compute  $\overline{BR}_u(S_i - \{v\})$  for all sleeping  $u$ , as well as new values  $h'_u$  and iterate. Values  $h'_u$  in later iterations refer to the minimum derivative values on all the *sleeping* edges neighboring  $u$ .

We still need a tie-breaking rule for choosing a node to wake up when there are several nodes with equal  $h'_v$ . Let  $s_v \in \overline{BR}_v(S_i)$  that we compute. For every edge  $e = (u, v)$  with  $h'_u = h'_v$ , we want to pick  $u$  with  $s_u(e) \leq s_v(e)$ . We claim that there is a node  $u$  such that this is true w.r.t. all its neighbors. Suppose a node  $u$  has two edges  $e = (u, v)$  and  $e' = (u, w)$  with  $h'_u = h'_v = h'_w$  and  $s_u(e) > s_v(e)$  but  $s_u(e') < s_w(e')$ . It can be shown that the functions on  $(u, v)$  and  $(u, w)$  are linear in this range. This implies that  $h_{e'}^+(s_u(e')) = h_e^-(s_u(e))$  because  $h_e^-(s_u(e)) = h'_u = h'_w = h_{e'}^-(s_w(e')) \leq h_{e'}^+(s_u(e'))$ , because  $h_{e'}$  is concave. Hence,  $u$  can move some amount of effort from  $e$  to  $e'$  and still form a best response. Continuing in this manner implies that there is  $u \in S_i$  with  $s_u \in \overline{BR}_u(S_i)$  such that  $s_u(e) \leq s_v(e)$  for all neighbors  $v$ , i.e., our tie-breaking is possible.

We first show that our algorithm forms a feasible solution, i.e., that the budget constraints are never violated. When node  $v$  is woken up and sets its  $s_v(e)$  on a newly awake edge  $e = (u, v)$ , the other sleeping player  $u$  must have enough available budget to match  $s_v(e)$ . In  $s_u \in \overline{BR}_u(S_{i-1})$  that our algorithm computes, let  $\overline{B_u}$  be the available budget of node  $u$ , that is,  $\overline{B_u} = B_u - \sum_{e=(u,w), w \in \overline{S_i}} s_w(e)$ . It is the maximum amount that  $u$  could assign to  $e$ .

For contradiction, assume that  $s_v(e) > \overline{B_u}$ . Then it must be that  $h_e^-(s_u(e)) \geq h_e^-(\overline{B_u}) \geq h_e^-(s_v(e))$ , since  $h_e$  is concave. By definition of  $h'_v$ , we know that  $h_e^-(s_v(e)) \geq h'_v$ , and so  $h_e^-(s_u(e)) \geq h'_v$ . Now let  $e' = (u, w)$  be the edge that achieves the value  $h'_u$ , i.e.,  $h_{e'}^-(s_u(e)) = h'_u$ . If  $h_{e'}^-(s_u(e)) < h'_v$ , then  $h_{e'}^-(s_u(e)) < h_e^-(\overline{B_u}) \leq h_e^+(s_u(e))$ , so  $s_u$  cannot be a best response, since  $u$  could earn more reward by switching some amount of effort from  $e'$  to  $e$ . Therefore, we know that  $h'_u \geq h'_v$ . If this is a strict inequality, then we have a contradiction, since  $u$  would have been woken up before  $v$ . Therefore, it must be that  $h'_u = h'_v$ . But this contradicts our tie-breaking rule – we would choose  $u$  before  $v$  because it puts less effort onto edge  $e$  in our choice from  $\overline{BR}_u(S_{i-1})$  than  $v$  does in  $\overline{BR}_v(S_{i-1})$ . Therefore, our algorithm creates a feasible solution.

Re-number the nodes  $v_1, v_2, \dots, v_n$  in the order that we wake them. We need to prove that the algorithm computes a pairwise equilibrium. We can show that all the contributions in the final solution are symmetric, and that node  $v_i$  gets exactly the reward  $\overline{BR}_{v_i}(S_i)$  in the final solution.

To prove that the above algorithm computes a pairwise equilibrium, we show by induction on  $i$  that node  $v_i$  will never have incentive to deviate, either unilaterally or bilaterally. This is clearly true for  $v_1$ , since it obtains the maximum possible reward that it could have in *any* solution, which proves the base case.

We now assume that this is true for all nodes earlier than  $v_i$ , and prove it for  $v_i$  as well. It is clear that  $v_i$  would not deviate unilaterally, since it is getting the reward of  $BR_{v_i}(S_i)$ . This is at least as good as any best response when it cannot control the strategies of any nodes except itself. By the inductive hypothesis,  $v_i$  would not deviate bilaterally with a node  $v_j$  such that  $j < i$ .  $v_i$  would also not deviate bilaterally with a node  $v_j$  such that  $j > i$ , since when forming  $\overline{BR}_{v_i}(S_i)$  node  $v_i$  can set the strategy of node  $v_j$ . So in  $\overline{BR}_{v_i}(S_i)$  node  $v_i$  achieves a reward better than any deviation possible with nodes from  $S_i$ . This completes the proof that our algorithm always finds a pairwise equilibrium.

A complete proof and a proof of uniqueness for strictly concave functions appears in the full version [6] of the paper.  $\square$

## 4 Additional Aspects

We showed above several classes of functions for which pairwise equilibrium exists and the price of anarchy is small. Another class that has this property are maximum effort games, in which  $f_e(x, y) = h_e(\max(x, y))$  for arbitrary increasing  $h_e$ . In the full version we show that in this case a pairwise equilibrium always exists, the price of stability is 1, and the price of anarchy is 2.

If we consider *approximate* equilibria, the following theorem says that an optimal and near-stable state always exists. An  $\alpha$ -approximate equilibrium is a state where players may gain utility by deviating (either unilaterally or bilaterally) by no more than a factor of  $\alpha$ .

**Theorem 10.** *In network contribution games a social optimum  $s^*$  is a 2-approximate equilibrium for any class of nonnegative reward functions.*

Our results on convergence are deferred to the full version. We can show that: (1) for general convex functions, natural game dynamics converge to a pairwise equilibrium (when one exists) in polynomial time; (2) for minimum effort games with convex functions, such convergence may not occur; and (3) for minimum effort games with concave functions, natural dynamics converge to a pairwise equilibrium (when one exists), but the convergence time may be unbounded.

## 5 Conclusions

In this paper we propose and study simple models of contribution games, in which agents can invest a fixed budget into different projects. An important issue is to extend our results for network contribution games to general contribution games, where a project may consist of more than two participants, i.e., the graph of projects is actually a hypergraph.

It is also worthwhile to explore cases in which a player has a function characterizing “payments” for the total effort that he invests in all projects. Such “price” functions are often assumed to be linear or convex (e.g., in [9, 24]).

## References

1. Ackermann, H., Goldberg, P., Mirrokni, V., Röglin, H., Vöcking, B.: Uncoordinated two-sided matching markets. In: Proc. 9th EC, pp. 256–263 (2008)
2. Alós-Ferrer, C., Weidenholzer, S.: Imitation in minimum effort network games (2010) (unpublished manuscript)
3. Andelman, N., Feldman, M., Mansour, Y.: Strong price of anarchy. *Games Econ. Behav.* 65(2), 289–317 (2009)
4. Anderson, S., Goeree, J., Holt, C.: Minimum-effort coordination games: Stochastic potential and logit equilibrium. *Games Econ. Behav.* 34(2), 177–199 (2001)
5. Anshelevich, E., Das, S., Naamad, Y.: Anarchy, stability, and utopia: Creating better matchings. In: Proc. 2nd SAGT, pp. 159–170 (2009)
6. Anshelevich, E., Hoefer, M.: Contribution Games in Social Networks. arXiv preprint 1004.1854
7. Babaioff, M., Feldman, M., Nisan, N.: Combinatorial agency. In: EC 2006, p. 28 (2006)
8. Ballester, C., Calvó-Armengol, A., Zenou, Y.: Who's who in networks. Wanted: The key player. *Econometrica* 74(5), 1403–1417 (2006)
9. Bramoullé, Y., Kranton, R.: Public goods in networks. *J. Econ. Theory* 135(1), 478–494 (2007)
10. Chakraborty, T., Kearns, M., Khanna, S.: Network bargaining: Algorithms and structural results. In: Proc. 10th EC, pp. 159–168 (2009)
11. Chaudhuri, A., Schotter, A., Sopher, B.: Talking ourselves to efficiency: Coordination in inter-generational minimum effort games with private, almost common and common knowledge of advice. *The Economic Journal* 119(534), 91–122 (2008)
12. Corbo, J., Calvó-Armengol, A., Parkes, D.: The importance of network topology in local contribution games. In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 388–395. Springer, Heidelberg (2007)
13. Daskalakis, C., Papadimitriou, C.: On a network generalization of the Minmax theorem. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 423–434. Springer, Heidelberg (2009)
14. Devetag, G., Ortmann, A.: When and why? A critical survey on coordination failure in the laboratory. *Experimental Economics* 10(3), 331–344 (2007)
15. Dufwenberg, M., Gneezy, U.: Gender and coordination. In: Rapoport, A., Zwick, R. (eds.) Experimental Business Research, vol. 3, pp. 253–262. Kluwer, Dordrecht (2005)
16. Fabrikant, A., Luthera, A., Maneva, E., Papadimitriou, C., Shenker, S.: On a network creation game. In: Proc. 22nd PODC, pp. 347–351 (2003)
17. Gusfield, D., Irving, R.: *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge (1989)
18. Hoefer, M., Suri, S.: Dynamics in network interaction games. In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 294–308. Springer, Heidelberg (2009)
19. Jackson, M.: *Social and Economic Networks*. Princeton University Press, Princeton (2008)
20. Jackson, M., Wolinsky, A.: A strategic model of social and economic networks. *J. Econ. Theory* 71(1), 44–74 (1996)
21. Kleinberg, J., Suri, S., Tardos, É., Wexler, T.: Strategic network formation with structural holes. In: Proc. 9th EC 2008, pp. 284–293 (2008)
22. Kleinberg, J., Tardos, É.: Balanced outcomes in social exchange networks. In: Proc. 40th STOC 2008, pp. 295–304 (2008)
23. Laoutaris, N., Poplawski, L., Rajaraman, R., Sundaram, R., Teng, S.-H.: Bounded budget connection (BBC) games or how to make friends and influence people, on a budget. In: Proc. 27th PODC 2008, pp. 165–174 (2008)
24. van Huyck, J., Battalio, R., Beil, R.: Tacit coordination games, strategic uncertainty and coordination failure. *Amer. Econ. Rev.* 80(1), 234–248 (1990)

# Improved Bounds for Online Stochastic Matching

Bahman Bahmani\* and Michael Kapralov\*

Stanford University  
`{bahman,kapralov}@stanford.edu`

**Abstract.** We study the online stochastic matching problem in a form motivated by Internet display advertisement. Recently, Feldman et al. gave an algorithm that achieves 0.6702 competitive ratio, thus breaking through the  $1 - 1/e$  barrier. One of the questions left open in their work is to obtain a better competitive ratio by generalizing their two suggested matchings (TSM) algorithm to  $d$ -suggested matchings ( $d$ -SM).

We show that the best competitive ratio that can be obtained with the *static* analysis used in the  $d$ -SM algorithm is upper bounded by 0.76, even for the special case of  $d$ -regular graphs, thus suggesting that a *dynamic* analysis may be needed to improve the competitive ratio significantly. We make the first step in this direction by showing that the RANDOM algorithm, which assigns an impression to a randomly chosen eligible advertiser, achieves  $1 - e^{-d} d^d / d! = 1 - O(1/\sqrt{d})$  competitive ratio for  $d$ -regular graphs, which converges to 1 as  $d$  increases. On the hardness side, we improve the upper bound of 0.989 on the competitive ratio of any online algorithm obtained by Feldman et al. to  $1 - 1/(e + e^2) \approx 0.902$ . Finally, we show how to modify the TSM algorithm to obtain an improved 0.699 approximation for general bipartite graphs.

## 1 Introduction

Bipartite matching problems are among the central problems in combinatorial optimization with numerous applications. In this paper, we study a variant of the online bipartite matching problem motivated by applications in Internet advertising.

We are given a graph  $G = (A, I, E)$ , where  $A$  is a set of advertisers,  $I$  is the set of impression types and  $E$  is the set of edges between them. For  $a \in A, i \in I$  the presence of an edge  $(a, i)$  indicates that advertiser  $a$  is bidding for impression type  $i$ . Advertisers are fixed and known in advance, while impressions of different types come online, and the task of the algorithm is to assign each impression to an advertiser as soon as the impression arrives or discard the impression. The objective is to maximize the number of matched impressions, subject to each advertiser being assigned at most one impression and each impression being assigned to at most one advertiser, i.e. subject to the allocation forming a matching.

---

\* Research supported by a Stanford Graduate Fellowship.

If nothing is known about the graph  $G$  in advance and edges incident to impression types are revealed online, then we are in the online adversarial setting, for which an algorithm achieving the optimal approximation ratio of  $1 - 1/e$  was given in [1]. A less restrictive model is the random order model, which has received a significant amount of attention recently. This model assumes that impression types and the graph  $G$  are unknown but appear in the random order. Even the greedy algorithm has a competitive ratio of  $1 - 1/e$  in this model [2]. It is also known that no deterministic algorithm can achieve approximation ratio better than 0.75 and no randomized algorithm better than 0.83 [2]. However, the adversarial model or random order model may be too restrictive for applications, where statistics about frequencies of various impression types may be known. In the iid model, which we consider in this paper, we assume that impressions are drawn from a known distribution  $\mathcal{D}$  on the set of *impression types* for which the edges to advertisers are known in advance. The  $1 - 1/e$  bound of [1] was the best known approximation for the stochastic online model, until recently Feldman et al. gave an algorithm that achieves 0.6702 competitive ratio. They also showed that no algorithm can achieve a competitive ratio better than 0.989. Their main algorithmic technique is a novel application of the power of two choices consisting of carefully computing two edge-disjoint (near)-matchings offline to guide the online allocation. One of the questions left open by Feldman et al. is to obtain a better competitive ratio by generalizing the two suggested matchings (TSM) idea to  $d$ -suggested matchings ( $d$ -SM).

## 1.1 Our Results and Techniques

In this paper we give improved upper and lower bounds for three questions related to the stochastic matching problem.

First, we show that the *static* analysis inherent in the  $d$ -suggested matchings algorithm of [3] cannot achieve an competitive ratio better than 0.76, suggesting that a *dynamic* analysis of the allocation process would be needed to achieve a higher competitive ratio, even for the special case of  $d$ -regular graphs. We make a first step in that direction by proving.

**Theorem 1.** *For any  $\epsilon > 0$  as the size of the  $d$ -regular expected graph  $G = (A, I, E)$  goes to  $\infty$ , with high probability the algorithm RANDOM has a competitive ratio at least as large as  $1 - e^{-d}d^d/d! - \epsilon$ .*

The analysis is based on a linear program that lower bounds the evolution of the allocation process as new impressions arrive. Interestingly, this bound coincides with the static bounds of  $1 - 1/e$  and  $1 - 2/e^2$  for  $d = 1, 2$ , but is strictly better than the static bound for any  $d > 2$ .

The proof of Theorem 1 proceeds by showing that RANDOM obtains a matching of size at least  $(1 - e^{-d}d^d/d! - \epsilon)n$  whp. It is interesting to note that this bound is tight, i.e. there exists a family of  $d$ -regular graphs for which the expected size of the matching constructed by RANDOM is  $(1 - e^{-d}d^d/d!)n + o(n)$ .

On the upper bound side, we prove

**Theorem 2.** *The expected competitive ratio of any algorithm for online stochastic matching is bounded above by  $\frac{1-1/e+1/e^2}{1+1/e} \approx 0.901$ .*

This improves upon the 0.9898 hardness result obtained in [3].

Finally, we show that the  $\frac{1-2/e^2}{4/3-2/3e} \approx 0.67$  approximation for general bipartite graphs obtained in [3] can be slightly improved without going beyond subgraphs of degree 2:

**Theorem 3.** *There exists an algorithm that for any  $\epsilon > 0$  one has competitive ratio at least  $0.699 - \epsilon$  with high probability.*

It is interesting to point out that the worst case competitive ratio is achieved in Theorem 3 in the regime when a bound on the performance of any online algorithm similar to Theorem 2 holds. In particular, a slightly better bound can be proved if one compares the performance of the algorithm to be best possible performance of an *online* algorithm. However, we do not pursue this direction here.

## 1.2 Other Related Work

The related online decision problem, the *ad allocation problem*, which is motivated by sponsored search auctions, has been studied extensively in the literature. In this problem, every edge  $(a, i) \in E$  has a weight that corresponds to the bid of advertiser  $a$  for impression  $i$ , and each advertiser has a budget. Now in addition to forming a matching, the allocation of impressions to advertisers now has to satisfy budget constraints, and the objective is to maximize the total value of bids for the assigned impression. The offline version of the problem is NP-hard ([4]) and several approximation algorithms have been designed ([2], [5]). Approximation algorithms have also been designed for the online setting ([6], [2], [7], [8]), typically achieving  $1 - 1/e$  approximation under the assumption that bids are small compared to budgets. An  $1 - \epsilon$  approximation for any  $\epsilon > 0$  has recently been obtained by [9] in the *random permutation model* under the assumption that the optimum is significantly larger than the maximum bid.

## 1.3 Preliminaries

We start with a formal definition of the problem. We are given a bipartite graph  $G = (A, I, E)$  of advertisers  $A$  and impression types  $I$ , together with an integer number of impressions of type  $i$  that we expect to see. We denote this number by  $e_i$  and define  $n := \sum_{i \in I} e_i$ . We assume that at each step an impression of type  $i$  arrives with probability  $e_i/n$  and denote this distribution by  $\mathcal{D}$ . Then,  $n$  i.i.d. draws  $i \sim \mathcal{D}$ , of impression types arrive, and as soon as an impression  $i$  arrives, we should either assign it to an advertiser  $a$  such that  $(i, a) \in E$ , or not assign  $i$  at all. Each advertiser  $a \in A$  may be assigned at most once. Our goal is to maximize the number of assigned impressions.

More formally, If  $D(i)$  is the set of arrivals of the impression type  $i$ , the *realization graph*  $\hat{G}$  is defined as  $\hat{G} = (A, \hat{I}, \hat{E})$ , where  $\hat{I} = \cup_{i \in I} D(i)$  and  $\hat{E} = \{(a, i') \mid i' \in D(i) \text{ and } (a, i) \in E\}$ . Then, we would like to design an algorithm  $ALG$ , which having access to  $G, \mathcal{D}$  and  $n$  finds a matching, in an online fashion, in  $\hat{G}$ , such that with high probability  $ALG(\hat{G})/OPT(\hat{G}) \geq \alpha$  for some  $\alpha$ , where  $OPT(\hat{G})$  is the size of the maximum matching in  $\hat{G}$ .

It is shown in [3] that one can assume without loss of generality that  $\mathcal{D}$  is the uniform distribution (indeed, it is sufficient to duplicate impression types in  $I$  an appropriate number of times to achieve that). Hence, in the rest of this paper we will make this assumption, and thus also have that  $n = |I|$ .

#### 1.4 $d$ -Suggested Matchings Algorithm

A family of algorithms for the introduced problem, namely  $d$ -Suggested Matching ( $d$ -SM), was presented in [3]. The  $d$ -SM algorithm works by finding  $d$  edge-disjoint (near) matchings in the expected graph,  $G$ , and then use these matchings to guide the online assignments of impressions.

More precisely, the algorithm finds (e.g. by using max flows in a boosted graph)  $d$  edge-disjoint near-matchings  $M_1, M_2, \dots, M_d$ , and then for the  $j^{th}$  arrival of the impression type  $i$ , the algorithm tries to match the impression to the advertiser  $a$  connected to  $i$  in  $M_j$ . If  $a$  is already matched, then the impression is not assigned at all.

It is proved in [3] that the 1-SM algorithm achieves  $1 - 1/e = 0.63$  competitive ratio, and that two near-matchings can be carefully chosen so that 2-SM (aka TSM) achieves  $\frac{1-2/e^2}{4/3-2/3e} \approx 0.67$  competitive ratio for general bipartite graphs. But, the problem of finding a decomposition of general bipartite graphs into  $d$  edge-disjoint near-matchings to achieve a better competitive ratio is left open. This problem is trivial for  $d$ -regular graphs, which can be decomposed into a union of  $d$  edges-disjoint perfect matchings. However, we show that the  $d$ -SM algorithm cannot achieve a competitive ratio better than 0.76 for *any*  $d$ , thus suggesting that a *dynamic* analysis is needed to improve the current competitive ratio significantly. We make a first step towards such dynamic analysis by proving that the RANDOM algorithm achieves a competitive ratio of  $1 - e^{-d} d^d / d! = 1 - O(1/\sqrt{d})$  on  $d$ -regular graphs.

#### 1.5 Organization

In section 2 we prove an upper bound on the performance of the  $d$ -SM algorithm and analyze the performance of RANDOM algorithm on  $d$ -regular graphs. Section 3 contains the improved hardness result, and section 4 presents an improved version of the TSM algorithm.

## 2 Dynamic Analysis for Regular Graphs

In this section we first upper bound the performance of the  $d$ -SM algorithm for any  $d$  and then analyze the RANDOM algorithm for allocation on regular graphs, showing that it achieves a competitive ratio of  $1 - e^{-d} d^d / d!$  on  $d$ -regular graphs.

## 2.1 Upper-Bounding the Performance of $d$ -SM

In this section, we give an upper-bound on the performance of any  $d$ -SM algorithm (for any value of  $d$ ). In particular, we show that (with  $|I| = n$ ) no  $d$ -SM algorithm can produce a matching of a larger size than  $0.76n$ , even on regular graphs:

**Theorem 4.** *No  $d$ -SM algorithm (for any value of  $d$ ) can achieve a larger matching size than  $0.76n$ , even in expectation on regular graphs.*

The proof of this theorem is deferred to the full version of the paper. The bottleneck in the performance of the  $d$ -SM algorithm is that it is inherently a static algorithm. That is after it calculates the offline matchings, it doesn't consider the dynamics of the arrival sequence. In particular, near the end of the sequence, many of the advertisers are already matched, so when an impression arrives and checks its corresponding advertiser, there is a good chance that advertiser is already matched and the impression is thrown away. However, there may be other advertisers connected to this impression that are not yet matched and could be used for this impression but don't get utilized. In the next section, we present a simple dynamic algorithm that can get a much better competitive ratio for regular graphs.

## 2.2 The Random Algorithm

As in the previous section,  $G = (A, I, E)$  is assumed to be a  $d$ -regular bipartite graph with  $|I| = n$ . The algorithm simply assigns each arriving impression to an unmatched connected advertiser chosen uniformly at random.

In this section, we analyze the performance of this algorithm and show that it achieves a significantly better competitive ratio than  $d$ -SM on regular graphs.

We give a few definitions first.

**Definition 1.** – We denote the  $t^{th}$  arriving impression by  $i_t$ . Whenever we refer to time  $t$ , we mean the time just before the arrival of  $i_t$ .

- $A_u(t)$  is the set of unmatched advertisers at time  $t$ .
- $S_t$  is all the state information at time  $t$ , i.e. the sequence of all impressions that have arrived by time  $t$  as well as the sequence of all assignments made by the algorithm up to time  $t$ .
- The set of impressions of remaining degree  $k$  at time  $t$  is denoted by  $I_k(t)$ . In other words,

$$I_k(t) = \{i \mid |N(i) \cap A_u(t)| = k\} \quad \forall 0 \leq k \leq d$$

where  $N(i)$  is the neighborhood of impression  $i$ .

- The fraction of impressions of remaining degree  $k$  at time  $t$ :

$$R_k(t) = \frac{|I_k(t)|}{n}$$

- For any  $i \in I$ ,  $a \in A$ , and  $1 \leq t \leq n$ :

$$P(i, a, t) = \Pr[i \text{ gets matched to } a \text{ at time } t \mid i_t = i, S_t]$$

- for any  $i \in I$ ,

$$P(i, t) = \Pr[\text{the remaining degree of } i \text{ changes at time } t \mid S_t]$$

- For  $1 \leq k \leq d$ ,

$$P_k(t) = E[\text{fraction of impression types whose remaining degree changes from } k \text{ to } k-1 \text{ at time } t \mid S_t]$$

- We denote the expectations of the random variables defined above by corresponding lower-case letters. That is,

$$\begin{aligned} r_k(t) &= E[R_k(t)], p(i, a, t) = E[P(i, a, t)], \\ p(i, t) &= E[P(i, t)], p_k(t) = E[P_k(t)] \end{aligned}$$

where all expectations are with respect to the randomness in  $S_t$ .

- We denote the matching constructed by the algorithm up to time  $t$  by  $M_t$  and the final matching (i.e. after the arrival of  $i_n$ ) by  $M$ .

**Fact 5.** The following are clear:

$$\begin{aligned} P(i, t) &= \sum_{a \in N(i) \cap A_u(t)} \sum_{j \in N(a)} P(j, a, t) / n \\ P_k(t) &= \frac{1}{n} \sum_{i \in I_k(t)} P(i, t) \end{aligned}$$

We would like to analyze  $|M|$ , the size of the final matching constructed by the algorithm. We start with a lemma, which is proved in the full version of the paper.

**Lemma 1.**  $|M|$  is sharply concentrated around  $E[|M|]$ .

So, we only need to analyze the expectations (or in other words, only  $E[|M|]$ ). We have the following lemmas, proved in the full version of the paper:

**Lemma 2**

$$E[|M|] = \sum_{t=1}^n (1 - r_0(t))$$

**Lemma 3**

$$r_k(t+1) - r_k(t) = p_{k+1}(t) - p_k(t) \quad \forall 1 \leq k < d$$

$$\begin{aligned} r_0(t+1) - r_0(t) &= p_1(t) \\ r_d(0) = 1, \sum_{k=0}^d r_k(t) &= 1, r_k(t) \geq 0 \end{aligned}$$

The next lemma is essential in our analysis. But, before presenting it, we notice that since the algorithm picks a connected unmatched advertiser uniformly at random at each time, we have:

$$P(i, a, t) = \frac{1\{a \in N(i) \cap A_u(t)\}}{|N(i) \cap A_u(t)|}$$

This fact is used in the proof of the lemma, which is given in the full version of the paper:

**Lemma 4**

$$\sum_{k'=1}^k p_{k'}(t) \leq \frac{d}{n} \sum_{k'=1}^k r_{k'}(t)$$

From the previous two lemmas, we get that to lower-bound  $E[|M|]$ , we can solve the following LP:

$$\begin{aligned} & \text{maximize} && \sum_{t=1}^n r_0(t) \\ \text{s.t.} & && r_0(t+1) - r_0(t) = p_1(t) \quad 1 \leq t \leq n-1 \\ & && r_k(t+1) - r_k(t) = p_{k+1}(t) - p_k(t) \quad \forall 1 \leq k < d, 1 \leq t \leq n-1 \\ & && r_d(t+1) - r_d(t) = -p_d(t) \quad 1 \leq t \leq n-1 \\ & && r_d(1) = 1, \quad r_k(1) = 0 \quad \forall 0 \leq k < d \\ & && \sum_{k'=1}^k p_{k'}(t) \leq \frac{d}{n} \sum_{k'=1}^k r_{k'}(t) \quad \forall 1 \leq k \leq d, 1 \leq t \leq n \end{aligned}$$

We present the solution to the above LP. To simplify the presentation of the solution, we introduce the following notation:

**Definition 2.** For any non-negative integers  $k, t$ :

$$\theta(k, t) = \left(\frac{d}{n}\right)^k \left(1 - \frac{d}{n}\right)^{t-k-1} \binom{t-1}{k}$$

By the properties of the binomial coefficients, we have  $\theta(k, t) = 0$  for any  $k > t - 1$ , and also  $\sum_{k=0}^{t-1} \theta(k, t) = 1$ .

Now, the following proposition, proven in the full version of the paper, gives the LP solution:

**Proposition 1.** The solution to the above LP is as follows:

$$\begin{aligned} r_{d-k}(t) &= \theta(k, t) \quad \forall 0 \leq k < d, 1 \leq t \leq n \\ r_0(t) &= 1 - \sum_{j=1}^d r_j(t) \quad \forall 1 \leq t \leq n \\ p_j(t) &= \frac{d}{n} r_j(t) \quad \forall 1 \leq j \leq d, 1 \leq t \leq n \end{aligned}$$

From the above proposition, we get that the performance of the algorithm is bounded as follows:

$$\frac{E[|M|]}{n} \geq \frac{1}{n} \sum_{k=0}^{d-1} \sum_{t=1}^n \left(\frac{d}{n}\right)^k \left(1 - \frac{d}{n}\right)^{t-1-k} \binom{t-1}{k}$$

It can be seen that as  $n \rightarrow \infty$ , the above summation converges to:

$$\sum_{k=0}^{d-1} \int_0^1 e^{-dx} (dx)^j / j! dx = 1 - e^{-d} d^d / d!$$

This and the sharp concentration of  $|M|$  give the following result:

**Theorem 6.** *For any  $\epsilon > 0$  as the size of the  $d$ -regular expected graph  $G = (A, I, E)$  converges to  $\infty$ , with high probability the algorithm RANDOM achieves a competitive ratio at least as large as  $1 - e^{-d} d^d / d! - \epsilon$ .*

It remains to note that the bound on the size of the matching is tight: it is achieved for a family of graphs  $G_n$  that consist of a disjoint union of  $n/d$  copies of  $K_{d,d}$ .

Comparing this lower bound on the performance of RANDOM (which converges to 1 as  $1 - 1/\sqrt{d}$  when  $d \rightarrow \infty$ ) with the constant (0.76) upper bound on the performance of  $d$ -SM, we conclude that the RANDOM algorithm achieves a significantly better competitive ratio than  $d$ -SM on regular graphs.

### 3 Hardness

Feldman et al. [3] prove that no online algorithm for the stochastic matching problem can achieve a better competitive ratio than 0.99. In this section, we prove that in fact no online algorithm can achieve a better competitive ratio than 0.902:

**Theorem 7.** *The expected competitive ratio of any algorithm for online stochastic matching is bounded above by  $\frac{1-1/e+1/e^2}{1+1/e} \approx 0.901062$ .*

*Proof.* We exhibit an instance  $G = (A, I, E)$  of the problem for which no online algorithm yields a matching of expected size larger than  $\frac{1-1/e+1/e^2}{1+1/e} |I|$ , even though a matching of size  $(1 - \epsilon)|I|$  exists with high probability. Let  $G = (I_1 \cup I_2, A_1 \cup A_2, E)$  be defined as follows:

1.  $|I_1| = |A_2| = n$ ,  $|A_1| = |I_2| = n/e$ .
2. There is a perfect matching  $M$  between  $I_1$  and  $A_2$ .
3. There is a complete graph between  $I_2$  and  $A_2$ , and between  $I_1$  and  $A_1$ .

We first show that with high probability there exists a matching of size  $(1 + 1/e - \epsilon)n$  in the realization graph. By the balls and bins analysis, there will be  $(1 - 1/e)n$  distinct arrivals in  $I_1$ . Route the first arrivals in  $I_1$  to  $A_2$  and the

rest to  $A_1$  (this is possible since there is a complete graph between  $I_1$  and  $A_1$ , and  $|A_1| = |I_1|/e$ ). This leaves  $|A_2|/e$  advertisers in  $A_2$  unmatched. These are matched to impressions from  $I_2$ . Hence, a matching of size  $(1 + 1/e - \epsilon)n$  exists with high probability.

Consider an advertiser  $a \in A_2$  that is matched by  $ALG$  to an impression in  $I_2$ . Fix a time  $t \in [1, (1 + 1/e)n]$  (recall that time  $t$  corresponds to the moment just before the arrival of the  $t^{th}$  impression). We call advertiser  $a$  *good* if  $M(a)$  has not arrived and *bad* otherwise (where  $M(a)$  is the impression connected to  $a$  by an edge in  $M$ ). Denote the number of good advertisers at time  $t$  by  $X_t$ , and the number of good advertisers at the end of the sequence by  $X = X_{(1+1/e)n+1}$ . Note that the size of the final matching constructed by  $ALG$  is upper bounded by  $n/e + (1 - 1/e)n + X$ .

At time  $t$  an impression  $i \in I_1$  arrives with probability  $1/(1 + 1/e)$  and it is a unique arrival incident on a good advertiser with probability  $X_t/n$ . Hence, we have:

$$\mathbf{E}[X_{t+1} - X_t | X_t] \leq \frac{1/e}{1 + 1/e} - \frac{1}{1 + 1/e} X_t/n, X_1 = 0,$$

which implies that

$$E[X_t] \leq (n/e) \left( 1 - \left( 1 - \frac{1}{(1 + 1/e)n} \right)^{t-1} \right)$$

This yields that  $E[X] \leq (n/e)(1 - e^{-1})$ , and hence, in expectation, at most  $\frac{1}{e}(1 - e^{-1})n$  of the advertisers that are matched to  $I_2$  are good at the end of the sequence. This concludes that the expected size of the matching constructed by  $ALG$  can not be larger than  $n/e + (1 - 1/e)n + 1/e(1 - 1/e)n = (1 + 1/e - 1/e^2)n$ , and hence  $ALG$ 's competitive ratio can not be better than  $\frac{1+1/e-1/e^2}{1+1/e} \approx 0.901062$ , which finishes the proof.  $\square$

## 4 Improved Competitive Ratio for General Graphs

In this section we present an algorithm for the online stochastic matching problem in general graphs that yields an 0.699-approximation to the offline optimum.

We start by giving an outline of the algorithm of [3]. In the offline phase, the TSM algorithm constructs a boosted flow graph  $G_f$ , where each  $a \in A$  is connected to a source  $s$  by an edge with capacity 2, each  $i \in I$  is connected to a sink  $t$  by an edge with capacity 2 and each edge of  $G$  is directed from  $A$  to  $I$  and assigned capacity 1. One then finds an integral maxflow in  $G_f$ . Denote the edges that carry flow by  $E_f$ . The flow edges form a union of paths and cycles, which are colored blue and red in an alternating fashion (with some extra care taken for various types of paths). The online algorithm proceeds as follows: (1) assigns the first arrival along the blue edge if it is still available, and discards otherwise, (2) assigns the second arrival to the red edge if it is still available, discards otherwise.

It was shown in [3] that this algorithm yields an  $(1 - 2/e^2)/(4/3 - 2/(3e)) \approx 0.67029$  approximation, which is tight for their algorithm. In what follows we show how to modify the flow graph constructed by the TSM algorithm to obtain an approximation ratio of 0.699 against the optimal offline algorithm.

Our algorithm works on the flow graph constructed by the algorithm of [3]. As in [3], denote the reachability min-cut corresponding to the max-flow in the boosted flow graph  $G_f$  by  $(A_S \cup I_S, A_T \cup I_T)$ , where  $A_S \cup I_S$  is the source side of the cut and  $A_T \cup I_T$  is the sink side, and denote the set of edges crossing the cut by  $E_\delta$ . It was shown in [3] that the min-cut can always be chosen so that the edges in  $E_\delta$  form a matching, and we make that assumption on  $E_\delta$ .

The intuition behind the algorithm is as follows. Recall that the analysis of the TSM algorithm uses the reachability cut  $(A_S \cup I_S, A_T \cup I_T)$  in the boosted flow graph to bound the optimum in  $\hat{G}$ . The key insight is that this bound on  $OPT$  can sometimes be improved by using a different cut in  $\hat{G}$ . In order to exploit this fact, however, we first modify the flow graph obtained in the TSM algorithm as described in Algorithm 1 below. Intuitively, the modification is based on the fact that the value of the flow in the boosted graph  $G_f$  does not translate directly into the performance of TSM on the subgraph given by  $E_f$ . In particular, the performance of the algorithm improves if the flow obtained via the max-flow computation in  $G_f$  is rerouted so that it is more evenly spread among vertices in  $A_S$  and  $I_T$ . This can be done using two max-flow computations. The two min-cuts obtained from these computations are then used to define a subgraph  $H$  of  $G$  for which we can bound  $OPT$  more carefully. The cut that we use to bound  $OPT(\hat{H})$  then depends on the structure of the new set of flow edges that was obtained.

Denote by  $G_S$  the subgraph induced by vertices of  $A_S \cup I_S$  in  $G$  and by  $G_T$  the subgraph induced by vertices of  $A_T \cup I_T$  in  $G$ . For  $k = 0, 1, 2$  define

$$\begin{aligned} A_S^k &= \{a \in A_S : a \text{ carries } k \text{ units of flow in } E_f\} \\ I_T^k &= \{i \in I_T : i \text{ carries } k \text{ units of flow in } E_f\}. \end{aligned}$$

### Algorithm 1

**Input:**  $G = (A, I, E)$ , the set of edges  $E_f$  carrying max-flow in  $G_f$ . The min-cut  $(A_S \cup I_S, A_T \cup I_T)$ .

**Output:**  $E^*$  - a modified set of paths and cycles in  $G$ .

1. Orient edges of  $G_S$  from  $A_S$  to  $I_S$ , orient flow edges from  $I_S$  to  $A_S$ .
2. Connect vertices in  $A_S^0$  to a source by edges of capacity 1, vertices in  $A_S^2$  to a sink by edges of capacity 1, assign capacity 1 to edges of  $G$ .
3. Find max-flow in  $G_S$ . Denote the set of edges carrying flow by  $E_f^S$ .
4. Orient edges of  $G_T$  from  $I_T$  to  $A_T$ , orient flow edges from  $A_T$  to  $I_T$ .
5. Connect vertices in  $I_T^0$  to a source by edges of capacity 1, vertices  $I_T^2$  to a sink by edges of capacity 1, assign capacity 1 to edges of  $G$ .
6. Find max-flow in  $G_T$ . Denote the set of edges carrying flow by  $E_f^T$ .
7. Set  $E^* := E_\delta \cup E_f^T \cup E_f^S$ .
8. Decompose  $E^*$  into a disjoint union of paths and cycles. Color the edges of the paths and cycles as follows (the same as in [3], given here for completeness):

- Color cycles alternately blue and red;
  - Color odd length paths alternately blue and red, with more blue than red;
  - For even paths that start and end in  $I$ , color the first two edges blue, then alternate red and blue.
  - For even paths that start and end in  $A$ , alternate blue and red.
- 9. (*TSM*)** At runtime, assign the first arrival along the blue edge if it is still available, discard otherwise. Assign the second arrival to the red edge if it is still available, discard otherwise.

We now analyze the performance of Algorithm 1. We start with some definitions. Denote by the  $\mathcal{P}^\delta$  and  $\mathcal{C}^\delta$  the set of paths and cycles respectively in  $E^*$  that contain edges from  $E_\delta$ . Note that since there are no flow edges between  $I_S$  and  $A_T$ , paths and cycles in the flow graph are either contained in one of  $G_S$  and  $G_T$  or contain an edge of  $E_\delta$ . Here and in what follows we will sometimes view the set of paths and cycles  $\mathcal{P}^\delta \cup \mathcal{C}^\delta$  as a set of vertices.

Orient edges of  $G_S$  and  $G_T$  as described above (note that edges from  $E_\delta$  are not oriented since reachability is defined only within  $G_S$  and  $G_T$ ). Denote by  $\mathcal{P}_S^*$  the set of paths in  $E^* \cap E(G_S)$  that are reachable from  $A_S \cap (\mathcal{P}^\delta \cup \mathcal{C}^\delta)$  using edge orientation in  $G_S$  and by  $\mathcal{P}_T^*$  the set of paths in  $E^* \cap E(G_T)$  that are reachable from  $I_T \cap (\mathcal{P}^\delta \cup \mathcal{C}^\delta)$  using edge orientation in  $G_T$ . Also, define  $\mathcal{P}^* := \mathcal{P}_S^* \cup \mathcal{P}_T^*$ .

Let  $H$  be the subgraph induced by  $\mathcal{P}^\delta \cup \mathcal{C}^\delta \cup \mathcal{P}^*$ . Define  $\tilde{A}_T := A_T \setminus V(H)$ ,  $\tilde{I}_T := I_T \setminus V(H)$ ,  $\tilde{I}_S := I_S \setminus V(H)$ ,  $\tilde{A}_S := A_S \setminus V(H)$ .

All proofs from this section have been deferred to the full version of the paper.

The following lemmas are important for our analysis:

**Lemma 5.** *There are no edges between  $\tilde{I}_T$  and  $A \setminus \tilde{A}_T$ , and no edges between  $\tilde{A}_S$  and  $I \setminus \tilde{I}_S$*

**Lemma 6.** *For any  $\epsilon > 0$  one has*

$$\begin{aligned} ALG(\hat{G}) &\geq (1 - 2/e^2)|\tilde{A}_T| + (1 - 2/e^2)|\tilde{I}_S| + ALG(\hat{H}) - \epsilon \\ OPT(\hat{G}) &\leq |\tilde{A}_T| + |\tilde{I}_S| + OPT(\hat{H}) + \epsilon \end{aligned}$$

with high probability.

We now proceed to prove that Algorithm 1 gives a 0.699-approximation to the best offline solution. In light of Lemma 6 and the fact that  $0.699 < 1 - 2/e^2$  it suffices to prove this guarantee for the graph  $H$ .

**Lemma 7.** *For any  $\epsilon > 0$  one has  $ALG(\hat{H})/OPT(\hat{H}) \geq 0.699 - \epsilon$  whp.*

We can now prove Theorem 3:

**Proof of Theorem 3:** It follows from Lemma 7 and Lemma 6 that for any  $\epsilon > 0$  Algorithm 1 achieves a competitive ratio of at least  $0.699 - \epsilon$ .  $\square$

**Remark 1.** It can be seen from the proof of Lemma 7 that the worst case performance of Algorithm 1 occurs when  $(1 - 1/e)|E_\delta| = |A'_S| - |I'_S|$ , i.e. in the

regime in which a bound similar to Theorem 2 on the performance of any online algorithm holds. It is in fact possible to get a slightly better bound if one compares the performance of Algorithm 1 against the best possible performance of an online algorithm, but we do not present that analysis here for the sake of clarity. It can also be noted that the original analysis of the TSM algorithm is tight even when compared against the performance of the best possible online algorithm.

## References

1. Karp, R., Vazirani, U., Vazirani, V.: An optimal algorithm for online bipartite matching. In: STOC (1990)
2. Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to adwords. In: SODA (2008)
3. Feldman, J., Mehta, A., Mirrokni, V., Muthukrishnan, S.: Online stochastic matching: Beating  $1 - 1/e$ . In: FOCS (2009)
4. Azar, Y., Birnbaum, B., Karlin, A., Mathieu, C., Nguyen, C.: Improved approximation algorithms for budgeted allocations. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 186–197. Springer, Heidelberg (2008)
5. Srinivasan, A.: Budgeted allocations in the full-information setting. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 247–253. Springer, Heidelberg (2008)
6. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: Adwords and generalized online matching. In: FOCS (2005)
7. Buchbinder, N., Jain, K., Naor, J.: Online primal-dual algorithms for maximizing ad-auctions. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 253–264. Springer, Heidelberg (2007)
8. Kalyanasundaram, B., Pruhs, K.R.: An optimal deterministic algorithm for online  $b$ -matching. Theoretical Computer Science (2000)
9. Devanur, N., Hayes, T.: The adwords problem: online keyword matching with budgeted bidders under random permutations. In: EC (2009)
10. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)

# Online Stochastic Packing Applied to Display Ad Allocation

Jon Feldman<sup>1</sup>, Monika Henzinger<sup>2</sup>, Nitish Korula<sup>3</sup>,  
Vahab S. Mirrokni<sup>1</sup>, and Cliff Stein<sup>4</sup>

<sup>1</sup> Google Research, 76 9th Ave, New York, NY 10011, U.S.A.

{jonfeld, mirrokni, cstein}@google.com

<sup>2</sup> University of Vienna, Austria

monika.henzinger@univie.ac.at

<sup>3</sup> University of Illinois at Urbana-Champaign

nkorula2@illinois.edu

<sup>4</sup> Google Research and Columbia University, New York, NY

**Abstract.** Inspired by online ad allocation, we study online stochastic packing integer programs from theoretical and practical standpoints. We first present a near-optimal online algorithm for a general class of packing integer programs which model various online resource allocation problems including online variants of routing, ad allocations, generalized assignment, and combinatorial auctions. As our main theoretical result, we prove that a simple dual training-based algorithm achieves a  $(1 - o(1))$ -approximation guarantee in the random order stochastic model. This is a significant improvement over logarithmic or constant-factor approximations for the adversarial variants of the same problems (e.g. factor  $1 - \frac{1}{e}$  for online ad allocation, and  $\log(m)$  for online routing). We then focus on the online display ad allocation problem and study the efficiency and fairness of various training-based and online allocation algorithms on data sets collected from real-life display ad allocation system. Our experimental evaluation confirms the effectiveness of training-based algorithms on real data sets, and also indicates an intrinsic trade-off between fairness and efficiency.

## 1 Introduction

Online stochastic optimization is a central problem in operations research with many applications in dynamic resource allocation. In these settings, given a set of resources, demands for the resources arrive online, with associated values; given a general prior about the demands, one has to decide whether and how to satisfy (i.e., allocate the desired resources to) a demand when it arrives. The goal is to find a valid assignment with maximum total value. Such problems appear in many areas including online routing [7, 3], online combinatorial auctions [9], online ad allocation problems [25, 10, 12], and online dynamic pricing and inventory management problems. For example, in routing problems, we are given a network with capacity constraints over edges; customers arrive online and bid for a subset of edges (typically a path) in the network, and the goal

is to assign paths to new customers so as to maximize the total social welfare. Similarly, in online combinatorial auctions, bidders arrive online and may bid on a subset of resources; the auctioneer should decide whether to sell those resources to the bidder. In the display ads problem, when users visit a website, the website publisher has to choose ads to show them so as to maximize the value of the displayed ads. In this paper, we study these online stochastic resource allocation problems from theoretical and practical standpoints. Our theoretical results apply to a general set of problems including all those discussed above. Our practical results apply to the problem of display ads and give additional validation of our theoretical models and results.

More specifically, we consider the following general class of packing integer programs (PIP): Let  $J$  be a set of  $m$  *resources*; each resource  $j \in J$  has a capacity  $c_j$ . The set of resources and their capacities are known in advance. Let  $I$  be a set of  $n$  *agents* that arrive one by one online, each with a set of *options*  $O_i$ . Each option  $o \in O_i$  of agent  $i$  has an associated *value*  $w_{io} \geq 0$  and requires  $a_{ioj} \geq 0$  units of each resource  $j$ . The set of options and the values  $w_{io}$  and  $a_{ioj}$  arrive together with agent  $i$ . When an agent arrives, the algorithm has to immediately decide whether to assign the agent and if so, which option to choose. The goal is to find a maximum-value allocation that does not allocate more of any resource than is available.

In the *adversarial* or worst-case setting, no online algorithm can achieve any non-trivial competitive ratio; consider the simple case of one resource with capacity one and two agents. For each agent there are just two options, namely to get the resource or not to get it. If an agent gets the resource, he uses its whole capacity. The first agent has value 100 for getting the resource and value 0 for not getting the resource. If he is assigned the resource, then the value of the second agent for getting the resource is 10000, otherwise it is 1. In both cases the algorithm achieves less than 1/100th of the value of the optimal solution. This example can easily be generalized to show that even randomized algorithms cannot achieve non-trivial competitive ratios, even if there is a single resource:

**Theorem 1.** *There is no  $o(\log n / \log \log n)$ -approximation for the online PIP.*

Since in the adversarial setting the lack of prior information about the arrival rate of different types of agents implies strong impossibility results, it is natural to consider *stochastic* settings for online allocation problems, where we may have some prior information about the arrival rate of different types of agents. In particular, we consider the *random-order stochastic model*, in which the agents, their options and associated values may be chosen by an adversary, but the order in which agents arrive is random. We present a training-based online algorithm for the general class of packing integer programs described above and prove that in the random-order stochastic model, it achieves an approximation ratio of  $1 - \varepsilon$ , where  $\varepsilon$  is a function of the parameters of the integer program<sup>1</sup>; more precisely,  $\varepsilon$

---

<sup>1</sup> In this context, an “ $\alpha$ -approximation” means that with high probability under the randomness in the stochastic model, the algorithm achieves at least an  $\alpha$  fraction of the value (efficiency) of the offline optimal solution for the actual instance.

measures how large a fraction of any resource can be demanded by a single agent, or how much a single agent's value contributes to the total objective. Thus, as agents become infinitesimally small, we obtain nearly optimal solutions. This result also implies the same result in the i.i.d. model<sup>2</sup>.

Our *dual-based* algorithm for the stochastic PIP problem observes the first  $\varepsilon$  fraction of the input and then solves an LP on this instance. (This requires knowing the number of agents in advance, at least approximately; Theorem II can be generalized to show that this is unavoidable for any sub-logarithmic approximation.) For each resource, the corresponding dual variable extracted from this LP serves as a (*posted*) *price per unit* of the resource for the remaining agents. The algorithm allocates to each remaining agent the option maximizing his *utility*, defined as the difference between the value of an option and the price he must pay to obtain the necessary resources. We prove that this algorithm provides a  $1 - \varepsilon$  approximation for the large class of natural packing problems we consider, provided that no individual option for any agent consumes too much of any resource or provides too large a fraction of the total value. Specifically we show the following result. Recall that  $n$  and  $m$  denote the number of agents and resources respectively;  $q$  denotes  $\max_i |O_i|$  and  $\text{OPT}$  the value of an optimal off-line solution to the PIP problem.

**Theorem 2.** *The Dual-Based algorithm is  $(1 - O(\varepsilon))$ -competitive for the online stochastic PIP problem with high probability, as long as:*

$$(1) \max_{i,o} \left\{ \frac{w_{io}}{\text{OPT}} \right\} \leq \frac{\varepsilon}{(m+1)(\ln n + \ln q)} \text{ and } (2) \max_{i,o,j} \left\{ \frac{a_{ioj}}{c_j} \right\} \leq \frac{\varepsilon^3}{(m+1)(\ln n + \ln q)}.$$

**Applications:** Theorem 2 has many applications; we elaborate on several, including routing problems, online combinatorial auctions, the display ad problem, and the adword allocation problem. For each of these problems, we improve on the known results for the online version. In each, we will comment on the interpretation of the two conditions of Theorem 2 in that application. Note that in condition (2), one might wish the dependence of  $\varepsilon$  on the input parameters to be linear; this does not seem possible in general. However, for specific applications, one may be able to exploit the structure of the LP to prove tighter bounds; we omit details from this extended abstract. Our experimental results show that in practice, one may be able to use  $\varepsilon$  much smaller than required by the theorem; in particular, for the DA problem, we sample only 1% of the input and obtain a competitive ratio of  $\approx 0.89$ .

In the *online routing* problem, we are initially given a network with capacity constraints over the  $m$  edges. When a customer  $i \in I$  arrives online, she wishes to send  $d_i$  units of flow between some vertices  $s_i$  and  $t_i$ , and derives  $w_i$  units of value from sending such flow. Thus, the set of options  $O_i$  for customer  $i$  is the set of all  $s_i - t_i$  paths in the network. The algorithm must pick a set of customers  $I^* \subseteq I$ , and satisfy their demands by allocating a path to each of them while respecting the capacity constraints on each edge; the goal is to maximize the

---

<sup>2</sup> In the i.i.d model each agent arrives independently and identically drawn from a fixed but unknown probability distribution over the set of possible types of agents [17]. Our random-order stochastic model captures the i.i.d model.

total value of satisfied customers. For this problem, the dual variables learned from the sample yield a price for each edge; each customer is allocated the minimum-cost  $s_i - t_i$  path if its cost is no more than  $w_i$ . In road networks, for instance, these dual variables can be interpreted as the tolls to be charged to prevent congestion. Theorem 2 applies when the contributions of individual agents/vehicles to the total objective or to road congestion are small. As one such example, consider congestion pricing for Manhattan: over a million vehicles enter or leave the island daily, and each of the 12 bridges and tunnels has an annual average daily traffic of over 50,000; using extremely conservative estimates of bridge capacity, we obtain  $\varepsilon \approx 0.15$ . Online routing problems have been studied extensively in the adversarial model when demands can be large, and there are *(poly)-logarithmic* lower and upper bounds even for special cases [37]. Our approach gives a  $(1 - o(1))$ -approximation for the described stochastic variants of these problems when the demands of individual agents are small.

In the *combinatorial auction* problem, we are initially given a set  $J$  of  $m$  goods, with  $c_j$  units for each good  $j \in J$ . Agents arrive online, and the options for agent  $i$  may include different bundles of goods he values differently; option  $o \in O_i$  provides  $w_{io}$  units of value, and requires  $a_{ioj}$  units of good  $j$ . We wish to find a valid allocation maximizing social welfare. Here, the dual variables learned from the sample yield a price per unit of each good; each agent picks the option that maximizes his utility. Here Theorem 2 applies as long as no individual agent controls a large fraction of the market, and as long as the set of options for any single agent is at most exponential in the number of resources. These conditions often hold, as in cases when bidders are single-minded or the number of bundles they are interested in is polynomial in  $n$ , or if their options correspond to using different subsets of the resources. We also observe that the posted prices result in a take-it-or-leave it auction, and thus a truthful online allocation mechanism. Revenue maximization in online auctions using sequence item pricing has been explored recently in the literature [59]. [9] achieves constant-factor approximations for these problems in more general models than we consider.

In the *Display Ads Allocation (DA)* problem [12], there is a set  $J$  of  $m$  advertisers who have contracted with a web publisher for their ads to be shown to visitors to the website. When a visitor is shown an ad, this is called an “impression”. The contract an advertiser  $j$  buys specifies an integer upper bound on the number  $n(j)$  of impressions that he is willing to pay for. A set  $I$  of visitors arrives online; visitor  $i$  has value  $w_{ij} \geq 0$  for advertiser  $j$ . Each visitor can be assigned to at most one advertiser, i.e., there are  $m$  options for an impression, and each option  $o$  has  $a_{ioj} = 1$  for advertiser  $j$ . The goal is to maximize the value of all the assigned impressions. The dual variables learned from the sample yield a discount factor  $\beta_j$  for each advertiser  $j$ ; the algorithm is to assign an impression to advertiser  $j$  that maximizes  $w_{ij} - \beta_j$ . Contracts for advertisers may involve hundreds of thousands of impressions, so the contribution of any one impression/agent is small. As publishers vary greatly in the amount of web traffic they receive, it is relatively meaningless to define a “typical” instance; however, a publisher with 30 million impressions, and 20 advertisers who have

each signed contracts for at least 500,000 impressions, satisfies the hypotheses of Theorem 2 with  $\varepsilon \approx 0.089$ . The adversarial online DA problem was considered in [12], which showed that the problem was inapproximable without exploiting *free disposal*; using this property (that advertisers are at worst indifferent to receiving more impressions than required by their contract), a simple greedy algorithm is  $\frac{1}{2}$ -competitive, which is optimal. When the demand of each advertiser is large, a  $(1 - \frac{1}{e})$ -competitive algorithm exists [12], and this is the best possible. For the *unweighted* (max-cardinality) version of this problem in the i.i.d. model, a 0.67-competitive algorithm has been recently developed [13]; this improves the known  $1 - \frac{1}{e}$ -approximation algorithm for online stochastic matching [19].

The *AdWords (AW)* problem [25,10] is related to the DA problem; here we allocate impressions resulting from search queries. Advertiser  $j$  has a budget  $b(j)$  instead of a bound  $n(j)$  on the number of impressions. Assigning impression  $i$  to advertiser  $j$  consumes  $w_{ij}$  units of  $j$ 's budget instead of 1 of the  $n(j)$  slots, as in the DA problem. Several approximation algorithms have been designed for the *offline* AW problem [8,26,4]. For the online setting, if every weight is very small compared to the corresponding budget, there exist  $(1 - \frac{1}{e})$ -competitive online algorithms [25,6,18,2], and this factor is tight. In order to go beyond the competitive ratio of  $1 - \frac{1}{e}$  in the adversarial model, stochastic online settings have been studied, such as the random order and i.i.d models [18]. Devanur and Hayes [10] described a dual-based  $(1 - \varepsilon)$ -approximation algorithm for this problem in the random order model, with the assumption that OPT is larger than  $O(\frac{m^2}{\varepsilon^3})$  times each  $w_{ij}$ , where  $m$  is the number of advertisers; Theorem 2 can be viewed as generalizing this result to a much larger class of problems.

**Experimental Validation:** For the applications described above, stochastic models are reasonable as the algorithm often has an idea of what agents to expect. For example, in the Display Ad Allocation problem, agents correspond to users visiting the website of a publisher who has sold contracts to advertisers. As the publisher most likely sees similar user traffic patterns from day to day, he has an idea of the available ad inventory based on historical data. In Section 3, we perform preliminary experiments on real instances of the DA problem, using actual display ad data for a set of anonymous publishers. As with any real application, there are additional features of the problem, and in the one we considered, both *fairness* and efficiency were important metrics. Hence, we also evaluated our algorithms for fairness (see Section 3 for a brief discussion); we compared our training-based algorithm with algorithms from [12] designed for the adversarial setting, as well as hybrid algorithms combining the two approaches. Our experimental results validate Theorem 2 for this application, as they show that on this real data set, even sampling 1% of the input (i.e. choosing  $\varepsilon = 0.01$ ) suffices to obtain efficiency of  $\approx 89\%$ , significantly better than the pure online algorithms from [12], which are in turn much better than a simple greedy approach.

**Other Related Work:** Our proof technique is similar to that of [10] for the AW problem; it is based on their observation that dual variables satisfy the complementary slackness conditions of the first  $\varepsilon$  fraction of impressions and *approximately* satisfy these conditions on the entire set. A key difference is that

in the AW problem, the coefficients for variable  $x_{ij}$  in the linear program are the *same* in both the constraint and the objective function. That is, the contribution an impression makes to an advertiser's value is identical to the amount of budget it consumes; in contrast, these coefficients are unrelated in the general class of packing problems that we study. Further, the structure of the LP considered by [10] is highly restricted, with each variable appearing in precisely two constraints. Thus, our paper provides a simple proof of a more general result than [10], and we experimentally validate these results for the DA problem.

The random-order model has been considered for several problems, often called *secretary* problems. Without additional assumptions (as in Theorem 2), constant-competitive (but no better) algorithms can be obtained for problems such as online Knapsack, or finding maximum weight independent sets in classes of matroids. (See [5] for a survey.) Specifically for the DA problem, the results of [21] imply that the random-order model permits a 1/8-competitive algorithm even without using the free disposal property or the conditions of Theorem 2.

There have been recent results regarding ad allocation strategies in display advertising in hybrid settings with both contract-based advertisers and spot market advertisers [15, 14]. Our results in this paper may be interpreted as a class of *representative bidding strategies* that can be used on behalf of contract-based advertisers competing with the spot market bidders [15].

Results similar to ours were recently posted in a working paper [1]. The dependence of  $\varepsilon$  on the input parameters is almost identical; however, the authors of [1] also observe that if one is willing to “reprice” the resources after every  $\varepsilon$  fraction of the input, the cubic dependence of  $\varepsilon$  can be reduced to quadratic. Also recently, Vee *et al.* [27] showed that for assignment-type problems (including the DA and AW problems, but not other problems considered in this paper), if one knows the distribution from which inputs are drawn, a similar dual-based algorithm can be used to obtain online allocations that are nearly optimal for certain “well-conditioned” convex objectives. The results from both these papers were obtained independently of ours, and posted publicly subsequent to the submission of an earlier version of this paper, which included our main result.

## 2 A Dual-Based Algorithm

In this section, we present the dual-based algorithm for the online stochastic packing problem and prove that, under our (practically-motivated) assumptions, it achieves an approximation factor of  $1 - \varepsilon$ . Recall that there is a set  $I$  of “agents”; agent  $i \in I$  has a set of mutually exclusive options  $O_i$ , and we use an indicator variable  $x_{io}$  to denote whether agent  $i$  selects alternative  $o \in O_i$ . Each option for an agent may have a different “size” in each constraint; we use  $a_{ioj}$  to denote the size in constraint  $j$  of option  $o$  for agent  $i$ .

Recall that  $w_{io}$  is the value from selecting option  $o$  for agent  $i$ , and  $c_j$  is the “capacity” of constraint  $j$ . That is, our goal is to maximize  $w^T x$  while picking at most one option for each agent, and subject to  $Ax \leq c$ . Subsequently, we normalize  $A, c$  such that  $c$  is the all-1's vector, and write the (normalized) primal

linear program below. We also use the dual linear program, which introduces a variable  $\beta_j$  for each constraint  $j$ .

Primal-LP	Dual-LP
$\max \sum_i \sum_{o \in O_i} w_{io} x_{io}$	$\min \sum_j \beta_j + \sum_i z_i$
$\sum_{o \in O_i} x_{io} \leq 1 \quad (\forall i)$	$z_i + \sum_j \beta_j a_{ioj} \geq w_{io} \quad (\forall i, o)$
$\sum_{i,o} a_{ioj} x_{io} \leq 1 \quad (\forall j)$	$\beta_j, z_i \geq 0 \quad (\forall i, j)$
$x_{io} \geq 0 \quad (\forall i, o)$	

Let  $n$  be the total number of agents,  $q = \max_i |O_i|$  be the maximum number of options for any agent, and  $m$  be the number of constraints. We say that the *gain* from option  $o \in O_i$  is  $w_{io} - \sum_j \beta_j^* a_{ioj}$ . The Dual-Based Algorithm proceeds as follows:

1. Let  $S$  denote the first  $\varepsilon n$  agents in the sequence. For the purposes of analysis, these agents are not selected. (Our implementations may assign these agents according to some online algorithm.)
2. Solve the **Dual-LP** on the agents in  $S$ , with the objective function containing the term  $\varepsilon \beta_j$  instead of  $\beta_j$  for each  $j \in [m]$ . (This is equivalent to reducing the capacity of a constraint from 1 to  $\varepsilon$ ; we refer to this as a *reduced instance*.) Let  $\beta_j^*$  denote the value of the dual variable for constraint  $j$  in this optimal solution.
3. For each subsequent agent  $i$ , if there is an option  $o$  with non-negative gain, select the option<sup>3</sup>  $o$  of maximum gain, and set  $z_i = \text{gain}(o)$ .

We will refer to a variant of this algorithm in Section 3 as the **DualBase** algorithm. The intuition behind this algorithm is simple; the dual variables  $\beta_j^*$  can be thought of as specifying a value/size ratio necessary for an option to be selected. An optimal choice for each  $\beta_j$  gives an optimal solution to the packing problem; this fact is proven implicitly in the next section, where we further show that with high probability, the optimal choice  $\beta_j^*$  on the sample  $S$  leads to a near-optimal solution on the entire instance.<sup>4</sup> In the following, let  $w_{\max} = \max_{i,o} \{w_{io}\}$ , and let  $a_{\max} = \max_{i,o,j} \{a_{ioj}\}$ .

<sup>3</sup> Assume for simplicity that there are no ties, and so there is a unique such option. This can be effectively achieved by adding a random perturbation to the weights; we omit details from this extended abstract.

<sup>4</sup> Technically, the solution returned may violate some constraints by a small factor, but it is easy to modify the algorithm to avoid this; see the discussion after Lemma 2.

## 2.1 Proof Sketch

We now sketch a proof of Theorem 2 showing that the above training-based algorithm is a  $(1 - O(\varepsilon))$ -approximation. (Proofs of some claims are omitted.) Let  $I^* \subseteq I$  denote the set of agents  $i$  with some option  $o$  having non-negative gain, and let  $\mathcal{O}^*$  denote the set of pairs  $\{(i, o) \mid i \in I^*, o = \arg \max_{o \in O(i)} \text{gain}(o)\}$ . We abuse notation by writing  $i \in \mathcal{O}^*$  if there exists  $o \in O(i)$  such that  $(i, o) \in \mathcal{O}^*$ . We use  $\mathcal{O}^*(S)$  to denote  $\mathcal{O}^* \cap S$ ; note that  $\mathcal{O}^* - \mathcal{O}^*(S)$  represents the options selected by the algorithm (for the purposes of analysis, we do not select any options for agents in  $S$ ).

Given any vector  $\beta^*$ , we obtain a feasible solution to **Dual-LP** by selecting for each item in  $I^*$ , the option  $o$  such that  $(i, o) \in \mathcal{O}^*$  and setting  $z_i = \text{gain}(o)$ ; for each item  $i \notin I^*$ , we set  $z_i = 0$ .

**Definition 1.** Let  $W = \sum_{(i,o) \in \mathcal{O}^*} w_{io}$  be the total weight of selected options, and let  $W(S) = \sum_{(i,o) \in \mathcal{O}^*(S)} w_{io}$ . Let  $C(j) = \sum_{(i,o) \in \mathcal{O}^*} a_{ioj}$  and  $C(j, S) = \sum_{(i,o) \in \mathcal{O}^*(S)} a_{ioj}$ .

For any fixed vector  $\beta^*$ ,  $\mathcal{O}^*$  and hence  $W$  and each  $C(j)$  are independent of the choice of the sample  $S$ ; the expected value of  $W(S)$  is  $\varepsilon W$ , and that of  $C(j, S)$  is  $\varepsilon C(j)$ .<sup>5</sup> For fixed  $\beta^*$ , we can conclude that the the values of  $W(S)$  and  $C(j, S)$  will be close to their expectations with high probability. However, the  $\beta^*$  computed by the algorithm is itself a function of  $S$ ; the main idea of the proof (similar to that of [10] on the special case of the AW problem) is that one can still show that if  $\beta^*$  satisfies the complementary slackness conditions on the first  $\varepsilon n$  agents (being an optimal solution), w.h.p. it *approximately* satisfies these conditions on the entire set. Two main differences from the proof of [10] are that the structure of our LP is unrestricted, and that the weight and demand coefficients of a given option may be completely unrelated; dealing with these requires additional care, and a weaker dependence of  $\varepsilon$  on the input parameters.

**Definition 2.** For a sample  $S$  and  $j \in [m]$ , let  $r_j(S) = |C(j, S) - \varepsilon C(j)|$ , and let  $t(S) = |W(S) - \varepsilon W|$ . When the context is clear, we will abbreviate  $r_j(S)$  by  $r_j$  and  $t(S)$  by  $t$ .

1. The sample  $S$  is  $r_j$ -bad if:

$$r_j \geq (m+1)(\ln n + \ln q)a_{\max} + \sqrt{C(j)} \cdot \left(2\sqrt{\varepsilon(m+1)(\ln n + \ln q)a_{\max}}\right).$$

2. The sample  $S$  is  $t$ -bad if:

$$t \geq (m+1)(\ln n + \ln q)w_{\max} + \sqrt{W} \cdot \left(2\sqrt{\varepsilon(m+1)(\ln n + \ln q)w_{\max}}\right).$$

**Lemma 1.**  $\Pr[S \text{ is } r_j\text{-bad}] \leq \frac{1}{m \cdot (nq)^{m+1}}$  for each  $j$ , and  $\Pr[S \text{ is } t\text{-bad}] \leq \frac{1}{(nq)^{m+1}}$ .

We argue below that if  $S$  is *not*  $t$ -bad or  $r_j$ -bad for any  $j$ , we obtain a good solution. We use the following simple proposition:

<sup>5</sup> Though  $\beta^*$  depends on  $S$ , many distinct samples  $S$  may lead to the same vector  $\beta^*$ .

Also, we take expectations over *all* choices of  $S$ , not just those leading to the given  $\beta^*$ . We will later use the union bound over all possible vectors  $\beta^*$ .

**Proposition 1.** Let  $j \in [m]$  be a constraint such that  $C(j, S) = \varepsilon$ . If  $S$  is not  $r_j$ -bad, we have  $1 - 2\varepsilon \leq C(j) \leq 1 + 3(\varepsilon + \varepsilon^2)$ .

**Lemma 2.** If the sample  $S$  is not  $t$ -bad or  $r_j$ -bad for any constraint  $j$ , the value of the options selected by the algorithm is  $(1 - O(\varepsilon))\text{OPT}$ .

Note that the options selected by the algorithm, as described above, may not be feasible even if  $S$  is not  $r_j$ -bad; Proposition 1 only implies that  $C(j) \leq 1 + 3(\varepsilon + \varepsilon^2)$ . Thus, we might violate some constraints by a small amount. This is easily fixed: simply decrease the capacities of all constraints by a factor of  $1 + O(\varepsilon)$ . This reduces the value of the optimal solution by no more than the same factor; though our algorithm might violate the reduced capacity of constraint  $j$  by a factor of  $1 + O(\varepsilon)$ , we respect the original capacity when  $S$  is not  $r_j$ -bad. Thus, when  $S$  is not  $t$ -bad or  $r_j$ -bad for any  $j$ , we obtain a feasible solution with value  $(1 - O(\varepsilon))\text{OPT}$ .

Finally, Lemma 1 implies that for any fixed  $\beta^*$ , the probability that a random sample  $S$  of agents is bad is less than  $\frac{2}{(nq)^{m+1}}$ . The following lemma shows that there are at most  $(nq)^m$  distinct choices for  $\beta^*$ ; as a result, the sample is good for any  $\beta^*$  with high probability. Therefore, with high probability, our algorithm returns a feasible solution with value at least  $(1 - O(\varepsilon))\text{OPT}$ , proving Theorem 2.

**Lemma 3.** There are fewer than  $(nq)^m$  distinct solutions  $\beta^*$  that are returned by the algorithm after step 2.

### 3 Experimental Evaluation: Efficiency and Fairness

We now present experimental results in the Display Ad setting, evaluating our algorithm and comparing its performance, in terms of efficiency and *fairness*, to other online algorithms and also to “ideal” solutions computed offline.

Our data set consists of a uniform sample of a set of impressions and a set of advertisers for six different publishers (A-F) from one week in September 2009. The number of arriving impressions varies from 200k to 1,500k impressions. The number of advertisers per publisher varies from 100 to 2,600 advertisers. Each impression is tagged with its set of eligible advertisers and an *edge weight* for each eligible advertiser capturing the “targeting quality” for this advertiser. We compare the algorithms both in terms of the social efficiency and the *fairness* of their solutions.

**Fairness in Ad Allocation.** Besides efficiency, fairness plays an important role in measuring the performance of an ad allocation. An allocation that achieves large total value while delivering very few impressions to some advertisers is undesirable; advertisers whose contracts are unfulfilled must typically be paid a penalty, and the publisher may have trouble retaining such advertisers.

Different notions of fairness in resource allocation have been explored extensively in the literature [20, 22, 16, 23]; perhaps the most common of these is *max-min* fairness. Such a metric is not appropriate in the DA setting, where the contract of one advertiser may specify many fewer impressions than that of others, and thus the total value he obtains *should* be lower.

For a detailed discussion of fairness metrics, see the full version of our paper at <http://arxiv.org/abs/1001.5076> [11]. Briefly, given an allocation  $x$ , we define its *value vector*  $v(x)$  such that the  $j$ th component is the total value obtained by advertiser  $j$ . We roughly define the fairness metric as the  $l_1$  distance between the value vector of  $x$  and that of an (appropriately normalized) ideal allocation  $x^*$ . In the full version [11], we consider various candidate ideal allocations; the one we focus on (fractionally) allocates an impression by dividing it equally among all advertisers who are *interested* in it; advertisers are interested in as many of the impressions they value highly as needed to satisfy their contracts.

**The Algorithms.** We examine (a) 3 pure online algorithms, (b) 2 training-based online algorithms, and (c) 2 offline algorithms.

(a) The pure online algorithms are GREEDY, PD\_AVG, and PD\_EXP, that are developed and analyzed in a previous paper [12], and achieve worst-case competitive ratios of  $\frac{1}{2}$ ,  $\frac{1}{2}$ , and  $1 - \frac{1}{e}$ . These algorithms are primal-dual algorithms that proceed as follows: we compute a discounting factor  $\beta_a$  based on the set of impressions already assigned to advertiser  $a$ , and then upon arrival of a new impression  $i$ , we assign this impression  $i$  to an advertiser  $a$  maximizing  $w_{ia} - \beta_a$ . The difference between these algorithms is in computing  $\beta_a$ : Let  $w_1, w_2, \dots, w_{n(a)}$  be the weights of impressions currently assigned to advertiser  $a$ , sorted in non-increasing order. In GREEDY, PD\_AVG, and PD\_EXP, we set  $\beta_a = w_{1a}$ ,  $\beta_a = \frac{\sum_{1 \leq j \leq n(a)} w_j}{n(a)}$ , and  $\beta_a = \frac{1}{n(a) \cdot ((1+1/n(a))^{n(a)} - 1)} \sum_{j=1}^{n(a)} w_j \left(1 + \frac{1}{n(a)}\right)^{j-1}$ , respectively.

(b) The training-based online algorithms are the dual-based algorithm DualBase from Section 2 and a HYBRID algorithm in which we set  $\beta_a$  for each advertiser  $a$  to be a convex combination of the DualBase and PD\_AVG algorithms, i.e., we start using  $\beta_a$  from DualBase, and then slowly move to using  $\beta_a$  from PD\_AVG. (The hybrid algorithm is inspired by ideas from [24].) To train and test them we used for each data set a random sample of 1% of the impressions for training and the remaining 99% for testing. This proxies the random order model, where a sample from the beginning part of the sequence is equivalent to a sample of the whole data.

(c) The offline algorithms are the fair algorithm FAIR using *equal sharing* described above, and the optimum efficient offline algorithm LP\_WEIGHT.

**Experimental Results.** The normalized efficiency and normalized fairness of each of the algorithms are summarized in Table 1. Regarding efficiency it shows that (1) the training-based algorithms perform very similarly (except for one publisher) and outperform the pure online algorithms (5-12% improvement), (2) of the pure online algorithms, both PD\_AVG and PD\_EXP outperform GREEDY, even though both PD\_AVG and GREEDY are 1/2-competitive in the worst case, (3) PD\_EXP shows only a 5% overall improvement over PD\_AVG, even though the worst-case competitive analysis of PD\_EXP is much better than PD\_AVG. Since the value of fairness depends on the values assigned to advertisers and different publishers have different advertisers, we normalized the fairness values for each publisher so that the *least* fair algorithm achieves a score of 100 and the best achieves a score of 0. Normalizing allows us to compute the average over

**Table 1.** Normalized efficiency and fairness of different algorithms for different publishers and averaged over all publishers. All numbers are normalized between 0 and 100 such that the efficiency of OPT = LP\_WEIGHT is 100 and 0 is the most fair solution.

Publishers	Normalized Efficiency						Normalized Fairness							
	A	B	C	D	E	F	Avg	A	B	C	D	E	F	Avg
LP_WEIGHT	100	100	100	100	100	100	100	34.6	47.7	98.8	100	70.3	90.1	73.6
FAIR	88.2	98.4	73.6	42.3	74.6	53.3	71.7	0	0	0	0	0	0	0
DualBase	85	93	85.7	74	91.8	93.5	87.2	69.5	62.5	96.7	43.1	87.9	88.6	74.7
HYBRID	85	93.8	95.2	73.8	92.7	93.5	89	69.4	63.1	100	41.9	83.7	88.6	74.5
PD_AVG	72	93.2	75.3	65.3	71.7	89.5	77.8	73	72	82.7	31.7	91.9	85.3	72.8
PD_EXP	72.6	89.7	73.9	90.8	72.6	96.3	82.6	69.7	59.5	86.1	71	88.8	100	79.2
GREEDY	64	90.5	69.7	53.6	55	86.2	69.8	100	100	98.6	45	100	100	90.6

different publishers. The results in the table indicate that GREEDY is the least fair algorithm and the remaining algorithms perform roughly the same, though their performance differs over different publishers. For a more detailed analysis of the results see the appendix.

## 4 Concluding Remarks

This paper motivates many open problems to explore: (i) Can we achieve an algorithm that is simultaneously good both in the worst case and in stochastic settings? Such an algorithm would be of use when the actual distribution of agents is different from the one predicted/learnt from a sample; in the display ad setting, this occurs when there is a sudden spike in traffic to a website, perhaps in response to a breaking news event, or links from an extremely high-traffic source. (ii) Can we design an online allocation algorithm that provably achieves approximate efficiency and approximate fairness (for some an appropriate notion of fairness) at the same time? (iii) Can we prove that in certain settings that appear in practice, the PD\_AVG algorithm achieves an improved approximation factor (i.e., better than  $\frac{1}{2}$ )? (iv) Can we extend the online stochastic algorithm studied in this paper to other stochastic process models such as Markov-based stochastic models? Answering these questions is an interesting subject of future research.

**Acknowledgments.** This paper is a followup of our previous work with S. Muthukrishnan and Martin Pál, and some of the results and discussions in this paper are inspired by our initial discussions with them. We thank Martin and Muthu for their contributing insights toward this paper. We also thank the Google display ad team, and especially Scott Benson for helping us with data sets used in this paper.

## References

1. Agrawal, S., Wang, Z., Ye, Y.: A dynamic near-optimal algorithm for online linear programming, Working paper posted at <http://www.stanford.edu/~yyye/>
2. Alaei, S., Malekian, A.: Maximizing sequence-submodular functions (2009) (manuscript)

3. Awerbuch, B., Azar, Y., Plotkin, S.: Throughput-competitive on-line routing. In: FOCS, vol. 34, pp. 32–40 (1993)
4. Azar, Y., Birnbaum, B., Karlin, A., Mathieu, C., Nguyen, C.: Improved Approximation Algorithms for Budgeted Allocations. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssón, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 186–197. Springer, Heidelberg (2008)
5. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Online auctions and generalized secretary problems. SIGecom Exchanges 7(2) (2008)
6. Buchbinder, N., Jain, K., Naor, J.: Online Primal-Dual Algorithms for Maximizing Ad-Auctions Revenue. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, p. 253. Springer, Heidelberg (2007)
7. Buchbinder, N., Naor, J.: Improved bounds for online routing and packing via a primal-dual approach. In: FOCS, pp. 293–304 (2006)
8. Chakrabarty, D., Goel, G.: On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and GAP. In: Proc. FOCS, pp. 687–696 (2008)
9. Chawla, S., Hartline, J.D., Malec, D., Sivan, B.: Sequential posted pricing and multi-parameter mechanism design. In: Proc. of ACM STOC, pp. 311–320 (2010)
10. Devanur, N., Hayes, T.: The adwords problem: Online keyword matching with budgeted bidders under random permutations. In: ACM EC (2009)
11. Feldman, J., Henzinger, M., Korula, N., Mirrokni, V.S., Stein, C.: Online stochastic ad allocation: Efficiency and fairness, <http://arxiv.org/abs/1001.5076>
12. Feldman, J., Korula, N., Mirrokni, V., Muthukrishnan, S., Pal, M.: Online ad assignment with free disposal. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 374–385. Springer, Heidelberg (2009)
13. Feldman, J., Mehta, A., Mirrokni, V., Muthukrishnan, S.: Online stochastic matching: Beating 1 - 1/e. In: FOCS (2009)
14. Ghosh, A., McAfee, P., Papineni, K., Vassilvitskii, S.: Bidding for representative allocations for display advertising. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 208–219. Springer, Heidelberg (2009)
15. Ghosh, A., Rubinstein, B.I.P., Vassilvitskii, S., Zinkevich, M.: Adaptive bidding for display advertising. In: WWW, pp. 251–260 (2009)
16. Goel, A., Meyerson, A., Plotkin, S.A.: Combining fairness with throughput: online routing with multiple objectives. In: STOC, pp. 670–679 (2000)
17. Goel, G., Mehta, A.: Adwords auctions with decreasing valuation bids. In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 335–340. Springer, Heidelberg (2007)
18. Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to adwords. In: SODA, pp. 982–991 (2008)
19. Karp, R., Vazirani, U., Vazirani, V.: An optimal algorithm for online bipartite matching. In: Proc. STOC (1990)
20. Kleinberg, J.M., Rabani, Y., Tardos, É.: Fairness in routing and load balancing. J. Comput. Syst. Sci. 63(1), 2–20 (2001)
21. Korula, N., Pal, M.: Algorithms for secretary problems on graphs and hypergraphs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 508–520. Springer, Heidelberg (2009)
22. Kumar, A., Kleinberg, J.M.: Fairness measures for resource allocation. SIAM J. Comput. 36(3), 657–680 (2006)
23. Lipton, R., Markakis, E., Mossel, E., Saberi, A.: On approximately fair allocations of indivisible goods. In: ACM EC (2004)

24. Mahdian, M., Nazerzadeh, H., Saberi, A.: Allocating online advertisement space with unreliable estimates. In: ACM EC, pp. 288–294 (2007)
25. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: Adwords and generalized online matching. In: FOCS (2005)
26. Srinivasan, A.: Budgeted Allocations in the Full-Information Setting. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 247–253. Springer, Heidelberg (2008)
27. Vee, E., Vassilvitskii, S., Shanmugasundaram, J.: Optimal online assignment with forecasts. In: Proc. of ACM EC, pp. 109–118 (2010)

# Caching Is Hard – Even in the Fault Model

Marek Chrobak<sup>1</sup>, Gerhard J. Woeginger<sup>2</sup>, Kazuhisa Makino<sup>3</sup>, and Haifeng Xu<sup>4</sup>

<sup>1</sup> Department of Computer Science, University of California, Riverside, USA

<sup>2</sup> Department of Mathematics and Computer Science, TU Eindhoven, Netherlands

<sup>3</sup> Department of Mathematical Informatics, Graduate School of Information and Technology, University of Tokyo, Japan

<sup>4</sup> Department of Mathematics, Zhejiang University, Hangzhou, China

**Abstract.** We prove strong NP-completeness for the four variants of caching with multi-size pages. These four variants are obtained by choosing either the fault cost or the bit cost model, and by combining it with either a forced or an optional caching policy. This resolves two questions in the area of paging and caching that were open since the 1990s.

## 1 Introduction

The *Caching Problem* deals with page replacement policies in two-level memory systems consisting of a small, fast cache and a large but slow main memory. This is a classical and well-studied problem in the area of on-line algorithms (see, for example, [4]), but in this paper we will be solely interested in its off-line version.

Formally, a caching instance specifies a sequence  $R$  of requests for memory pages. The pages in  $R$  are requested one by one, and for each page  $p$  we are given its size  $\text{SIZE}(p)$  and its faulting cost  $\text{COST}(p)$ . The cache, whose size  $C$  is also specified in the instance, can store a subset of memory pages whose total size does not exceed  $C$ . When the requested page  $p$  is in the cache, the request is served at no cost. When the requested page  $p$  is not in the cache, a page fault of cost  $\text{COST}(p)$  occurs. In response to a fault,  $p$  may be fetched into the cache. (Without loss of generality, we assume that pages are fetched only in response to faults.) In order to make room for  $p$ , other pages may have to be evicted from the cache. The objective is to decide which pages one should retain in the cache at each step so as to minimize the overall page fault cost.

There are two basic policies that determine how a page fault is resolved:

**Forced:** The faulted page  $p$  must be loaded and stored in the cache, where it occupies  $\text{SIZE}(p)$  bits.

**Optional:** The faulted page  $p$  can either be loaded for later use into the cache (where it occupies  $\text{SIZE}(p)$  bits), or it can be left outside the cache. In the latter case, the next request to  $p$  will necessarily cause a fault.

We stress that the forced policy is the standard in the literature, and all results mentioned later-on in this section assume the forced policy. The optional policy was introduced by Irani [8] in the context of web caching.

The literature contains four fundamental models of caching, defined by imposing different assumptions on page sizes and fault costs (this classification can be found in the work of Albers, Arora & Khanna [1]).

**Bit model:** For each page  $p$  we have  $\text{COST}(p) = \text{SIZE}(p)$ . The fault cost is proportional to the time it takes to bring the page into the cache. This model goes back to Irani [8].

**Cost model:** For each page  $p$  we have  $\text{SIZE}(p) = 1$ . (This model is also known as the *weighted caching* problem.) All pages have more or less the same size, but they may have varying fault costs. This model goes (at least) back to Chrobak, Karloff, Payne & Vishwanathan [5].

**Fault model:** For each page  $p$  we have  $\text{COST}(p) = 1$ . The setup cost for a fault is huge, and hence the exact page sizes have no real influence on the fault cost. This model was introduced by Irani [8].

**General model:** For each page cost and size can be arbitrary. This model was introduced by Young [9].

What positive results are known about off-line caching? The simplest variant combines the properties of bit, cost, and fault model, and only considers pages of unit size and unit cost; it can be solved to optimality by Belady's rule [3]: "Always evict the cached page whose next request is furthest in the future". Caching in the cost model can be solved in polynomial time using network flow methods; see Chrobak & al. [5]. Albers & al. [1] derived the first off-line approximation results for the bit, fault, and general model. The strongest currently known approximation result for the general model is a polynomial time 4-approximation algorithm by Bar-Noy, Bar-Yehuda, Freund, Naor & Schieber [2].

What negative results are known about off-line caching? In 1997, Fiat [6] constructed a reduction from the PARTITION problem to caching in the bit model, which implies weak NP-completeness for the bit model as well as for the general model. In 1999 Albers & al. [1] wrote in their concluding remark: "*The hardness results for caching problems are very inconclusive. The NP-hardness result for the bit model uses a reduction from PARTITION, which has pseudo-polynomial algorithms. Thus a similar algorithm may well exist for the bit model. We do not know whether computing the optimum in the fault model is NP-hard.*" In fact, this quote provides an exact summary of the current state of knowledge (just before our paper), and the open questions about the complexity of these problems have been formulated repeatedly in the caching literature since 1999. The only other relevant work we are aware of is that of Brehob *et al* [10], who proved NP-hardness of caching in non-standard cache architectures.

*Contribution of this paper.* We establish that caching in the fault model and caching in the bit model are strongly NP-complete, under the forced as well as under the optional policy. Note that for the bit model, our result excludes the possibility of a pseudo-polynomial algorithm (unless P = NP). These results finally settle the complexity status of all the caching variants discussed above.

The paper is organized as follows. In Section 2 we introduce two interval packing problems that play the central role in the paper and we show that strong

$\text{NP}$ -completeness of interval packing problems implies strong  $\text{NP}$ -completeness of caching. The rest of the paper contains our main technical contribution: Section 3 discusses the used gadgets and how they interact. Section 4 proves intractability of an intermediate auxiliary problem, and Sections 5 and 6 finally contain the hardness proofs for interval packing.

## 2 Caching Versus Interval Packing

This section introduces an auxiliary weighted interval packing problem where we wish to choose a maximum number of given weighted intervals, subject to the constraint that for any point the total weight of the covering intervals does not exceed a given threshold.

We now give a more formal definition. Suppose we are given a set of  $N$  intervals  $(s_i, t_i)$ ,  $i = 0, \dots, N - 1$ . We will identify these intervals by their indices, that is “interval  $i$ ” will refer to  $(s_i, t_i)$ . For a subset  $S \subseteq \{0, 1, \dots, n - 1\}$  of intervals, we define its weight in the natural way as  $w(S) = \sum_{i \in S} w_i$ . Also, for any real number  $\gamma$ , we define  $\text{cut}_\gamma(S) = \{i : s_i < \gamma < t_i\}$  to be the so-called *cut* of  $S$  at  $\gamma$ , that is, the intervals which contain  $\gamma$ .

The weighted interval packing problem is to choose a maximum-cardinality subset of intervals that satisfies  $w(\text{cut}_\gamma(S)) \leq W$  for all  $\gamma$ . The decision version of this problem, denoted INTVPACK-CARD, is formulated as follows.

*Problem:* INTVPACK-CARD

*Instance:* A set of  $N$  open intervals  $(s_i, t_i)$  for  $i = 0, \dots, N - 1$ , where each interval  $i$  has weight  $w_i \geq 0$ . Positive integers  $W$  and  $\ell$ .

*Question:* Is there a subset  $S$  of  $\ell$  intervals that satisfies  $w(\text{cut}_\gamma(S)) \leq W$  for all real numbers  $\gamma$ ?

In the following variation of INTVPACK-CARD, the objective changes from finding a subset of large cardinality to finding a subset of large weight.

*Problem:* INTVPACK-WEIGHT

*Instance:* A set of  $N$  open intervals  $(s_i, t_i)$  for  $i = 0, \dots, N - 1$ , where each interval  $i$  has weight  $w_i \geq 0$ . Positive integers  $W$  and  $L$ .

*Question:* Is there a subset  $S$  of the intervals with  $w(S) \geq L$  that satisfies  $w(\text{cut}_\gamma(S)) \leq W$  for all real numbers  $\gamma$ ?

We will prove in Theorems 4 and 5 that INTVPACK-CARD and INTVPACK-WEIGHT are strongly  $\text{NP}$ -complete.

Now let us draw the connection between interval packing and caching problems. Here is a generic decision version of the caching variants that will be proved to be intractable:

*Problem:* CACHING

*Instance:* A set of pages  $p_1, \dots, p_k$  with sizes  $\text{SIZE}(p_1), \dots, \text{SIZE}(p_k)$ . A request sequence  $r_1, \dots, r_m \in \{p_1, \dots, p_k\}$ . A cache size  $C$ , and a cost bound  $F$ .

*Question:* Is there a replacement policy that serves  $r_1, \dots, r_m$  with a cache of size  $C$  and incurs a total fault cost at most  $F$ ?

The four caching variants that arise from combining the generic decision problem with the fault/bit model under an optional/forced caching policy are respectively denoted as CACHING(FAULT,OPTIONAL), CACHING(FAULT,FORCED), CACHING(BIT,OPTIONAL), and CACHING(BIT,FORCED).

## 2.1 Hardness for Optional Policies

Our first reduction is from INTVPACK-CARD to CACHING(FAULT,OPTIONAL). In a preprocessing step we perturb the INTVPACK-CARD instance such that the endpoints of the  $N$  intervals become pairwise distinct and coincide with the integer points  $1, 2, \dots, 2N$ . This can be done while preserving the intersection patterns of the intervals.

Now let us construct an instance of CACHING(FAULT,OPTIONAL). For every interval  $(s_i, t_i)$  we introduce a corresponding page  $p_i$  with  $\text{SIZE}(p_i) = w_i$ . The request sequence  $R$  consists of  $2N$  requests. Every page  $p_i$  is requested exactly twice, once at position  $s_i$  and once at position  $t_i$  of the request sequence. The cache size is  $W$ , and the bound on the number of page faults is  $F = 2N - \ell$ .

(i) Suppose the INTVPACK-CARD instance has a solution set  $S$ . Then while serving the page requests, we only load pages  $p_i$  with  $i \in S$  into the cache, and we evict them right away after they have been requested for the second time. The cut condition guarantees that at every point in time the cache can accommodate all loaded pages. Since every page  $p_i$  with  $i \in S$  faults once and every page  $p_i$  with  $i \notin S$  faults twice, this yields a total of at most  $2N - \ell$  page faults. (ii) Next, suppose that the caching instance has a solution with at most  $F = 2N - \ell$  page faults. Every page  $p_i$  must fault when it is requested the first time at  $s_i$ . Let  $S$  contain all intervals  $i$  for which  $p_i$  does not fault when it is requested the second time; this implies  $|S| \geq \ell$ . Since the pages  $p_i$  with  $i \in S$  occupy space in the cache from request  $s_i$  till request  $t_i$ , the cache size  $W$  ensures that all cuts have weight bounded by  $W$ .

All in all, this yields that the INTVPACK-CARD instance has a solution if and only if the CACHING(FAULT,OPTIONAL) instance has a solution. In an almost identical fashion, we can reduce INTVPACK-WEIGHT to CACHING(BIT,OPTIONAL). The only difference is that this time we define the bound on the total fault cost as  $F = 2 \sum_{i=0}^{N-1} w_i - L$ . All remaining arguments go through as before. With Theorems 4 and 5, this yields the following.

**Theorem 1.** *The decision problems CACHING(FAULT,OPTIONAL) and CACHING(BIT,OPTIONAL) are strongly N $\bar{P}$ -complete.*  $\square$

## 2.2 Hardness for Forced Policies

Our next reduction will be from CACHING(FAULT,OPTIONAL) to CACHING(FAULT,FORCED), and it is very simple. Take an instance of

$\text{CACHING}(\text{FAULT}, \text{OPTIONAL})$ , keep all the old pages, and create two new pages  $p^*$  and  $p^{**}$  with  $\text{SIZE}(p^*) = \text{SIZE}(p^{**}) = C + 1$ . The new cache size is  $C^f = 2C + 1$ . The new request sequence  $R^f$  has length  $3m$ , and it results from the old request sequence  $R$  by replacing every request  $r_j$  by the three consecutive requests  $r_j, p^*, p^{**}$ . The new bound on the number of page faults is set to  $F^f = F + 2m$ . Then  $R^f, C^f, F^f$  specify an instance of  $\text{CACHING}(\text{FAULT}, \text{FORCED})$ .

(i) Suppose the  $\text{CACHING}(\text{FAULT}, \text{OPTIONAL})$  instance has a solution with cost  $F$ . In the cache of size  $C^f = 2C + 1$ , we reserve a segment of length  $C$  for handling the old pages. We serve request sequence  $R^f$  by mimicking the serving of sequence  $R$ : Whenever the policy for  $R$  loads an old page into the cache, we load the same old page into the reserved segment of the cache. The unreserved segment of length  $C + 1$  is used for loading the other old pages (which the policy for  $R$  does not load) and the new pages  $p^*$  and  $p^{**}$ . Then we only incur  $2m$  additional faults for the  $2m$  requests to  $p^*$  and  $p^{**}$ , and sequence  $R^f$  is served at a cost of  $F + 2m$ . (ii) Next suppose that the  $\text{CACHING}(\text{FAULT}, \text{FORCED})$  instance has a solution with cost  $F + 2m$ . Since the pages  $p^*$  and  $p^{**}$  do not fit simultaneously into the cache, this solution must fault on every request to  $p^*$  and  $p^{**}$ . The old pages are served at a total fault cost of at most  $F$ , and this induces a solution under the optional policy of cost at most  $F$ .

In the bit model, our reduction from  $\text{CACHING}(\text{BIT}, \text{OPTIONAL})$  to  $\text{CACHING}(\text{BIT}, \text{FORCED})$  is similar: Create a new instance by using new pages  $p^*$  and  $p^{**}$ , a new cache size  $2C + 1$ , and a new request sequence with  $3m$  requests. However, the new bound on the cost of page faults this time is set to  $F + 2m(C + 1)$ . Other than this, the proof remains essentially the same.

**Theorem 2.** *The decision problems  $\text{CACHING}(\text{FAULT}, \text{FORCED})$  and  $\text{CACHING}(\text{BIT}, \text{FORCED})$  are strongly  $\text{NP}$ -complete.*  $\square$

### 3 Setting Up the $\text{NP}$ -completeness Proof

The hardness proof for  $\text{INTVPACK-CARD}$  is by reduction from the well-known  $\text{NP}$ -complete  $\text{VERTEXCOVER}$  problem; see Garey & Johnson [7]. An instance of  $\text{VERTEXCOVER}$  consists of an undirected graph  $G = (V, E)$  with  $n = |V|$  vertices and  $m = |E|$  edges, and an integer  $k$ ,  $0 \leq k \leq n$ . The objective is to determine if  $G$  has a vertex cover of cardinality  $k$ .

We will present a reduction that maps an instance  $G, k$  of  $\text{VERTEXCOVER}$  into a corresponding instance of  $\text{INTVPACK-CARD}$ . Our construction can be viewed as consisting of two somewhat independent gadgets: one gadget is responsible for choosing a  $k$ -element vertex set – a candidate cover of  $G$ , while the other one verifies whether this chosen set is indeed a correct cover.

We describe the reduction in several stages. In this section we introduce the main ideas behind the cover-choosing gadget. Section 4 gives a construction for a variant of interval packing with more complicated constraints on cut weights. Finally, in Sections 5 and 6 we will show how to wrap-up this construction and derive hardness of  $\text{INTVPACK-CARD}$  and  $\text{INTVPACK-WEIGHT}$ .

*The set dominance relation.* For two sets  $X, Y \subseteq \{0, \dots, n-1\}$  such that  $|X| = |Y| = k$ , we write  $X \preceq Y$  if there is a 1-1 mapping (matching)  $f : X \rightarrow Y$  such that  $f(x) \geq x$  for all  $x \in X$ . We will also say that  $Y$  dominates  $X$ . We will write  $X \prec Y$  if  $X \preceq Y$  and  $X \neq Y$ . It is easy to show (and well-known) that  $Y$  dominates  $X$  if and only if  $|X|_{\leq x} \geq |Y|_{\leq x}$  for all  $x$ , where  $|Z|_{\leq x} = |\{z \in Z : z \leq x\}|$ . The dominance relation is clearly transitive. Further, it satisfies the following important property:

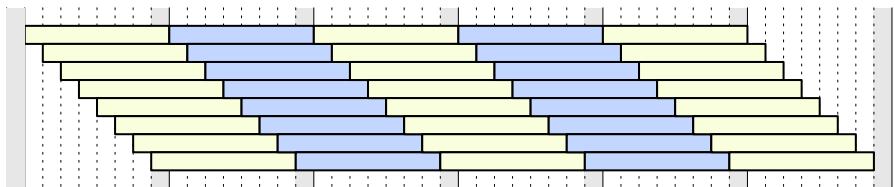
**Lemma 1.** Suppose that  $Z_0 \prec Z_1 \prec \dots \prec Z_r$ . Then  $r \leq k(n-k)$ .

*Proof.* let  $\phi_i = \sum_{z \in Z_i} z$ . We have  $\phi_0 \geq \binom{k}{2}$ ,  $\phi_r \leq \binom{n}{2} - \binom{n-k}{2}$ , so  $\phi_r - \phi_0 \leq k(n-k)$ . Since  $\phi_{i+1} > \phi_i$  for all  $i$ , the lemma follows.

*Cover chooser.* Let  $P = k(n-k) + 1$  and  $B = mP + 1$ . We now consider the instance of INTVPACK-CARD with bounds  $W = k$  and  $\ell = kB$ , and with  $N = nB$  intervals, each of length  $n$ :  $(s_{b,z}, t_{b,z}) = (bn+z, bn+z+n)$ , for  $b = 0, \dots, B-1$  and  $z = 0, \dots, n-1$ . All intervals have weight 1. (See Figure 1 for illustration.) The intervals are grouped into  $B$  bundles and all bundles, except for the last one, are grouped into  $P$  phases, as follows:

- Bundle  $b$ ,  $0 \leq b \leq B-1$ , consists of the intervals  $(s_{b,z}, t_{b,z})$ ,  $z = 0, \dots, n-1$ .
- Phase  $p$ ,  $0 \leq p \leq P-1$ , consists of the  $m$  bundles numbered  $pm, pm+1, \dots, pm+m-1$ .

The last bundle  $B-1$  does not belong to any phase.

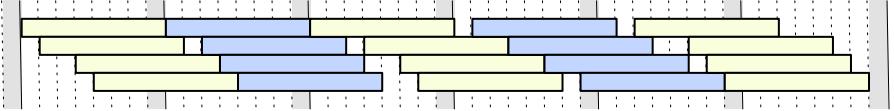


**Fig. 1.** A cover chooser for  $n = 8$ . The picture shows only a portion of the instance, with bundles colored with alternating colors. Central slots are shaded.

For an integer  $\sigma$ ,  $0 \leq \sigma \leq nB + n$ , the unit interval  $(\sigma, \sigma + 1)$  is called a *slot*. For each bundle  $b$ , the slot  $(\lambda_b, \lambda'_b) = (s_{b,n-1}, t_{b,0}) = (bn + n - 1, bn + n)$  is called the *central slot* of this bundle.

Consider some solution  $S$  of INTVPACK-CARD. Since all intervals in bundle  $b$  overlap its central slot,  $S$  contains at most  $k$  intervals from each bundle. On the other hand,  $S$  contains at least  $\ell = kB$  intervals, so it must contain exactly  $k$  intervals from each bundle. Denote by  $S_b$  the set of  $k$  intervals from bundle  $b$  that are in  $S$ . We will identify the intervals in  $S_b$  by their index with respect to the bundle, that is  $S_b$  contains those  $z$  for which  $(s_{b,z}, t_{b,z})$  is in  $S$ .

**Lemma 2.** There is a phase  $p$ ,  $0 \leq p \leq P-1$  for which  $S_{mp} = S_{mp+1} = \dots = S_{mp+m-1}$ .



**Fig. 2.** An illustration of the bundle dominance in the proof of Lemma 2. The instance is for  $n = 8$ . Here we have five consecutive sets  $S_b = \{0, 1, 3, 4\}$ ,  $S_{b+1} = \{0, 2, 3, 4\}$ ,  $S_{b+2} = \{0, 3, 5, 6\}$ ,  $S_{b+3} = \{1, 3, 5, 7\}$ ,  $S_{b+4} = \{2, 5, 6, 7\}$ .

*Proof.* We start with the following claim:  $(*) S_b \preceq S_{b+1}$  for  $b = 0, \dots, B - 2$ . We prove this claim by contradiction. Suppose that  $S_b \not\preceq S_{b+1}$  for some  $b$ . Then choose any  $x$  for which  $|S_b|_{\leq x} < |S_{b+1}|_{\leq x}$ . Let  $X = \{z \in S_b : z > x\}$  and  $Y = \{z \in S_{b+1} : z \leq x\}$ . Then  $|X \cup Y| > k$  and each interval in  $X \cup Y$  contains the slot  $I = (bn + x + n, bn + x + n + 1)$ . (More precisely, if  $z \in X$  then  $I$  is contained in  $(s_{b,z}, t_{b,z})$ , and if  $z \in Y$  then  $I$  is contained in  $(s_{b+1,z}, t_{b+1,z})$ .) This contradicts the feasibility of  $S$ .

Call a phase *good* if it satisfies the lemma and *bad* otherwise. Each bad phase must contain a bundle  $b$  for which  $S_b \prec S_{b+1}$ . By Lemma 1, the number of bad phases is at most  $P - 1$ , so, by the pigeon-hole principle, there must exist a good phase, and the lemma follows.

The intuition is this: The sets  $S_b$  will correspond to a vertex cover and transitions between consecutive bundles will be used to verify the correctness of this cover. Each phase has  $m$  such transitions, each one corresponding to one edge and verifying if this edge is covered. In order for this to work, all edges must be verified against the same set  $S_b$ . Lemma 2 above guarantees that there will be some phase in which all the sets  $S_b$  will indeed be the same.

## 4 An Extension of Interval Packing

We now extend INTVPACK-CARD as follows: in addition to  $W$ ,  $\ell$  and the set of intervals  $(s_i, t_i)$ ,  $i = 0, \dots, N - 1$ , we are also given a set  $\Gamma$  of pairs  $(\alpha, \beta)$  of numbers. We want to decide whether there is a subset  $S$  of at least  $\ell$  intervals that satisfies the following two conditions:

- (i)  $w(cut_\gamma(S)) \leq W$  for all  $\gamma$  (as before), and
- (ii)  $\min \{w(cut_\alpha(S)), w(cut_\beta(S))\} \leq W - 1$  for each  $(\alpha, \beta) \in \Gamma$ .

Intuitively, each pair  $(\alpha, \beta) \in \Gamma$  represents a “bottleneck pair”, where the weight bound is lower by 1, but only one of these tighter bounds needs to be met, not necessarily both. We will refer to this version as EXTINTVPACK.

Our goal in this section is to establish NP-completeness of EXTINTVPACK. We transform the given instance  $G = (V, E)$ ,  $k$  of VERTEXCOVER into an instance of EXTINTVPACK. As in the previous section, let  $P = k(n-k)+1$  and  $B = mP+1$ . The instance of INTVPACK-CARD will have bounds  $W = k$ ,  $\ell = kB$ , and will contain  $N = nB$  unit-weight intervals

$$(s_{b,z}, t_{b,z}) = (bn + z, bn + z + n - \delta_{b,z}),$$

for  $b = 0, \dots, B-1$  and  $z = 0, \dots, n-1$ , where each  $\delta_{b,z} \in \{0, \frac{1}{2}\}$  is determined as follows. For the last bundle  $B-1$ , we let all  $\delta_{B-1,z} = 0$ . Let  $b \leq B-2$ . Each such  $b$  is associated with one edge, with all bundles in each phase associated with different edges. Assume that the edges of  $G$  are numbered, say  $E = \{e_0, \dots, e_{m-1}\}$ . If  $b = mp + a$ , for some phase  $p$ , then we say that  $b$  is *associated* with edge  $e_a$ . If  $e_a = (u, v)$ , then we set  $t_{b,z} = \frac{1}{2}$  for  $z \in \{u, v\}$  and  $t_{b,z} = 0$  for  $z \notin \{u, v\}$ .

Next we need to define  $\Gamma$ . We let  $\Gamma = \{(\alpha_b, \beta_b)\}_{b=0, \dots, B-2}$ , where each pair  $(\alpha_b, \beta_b)$  is defined as follows: If  $e_a = (u, v)$  is the edge associated with  $b$ , then  $\alpha_b = t_{b,u} + \frac{1}{4} = bn + u + n - \frac{1}{4}$  and  $\beta_b = t_{b,v} + \frac{1}{4} = bn + v + n - \frac{1}{4}$ .

**Theorem 3.** *Problem EXTINTVPACK is strongly NP-complete.*

*Proof.* Let  $\mathcal{I}$  be the instance of EXTINTVPACK constructed above. It is sufficient to prove the following claim: (\*)  $G$  has a vertex cover of size  $k$  if and only if  $\mathcal{I}$  has a solution.

( $\Rightarrow$ ) Suppose that  $U$  is a vertex cover of  $G$  of cardinality  $k$ . We will specify the solution  $S$  by the sets  $S_b$  of intervals selected from each bundle  $b$ . We simply let  $S_b = U$  for each  $b$ . Since we choose the same  $k$  intervals from each bundle, we have  $w(cut_\gamma(S)) \leq k$  for all reals  $\gamma$ ; thus condition (i) is satisfied. To verify (ii), consider any  $(\alpha_b, \beta_b) \in \Gamma$ , for a bundle  $b$  associated with an edge  $e_a = (u, v)$ . Since  $U$  is a vertex cover, we either have  $u \in U$  or  $v \in U$ . Without loss of generality, assume  $u \in U$  (the other case is symmetric). Then, by the construction of the intervals in  $\mathcal{I}$ , we have  $t_{b,u} < \alpha_b < s_{b+1,u}$ , which means that the intervals in  $S$  corresponding to  $u$  do not intersect  $\alpha_b$ ; thus  $w(cut_{\alpha_b}(S)) \leq k-1$ , proving that condition (ii) holds.

( $\Leftarrow$ ) Now, suppose that  $\mathcal{I}$  has a solution  $S$ . As before, letting each  $S_b$  be the set of intervals from bundle  $b$  that are in  $S$ , we must have  $|S_b| = k$  for all  $b$ . By Lemma 2, there is a phase  $p$  for which  $S_{pm} = S_{pm+1} = \dots = S_{pm+m-1}$ . (Note that, even though we adjusted some end-points of the intervals, Lemma 2 still holds, since we decreased these endpoints only by  $\frac{1}{2}$ .) We take  $U = S_{pm}$ , and we claim that  $U$  is a vertex cover. Indeed, let  $e_a = (u, v) \in E$  be any edge and take the bundle  $b = pm + a$  in phase  $p$  associated with edge  $e_a$ . By condition (ii) in the definition of EXTINTVPACK, we have  $w(cut_{\alpha_b}(S)) \leq k-1$  or  $w(cut_{\beta_b}(S)) \leq k-1$ . Without loss of generality, assume that  $w(cut_{\alpha_b}(S)) \leq k-1$  (the other case is symmetric). All intervals  $\{z \in S_b : z > u\}$  from bundle  $b$  and all intervals  $\{z \in S_{b+1} : z < u\}$  from bundle  $b+1$  intersect  $\alpha_b$ . Since  $S_b = S_{b+1} = U$ , this means that  $u \in U$ , for otherwise this would give us  $k$  intervals intersecting  $\alpha_b$ , violating the bottleneck bound at  $\alpha_b$ . This holds for all edges  $e_a$ ; therefore we can conclude that  $U$  is indeed a vertex cover of  $G$  of size  $k$ .

## 5 Strong NP-Completeness of INTVPACK-CARD

We now show how to “implement” the construction from the previous section using INTVPACK-CARD. Again, let  $G = (V, E)$ ,  $k$  be an instance of VERTEXCOVER. To streamline the argument, we assume that  $k \leq n - 2$  and that vertices 0 and  $n - 1$  of  $G$  are isolated, so that, without loss of generality, they will not belong

to any vertex cover. We transform  $G, k$  into an instance  $\mathcal{J}$  of INTVPACK-CARD.  $\mathcal{J}$  will contain the same intervals as  $\mathcal{I}$  from the previous section, plus some additional ones. However, we change the bounds  $W$  and  $\ell$ , and we add many more small intervals that will be used to enforce the bound of  $k$  on the number of intervals chosen from each bundle and to simulate the bottleneck pairs.

We choose first some large even constant  $D$ , say  $D = 2n^5$ . The bounds in the instance of INTVPACK-CARD will be  $W = 2k + 1$  and  $\ell = kB + BD + n(B - 1)D$ . We include in  $\mathcal{J}$  the same  $nB$  unit-weight intervals as in  $\mathcal{I}$ : namely all  $(s_{b,z}, t_{b,z}) = (bn + z, bn + z + n - \delta_{b,z})$ , for  $b = 0, \dots, B - 1$  and  $z = 0, \dots, n - 1$ , where each  $\delta_{b,z} \in \{0, \frac{1}{2}\}$  is determined as before, that is,  $\delta_{b,z} = \frac{1}{2}$  for intervals  $z$  that correspond to the endpoints of the edge associated with  $b$  (see the previous section). We will refer to these intervals as *bundle intervals*.

Now we add to  $\mathcal{J}$  new intervals called *obstacles*. Most of them will have length  $1/D$  and weight  $k + 1$ , but a few of them (two per bundle) will have length  $1/2D$  and weight  $k + 2$ .

The first category of obstacle intervals is called *plain obstacles*. For each bundle  $b$ , we introduce  $D$  disjoint obstacles that fill its central slot  $(\lambda_b, \lambda'_b)$ , namely intervals  $(\lambda_b + g/D, \lambda_b + (g + 1)/D)$ , for  $g = 0, \dots, D - 1$ . All these intervals have weight  $k + 1$ . We have  $BD$  plain obstacles in the central slots.

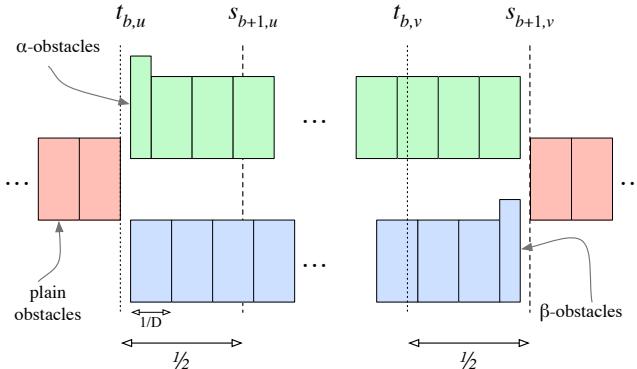
More plain obstacles are introduced between central slots of any two consecutive bundles. For each bundle  $b \leq B - 2$  we proceed as follows. Let  $e_a = (u, v)$  be the edge associated with  $b$ , where  $u < v$ . We fill intervals  $(\lambda'_b, t_{b,u})$  and  $(s_{b+1,v}, \lambda_{b+1})$  with plain obstacles. These obstacles are  $(\lambda'_b + g/D, \lambda'_b + (g + 1)/D)$ , for  $g = 0, \dots, D(u - \frac{1}{2}) - 1$ , and  $(s_{b+1,v} + g/D, s_{b+1,v} + (g + 1)/D)$ , for  $g = 0, \dots, D(n - v) - 1$ .

Now, we introduce two groups of obstacles in the interval  $(t_{b,u}, s_{b+1,v})$ . The intervals in the first group are called  $\alpha$ -obstacles and those in the other group  $\beta$ -obstacles. We have  $D(v - u + \frac{1}{2})$  obstacles of each of these two types. The first  $\alpha$ -obstacle is called the  $\alpha$ -bottleneck, and it is the interval  $(t_{b,u} + 1/4D, t_{b,u} + 3/4D)$ , with weight  $k + 2$ . The remaining  $\alpha$ -obstacles are  $(t_{b,u} + 3/4D + g/D, t_{b,u} + 3/4D + (g + 1)/D)$ , for  $g = 0, \dots, D(v - u + \frac{1}{2}) - 2$  and they all have weight  $k + 1$ . Analogously, the  $\beta$ -obstacles (other than the last one) are  $(t_{b,u} + 1/4D + g/D, t_{b,u} + 1/4D + (g + 1)/D)$ , for  $g = 0, \dots, D(v - u + \frac{1}{2}) - 2$ , all with weight  $k + 1$ . The last  $\beta$ -obstacle, called the  $\beta$ -bottleneck, is  $(s_{b+1,v} - 3/4D, s_{b+1,v} - 1/4D)$ , and it has weight  $k + 2$ . (See Figure 3 for illustration.)

Note that between  $\lambda'_b$  and  $\lambda_{b+1}$  we have  $D(u - \frac{1}{2}) + D(n - v)$  plain obstacles,  $D(v - u + \frac{1}{2})$   $\alpha$ -obstacles, and  $D(v - u + \frac{1}{2})$   $\beta$ -obstacles.

**Lemma 3.** (a) *Any solution of instance  $\mathcal{J}$  has at most  $BD + n(B - 1)D$  obstacle intervals.* (b) *If some solution of instance  $\mathcal{J}$  has exactly  $BD + n(B - 1)D$  obstacle intervals then, for any bundle  $b \leq B - 1$ , it must contain either the  $\alpha$ -bottleneck or the  $\beta$ -bottleneck between  $\lambda'_b$  and  $\lambda_{b+1}$ .*

*Proof.* (a) Consider a bundle  $b \leq B - 1$ , and let  $(u, v)$  be the edge associated with  $b$ , where  $u < v$ . We claim that any solution contains at most  $D(v - u + \frac{1}{2})$  obstacles between  $t_{b,u}$  and  $s_{b+1,v}$ . This can be justified as follows: Order all the obstacles in this range in order of increasing left endpoints, starting with the



**Fig. 3.** Three types of obstacle intervals. The heights of the rectangles represent the weights,  $k + 1$  or  $k + 2$ .

$\alpha$ -bottleneck (and ending with the  $\beta$ -bottleneck). This will give us a sequence of  $2D(v - u + \frac{1}{2})$  intervals where each one (except the last one) intersects the next one. No two intersecting obstacles can be in the solution, because of the weight constraint. Therefore at most half of the obstacles in this sequence can be in the solution – proving our claim.

There are  $D(u - \frac{1}{2}) + D(n - v)$  plain obstacles between  $\lambda'_b$  and  $\lambda_{b+1}$ . By the previous paragraph, any solution can contain at most  $D(v - u + \frac{1}{2})$   $\alpha$ - or  $\beta$ -obstacles in this range, for the total of  $nD$  obstacles. Multiplying it by  $B - 1$  bundles and adding  $BD$  plain obstacles in central slots, we obtain (a).

(b) Consider a bundle  $b \leq B - 1$  whose associated edge is  $(u, v)$ , for  $u < v$ . It is sufficient to show that if a solution contains  $D(v - u + \frac{1}{2})$  obstacles between  $t_{b,u}$  and  $s_{b+1,v}$ , then it must contain at least one of the two bottlenecks in this range. Suppose, towards contradiction, that it does not. Order these obstacles from left to right, as in (a). Without the two bottlenecks, the ordering will contain  $2D(v - u + \frac{1}{2}) - 2$  obstacles, and the solution can contain at most half of them, that is at most  $D(v - u + \frac{1}{2}) - 1$  intervals – a contradiction.

**Theorem 4.** Problem INTVPACK-CARD is strongly NP-complete.

*Proof.* Let  $\mathcal{J}$  be the instance of INTVPACK-CARD constructed above. It is sufficient to prove the following claim: (\*)  $G$  has a vertex cover of size  $k$  if and only if  $\mathcal{J}$  has a solution. The proof mimics the proof of Theorem 3, “simulating” the constraints from that proof using obstacle intervals.

( $\Rightarrow$ ) Suppose that  $U$  is a vertex cover of  $G$  of cardinality  $k$ . We define a solution  $S$  of  $\mathcal{J}$ . The bundle intervals in  $S$  are specified by the sets  $S_b$  of intervals selected from each bundle  $b$ . Here, we simply let  $S_b = U$  for each  $b$ . This will give us  $kB$  intervals. Since we choose the same  $k$  intervals from each bundle, these bundle intervals will have weight at most  $k$  at each cut point. Next, we add to  $S$  all  $BD$  plain obstacles in central slots. Finally, for each bundle  $b \leq B - 1$ , we proceed as follows: We add to  $S$  all plain obstacles between  $\lambda'_b$  and  $\lambda_{b+1}$ . If  $e_a = (u, v)$ , where  $u < v$ , is the edge associated with  $b$ , then either  $u \in U$  or  $v \in U$ . If  $u \in U$

then we add to  $S$  all  $\alpha$ -obstacles between  $\lambda'_b$  and  $\lambda_{b+1}$ ; otherwise we add to  $S$  all  $\beta$ -obstacles in this range. The total number of obstacles between  $\lambda'_b$  and  $\lambda_b$  will be  $nD$ , so, overall, we will have  $|S| = kB + BD + n(B - 1)D = \ell$ .

It remains to verify the bound on weight. Consider any  $\gamma$ . This  $\gamma$  is intersected by at most one obstacle and at most  $k$  bundle intervals. If this obstacle is not an  $\alpha$ - or  $\beta$ -bottleneck, then its weight is  $k + 1$ , so  $w(cut_\gamma(S)) \leq k + (k + 1) = W$ . Suppose that this obstacle is the  $\alpha$ -obstacle (the case of the  $\beta$ -obstacle is symmetric) between  $\lambda'_b$  and  $\lambda_{b+1}$ , where  $b$  is associated with an edge  $e_a = (u, v)$ , for  $u < v$ . By the definition of  $S$ , since we included the  $\alpha$ -bottleneck in  $S$ , we must have  $u \in U$ . Further, by the definition of the  $\alpha$ -bottleneck for  $b$ , we also have  $t_{b,u} < \gamma < s_{b+1,v}$  – in other words,  $\gamma$  is not contained in any bundle interval corresponding to  $u$ . Thus at most  $k - 1$  bundle intervals intersect  $\gamma$ , implying that  $w(cut_\gamma(S)) \leq k - 1 + (k + 2) = W$ .

( $\Leftarrow$ ) Now suppose that  $S$  is a solution for  $\mathcal{J}$ . For any slot  $(\sigma, \sigma + 1)$ , its sub-intervals  $(\sigma, \sigma + \frac{1}{2})$  and  $(\sigma + \frac{1}{2}, \sigma + 1)$  will be called *half-slots*.

First, we argue that, for each half-slot between  $\lambda_0$  and  $\lambda'_{B-1}$ ,  $S$  must contain at least one obstacle that intersects this half-slot. Indeed, we have  $2(\lambda'_{B-1} - \lambda_0) = 2[(n(B - 1) + n) - (n - 1)] = 2nB - 2n + 2$  such half-slots, and each half-slot can contribute at most  $D/2$  obstacles to  $S$ ; because obstacles have weight  $k + 1 > W/2$ . If we had a half-slot that has no obstacle in  $S$ , the total number of intervals in  $S$  (even if we include, generously, all bundle intervals) would be at most  $(2nB - 2n + 2) \cdot D/2 + nB = nBD - nD + D + nB < \ell$ .

By the previous paragraph,  $S$  has at least one obstacle interval in each half-slot. This implies that  $S$  contains at most  $k$  intervals from each bundle. Thus, applying Lemma 3, we obtain the following:

- (a)  $S$  has at most  $k$  intervals from each of the  $B$  bundles,
- (b)  $S$  has at most  $D$  plain obstacles in each of the  $B$  central slots,
- (c)  $S$  has at most  $nD$  obstacles (of type plain,  $\alpha$  or  $\beta$ ) in each of the  $B - 1$  intervals between two consecutive central slots.

Since  $\ell = kB + BD + n(B - 1)D$ ,  $S$  must contain the *exact* numbers of intervals given above in each of the categories (a), (b) and (c).

The first important consequence of the observation above is that  $S$  contains exactly  $k$  intervals from each bundle. Thus we can represent  $S$ , yet again, by the sets  $S_b$  of intervals from each bundle  $b$ , and we will have  $|S_b| = k$  for all  $k$ . Further, since, as we showed earlier, each half-slot contains an obstacle (actually, many of them), we will also have  $S_b \preceq S_{b+1}$  for  $b = 0, \dots, B - 1$ . By Lemma 2, there will be a phase  $p$  where  $S_{pm} = S_{pm+1} = \dots = S_{pm+m-1}$ .

As before, we claim that  $U = S_{pm}$  is a vertex cover. Recall that  $S$  contains  $BD + n(B - 1)D$  obstacle intervals. By Lemma 3, the only way this is possible is when  $S$  contains at least one bottleneck in each range between  $\lambda'_b$  and  $\lambda_{b+1}$ , for each bundle  $b = 0, \dots, B - 1$ . Let  $e_a = (u, v) \in E$ , where  $u < v$ , be any edge and take the bundle  $b = pm + a$  in phase  $p$  associated with edge  $e_a$ . Without loss of generality, suppose that  $S$  contains the  $\alpha$ -bottleneck between  $\lambda'_b$  and  $\lambda_{b+1}$ , that is the interval  $(t_{b,u} + 3/4D + g/D, t_{b,u} + 3/4D + (g + 1)/D)$ . Taking any  $\gamma$  from this interval, this  $\gamma$  must be intersected by at most  $k - 1$  bundle intervals, which

is possible only if  $u \in S_b = U$ . In other words,  $e_a$  is covered by  $U$ . Since this holds for any edge  $e_a$ , we can conclude that  $U$  is a vertex cover of  $G$ .

## 6 Strong NP-Completeness of INTVPACK-WEIGHT

The idea is this: Change the size of the obstacles to  $p(n)$ , for some large polynomial  $p()$ . As before, the bottlenecks are 1 unit higher,  $p(n) + 1$ . Now let's take  $W = p(n) + k$  and  $L = kB + M$ , where  $M$  is the total weight of plain obstacles plus the total weight of all alpha-obstacles. Since  $p(n)$  is large, any solution must take the maximum number of obstacles, which gives us the same constraint as before, and the rest of the argument remains the same.

**Theorem 5.** *Problem INTVPACK-WEIGHT is strongly NP-complete.* □

## Acknowledgements

This project was carried out, in part, at the workshop on ‘Adaptive, output-sensitive, on-line, and parameterized algorithms’, Schloss Dagstuhl, Germany, April 19–24, 2009. M. Chrobak has been supported by the NSF Grant CCF-0729071. G. Woeginger has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403, and by BSIK grant 03018.

## References

1. Albers, S., Arora, S., Khanna, S.: Page replacement for general caching problems. In: Proc. 10th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 1999), pp. 31–40 (1999)
2. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. Journal of the ACM 48, 1069–1090 (2000)
3. Belady, L.A.: A study of replacement algorithms for virtual-storage computer. IBM Systems Journal 5, 78–101 (1966)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Chrobak, M., Karloff, H.J., Payne, T.H., Vishwanathan, S.: New results on server problems. SIAM Journal on Discrete Mathematics 4, 172–181 (1991)
6. Fiat, A.: Unpublished manuscript (1997)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
8. Irani, S.: Page replacement with multi-size pages and applications to web caching. Algorithmica 33, 384–409 (1997)
9. Young, N.E.: On-line file caching. Algorithmica 33, 371–383 (1998)
10. Brehob, M., Wagner, S., Torng, E., Enbody, R.: Optimal replacement is NP-hard for non-standard caches. IEEE Trans. Computers 53, 73–76 (2004)

# Superselectors: Efficient Constructions and Applications

Ferdinando Cicalese and Ugo Vaccaro

Department of Computer Science and Applications “R.M. Capocelli”  
University of Salerno, via Ponte don Melillo, 84084 Fisciano, Italy

**Abstract.** We introduce a new combinatorial structure: *superselectors*. We show that superselectors subsume several important combinatorial structures used in the past few years to solve problems in group testing, compressed sensing, multi-channel conflict resolution and data security. We prove close upper and lower bounds on the size of superselectors and we provide efficient algorithms for their constructions. Albeit our bounds are very general, when they are instantiated on the combinatorial structures that are particular cases of superselectors (e.g.,  $(p, k, n)$ -selectors [15],  $(d, \ell)$ -list-disjunct matrices [25],  $MUT_k(r)$ -families [28],  $FUT(k, \alpha)$ -families [2], etc.) they match the best known bounds in terms of size of the structures (the relevant parameter in the applications). For appropriate values of parameters, our results also provide the first efficient deterministic algorithms for the construction of such structures.

## 1 Introduction

It is often the case where understanding and solving a problem means discovering the *combinatorics* at the heart of the problem. Equally time and again it happens that the crucial step towards the economical solution of problems arising in different areas hinges on the efficient construction of a *same* combinatorial object. An interesting example is that of superimposed codes [26] (also known as cover-free families [20], strongly selective families [10], disjunct matrices [16], ...). Superimposed codes represent the main tool for the efficient solution of several problems arising in compressed sensing [11], cryptography and data security [27], computational biology [3], multi-access communication [36], database theory [26], pattern matching [24][34][32], distributed colouring [29], and circuit complexity [4], among the others. Due to their importance, a lot of efforts has been devoted to the design of fast algorithms for the construction of superimposed codes of short length. In this line of research a main result is the paper by Porat and Rotschild [33] who presented a very efficient polynomial time algorithm for that purpose. More recently, Indyk *et al.* [25] showed that optimal nonadaptive group testing procedure (i.e. superimposed codes) can be efficiently constructed and decoded.

In the past few years it has also become apparent that combinatorial structures strictly related to superimposed codes lie at the heart of an even more vast series of problems. As quick examples, the selectors introduced in [9] were instrumental

to obtain fast broadcasting algorithms in radio networks, the  $(p, k, n)$ -selectors of [15] were the basic tool for the first two-stage group testing algorithm with an information theoretic optimal number of tests, the  $(d, \ell)$ -disjunct matrices of [25] were a crucial building block for the efficiently decodable non-adaptive group testing procedures mentioned above.

It is the purpose of this paper to introduce *superselectors*, a new combinatorial object that encompasses and unifies all of the combinatorial structures mentioned above (and more). We provide efficient methods for their constructions and apply their properties to the solutions of old and new problems for which constructive solutions have not been shown so far. In particular, superselectors extend at the same time superimposed codes and several different generalizations of theirs proposed in the literature.

When appropriately instantiated, our superselectors asymptotically match the best known constructions of  $(p, k, n)$ -selectors [15],  $(d, \ell)$ -list-disjunct matrices [25], monotone encodings and  $(k, \alpha)$ -FUT families [312],  $MUT_k(r)$ -families for multiaccess channel [28][1]. In some cases, e.g., for  $(p, k, n)$ -selectors and  $(d, \ell)$ -list-disjunct matrices, we also improve on the multiplicative constant in the O notation. We show that optimal size superselectors (and hence all the above structures) can be easily constructed in time polynomial in  $n$ , the main dimension of the structure, though exponential in the second parameter  $p$ . This might be satisfying in those applications, e.g., computational biology, where  $p \ll n$ . A major open question is whether it is possible to deterministically obtain optimal size superselectors (or even selectors) in time subexponential in  $p$ . However, in cases when  $p$  is constant we note that our results provide the first known polynomial construction of *optimal size*  $(p, k, n)$ -selectors (and related structures).

It should be also noticed that selectors, and similar combinatorial structures, generally have to be computed *only once*, since they can be successively used in different contexts without the need to recompute them from scratch. Therefore, it seems to make sense (and in absence of better alternatives) to have onerous algorithms that output structures of optimal size, (the crucial parameter that will affect the complexity of algorithms that uses selectors and the like structures in different scenarios) than more efficient construction algorithms that produce structures of suboptimal size. This brings us to another question. Most of the structures mentioned above, and subsumed by our superselectors, can also be obtained via expander graphs, or equivalently, randomness extractors. However, to the best of our knowledge, the best known explicit expander-based constructions give only suboptimal (w.r.t. to the size) selector-like structures. Table II summarizes how our results compare to the state of the art. The bounds are reported as they were given in the original papers, thus producing a slight level of disformity. However, if with this choice we might be requiring the reader to put a little bit of effort in the comparisons, we are not risking mistranslations of the bounds from one notation into another. The main aim of the data in the table is to show that the generalization provided by the superselectors in no case implies a loss in terms of optimality of the structure size. In addition, the number of

**Table 1.** Bounds attained via SUPER-SELECTORS against best known bounds

Structure	Lower Bounds on the Size	Our Upper Bounds on the Size construction time	Old Upper Bounds on the Size, construction time
$(p, k, n)$ -sel.	$\Omega\left(\frac{p^2}{p-k+1} \log \frac{n/p}{p-k+1} + O(1)\right)$ [7, 15]	$\frac{2p^2}{p-k+1} \log \frac{n}{p} (1 + o(1))$ time: $O(n^{p+1} \log n)$	$\frac{p^2}{p-k+1} \text{polylog}(n)$ , time: $\text{poly}(n)$ [4]
$(d, \ell)$ -list	$d \log \left(\frac{n}{e(d+\ell-1)}\right)$ if $d < 2\ell$ $\frac{d^2}{4\ell \log \frac{e d^2}{4\ell}} \log \frac{n-2\ell-d}{e d^2}$ if $d \geq 2\ell$ [15]	$O(2d \log \frac{n}{2d})$ , $d < \ell$ $O\left(\frac{(d+\ell)^2}{\ell} \log \frac{n}{d+\ell}\right)$ , $d \geq \ell$ time: $O(n^{d+\min\{\ell, d\}+1} \log n)$	$O\left(\frac{2(d+\ell) \log n + \log \left(\frac{d+\ell}{d}\right)}{d+\ell \left(\frac{d}{d+\ell}\right)^{d/\ell}}\right)$ , time: $\text{poly}\left(\binom{d+\ell}{d}, n^{d+\ell}, 2^{d+\ell}, \frac{\left(\frac{d}{d+\ell}\right)^{-\frac{d}{\ell}}}{\frac{d}{d+\ell}}\right)$ [25]
$MUT_k(p)$	$\Omega\left(\max\{k^2, p\} \log \frac{n}{p}\right)$ [28]	$O((p+k^2) \log \frac{n}{p})$ time: $O(n^{p+1} \log n)$	$O((p+k^2) \log \frac{n}{p})$ , non-constructive [11]
$(p, \alpha)$ -FUT	$\frac{p}{(1-\alpha) \log \frac{p}{1-\alpha}} \log n$ [2]	$\frac{p}{(1-\alpha)} \log \frac{n}{p}$ time: $O(n^{p+1} \log n)$	$\frac{p}{(1-\alpha)} \log \frac{n}{p}$ , non-constructive [2]
$p$ -cover free	$\left(\frac{p^2}{2 \log p} \log \frac{n}{p}\right)(1+o(1))$ [17]	$\frac{ep^2}{\log e} \log \frac{n}{p}(1+o(1))$ time: $O(n^{p+1} \log n)$	$p(p+1) \log e \log \frac{n}{p}$ , non-constructive [15, 16] $\Theta(p^2 \log n)$ , time: $\Theta(pn \log n)$ [33].
$(p, \mathbf{v}, n)$ -sel.	$\max_j \frac{j^2}{j-v_j+1} \frac{\log(n/j)}{\log \frac{n}{j-v_j+1} + O(1)}$	$\max_{j=1, \dots, p} \{\min\{\frac{j^2}{\log_2 e}, \frac{3pej}{j-v_j+1}\} \log \frac{n}{j}\}$	

applications of superselectors we shall present in Section 3 seems to suggest that they represent a basic structure, likely to be useful in many contexts.

## 2 The $(p, \mathbf{v}, n)$ -SUPER-SELECTOR

Given two vectors  $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ , we denote with  $\mathbf{x} \oplus \mathbf{y}$  the Boolean sum of  $\mathbf{x}$  and  $\mathbf{y}$ , i.e., their componentwise OR. Given an  $m \times n$  binary matrix  $M$  and an  $n$ -bit vector  $\mathbf{x}$ , we denote by  $M \odot \mathbf{x}$  the  $m$ -bit vector obtained by performing the Boolean sum of the columns of  $M$  corresponding to the positions of the 1's in  $\mathbf{x}$ . That is, if  $\mathbf{x}$  has a 1 in positions, say  $3, 7, 11, \dots$ , then  $M \odot \mathbf{x}$  is obtained by performing the  $\oplus$  of the 3rd, 7th, 11th,  $\dots$ , column of  $M$ . Given a set  $S \subseteq [n]$ , we use  $M(S)$  to denote the submatrix induced by the columns with index in  $S$ . Also we use  $\mathbf{a}_S$  to indicate the Boolean sum of the columns of  $M(S)$ . Given two  $n$ -bit vector  $\mathbf{x}, \mathbf{y}$  we say that  $\mathbf{x}$  is *covered* by  $\mathbf{y}$  if  $x_i \leq y_i$ , for each  $i = 1, \dots, n$ . Note that if  $\mathbf{x}$  is not covered by  $\mathbf{y}$  then it means that  $\mathbf{x}$  has a 1 in a position in which  $\mathbf{y}$  has a 0.

We first recall the definition of  $(p, k, n)$ -selector, as given in [15]. A  $(p, k, n)$ -selector is an  $m \times n$  binary matrix such that for any subset  $S$  of  $p \leq n$  columns, the submatrix  $M(S)$  induced by  $S$  contains at least  $k \leq p$  rows of the identity matrix  $I_p$ . The parameter  $m$  is the *size* of the selector.

**Definition 1.** Fix integers  $n, p$ , with  $p \leq n$  and an integer vector,  $\mathbf{v} = (v_1, \dots, v_p)$ , such that  $v_i \leq i$ , for each  $i = 1, \dots, p$ . We say that an  $m \times n$  binary matrix  $M$  is a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR if  $M$  is a  $(i, v_i, n)$ -selector for each  $i = 1, \dots, p$ . We call  $m$  the size of the SUPER-SELECTOR.

Our main result on SUPER-SELECTORS is summarized in the following theorem, whose proof will be given in Sections 4.

**Theorem 1.** *A  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR of size*

$$m = O\left(\max_{j=1,\dots,p} k_j \log(n/j)\right), \quad \text{where } k_j = \min\left\{\frac{3pej}{(j - v_j + 1)}, \frac{ej^2}{\log_2 e}\right\}$$

*can be constructed in time polynomial in  $n$  and exponential in  $k$ .*

The “identification” capability of a SUPER-SELECTOR are as follows.

**Lemma 1.** *Let  $M$  be a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR,  $\mathbf{v} = (v_1, \dots, v_p)$ . Let  $S$  be any set of  $x < v_p$  columns of  $M$ . Let  $\mathbf{a}_S$  denote the Boolean sum of the columns in  $S$ . Then, from  $\mathbf{a}_S$  it is possible to identify at least  $v_{x+y}$  of the columns in  $S$ , where  $y$  is the number of columns of  $M$  which are not in  $S$  but are covered by  $\mathbf{a}_S$ . Moreover,  $y < \min\{j \mid x < v_j\} - x$ .*

*Proof.* Let  $T = \{\mathbf{b} \mid \mathbf{b} \notin S \text{ and } \mathbf{b} \oplus \mathbf{a}_S = \mathbf{a}_S\}$ , i.e.,  $T$  is the set of columns not in  $S$  but covered by  $\mathbf{a}_S$ . Then,  $y = |T|$ . We first prove the last statement.

*Claim.*  $y < \min\{j \mid v_j > x\} - x$ . Let  $j^*$  be a value of  $j$  achieving the minimum. The claim is a consequence of  $M$  being a  $(j^*, v_{j^*}, n)$ -selector. To see this, assume, by contradiction, that  $|T| \geq j^* - x$ . Let  $T' \subseteq T$  and  $|T' \cup S| = j^*$ . Then, there are at least  $v_{j^*} > |S|$  columns in  $T' \cup S$  with a 1 in a row where all the other columns have a 0. Thus, there is at least one column of  $T'$  which has a 1 where all the column of  $S$  have a 0. This contradicts the fact that all the columns of  $T$  (and hence of  $T'$ ) are covered by  $\mathbf{a}_S$ .

Since  $x+y < j^* \leq p$ , and  $M$  is an  $(x+y, v_{x+y}, n)$ -selector, among the columns of  $S \cup T$  there are at least  $v_{x+y}$  which have a 1 where all the others have a 0. Let  $W$  be such set of columns. By an argument analogous to the one used in the claim we have that  $W \subseteq S$  and we can identify them.  $\square$

*Remark 1.* Notice that if  $v_i > v_{i-1}$ , for each  $i = 2, \dots, p$ , then we have a situation that, at a first look, might appear surprising: the larger is the number of spurious elements, i.e., columns not in  $S$  but covered by  $\mathbf{a}_S$ , the more information we get on  $S$ , i.e., the more are the columns of  $S$  that are identified.

*Remark 2.* The same argument used in the proof above shows that Lemma 1 also holds when  $\mathbf{a}_S$  is the component-wise arithmetic sum of the columns in  $S$ .

### 3 Applications of the SUPER-SELECTORS

**Approximate Group Testing.** In classical non-adaptive group testing [16], we want to identify a subset  $P \subseteq [n]$ , with  $|P| \leq p$ , by using the minimum possible set of tests  $T_1, \dots, T_m$ , where for each  $i = 1, \dots, m$ , we have  $T_i \subseteq [n]$ . The outcome of test  $T_i$  is a bit which is 1 iff  $T_i \cap P \neq \emptyset$ . If we require that the whole  $P$  is identified exactly, and non-adaptively, then it is known that  $\Omega(\frac{p^2}{\log p} \log \frac{n}{p})$  tests are necessary [16].

Cheraghchi [8], in the context of error-resilient group testing, Gilbert *et al.* [22], in the context of sparse signal recovery, and Alon and Hod [2] considered

the case when one is interested in identifying some approximate version of  $P$ . It turns out [8] that at least  $p \log \frac{n}{p} - p - e_0 - O(e_1 \log \frac{n-p-e_0}{e_1})$  tests are necessary if one allows the identification algorithm to report a set  $P'$ , such that  $|P' \setminus P| \leq e_0$  and  $|P \setminus P'| \leq e_1$ . In other words, the algorithm can report up to  $e_0$  false positives and up to  $e_1$  false negatives.

Let  $M$  be an appropriate  $(p + e_0, \mathbf{v}, n)$ -SUPER-SELECTOR, with the components of vector  $\mathbf{v}$  defined by  $v_i = i - \min\{e_0, e_1\} + 1$ . We can use  $M$  to attain approximate identification in the above sense. Proceeding in a standard way, map  $[n]$  to the indices of the columns of the super-selector and interpret the rows of the super-selector as the indicator vectors of the tests. Now the vector of the outcomes of the tests is the Boolean sum  $\mathbf{a}_P$  of those columns whose index is in  $P$ . Let  $P'$  be the set of the indices of the columns covered by  $\mathbf{a}_P$ . We have  $P \subseteq P'$  and by Lemma 11 also  $|P'| \leq |P| + e_0$ . Moreover, from Lemma 11 we also know that a set of positives  $P'' \subseteq P$  can be exactly identified, with  $|P''| \geq |P| - e_1$ . Therefore, any set  $P^*$  with  $P'' \subseteq P^* \subseteq P'$  satisfies the bounds on the false positives and false negatives.

Note that, for the interesting case of  $e_0, e_1 = \Theta(p)$ , the above group testing strategy is best possible since it uses  $O(p \log \frac{n}{p})$  tests which matches the lower bound of [8]. Cheraghchi [8] considers the case when some tests might be erroneous and only focuses on the case of zero false negatives. Alon and Hod [2] consider the case of zero false positives and obtain  $O(p \log(n/p))$  tests procedures, which are in fact optimal for this case. Gilbert *et al.* [22] allow both false positives and false negatives but their procedures uses  $O(p \log^2 n)$  tests. Moreover, our implementation guarantees the exact identification of at least  $p' - \min\{e_0, e_1\} + 1$  positives, where  $p' \leq p$  is the actual number of positive elements.

**Additive Group Testing.** We now consider exact group testing with *additive* tests. In this variant, the outcome of testing a subset  $T_i$  is the number of positives contained in  $T_i$ , i.e., the integer  $|T_i \cap P|$ .

It is known that  $\Omega(\frac{p}{\log p} \log \frac{n}{p})$  tests are necessary if we want to exactly identify  $P$  using additive tests (see, e.g., [23] and references therein).

Proceeding analogously to the case of Approximate Group Testing, we can reformulate the additive group testing problem as follows: given positive integers  $n$  and  $p < n$ , minimize the number  $m$  of rows of an  $m \times n$  0-1 matrix  $M$  such that any set  $P$  of up to  $p$  columns of  $M$  can be identified from their sum<sup>1</sup>  $\mathbf{a}_P$ .

Let  $M$  be an appropriate  $(2p, \mathbf{v}, n)$ -SUPER-SELECTOR, with the components of vector  $\mathbf{v}$  defined by  $v_i = i$ , for  $i = 1, \dots, \sqrt{p}$  and  $v_i = \lceil \frac{i}{2} \rceil + 1$ , for  $\sqrt{p} < i \leq 2p$ . We show that  $M$  provides a non-adaptive strategy for additive group testing with  $O(p \log(n/p))$  tests.

If  $|P| < \sqrt{p}$ , using the fact that  $v_{|P|+1} = |P| + 1$ , Lemma 11 and Remark 2 imply that from  $\mathbf{a}_S$  we can identify the whole set  $P$ .

If, otherwise,  $|P| \geq \sqrt{p}$ , by using the fact that  $v_{2|P|} > |P|$ , by Lemma 11 and Remark 2, from  $\mathbf{a}_P$  we can uniquely identify a subset  $R$  of  $P$ , such that  $|R| \geq p/2$  and confine the elements of  $P_1 = P \setminus R$  into a set  $S_1$  such that  $|S_1| \leq p$ . In

---

<sup>1</sup> Here sum is meant in the arithmetic way, i.e.,  $\mathbf{z} = \mathbf{x} + \mathbf{y}$  iff  $z_i = x_i + y_i$ , for each  $i$ .

particular  $S_1 \cup R$  is the set of all columns of  $M$  which are component-wise not larger than  $\mathbf{a}_P$ .

Now, let  $\mathbf{a}_{P_1} = \mathbf{a}_P - \sum_{i \in R} \mathbf{c}_i$ , where  $\mathbf{c}_i$  denotes the  $i$ th column of  $M$  and the additions and subtractions among vectors are meant component-wise. Clearly,  $\mathbf{a}_{P_1}$  is the sum of  $P_1$ , i.e., the columns that are still to be identified. Note also that  $\mathbf{a}_{P_1}$  can be computed from  $\mathbf{a}_P$  and the set  $R$  of identified columns *without* any additional test.

We have now a smaller instance of the same problem from which we started, namely identifying the columns of  $P_1$ , among the ones in  $M(S_1 \setminus R)$ , from their sum  $\mathbf{a}_{P_1}$ . Also notice that Lemma 1 still applies to the columns of  $M(S_1 \setminus R)$ . Therefore, repeatedly using the above argument we can eventually identify the whole set  $P$ . Again, no additional tests are required since we reinterpret, so to speak, the tests outcomes in light of new acquired knowledge.

Finally, by Theorem 1 a SUPER-SELECTOR  $M$  of size  $O(p \log \frac{n}{p})$  can be constructed in time  $O(n^p)$ , which gives the desired result. We hasten to remark that in [23] Grebinsky and Kucherov prove the existence of matrices  $M$  with an optimal  $O(\frac{p}{\log p} \log \frac{n}{p})$  number of rows for the Additive Group Testing described above. However, it's not clear whether their probabilistic construction can be derandomized, and at which cost. We thought worthwhile to mention that our combinatorial tool gives, for free, a solution to the Additive Group Testing problem using number of tests that differ from the optimal one for only a factor of  $\log p$ .

**Monotone Encodings.** Moran *et al.* posed the problem of efficiently constructing  $(n, k)$ -monotone encodings of size  $r$ , (denoted by  $ME(n, k, r)$ ), i.e., monotone injective functions mapping subsets of  $[n]$ , of size up to  $k$ , into  $2^{[r]}$  [31]. Monotone encodings are relevant to the study of tamper-proof data structures and arise also in the design of broadcast schemes in certain communication networks. A simple counting argument shows that  $ME(n, k, r)$  can only exist for  $r = \Omega(k \log n/k)$ . We can use our SUPER-SELECTOR for obtaining  $ME(n, k, O(k \log n/k))$  in the following way. Let  $M^{[t]}$  denote the  $(t, \mathbf{v}, n)$ -SUPER-SELECTOR defined by the vector  $\mathbf{v}$  whose  $i$ th component is  $v_i = \lfloor i/2 \rfloor + 1$  for each  $i = 1, \dots, t$ . By Lemma 1, we have that for any  $S \subseteq [t/2]$ , from  $\mathbf{a}_S$  we can identify at least  $|S|/2$  of the columns in  $M^{[t]}(S)$ . Let  $S^{yes}$  (resp.  $S^{no}$ ) be the subset of these columns which we can (resp. cannot) identify from  $\mathbf{a}_S$ .

We can obtain our mapping in the following way. Given  $S_0 \in \binom{[n]}{\leq k}$ , we map it to the concatenation of the vectors  $\mathbf{a}_0 \mathbf{a}_1, \dots, \dots, \mathbf{a}_{\log k}$ , where  $\mathbf{a}_i$  is the Boolean sum of the columns of  $M^{[k/2^{i-1}]}(S_i)$ , with  $S_i = S_{i-1}^{no}$ .

The mapping is of size  $\sum_{j=0}^{\log k} \frac{2k}{2^j} \log \frac{n2^j}{2k} = O(k \log n/k)$ , therefore of optimal size. Moreover, by observing that for each  $S \subseteq T$  we have  $\mathbf{a}_S \leq \mathbf{a}_T$  and  $S^{no} \subseteq T^{no}$ , we also have that the mapping is monotone. By our Theorem 1 such mapping can be deterministically computed in  $O(n^k)$ -time.

Alon and Hod [2] defined  $(k, \alpha)$ -FUT families in order to obtain  $ME(n, k, O(k \log \frac{n}{k}))$  in a way analogous to the one we depicted above, i.e., by

chaining  $(\frac{k}{2^t}, \frac{1}{2})$ -FUT families<sup>2</sup> of cardinality  $n$  for  $t = 0, 1, \dots, \log k$ . However, for optimal, i.e.,  $O(k \log n/k)$ -size monotone encodings no explicit deterministic construction has been provided so far [231].

**Selector-based data compression.** Let  $M$  be a  $(p+1, 2p, n)$ -selector of size  $m = O(p \log(n/p))$ . Let  $\mathbf{x}$  be a binary vector with  $\|\mathbf{x}\|_0 \leq p$ . Define the encoding of  $\mathbf{x}$  as the vector  $\mathbf{y}$  equal to the componentwise OR of columns of  $M$  corresponding to the positions of the 1's in  $\mathbf{x}$ . Let  $x_{i_1}, \dots, x_{i_d}$ ,  $d \leq p$ , be all the components of  $\mathbf{x}$  such that  $x_{i_1} = \dots = x_{i_d} = 1$ . By Lemma 1, there exist at most  $t$  other columns  $m_{j_1}, \dots, m_{j_t}$  of matrix  $M$ ,  $t \leq p$ , such that  $\mathbf{y} = m_{j_1} \vee \dots \vee m_{j_t} \vee m_{i_1} \vee \dots \vee m_{i_d}$ .

Now, think of an “encoder” that works as follows: for a given vector  $\mathbf{x}$  it first computes its encoding  $\mathbf{y}$ , then it computes  $A = \{i_1, \dots, i_d\}, B = \{j_1, \dots, j_t\}$ , and subsequently it computes an ordered list  $L$  from  $A \cup B$ . Finally, the encoder computes a binary vector  $\mathbf{z}$  of length  $2p$  such that  $z_k = 1$  if and only if the  $k$ -th element of the ordered list  $L$  is an element of  $A$ . The encoding of  $\mathbf{x}$  is now the concatenated binary vector  $\mathbf{yz}$  of length  $O(p \log(n/p)) + 2p = O(p \log(n/p))$ . One can see that  $\mathbf{x}$  can be (efficiently) recovered from  $\mathbf{yz}$  and that the length of the encoding  $\mathbf{yz}$  of  $\mathbf{x}$  is information theoretically optimal.

An extension of the above reasoning can be carried out also to a scenario where  $\mathbf{x}$  is generated by a probabilistic source, provided that  $\Pr\{\|\mathbf{x}\|_0 > p\}$  goes to zero as the length  $n$  of  $\mathbf{x}$  grows.

The above encoding procedure has some features which might be of some interest in the area of data compression. Specifically, it does not require construction of code dictionary, nor it is based on statistical analysis of the sequences to be compressed. Moreover, the encoding/decoding procedure only involves simple operations on Boolean vectors (OR's of them and checks for containments), which leads to fast implementation. Furthermore, the above procedure provides a faster alternative for optimal size enumerative encoding of low-weight binary sequences. [1235]. In particular, for binary vectors of Hamming weight at most  $d$ , our encoding/decoding procedures require time  $O(nd \log(n/d))$ , whereas the procedures given in [35] require time  $O(n \log^2 n \log \log n)$  for the encoding, and time  $O(n \log^3 n \log \log n)$  for the decoding.

**Tracing many users (or finding many positives).** In [28] the authors introduced  $k$ -out-of- $r$  Multi User Tracing families, aka  $MUT_k(r)$ . A family  $\mathcal{F}$  of  $n$  many subsets of  $[m]$  is  $MUT_k(r)$  if given the union of  $\ell \leq p$  of the sets in  $\mathcal{F}$ , one is able to identify at least  $k$  of them, or all if  $\ell < k$ . Such definition is motivated by applications in multiple access channel communication and DNA computing (see [28] and references quoted therein).

In [1] it was proved that  $MUT_k(r)$  families exist for  $m = O((r+k^2) \log \frac{n}{r})$ , determining the maximum possible rate  $\frac{\log n}{m}$  for all  $k \leq \sqrt{r}$  up to a constant factor. Somehow surprisingly, in all this range the rate is  $\Theta(\frac{1}{r})$ , independently of  $k$ . However, no constructive proof of such “optimal” rate families has been provided so far.

<sup>2</sup> In fact, via SUPER-SELECTORS, we can provide constructions of optimal size  $(k, \alpha)$ -FUT families, for any  $1/2 < \alpha < 1 - \frac{1}{k}$ .

We can use our SUPER-SELECTORS to match such result: Let  $M$  be a  $(2r, \mathbf{v}, n)$ -SUPER-SELECTOR where the vector  $\mathbf{v} = (v_1, \dots, v_{2r})$  is defined by:  $v_i = i$  for  $i = 1, \dots, k$ ;  $v_i = k$ , for  $i = k + 1, \dots, 2r - 1$ , and  $v_{2r} = r + 1$ .

First, we notice that  $M$  is a  $(k, k, n)$ -selector, i.e., a  $(k-1)$ -superimposed code, hence every union of up to  $k-1$  columns is unique. Moreover, for any  $k \leq \ell \leq r$ , by Lemma 1 we have that at least  $k$  columns out of  $\ell$  can be identified by their Boolean sum. These two properties show that the sets whose indicator vectors coincide with the columns of  $M$ , form an  $MUT_k(r)$  family. Therefore, Theorem 1 applied to  $M$  provides the best known bound on the size of  $MUT_k(r)$  families, i.e., the  $O(\max\{r, k^2\} \log n/r)$  of [1]. Our main theorem also explicitly shows that the result of [1] can be attained by a constructive  $O(n^k)$  strategy.

**The  $(d, \ell)$ -list disjunct matrices.** Indyk *et al.* [25] studied  $(d, \ell)$ -list disjunct matrix which are  $m \times n$  binary matrix such that the following holds: for any disjoint subsets  $S, T$  of columns, such that  $|S| \leq d$  and  $|T| \geq \ell$ , there exists a row where there is a 1 among the columns in  $T$ , while all the columns in  $S$  have a 0. Such structure was also considered in [14, 15, 19, 8].

One can easily verify that a  $(d+\ell, d+1, n)$ -selector is also a  $(d, \ell)$ -list disjunct matrix. As a consequence, our Lemma 3 (below) provides improved bounds on construction of  $(d, \ell)$ -list disjunct matrices<sup>3</sup> compared to the ones given in [25].

For any  $d \geq \ell$ , by using  $(d+\ell, d+1, n)$ -selector, we obtain  $(d, \ell)$ -list disjunct matrices of size  $O\left(\frac{(d+\ell)^2}{\ell} \log \frac{n}{\ell}\right)$  for any constant  $d$  and  $\ell$ . This improves on [25], particularly for  $d$  large compared to  $\ell$ . Also for  $\ell = \Theta(d)$  and particularly for  $(d, d)$ -list disjunct matrices our bound compares favorably with the  $O((d \log n)^{1+o(1)})$  size bound given in [25] and the  $O(d^{1+o(1)} \log n)$  size bound given in [8]. Alternatively, for  $d < \ell$  one can see that a  $(2d, d+1, n)$  selector is also a  $(d, \ell)$ -list disjunct matrix. Such a selector can be constructed of size  $O(d \log n/d)$ , in time  $n^{2d+o(1)}$ .

We remark that the above results on the size of  $(d, \ell)$ -list disjunct matrices via selectors, are tight with respect to the lower bounds provided in [15, Theorem 2], as reported in Table 1.

## 4 Bounds on the Size of a $(p, \mathbf{v}, n)$ -SUPER-SELECTOR

In this section we prove the bound on the size of a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR as announced in Theorem 1. First we present an immediate lower bound following from the ones of [7, 15] on the size of  $(p, k, n)$ -selectors.

**Theorem 2.** *The size of a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR has to be*

$$\Omega\left(\max_{j=1, \dots, p} \frac{j^2}{j - v_j + 1} \frac{\log(n/j)}{\log(j/(j - v_j + 1)) + O(1)}\right).$$

---

<sup>3</sup> Analogous bounds, in terms of size, are derivable from [15] via  $(p, k, n)$ -selectors. However, their construction time is exponential in  $n$ .

For the upper bound, we first give a proof based on the probabilistic method and then derandomize it. We need the following two lemmas.

**Lemma 2.** *There exists a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR of size*

$$m = O\left(\max_{j=1,\dots,p} \frac{3pej}{(j - v_j + 1)} \log(n/j)\right).$$

*Proof.* Generate the  $m \times n$  binary matrix  $M$  by choosing each entry randomly and independently, with  $\Pr(M[i, j] = 0) = (p - 1)/p = x$ . Fix an integer  $j \leq p$ . Fix  $S \in \binom{[n]}{j}$ . For any subset  $R$  of  $j - v_j + 1$  rows of  $I_j$  let  $E_{R,S}$  be the event that the submatrix  $M(S)$  does not contain *any* of the  $(j - v_j + 1)$  rows of  $R$ . We have

$$\Pr(E_{R,S}) = (1 - (j - v_j + 1)x^{j-1}(1 - x))^m \quad (1)$$

Let  $R_1, \dots, R_t$ ,  $t = \binom{j}{j - v_j + 1}$  be all possible subsets of exactly  $j - v_j + 1$  rows of the matrix  $I_j$ , and let  $N_S$  be the event that, for some index  $i \in \{1, \dots, t\}$ , the sub-matrix  $M(S)$  does not contain *any* of the rows of the subset  $R_i$ . By the union bound we have

$$\Pr(N_S) = \Pr\left(\bigvee_{i=1}^t E_{R_i,S}\right) \leq \binom{j}{j - v_j + 1} (1 - (j - v_j + 1)x^{j-1}(1 - x))^m \quad (2)$$

One can see that  $N_S$  coincides with the event that the sub-matrix  $M(S)$  contains strictly less than  $v_j$  rows of  $I_j$ . To see this, it is enough to observe that if  $M(S)$  contains less than  $v_j$  rows of  $I_j$  it means that there is some  $i$  such that  $M(S)$  does not contain any of the rows in  $R_i$ .

Let  $Y_M$  denote the event that the matrix  $M$  is a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR. We can use again the union bound to estimate the probability of the negated event  $\overline{Y_M}$ . If  $M$  is not a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR then there exists an integer  $j \in [p]$  such that for some  $S \in \binom{[n]}{j}$  the event  $N_S$  happens. Therefore,

$$\Pr(\overline{Y_M}) = \Pr\left(\bigvee_{j=1}^p \bigvee_{S \in \binom{[n]}{j}} N_S\right),$$

whence, we obtain:

$$\Pr(Y_M) \geq 1 - \sum_{j=1}^p \binom{n}{j} \binom{j}{j - v_j + 1} (1 - (j - v_j + 1)x^{j-1}(1 - x))^m. \quad (3)$$

By the probabilistic method, there exists a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR of size  $m^* = \operatorname{argmin}_{m \geq 1} \Pr(Y_M) > 0$ . The rest of the proof will consist in showing that  $m^*$  satisfies the bound claimed.

Let us focus on the value  $c_j$  such that the  $j$ -th summand in (3) satisfies the following inequality

$$\binom{n}{j} \binom{j}{j - v_j + 1} (1 - (j - v_j + 1)x^{j-1}(1 - x))^{c_j j \log n / j} \leq 1/p \quad (4)$$

We shall use the following two inequalities

$$(1 - (j - v_j + 1)x^{j-1}(1-x))^{c_j j \log(n/j)} \leq \left(\frac{n}{j}\right)^{-\frac{(j-v_j+1)c_j j}{ep}} \quad (5)$$

$$\binom{n}{j} \binom{j}{j-v_j+1} \leq n^j 2^{\frac{j}{2}} e^{\frac{3j}{2}} j^{-j} \quad (6)$$

By (5)-(6), we have that the left-hand-side of (4) can be upper bounded by

$$n^{j - \frac{c_j(j-v_j+1)j}{pe}} 2^{\frac{j}{2}} e^{\frac{3j}{2}} j^{-\left(j - \frac{c_j(j-v_j+1)j}{pe}\right)} = n^{j - \frac{c_j(j-v_j+1)j}{pe}} 2^{\frac{j}{2}} e^{\frac{3j}{2}} j^{-j + \frac{c_j(j-v_j+1)j}{pe}}, \quad (7)$$

Therefore, if we take  $c_j = \frac{3pe}{(j-v_j+1)}$  we have that (7) can be further upper bounded with  $n^{-2j} e^{2j} j^{2j}$  which is not larger than  $1/p$  for all  $n \geq 20$  and  $n > p \geq j > 0$ . Therefore, by taking

$$m = \max_{j=1,\dots,p} c_j \log(n/j) = \max_{j=1,\dots,p} \frac{3pej}{(j-v_j+1)} \log \frac{n}{j} \quad (8)$$

we can have each of the summands in (3) smaller than  $1/p$ , hence guaranteeing  $\Pr(Y_M) > 0$ . By definition  $m^* \leq m$  which concludes the proof.  $\square$

The same analysis as above, tailored for a  $(p, k, n)$ -selector gives the following bounds.

**Lemma 3.** *For each  $0 \leq k < p < n$ , there exists a  $(p, k, n)$ -selector of size*

$$m = \left( \log_2 \frac{e}{e - 1 + \frac{k}{p}} \right)^{-1} p \log \frac{n}{p} (1 + o(1)) \leq \frac{2p^2}{p - k + 1} \log \frac{n}{p} (1 + o(1)). \quad (9)$$

Moreover, there exists a  $(p, p, n)$ -selector of size  $m = \frac{ep^2}{\log_2 e} \log(n/p) (1 + o(1))$ .

We can now combine the last two lemmas to obtain the main result of this section, providing an almost tight upper bound on the size of a SUPER-SELECTOR.

**Theorem 3.** *There exists a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR of size*

$$m = O(\max_{j=1,\dots,p} k_j \log(n/j)), \quad \text{where } k_j = \min \left\{ \frac{3pej}{(j-v_j+1)}, \frac{ej^2}{\log_2 e} \right\}$$

*Proof.* Fix  $k = \max \left\{ j \mid \frac{3pej}{(j-v_j+1)} > \frac{ej^2}{\log_2 j} \right\}$ . Let  $M_1$  be a minimum size  $(k, k, n)$ -selector. In particular this is a  $(k, < 1, 2, \dots, k >, n)$ -SUPER-SELECTOR hence *a fortiori* it is also a  $(k, (v_1, \dots, v_k), n)$ -SUPER-SELECTOR.

Let  $M_2$  be a minimum size  $(p, (0, \dots, 0, v_{k+1}, \dots, v_p), n)$ -SUPER-SELECTOR.

Let  $M$  be the binary matrix obtained by pasting together, one on top of the other,  $M_1$  and  $M_2$ . It is not hard to see that  $M$  is a  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR. By Lemmas 3 and 2,  $M$  satisfies the desired bound. The proof is complete.  $\square$

*Remark 3.* Note that, if there exists a constant  $\alpha$  such that  $v_j \leq \alpha j$  for each  $\sqrt{p} < j \leq p$ , then the size of the SUPER-SELECTOR is  $O(p \log \frac{n}{p})$ , matching the information theoretic lower bound. Particular cases are given by instances where for each  $j$ , we have  $v_j = f_j(j)$  for some function  $f_j$  such that  $f_j(j) = o(j)$ .

**Deterministic construction.** By using the method of the conditional expectations (see, e.g., [B0]) we can derandomize the result of the previous section and provide a deterministic construction of the  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR of Theorem 3 which is polynomial in  $n$  but exponential in the second parameter  $p$ . More precisely we obtain the following result, whose proof is deferred to full version of the paper.

**Theorem 4.** *There exists a deterministic  $O(p^3 n^{p+1} \log n)$  construction of the  $(p, \mathbf{v}, n)$ -SUPER-SELECTOR given by Theorem 3.*

## References

1. Alon, N., Asodi, V.: Tracing many users with almost no rate penalty. *IEEE Trans. on Information Theory* 53(1), 437–439 (2007)
2. Alon, N., Hod, R.: Optimal Monotone Encodings. *IEEE Trans. on Information Theory* 55(3), 1343–1353 (2009)
3. Balding, D.J., et al.: A comparative survey of non-adaptive pooling design. In: Speed, T.P., Waterman, M.S. (eds.) *Genetic mapping and DNA sequencing*, IMA Volumes in Mathematics and its Appl., pp. 133–154. Springer, Heidelberg (1996)
4. Chaudhuri, S., Radhakrishnan, J.: Deterministic restrictions in circuit complexity. In: Proc. of 28th STOC, pp. 30–36 (1996)
5. Cheng, Y., Du, D.Z.: New Constructions of One- and Two-Stage Pooling Designs. *Journal of Computational Biology* 15(2), 195–205 (2008)
6. Cheng, Y., Du, D.Z., Lin, G.: On the upper bounds of the minimum number of rows of disjunct matrices. *Optimization Letters* 3, 297–302 (2009)
7. Chlebus, B.S., Kowalski, D.R.: Almost Optimal Explicit Selectors. In: Liśkiewicz, M., Reischuk, R. (eds.) *FCT 2005*. LNCS, vol. 3623, pp. 270–280. Springer, Heidelberg (2005)
8. Cheraghchi, M.: Noise-resilient group testing: Limitations and constructions. In: Proc. of FCT 2009 (2009)
9. Chrobak, M., Gasieniec, L., Rytter, W.: Fast Broadcasting and Gossiping in Radio Networks. In: FOCS 2000, pp. 575–581 (2000)
10. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: Proc. of Symp. on Discrete Algorithms (SODA 2001), pp. 709–718 (2001)
11. Cormode, G., Muthukrishnan, S.: Combinatorial Algorithms for Compressed Sensing. In: Flocchini, P., Gasieniec, L. (eds.) *SIROCCO 2006*. LNCS, vol. 4056, pp. 280–294. Springer, Heidelberg (2006)
12. Cover, T.: Enumerative source encoding. *IEEE Trans. Inf. Th.* 19, 73–77 (1973)
13. Damaschke, P.: Adaptive versus Nonadaptive Attribute-Efficient Learning. In: STOC 1998, pp. 590–596 (1998)
14. De Bonis, A., Vaccaro, U.: Constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. *Theoretical Computer Science* 306, 223–243 (2003)

15. De Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. *SIAM J. on Comp.* 34(5), 1253–1270 (2005)
16. Du, D.Z., Hwang, F.K.: Pooling Design and Nonadaptive Group Testing. World Scientific, Singapore (2006)
17. D'yachkov, A.G., Rykov, V.V.: Bounds of the length of disjunct codes. *Problems Control Inform. Theory* 11, 7–13 (1982)
18. D'yachkov, A.G., Rykov, V.V., Rashad, A.M.: Superimposed distance codes. *Problems Control Inform. Theory* 18, 237–250 (1989)
19. Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved Combinatorial Group Testing Algorithms for Real-World Problem Sizes. *SIAM J. on Comp.* 36, 1360–1375 (2007)
20. Erdős, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of  $r$  others. *Israel J. of Math.* 51, 75–89 (1985)
21. Ganguly, S.: Data stream algorithms via expander graph. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) *ISAAC 2008. LNCS*, vol. 5369, pp. 52–63. Springer, Heidelberg (2008)
22. Gilbert, A.C., Iwen, M.A., Strauss, M.J.: Group Testing and Sparse Signal Recovery. In: 42nd Asilomar Conf. on Signals, Systems, and Computers, pp. 1059–1063 (2008)
23. Grebinsky, V., Kucherov, G.: Optimal Reconstruction of Graphs under the Additive Model. *Algorithmica* 28(1), 104–124 (2000)
24. Indyk, P.: Deterministic superimposed coding with application to pattern matching. In: Proc. of 39th FOCS 1997, pp. 127–136 (1997)
25. Indyk, P., Ngo, H.Q., Rudra, A.: Efficiently Decodable Non-adaptive Group Testing. In: Proc. of 20th SODA, pp. 1126–1142 (2010)
26. Kautz, W.H., Singleton, R.R.: Nonrandom binary superimposed codes. *IEEE Trans. on Inform. Theory* 10, 363–377 (1964)
27. Kumar, R., Rajagopalan, S., Sahai, A.: Coding constructions for blacklisting problems without computational assumptions. In: Wiener, M. (ed.) *CRYPTO 1999. LNCS*, vol. 1666, pp. 609–623. Springer, Heidelberg (1999)
28. Laczay, B., Ruszikó, M.: Multiple User Tracing Codes. In: Proc. of ISIT 2006, pp. 1900–1904 (2006)
29. Linial, N.: Locality in distributed graph algorithms. *SIAM J. on Computing* 21, 311–312 (1992); *Discrete Mathematics* 162, 311–312 (1996)
30. Mitzenmacher, M., Upfal, E.: *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Cambridge (2005)
31. Moran, T., Naor, M., Segev, G.: Deterministic history-independent strategies for storing information on write-once memories. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) *ICALP 2007. LNCS*, vol. 4596, pp. 303–315. Springer, Heidelberg (2007)
32. Porat, B., Porat, E.: Exact and Approximate Pattern Matching in the Streaming Model. In: Proc. 50th FOCS, pp. 315–323 (2009)
33. Porat, E., Rothschild, A.: Explicit non-adaptive combinatorial group testing schemes. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I. LNCS*, vol. 5125, pp. 748–759. Springer, Heidelberg (2008)
34. Clifford, R., Efremenko, K., Porat, E., Rothschild, A.:  $k$ -Mismatch with Don't Cares. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007. LNCS*, vol. 4698, pp. 151–162. Springer, Heidelberg (2007)
35. Ryabko, B.: Fast Enumerative Source Coding. In: Proc. of 1995 IEEE Intern. Symp. on Inf. Th., p. 395 (1995)
36. Wolf, J.: Born again group testing: Multiaccess Communications. *IEEE Trans. Information Theory* 31, 185–191 (1985)

# Estimating the Average of a Lipschitz-Continuous Function from One Sample

Abhimanyu Das and David Kempe

University of Southern California  
`{abhimand,dkempe}@usc.edu`

**Abstract.** We study the problem of estimating the average of a Lipschitz continuous function  $f$  defined over a metric space, by querying  $f$  at only a single point. More specifically, we explore the role of randomness in drawing this sample. Our goal is to find a distribution minimizing the expected estimation error against an adversarially chosen Lipschitz continuous function. Our work falls into the broad class of estimating aggregate statistics of a function from a small number of carefully chosen samples. The general problem has a wide range of practical applications in areas such as sensor networks, social sciences and numerical analysis. However, traditional work in numerical analysis has focused on asymptotic bounds, whereas we are interested in the *best* algorithm. For arbitrary discrete metric spaces of bounded doubling dimension, we obtain a PTAS for this problem. In the special case when the points lie on a line, the running time improves to an FPTAS. For Lipschitz-continuous functions over  $[0, 1]$ , we calculate the precise achievable error as  $1 - \frac{\sqrt{3}}{2}$ , which improves upon the  $\frac{1}{4}$  which is best possible for deterministic algorithms.

## 1 Introduction

One of the fundamental problems in data-driven sciences is to estimate some aggregate statistic of a real-valued function  $f$ , by sampling  $f$  in few places. Frequently, obtaining samples incurs a cost in terms of computation, energy or time. Thus, researchers face an inherent tradeoff between the accuracy of estimating the aggregate statistic and the number of samples required. With samples a scarce resource, it becomes an important problem to determine where to sample  $f$ , and how to post-process the samples.

Naturally, there are many mathematical formulations of this estimation problem, depending on the aggregate statistic that we wish to estimate (such as the average, median or maximum value), the error objective that we wish to minimize (such as worst-case absolute error, average-case squared error, etc.), and on the conditions imposed on the function. In this paper, we study algorithms optimizing a worst-case error objective, i.e., we assume that  $f$  is chosen adversarially. Motivated by the applications described below, we use Lipschitz-continuity to impose a “smoothness” condition on  $f$ . (Note that without any smoothness conditions on  $f$ , we cannot hope to approximate any aggregate function in an adversarial setting without learning all function values.) That is, we assume that

the domain of  $f$  is a metric space, and that  $f$  is Lipschitz-continuous over its domain. Thus, nearby points are guaranteed to have similar function values.

Here, we focus on perhaps the simplest aggregation function: the average  $\bar{f}$ . Despite its simplicity, it has many applications. For example, in sensor networks covering a geographical area, the average of a natural phenomenon (such as temperature or humidity) is frequently one of the most interesting quantities. Here, nearby locations tend to yield similar measurements. Since energy is a scarce resource, it is desirable to sample only a few of the sensors. Another application is in numerical analysis, where one of the fundamental problems is numerical integration of a function. If the domain is continuous, this corresponds precisely to computing the average. If the function to be integrated is costly to evaluate, then again, it is desirable to sample a small number of points.

If  $f$  is to be evaluated at  $k$  points, chosen deterministically and non-adaptively, then previous work [4] shows that the optimum sampling locations for estimating the average of  $f$  form a  $k$ -median of the metric space. However, the problem becomes significantly more complex when the algorithm gets to randomize its choice of sampling locations. In fact, even the seemingly trivial case of  $k = 1$  turns out to be highly non-trivial, and is the focus of this paper. Addressing this case is an important step toward the ultimate goal of understanding the tradeoffs between the number of samples and the estimation error.

Formally, we thus study the following question: Given a metric space  $\mathcal{M}$ , a randomized *sampling algorithm* is described by (1) a method for sampling a location  $x \in \mathcal{M}$  from a distribution  $\mathbf{p}$ ; (2) a function  $g$  for predicting the average  $\bar{f}$  of the function  $f$  over  $\mathcal{M}$ , using the sample  $(x, f(x))$ . The expected estimation error is then  $E(\mathbf{p}, g, f) = \sum_{x \in \mathcal{M}} p_x \cdot |g(x, f(x)) - \bar{f}|$ . (The sum is replaced by an integral, and  $\mathbf{p}$  by a density, if  $\mathcal{M}$  is continuous.) The worst-case error is  $\hat{E}(\mathbf{p}, g) = \sup_{f \in L} E(\mathbf{p}, g, f)$ , where  $L$  is the set of all 1-Lipschitz continuous functions defined on  $\mathcal{M}$ . Our goal is to find a randomized sampling algorithm (i.e., a distribution  $\mathbf{p}$  and function  $g$ , computable in polynomial time) that (approximately) minimizes  $\hat{E}(\mathbf{p}, g)$ .

In this paper, we provide a PTAS for minimizing  $\hat{E}(\mathbf{p}, g)$ , for any discrete metric space  $\mathcal{M}$  with constant doubling dimension. (This includes constant-dimensional Euclidean metric spaces.) For discrete metric spaces  $\mathcal{M}$  embedded on a line, we improve this result to an FPTAS. Both of these algorithms are based on a linear program with infinitely many constraints, for which an approximate separation oracle is obtained.

We next study the perhaps simplest variant of this problem, in which the metric space is the interval  $[0, 1]$ . While the worst-case error of any deterministic algorithm is obviously  $\frac{1}{4}$  in this case, we show that for a randomized algorithm, the bound improves to  $1 - \frac{\sqrt{3}}{2}$ . We prove this by providing an explicit distribution, and obtaining a matching lower bound using Yao's Minimax Principle. Our result can also be interpreted as showing how “close” a collection of Lipschitz-continuous functions on  $[0, 1]$  can be.

Due to space constraints for this version, several discussions and proofs are relegated to a full version, which will be available on the authors' web sites.

### 1.1 Related Work

Estimating the integral of a smooth function  $f$  using its values at a discrete set of points is one of the core problems in numerical analysis. The tradeoffs between the number of samples needed and the estimation error bounds have been investigated in detail under the name of *Information Based Complexity (IBC)* [10,11]. More generally, IBC studies the problem of computing approximations to an operator  $S(f)$  on functions  $f$  from a set  $F$  (with certain “smoothness” properties) using a finite set of samples  $N(f) = [L_1(f), L_2(f), \dots, L_n(f)]$ . The  $L_i$  are functionals. For a given algorithm  $U$ , its error is  $E(U) = \sup_{f \in F} \|S(f) - U(f)\|$ . The goal in IBC is to find an  $\epsilon$ -approximation  $U$  (i.e., ensuring that  $E(U) \leq \epsilon$ ) with least information cost  $c(U) = n$ .

One of the common problems in IBC is multivariate integration of real-valued functions with a smoothness parameter  $r$  over  $d$ -dimensional unit balls. For such problems, Bakhvalov [2] designed a randomized algorithm providing an  $\epsilon$ -approximation with cost  $\Theta(\frac{1}{n^{2d/(d+2r)}})$ . Bakhvalov [2], and later, Novak [9] also show that this cost is asymptotically optimal. The papers by Novak [9] and Mathe [7] show that if  $r = 0$ , then simple Monte-Carlo integration algorithms (which sample from the uniform distribution) have an asymptotically optimal cost of  $\frac{1}{\epsilon^2}$ . In [12,13], Wozniakowski studied the average case complexity of linear multivariate IBC problems, and proved conditions under which the problems are tractable, i.e., have cost polynomial in  $\frac{1}{\epsilon}$  and  $d$ .

In [3], Baran et al. study the IBC problem for univariate integration of Lipschitz continuous functions, in an adaptive setting. That is, the sampling strategy can change adaptively based on the previously sampled values. They provide a deterministic and a randomized  $\epsilon$ -approximation algorithm, which use  $O(\log(\frac{1}{\epsilon \cdot \text{OPT}}) \cdot \text{OPT})$  and  $O(\text{OPT}^{4/3} + \text{OPT} \cdot \log(\frac{1}{\epsilon}))$  samples, respectively. Here, OPT is the optimal number of samples for the problem instance. They prove that their algorithms are asymptotically optimal.

There are two main differences between the results in IBC and our work: first, IBC treats the target approximation as given and minimizes the number of samples. Our goal is to minimize the expected worst-case error with a fixed number of samples (one). More importantly, results in IBC are traditionally *asymptotic*, ignoring constants. For a single sample, this would trivialize the problem: it is implicit in our proofs that sampling at the metric space’s median is a constant-factor approximation to the best randomized algorithm.

The deterministic version of our problem was studied previously in [4]. There, it was shown that the best sampling locations for reading  $k$  values non-adaptively are the optimal  $k$ -median of the metric space. Thus, the algorithm of Arya et al. [1] gives a polynomial-time  $(3 + \epsilon)$ -approximation algorithm.

## 2 Preliminaries

We are interested in real-valued Lipschitz-continuous functions over metric spaces of constant doubling dimension (e.g., [6]). Let  $(\mathcal{M}, d)$  be a compact metric space with distances  $d(x, y)$  between pairs of points. W.l.o.g., we assume that

$\max_{x,y \in \mathcal{M}} d(x, y) = 1$ . We require  $(\mathcal{M}, d)$  to have constant doubling dimension  $\beta$ , i.e., for every  $\delta$ , each ball of diameter  $\delta$  can be covered by at most  $c^\beta$  balls of diameter  $\delta/c$ , for any  $c \geq 2$ .

A function  $f$  is Lipschitz continuous (with constant 1) if for all  $x, y$ , we have  $|f(x) - f(y)| \leq d(x, y)$ . Let  $L$  be the set of all such Lipschitz-continuous functions  $f$ , i.e.,  $L = \{f \mid |f(x) - f(y)| \leq d(x, y) \text{ for all } x, y\}$ . We also define  $L_c = \{f \in L \mid |\int_x f(x) dx| \leq c\}$ . Notice that  $L_c$  is a compact set.

We wish to predict the average  $\bar{f} = \int_x f(x) dx$  of all the function values. When  $\mathcal{M}$  is finite of size  $n$ , then the average is of course  $\bar{f} = \frac{1}{n} \cdot \sum_x f(x)$  instead. The algorithm first gets to choose a single point  $x$  according to a (polynomial-time computable) density function  $\mathbf{p}$ ; it then learns the value  $f(x)$ , and may post-process it with a *prediction function*  $g(x, f(x))$  to produce its estimate of the average  $\bar{f}$ . The goal is to minimize the expected estimation error of the average, under the assumption that  $f$  is chosen adversarially from  $L$  with knowledge of the algorithm, but not its random choices. Formally, the goal is to minimize  $\hat{E}(\mathbf{p}, g) = \sup_{f \in L} (\int_x p_x \cdot |\bar{f} - g(x, f(x))| dx)$ . If  $\mathcal{M}$  is finite, then  $\mathbf{p}$  will be a probability distribution instead of a density, and the error now can be written as  $\hat{E}(\mathbf{p}, g) = \sup_{f \in L} (\sum_x p_x \cdot |\bar{f} - g(x, f(x))|)$ .

Formally, we consider an algorithm to be the pair  $(\mathbf{p}, g)$  of the distribution and prediction function. Let  $\mathcal{A}$  denote the set of all such pairs, and  $\mathcal{D}$  the set of all *deterministic* algorithms, i.e., algorithms for which  $\mathbf{p}$  has all its density on a single point. Our analysis will make heavy use of Yao's Minimax Principle [8]. To state it, we define  $\mathcal{L}$  to be the set of all probability distributions over  $L$ . We also define the estimation error  $\Delta(f, A) = \int_x p_x \cdot |\bar{f} - g(x, f(x))| dx$ , where  $A$  corresponds to the pair  $(\mathbf{p}, g)$ .

### Theorem 1 (Yao's Minimax Principle [8])

$$\sup_{q \in \mathcal{L}} \inf_{A \in \mathcal{D}} \mathbb{E}_{f \sim q} [\Delta(f, A)] = \inf_{A \in \mathcal{A}} \sup_{f \in L} \Delta(f, A).$$

The next theorem shows that without loss of generality, we can focus on algorithms whose post-processing is just to output the observed value, i.e., algorithms  $(\mathbf{p}, \text{id})$  with  $\text{id}(x, y) = y$ , for all  $x, y$ .

**Theorem 2.** *Let  $A^* = (\mathbf{p}^*, g^*)$  be the optimum randomized algorithm. Then, for every  $\epsilon > 0$ , there is a randomized algorithm  $A = (\mathbf{p}, \text{id})$  with  $\hat{E}(A) \leq \hat{E}(A^*) + \epsilon$ .*

## 3 Discrete Metric Spaces

In this section, we focus on finite metric spaces, consisting of  $n$  points. Thus, instead of integrals and densities, we will be considering sums and probability distributions. The result from Theorem 2 holds in this case as well; hence w.l.o.g., we assume that all algorithms simply output the value they observe. The problem of finding the best probability distribution for a single sample can be expressed as a linear program, with variables  $p_x$  for the sampling probabilities at each of the  $n$  points  $x$ , and a variable  $Z$  for the estimation error.

$$\begin{aligned}
& \text{Minimize} && Z \\
& \text{subject to} && \begin{aligned}
& \text{(i)} \quad \sum_x p_x = 1 \\
& \text{(ii)} \quad \sum_x p_x \cdot |\bar{f} - f(x)| \leq Z \quad \text{for all } f \in L \\
& \text{(iii)} \quad 0 \leq p_x \leq 1 \quad \text{for all points } x
\end{aligned}
\end{aligned} \tag{1}$$

Since this LP (which we refer to as the “exact LP”) has infinitely many constraints, our approach is to replace the set  $L$  in the second constraint with a set  $Q_\delta$ . We will choose  $Q_\delta$  carefully to ensure that it “approximates”  $L$  well, and such that the resulting LP (which we refer to as the “discretized LP”) can be solved efficiently.

To define the notion of approximation formally, let  $o$  be a 1-median of the metric space, i.e., a point minimizing  $\sum_x d(o, x)$ . Let  $m = \frac{1}{n} \sum_x d(o, x)$  be the average distance of all points from  $o$ . Since w.l.o.g.  $\max_{x,y \in \mathcal{M}} d(x, y) = 1$ , at least one point has distance at least  $\frac{1}{2}$  from  $o$ , and thus  $m \geq \frac{1}{2n}$ . The median value  $m$  forms a lower bound for randomized algorithms in the following sense.

**Lemma 1.** *The worst-case expected error for any randomized algorithm is at least  $\frac{1}{4 \cdot 6^\beta} \cdot m$ , where  $\beta$  is the doubling dimension of the metric space.*

**Proof.** Consider any randomized algorithm with probability distribution  $\mathbf{p}$ . Let  $R = \{x \mid \frac{m}{2} \leq d(x, o) \leq \frac{3m}{2}\}$  be the ring of points at distance between  $\frac{m}{2}$  and  $\frac{3m}{2}$  from  $o$ . We distinguish two cases:

1. If  $\sum_{x \in R} p_x \leq \frac{1}{2}$ , consider the Lipschitz-continuous function  $f(x) = d(x, o)$ . Then,  $\bar{f} = m$ . With probability at least  $\frac{1}{2}$ , the algorithm samples a point outside  $R$ , and thus outputs a value outside the interval  $[\frac{m}{2}, \frac{3m}{2}]$ , which incurs error at least  $\frac{m}{2}$ . Thus, the expected error is at least  $\frac{m}{4}$ .
2. If  $\sum_{x \in R} p_x > \frac{1}{2}$ , then consider a collection of balls  $B_1, \dots, B_k$  of diameter  $\frac{m}{2}$  covering all points in  $R$ . Because  $R$  is contained in a ball of diameter  $3m$ , the doubling constraint implies that  $k \leq 6^\beta$  balls are sufficient. At least one of these balls — say,  $B_1$  — has  $\sum_{x \in B_1} p_x \geq \frac{1}{2k}$ . Fix an arbitrary point  $y \in B_1$ , and define the Lipschitz-continuous function  $f$  as  $f(x) = d(x, y)$ . Because  $o$  was a 1-median, we get that  $\bar{f} \geq m$ . With probability at least  $\frac{1}{2k}$ , the algorithm will choose a point inside  $B_1$  and output a value of at most  $\frac{m}{2}$ , thus incurring an error of at least  $\frac{m}{2}$ . Hence, the expected error is at least  $\frac{1}{2k} \cdot \frac{m}{2} \geq \frac{1}{4 \cdot 6^\beta} \cdot m$ . ■

We now formalize our notion for a set of functions  $Q_\delta$  to be a good approximation.

**Definition 1 ( $\delta$ -approximating function classes).** *For any sampling distribution  $\mathbf{p}$ , let  $E_L(\mathbf{p}) = \max_{f \in L} \Delta(f, \mathbf{p})$  and  $E_{Q_\delta}(\mathbf{p}) = \max_{f \in Q_\delta} \Delta(f, \mathbf{p})$  be the maximum error of sampling according to  $\mathbf{p}$  against a worst-case function from  $L$  and  $Q_\delta$ , respectively, where  $\Delta(f, \mathbf{p}) = \sum_x p_x \cdot |\bar{f} - f(x)|$ . The class  $Q_\delta$   $\delta$ -approximates  $L$  if*

1. *For each  $f \in L$ , there is a function  $f' \in Q_\delta$  such that for all distributions  $\mathbf{p}$ , we have  $|\Delta(f', \mathbf{p}) - \Delta(f, \mathbf{p})| \leq \frac{\delta}{2} \cdot E_L(\mathbf{p})$ .*

2. For each  $f \in Q_\delta$ , there is a function  $f' \in L$  such that for all distributions  $\mathbf{p}$ , we have  $|\Delta(f', \mathbf{p}) - \Delta(f, \mathbf{p})| \leq \frac{\delta}{2} \cdot E_L(\mathbf{p})$ .

**Theorem 3.** Assume that for every  $\delta$ ,  $Q_\delta$  is a class of functions  $\delta$ -approximating  $L$ , such that the following problem can be solved in polynomial time (for fixed  $\delta$ ): Given  $\mathbf{p}$ , find a function  $f \in Q_\delta$  maximizing  $\Delta(f, \mathbf{p})$ .

Then, solving the discretized LP gives a PTAS for the problem of finding a sampling distribution that minimizes the worst-case expected error.

**Proof.** First, an algorithm to find a function  $f$  maximizing  $\sum_x p_x \cdot |\bar{f} - f(x)|$  gives a separation oracle for the discretized LP. Thus, using the Ellipsoid Method (e.g., [5]), an optimal solution to the discretized LP can be found in polynomial time, for any fixed  $\delta$ .

Let  $\mathbf{p}, \mathbf{q}$  be optimal solutions to the exact and discretized LPs, respectively. Let  $f_1 \in L$  maximize  $\sum_x q_x \cdot |\bar{f} - f(x)|$  over  $f \in L$ , and  $f_2 \in Q_\delta$  maximize  $\sum_x p_x \cdot |\bar{f} - f(x)|$  over  $f \in Q_\delta$ . Thus,  $\Delta(f_1, \mathbf{q}) = E_L(\mathbf{q})$  and  $\Delta(f_2, \mathbf{p}) = E_Q(\mathbf{p})$ .

Now, applying Definition 10 to  $f_1 \in L$  gives us a function  $f'_1 \in Q_\delta$  such that  $|\Delta(f'_1, \mathbf{q}) - E_L(\mathbf{q})| \leq \frac{\delta}{2} E_L(\mathbf{q})$ . Since  $E_Q(\mathbf{q}) \geq \Delta(f'_1, \mathbf{q})$ , we obtain that  $E_Q(\mathbf{q}) \geq E_L(\mathbf{q})(1 - \frac{\delta}{2})$ .

Similarly, applying Definition 10 to  $f_2 \in Q_\delta$ , gives us a function  $f'_2 \in L$  with  $|\Delta(f'_2, \mathbf{p}) - E_Q(\mathbf{p})| \leq \frac{\delta}{2} E_Q(\mathbf{p})$ . Since  $E_L(\mathbf{p}) \geq \Delta(f'_2, \mathbf{p})$ , we obtain that  $E_L(\mathbf{p}) \geq E_Q(\mathbf{p}) - \frac{\delta}{2} E_Q(\mathbf{p})$ , or  $E_L(\mathbf{p}) \geq \frac{E_Q(\mathbf{p})}{1 + \frac{\delta}{2}}$ . Also, by optimality of  $\mathbf{q}$  in  $Q_\delta$ ,  $E_Q(\mathbf{q}) \leq E_Q(\mathbf{p})$ . Thus,  $E_L(\mathbf{q}) \leq \frac{E_Q(\mathbf{q})}{1 - \frac{\delta}{2}} \leq \frac{E_Q(\mathbf{p})}{1 - \frac{\delta}{2}} \leq \frac{E_L(\mathbf{p})(1 + \frac{\delta}{2})}{1 - \frac{\delta}{2}} \leq E_L(\mathbf{p})(1 + 2\delta)$ . ■

### 3.1 A PTAS for Arbitrary Metric Spaces

We first observe that since the error for any translation of a function  $f$  is the same as for  $f$ , we can assume w.l.o.g. that  $f(o) = 0$  for all functions  $f$  considered in this section. Thus, we implicitly restrict  $L$  to functions with  $f(o) = 0$ .

We next describe a set  $Q_\delta$  of functions which  $\delta$ -approximate  $L$ . Roughly, we will discretize function values to different multiples of  $\gamma$ , and consider distance scales that are different multiples of  $\gamma$ . We later set  $\gamma = \frac{\delta}{48 \cdot 6^\beta + 6}$ . We show in Lemma 12 that  $Q_\delta$  has size  $n^{\log(2/\gamma)(2/\gamma)^\beta} = n^{O(1)}$  for constant  $\delta$ ; this immediately implies that the discretized LP can be solved in time  $O(\text{poly}(n) \cdot n^{\log(2/\gamma)(2/\gamma)^\beta})$  (using exhaustive search for the separation oracle), and we obtain a PTAS for finding the optimum distribution.

We let  $k = \log_2 \frac{1}{2m}$ , and define a sequence of  $k$  rings of exponentially decreasing diameter around  $o$ , dividing the space into  $k+1$  regions  $R_1, \dots, R_{k+1}$ . Specifically, we let  $R_{k+1} = \{x \mid d(x, o) \leq 2m\}$ , and  $R_i = \{x \mid 2^{-i} < d(x, o) \leq 2^{-(i-1)}\}$  for  $i = 1, \dots, k$ . Since  $m \geq \frac{1}{2n}$ , we have that  $k \leq \log n$ .

Since the metric space has doubling dimension  $\beta$ , each region  $R_i$  can be covered with at most  $(2/\gamma)^\beta$  balls of diameter  $2\gamma \cdot 2^{-i}$ . Let  $B_{i,j}$  denote the  $j^{\text{th}}$  ball from the cover of  $R_i$ . W.l.o.g., each  $B_{i,j}$  is non-empty and contained in  $R_i$  (otherwise, consider its intersection with  $R_i$  instead). We call  $B_{i,j}$  the  $j^{\text{th}}$  grid

ball for region  $i$ . Thus, the grid balls cover all points, and there are at most  $(2/\gamma)^\beta \cdot \log n$  grid balls.

For each grid ball  $B_{i,j}$ , let  $o_{i,j} \in B_{i,j}$  be an arbitrary, but fixed, *representative* of  $B_{i,j}$ . The exception is that for the grid ball containing  $o$ ,  $o$  must be chosen as the representative. We now define the class  $Q_\delta$  of functions  $f$  as follows:

1. For each  $i, j$ ,  $f(o_{i,j})$  is a multiple of  $\gamma \cdot 2^{-i}$ .
2. For all  $(i, j), (i', j')$ , the function values satisfy the *relaxed Lipschitz-condition*  $|f(o_{i,j}) - f(o_{i',j'})| \leq d(o_{i,j}, o_{i',j'}) + \gamma \cdot (2^{-i} + 2^{-i'})$ .
3. All points in  $B_{i,j}$  have the same function value, i.e.,  $f(x) = f(o_{i,j})$  for all  $x \in B_{i,j}$ .

**Lemma 2.** *The size of  $Q_\delta$  is at most  $n^{\log(2/\gamma)(2/\gamma)^\beta}$ .*

We need to prove that  $Q_\delta$  approximates  $L$  well, by verifying that for each function  $f \in L$ , there is a “close” function in  $Q_\delta$ , and vice versa. We first show that for any function satisfying the relaxed Lipschitz condition, we can change the function values slightly and obtain a Lipschitz continuous function. In Lemma 4, we then apply this result specifically to functions in  $Q_\delta$ . Finally, in Lemma 5 (whose proof is deferred to the full version, due to space constraints), we show the converse approximation direction.

**Lemma 3.** *For each  $x \in \mathcal{M}$ , let  $s_x$  be some non-negative number. Assume that  $f$  satisfies the “relaxed Lipschitz condition”  $|f(x) - f(y)| \leq d(x, y) + s_x + s_y$  for all  $x, y$ . Then, there is a Lipschitz continuous function  $f' \in L$  such that  $|f(x) - f'(x)| \leq s_x$  for all  $x$ .*

**Proof.** We describe an algorithm which runs in iterations  $\ell$ , and sets the value of one point  $x$  per iteration.  $S_\ell$  denotes the set of  $x$  such that  $f'(x)$  has been set. We maintain the following two invariants after the  $\ell^{\text{th}}$  iteration: 1)  $f'$  satisfies the Lipschitz condition for all pairs of points in  $S_\ell$ , and  $|f'(x) - f(x)| \leq s_x$  for all  $x \in S_\ell$ , and 2) For every function  $f''$  satisfying the previous condition,  $f'(x) \leq f''(x)$  for all  $x \in S_\ell$ .

Initially, this clearly holds for  $S_0 = \emptyset$ . And clearly, if it holds after iteration  $n$ , the function  $f'$  satisfies the claim of the lemma.

In iteration  $\ell$ , for each  $x \notin S_{\ell-1}$ , let  $t_x = \max_{y \in S_{\ell-1}} (f'(y) - d(x, y))$ . We show below that for all  $x$ , we have  $t_x \leq f(x) + s_x$ . Let  $x \notin S_{\ell-1}$  be a point maximizing  $\max(f(x) - s_x, t_x)$ , and set  $f'(x) = \max(f(x) - s_x, t_x)$ . It is easy to verify that this definition satisfies both parts of the invariant.

It remains to show that  $t_x \leq f(x) + s_x$  for all points  $x \notin S_{\ell-1}$ . Assume that  $t_x > f(x) + s_x$  for some point  $x$ . Let  $x_1$  be the point in  $S_{\ell-1}$  for which  $t_x = f'(x_1) - d(x, x_1)$ . By definition,  $f'(x_1) = f(x_1) - s_{x_1}$  or there is an  $x_2$  such that  $f'(x_1) = t_{x_1} = f'(x_2) - d(x_1, x_2)$ . Thus, we obtain a chain  $x_1, \dots, x_r$  with  $f'(x_i) = f'(x_{i+1}) - d(x_i, x_{i+1})$  for all  $i < r$ , and  $f'(x_r) = f(x_r) - s_{x_r}$ . Rearranging as  $f'(x_{i+1}) - f'(x_i) = d(x_i, x_{i+1})$ , and adding all these equalities for  $i = 1, \dots, r$  gives us that  $f(x_r) - f'(x_1) = s_{x_r} + \sum_{i=1}^{r-1} d(x_i, x_{i+1})$ . By assumption, we have  $f'(x_1) - d(x, x_1) = t_x > f(x) + s_x$ . Substituting the previous equality,

rearranging, and applying the triangle inequality gives us that  $f(x_r) - f(x_1) > s_x + s_{x_r} + d(x, x_1) + \sum_{i=1}^{r-1} d(x_i, x_{i+1}) \geq s_x + s_{x_r} + d(x, x_r)$ , which contradicts the relaxed Lipschitz condition for the pair  $x_1, x_r$ . ■

**Lemma 4.** *Let  $f \in Q_\delta$ . There exists an  $f' \in L$  such that for all distributions  $\mathbf{p}$ , we have  $|\Delta(f, \mathbf{p}) - \Delta(f', \mathbf{p})| \leq \frac{\delta}{2} \cdot E_L(\mathbf{p})$ .*

**Proof.** Because  $f$  is in  $Q_\delta$ , it must satisfy the relaxed Lipschitz condition  $|f(o_{i,j}) - f(o_{i',j'})| \leq d(o_{i,j}, o_{i',j'}) + \gamma \cdot (2^{-i} + 2^{-i'})$  for all  $(i, j), (i', j')$ . Thus, applying Lemma 3 with  $s_{o_{i,j}} = \gamma \cdot 2^{-i}$  gives us function values  $f'(o_{i,j})$  for all  $i, j$ , satisfying the Lipschitz condition, as well as  $f'(o_{i,j}) - f(o_{i,j}) \leq \gamma \cdot 2^{-i}$ . For any other point  $x$ , let  $L_{\max}(x, f) = \min_{i,j} (f'(o_{i,j}) + d(x, o_{i,j}))$  and  $L_{\min}(x, f) = \max_{i,j} (f'(o_{i,j}) - d(x, o_{i,j}))$ , and set  $f'(x) = \frac{1}{2} \cdot (L_{\max}(x, f) + L_{\min}(x, f))$ . It is easy to see that  $L_{\min}(x, f) \leq L_{\max}(x, f)$  for all  $x$ , and that this definition gives a Lipschitz continuous function  $f'$ . For a point  $x \in B_{i,j}$ , triangle inequality, the above construction, and the fact that  $B_{i,j}$  has diameter  $2\gamma \cdot 2^{-i}$  imply that  $|f'(x) - f(x)| \leq |f'(x) - f'(o_{i,j})| + |f'(o_{i,j}) - f(o_{i,j})| + |f(o_{i,j}) - f(x)| \leq 2\gamma \cdot 2^{-i} + \gamma \cdot 2^{-i} + 0 = 3\gamma \cdot 2^{-i}$ .

For each point  $x$ , let  $\iota(x)$  be the index of the region  $i$  such that  $x \in R_i$ . Now, using the triangle inequality and Lemma 1, we can bound  $|\overline{f'} - \overline{f}| \leq \frac{1}{n} \cdot \sum_x |f'(x) - f(x)| \leq \frac{1}{n} \cdot \sum_x 3\gamma \cdot 2^{-\iota(x)} \leq \frac{1}{n} \cdot (\sum_{x \notin R_{k+1}} 3\gamma \cdot d(x, o) + \sum_{x \in R_{k+1}} 3\gamma \cdot m) \leq \frac{1}{n} \cdot (3\gamma nm + 3\gamma nm) \leq 24 \cdot 6^\beta \cdot \gamma \cdot E_L(\mathbf{p})$ .

Similarly, we can bound  $\sum_x p_x \cdot |f'(x) - f(x)| \leq 3\gamma \cdot (\sum_{x \notin R_{k+1}} p_x \cdot d(x, o) + \sum_{x \in R_{k+1}} p_x m) \leq 3\gamma \cdot (m + \sum_{x \notin R_{k+1}} p_x \cdot d(x, o))$ .

Let  $f''$  be defined as  $f''(x) = d(x, o)$ . Clearly,  $f'' \in L$ ,  $\overline{f''} = m$ , and the estimation error for  $\mathbf{p}$  when the input is  $f''$  is  $\Delta(f'', \mathbf{p}) = \sum_x p_x \cdot |f''(x) - m| \geq \sum_{x \notin R_{k+1}} p_x \cdot |d(x, o) - m| \geq (\sum_{x \notin R_{k+1}} p_x \cdot d(x, o)) - m$ .

Combining these observations, and using Lemma 1 and the fact that  $\Delta(f'', \mathbf{p}) \leq E_L(\mathbf{p})$ , we get  $\sum_x p_x \cdot |f'(x) - f(x)| \leq 6\gamma \cdot m + 3\gamma \Delta(f'', \mathbf{p}) \leq (8 \cdot 6^\beta + 1) \cdot 3\gamma \cdot E_L(\mathbf{p})$ .

Now, by using that  $|\Delta(f, \mathbf{p}) - \Delta(f', \mathbf{p})| \leq |\overline{f'} - \overline{f}| + \sum_x p_x \cdot |f'(x) - f(x)|$ , and setting  $\gamma = \frac{\delta}{48 \cdot 6^\beta + 6}$ , we obtain the desired bound. ■

**Lemma 5.** *Let  $f \in L$ . There exists an  $f' \in Q_\delta$  such that for all distributions  $\mathbf{p}$ , we have  $|\Delta(f, \mathbf{p}) - \Delta(f', \mathbf{p})| \leq \frac{\delta}{2} \cdot E_L(\mathbf{p})$ .*

If the metric consists of a discrete point set on the line, then the PTAS can be improved to an FPTAS, as discussed in the full version of the paper.

## 4 Sampling in the Interval $[0, 1]$

In this section, we focus on what is probably the most basic version of the problem: the metric space is the interval  $[0, 1]$ . It is easy to see (and follows from a more general result in [4]) that the best deterministic algorithm samples the function at  $\frac{1}{2}$  and outputs the value read. The worst-case error of this algorithm is  $\frac{1}{4}$ . We prove that randomization can lead to the following improvement.

**Theorem 4.** An optimal distribution that minimizes the worst-case expected estimation error is to sample uniformly from the interval  $[2 - \sqrt{3}, \sqrt{3} - 1]$ . This sampling gives a worst-case error of  $1 - \frac{\sqrt{3}}{2} \approx 0.134$ .

In this section, we restrict our analysis w.l.o.g. to functions  $f \in L_0$ , i.e., we assume that  $\int_0^1 f(x)dx = 0$ . Then, the expected error of a distribution  $\mathbf{p}$  against input  $f$  is  $\Delta(f, \mathbf{p}) = \int_0^1 p_x |f(x)| dx$ . We say that  $f$  is a *worst-case function* for  $\mathbf{p}$  if it maximizes  $\Delta(f, \mathbf{p})$ ; because  $L_0$  is compact, this notion is well-defined.

The key part of the proof of Theorem 4 is to characterize worst-case functions for distributions  $\mathbf{p}$  that are uniform over an interval  $[c, 1 - c]$  for some  $c \leq \frac{1}{2}$ .

**Theorem 5.** If  $\mathbf{p}$  is uniform over  $[c, 1 - c]$ , then there exists a worst-case function for  $\mathbf{p}$  of the form  $f(x) = \frac{1}{2} + b^2 - b - |b - x|$ , for a parameter  $b$ .

All of Section 4.1 is devoted to the proof of Theorem 5. Here, we show how to use Theorem 5 to prove the upper bound from Theorem 4.

Let  $c = 2 - \sqrt{3}$ , so that the algorithm samples uniformly from  $[c, 1 - c]$ . Using Theorem 5, there exists a worst-case function for this distribution of the form  $f(x) = \frac{1}{2} + b^2 - b - |b - x|$ . We distinguish two cases:

1. If  $b \leq c$ , then  $\Delta(f, \mathbf{p}) = \frac{1}{1-2c} \cdot \int_c^{1-c} |\frac{1}{2} + b^2 - b - |b - x|| dx = \frac{1}{1-2c} \cdot (\frac{1}{2}(b^2 + \frac{1}{2} - c)^2 + \frac{1}{2}(1 - c - b^2)^2) = \frac{1}{1-2c} \cdot (b^4 + (\frac{1}{2} - c)^2)$ .
2. If  $b \geq c$ , then  $\Delta(f, \mathbf{p}) = \frac{1}{1-2c} \cdot \int_c^{1-c} |\frac{1}{2} + b^2 - b - |b - x|| dx = \frac{1}{1-2c} \cdot (2bf(b) + 2cb + f(b)^2 - c - b^2) = \frac{1}{1-2c} (b^4 - b^2 + 2cb + \frac{1}{4} - c) = \frac{1}{1-2c} (b^4 - (b - c)^2 + (\frac{1}{2} - c)^2)$ .

The first formula is increasing in  $b$ , and thus maximized at  $b = c$ ; at  $b = c$ , the value equals that of the second formula, so the maximization must happen for  $b \geq c$ . A derivative test shows that it is maximized for  $b = \frac{\sqrt{3}-1}{2}$ , giving an error of  $1 - \frac{\sqrt{3}}{2}$ .

Next, we prove optimality of the uniform distribution over  $[2 - \sqrt{3}, \sqrt{3} - 1]$ , by providing a lower bound on all randomized sampling distributions. Again, by Theorem 2, we focus only on algorithms which output the value  $f(x)$  after sampling at  $x$ , by incurring an error  $\epsilon > 0$  that can be made arbitrarily small. Our proof is based on Yao's Minimax principle: we explicitly prescribe a distribution  $q$  over  $L_0$  such that for any deterministic algorithm using the identity function, the expected estimation error is at least  $1 - \frac{\sqrt{3}}{2}$ . Since a deterministic algorithm is characterized completely by its sampling location  $x$ , this is equivalent to showing that  $E_{f \sim q} [|f(x)|] \geq 1 - \frac{\sqrt{3}}{2}$  for all  $x$ .

We let  $b = \frac{\sqrt{3}-1}{2}$ , and define two functions  $f, f'$  as  $f(x) = \frac{1}{2} + b^2 - b - |x - b|$  and  $f'(x) = f(1 - x)$ . The distribution  $q$  is then simply to choose each of  $f$  and  $f'$  with probability  $\frac{1}{2}$ . Fix a sampling location  $x$ ; by symmetry, we can restrict ourselves to  $x \leq \frac{1}{2}$ . Because  $\bar{f} = \bar{f}' = 0$ , the expected estimation error is  $\frac{1}{2}(|f(x)| + |f'(x)|) = \frac{1}{2}(|\frac{1}{2} + b^2 - b - |x - b|| + |\frac{1}{2} + b^2 - b - |1 - x - b||) = \begin{cases} \frac{1}{2} - b, & \text{if } x \leq b \\ \frac{1}{2} - x, & \text{if } b \leq x \leq \frac{1}{2} - b^2 \\ b^2, & \text{if } \frac{1}{2} - b^2 \leq x \leq \frac{1}{2}. \end{cases}$

This function is non-increasing in  $x$ , and thus minimized at  $x = \frac{1}{2}$ , where its value is  $b^2 = 1 - \frac{\sqrt{3}}{2}$ . Thus, even at the best sampling location  $x = \frac{1}{2}$ , the error cannot be less than  $1 - \frac{\sqrt{3}}{2}$ . This completes the proof of Theorem 4. ■

The proof of Theorem 4 has an interesting alternative interpretation. For a (finite) multiset  $S \subset L_0$  of Lipschitz continuous functions  $f$  with  $\int_x f(x)dx = 0$ , we say that  $S$  is  $\delta$ -close if there exist  $x, y$  such that  $\frac{1}{n} \cdot \sum_{f \in S} |f(x) - y| \leq \delta$ . In other words, the average distance of the functions from a carefully chosen reference point is at most  $\delta$ . Then, the proof of Theorem 4 implies:

**Theorem 6.** *Every set  $S \subseteq L_0$  is  $(1 - \frac{\sqrt{3}}{2})$ -close, and this is tight.*

#### 4.1 Characterization of Worst-Case Functions

We begin with the following lemma which guarantees that there exists a worst case function  $f$  with a finite number of points  $x$  such that  $f(x) = 0$ .

**Lemma 6.** *W.l.o.g., there are a finite number of points  $x$  such that  $f(x) = 0$ .*

We focus on points  $x \in (c, 1 - c)$  with  $f(x) = 0$ . Let  $c \leq z_1 \leq \dots \leq z_k \leq 1 - c$  be all such points. For ease of notation, we write  $z_0 = c$  and  $z_{k+1} = 1 - c$ . By continuity,  $f(x)$  has the same sign for all  $x \in (z_i, z_{i+1})$ , for  $i = 0, \dots, k$ . Next, we show that w.l.o.g.,  $f$  is as large as possible over areas of the same sign.

**Lemma 7.** *Assume w.l.o.g. that  $f(x) \geq 0$  for all  $x \in [z_i, z_j]$ , with  $j > i$ . Then, w.l.o.g.,  $f$  maximizes the area over  $[z_i, z_j]$  subject to the Lipschitz constraint and the function values at  $z_i$  and  $z_j$ . More formally, w.l.o.g.,  $f$  satisfies,*

1. *If  $1 \leq i < j \leq k$ , then  $f(x) = \min(x - z_i, z_j - x)$  for all  $x \in [z_i, z_j]$ .*
2. *If  $i = 0$ , then  $f(x) = \min(f(c) + (x - c), z_1 - x)$  for all  $x \in [c, z_1]$ , and if  $i = k$ , then  $f(x) = \min(f(1 - c) + (1 - c) - x, x - z_k)$  for all  $x \in [z_k, 1 - c]$ .*

**Proof.** We prove the first part here; the second is analogous and proved in the full version. Define a function  $f'$  as  $f'(x) = \min(x - z_i, z_j - x)$  for  $x \in [z_i, z_j]$ , and  $f'(x) = f(x)$  otherwise. Let  $f'' = f' - \bar{f}'$ , so that  $f''$  is renormalized to have integral 0. Since  $f'(x) \geq f(x)$  for all  $x$ , and  $\bar{f}' = 0$ , we have that  $\bar{f}' \geq 0$ . Then

$$\begin{aligned} & \int_c^{1-c} |f''(x)| - |f(x)| dx \\ &= \int_{z_i}^{z_j} |f''(x)| - |f(x)| dx + \int_c^{z_i} |f(x) - \bar{f}'| - |f(x)| dx + \int_{z_j}^{1-c} |f(x) - \bar{f}'| - |f(x)| dx \\ &\geq \int_{z_i}^{z_j} (|f'(x) - \bar{f}'| - |f'(x)|) + (|f'(x)| - |f(x)|) dx - (1 - 2c - (z_j - z_i))\bar{f}' \\ &\geq \int_{z_i}^{z_j} |f'(x)| - |f(x)| dx - \int_{z_i}^{z_j} \bar{f}' dx - (1 - 2c - (z_j - z_i))\bar{f}' \\ &= \int_{z_i}^{z_j} f'(x) - f(x) dx - (1 - 2c)\bar{f}' = 2c \cdot \bar{f}' \geq 0. \end{aligned}$$

Thus, the estimation error of  $f''$  is at least as large as the one for  $f$ , so w.l.o.g.,  $f$  satisfies the statement of the lemma. ■

**Lemma 8.** *W.l.o.g., there are at most two points  $x \in (c, 1 - c)$  where  $f(x) = 0$ .*

**Proof.** Assume that  $f(z_1) = f(z_2) = f(z_3) = 0$ . Mirror the function on the interval  $[z_1, z_3]$ , i.e., define  $f'(x) = f(z_3 - x)$  if  $x \in [z_1, z_3]$ , and  $f'(x) = f(x)$

otherwise. Clearly,  $f'$  is Lipschitz continuous and has the same average and same expected estimation error as  $f$ . However, the signs of  $f'$  on the intervals  $[c, z_1]$  and  $[z_1, z_1 + z_3 - z_2]$  are now the same; similarly for the intervals  $[z_1 + z_3 - z_2, z_3]$  and  $[z_3, 1 - c]$ . Thus, applying Lemma 7, we can further reduce the number of  $x$  with  $f(x) = 0$ , without decreasing the estimation error. ■

Hence, the worst-case function  $f$  must have at most two points  $z \in (c, 1 - c)$  with  $f(z) = 0$ . We distinguish three cases accordingly:

1. If there is no point  $z \in (c, 1 - c)$  with  $f(z) = 0$ , then  $f(c)$  and  $f(1 - c)$  have the same signs. Then, the expected error is maximized when  $\int_0^c f(x)dx$  and  $\int_{1-c}^1 f(x)dx$  are as positive as possible, subject to the Lipschitz condition and the constraint that  $\int_0^1 f(x)dx = 0$ . Otherwise, we could increase the value of  $\int_0^c f(x)dx$  and  $\int_{1-c}^1 f(x)dx$ , and then lower the function to restore the integral to 0. By doing this, the expected estimation error cannot decrease. Thus, by Lemma 7,  $f$  is of the form  $f(x) = |x - b| + f(b)$ , where  $b = \operatorname{argmin}_{x \in (c, 1-c)} f(x)$ .
2. If there is exactly one point  $z \in (c, 1 - c)$  with  $f(z) = 0$ , then  $f(c)$  and  $f(1 - c)$  have opposite signs. W.l.o.g., assume that  $f(c) > 0 > f(1 - c)$  and that  $z \leq \frac{1}{2}$  (otherwise, we consider  $f'(x) = f(1 - x)$  instead). The expected error is maximized when  $f(c)$  is as large as possible, and  $\int_z^{1-c} f(x)dx$  is as negative as possible, subject to the Lipschitz condition and the constraint that  $\int_0^1 f(x)dx = 0$ . Since  $z \leq \frac{1}{2}$  and the integral of the function  $f'(x) = z - x$  is thus negative, by starting from  $f'$ , then raising the function in the interval  $[1 - c, 1]$  and, if necessary, increasing  $f'(1 - c)$ , it is always possible to ensure that  $f(x) = z - x$  for all  $x \in [0, z]$ . Then,  $\int_z^{1-c} f(x)dx$  is as negative as possible if  $f(x) = -(x - z)$  for  $x \leq b$  (for some value  $b$ ), and  $f(x) = -(b - z) + (x - b) = z + x - 2b$  for  $x \geq b$ . Thus,  $f$  overall is of the form  $f(x) = |x - b| - (b - z)$ .
3. If there are two points  $z_1 < z_2 \in (c, 1 - c)$  with  $f(z_1) = f(z_2) = 0$ , then again, it can be shown that w.l.o.g  $f(x) = |x - \frac{z_1+z_2}{2}| - \frac{z_2-z_1}{2}$ . Due to space constraints, the formal proof is deferred to the full version of the paper.

In all three cases, we have thus shown that w.l.o.g.,  $f(x) = |x - b| - t$ , for some values  $b, t$ . Finally, the normalization  $\int_0^1 f(x)dx = 0$  implies that  $t = \frac{1}{2} + b^2 - b$ , completing the proof of Theorem 5.

## 5 Future Work

Our work is a first step toward obtaining optimal (as opposed to asymptotically optimal) randomized algorithms for choosing  $k$  sample locations to estimate an aggregate quantity of a function  $f$ . The most obvious extension is to extend our results to the case of estimating the average using  $k$  samples. It would be interesting whether approximation guarantees for the  $k$ -median problem (the deterministic counterpart) can be exceeded using a randomized strategy.

Also, our precise characterization of the optimal sampling distribution for functions on the  $[0, 1]$  interval should be extended to higher-dimensional continuous metric spaces. Another natural direction is to consider other aggregation goals, such as predicting the function's maximum, minimum, or median. For predicting the maximum from  $k$  deterministic samples, a 2-approximation algorithm was given in [4], which is best possible unless  $P=NP$ . However, it is not clear if equally good approximations can be achieved for the randomized case. For the median, even the deterministic case is open.

On a technical note, it would be interesting whether finding the best sampling distribution for the single sample case is NP-hard. While we presented a PTAS in this paper, no hardness result is currently known.

**Acknowledgments.** We would like to thank David Eppstein, Bobby Kleinberg, Alex Slivkins and several anonymous referees for helpful feedback.

## References

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. In: Proc. ACM Symposium on Theory of Computing (2001)
2. Bakhvalov, N.S.: On approximate calculation of integrals. Vestnik MGU, Ser. Mat. Mekh. Astron. Fiz. Khim 4, 3–18 (1959)
3. Baran, I., Demaine, E., Katz, D.: Optimally adaptive integration of univariate lipschitz functions. Algorithmica 50(2), 255–278 (2008)
4. Das, A., Kempe, D.: Sensor selection for minimizing worst-case prediction error. In: Proc. ACM/IEEE International Conference on Information Processing in Sensor Networks (2008)
5. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1, 169–197 (1981)
6. Gupta, A., Krauthgamer, R., Lee, J.R.: Bounded geometries, fractals, and low-distortion embeddings. In: Proc. IEEE Symposium on Foundations of Computer Science (2003)
7. Mathe, P.: The optimal error of monte carlo integration. Journal of Complexity 11(4), 394–415 (1995)
8. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1990)
9. Novak, E.: Stochastic properties of quadrature formulas. Numer. Math. 53(5), 609–620 (1988)
10. Traub, J.F., Wasilkowski, G.W., Woźniakowski, H.: Information-Based Complexity. Academic Press, New York (1988)
11. Traub, J.F., Werschulz, A.G.: Complexity and Information. Cambridge University Press, Cambridge (1998)
12. Woźniakowski, H.: Average case complexity of linear multivariate problems part 1: Theory. Journal of Complexity 8(4), 337–372 (1992)
13. Woźniakowski, H.: Average case complexity of linear multivariate problems part 2: Applications. Journal of Complexity 8(4), 373–392 (1992)

# Streaming Graph Computations with a Helpful Advisor

Graham Cormode<sup>1</sup>, Michael Mitzenmacher<sup>2,\*</sup>, and Justin Thaler<sup>2,\*\*</sup>

<sup>1</sup> AT & T Labs – Research

[graham@research.att.com](mailto:graham@research.att.com)

<sup>2</sup> Harvard University, School of Engineering and Applied Sciences

[{michaelm,jthaler}@seas.harvard.edu](mailto:{michaelm,jthaler}@seas.harvard.edu)

**Abstract.** Motivated by the trend to outsource work to commercial cloud computing services, we consider a variation of the streaming paradigm where a streaming algorithm can be assisted by a powerful helper that can provide annotations to the data stream. We extend previous work on such *annotation models* by considering a number of graph streaming problems. Without annotations, streaming algorithms for graph problems generally require significant memory; we show that for many standard problems, including all graph problems that can be expressed with totally unimodular integer programming formulations, only constant memory is needed for single-pass algorithms given linear-sized annotations. We also obtain a protocol achieving *optimal* tradeoffs between annotation length and memory usage for matrix–vector multiplication; this result contributes to a trend of recent research on numerical linear algebra in streaming models.

## 1 Introduction

The recent explosion in the number and scale of real-world structured data sets including the web, social networks, and other relational data has created a pressing need to efficiently process and analyze massive graphs. This has sparked the study of graph algorithms that meet the constraints of the standard streaming model: restricted memory and the ability to make only one pass (or few passes) over adversarially ordered data. However, many results for graph streams have been negative, as many foundational problems require either substantial working memory or a prohibitive number of passes over the data [1]. Apparently most graph algorithms fundamentally require flexibility in the way they query edges, and therefore the combination of adversarial order and limited memory makes many problems intractable.

To circumvent these negative results, variants and relaxations of the standard graph streaming model have been proposed, including the Semi-Streaming [2],

---

\* This work was supported in part by NSF grants CCF-0915922 and CNS-0721491, and in part by grants from Yahoo! Research, Google, and Cisco, Inc.

\*\* Supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

W-Stream [3], Sort-Stream [4], Random-Order [1], and Best-Order [5] models. In Semi-Streaming, memory requirements are relaxed, allowing space proportional to the number of vertices in the stream but not the number of edges. The W-Stream model allows the algorithm to write temporary streams to aid in computation. And, as their names suggest, the Sort-Stream, Random-Order, and Best-Order models relax the assumption of adversarially ordered input. The Best-Order model, for example, allows the input stream to be re-ordered arbitrarily to minimize the space required for the computation.

In this paper, our starting point is a relaxation of the standard model, closest to that put forth by Chakrabarti *et al.* [6], called the *annotation model*. Motivated by recent work on outsourcing of database processing, as well as commercial cloud computing services such as Amazon EC2, the annotation model allows access to a powerful advisor, or *helper* who observes the stream concurrently with the algorithm. Importantly, in many of our motivating applications, the helper is not a trusted entity: the commercial stream processing service may have executed a buggy algorithm, experienced a hardware fault or communication error, or may even be deliberately deceptive [5, 6]. As a result, we require our protocols to be *sound*: our *verifier* must detect any lies or deviations from the prescribed protocol with high probability.

The most general form of the annotation model allows the helper to provide additional annotations in the data stream *at any point* to assist the verifier, and one of the cost measures is the total length of the annotation. In this paper, however, we focus on the case where the helper’s annotation arrives as a single message after both the helper and verifier have seen the stream. The helper’s message is also processed as a stream, since it may be large; it often (but not always) includes a re-ordering of the stream into a convenient form, as well as additional information to guide the verifier. This is therefore stronger than the Best-Order model, which only allows the input to be reordered and no more; but it is weaker than the more general *online* model, because in our model the annotation appears only after the input stream has finished.

We argue that this model is of interest for several reasons. First, it requires minimal coordination between helper and verifier, since it is not necessary to ensure that annotation and stream data are synchronized. Second, it captures the case when the verifier uploads data to the cloud as it is collected, and later poses questions over the data to the helper. Under this paradigm, the annotation must come *after* the stream is observed. Third, we know of no non-trivial problems which separate the general online and our “at-the-end” versions of the model, and most prior results are effectively in this model.

Besides being practically motivated by outsourced computations, annotation models are closely related to Merlin-Arthur proofs with space-bounded verifiers, and studying what can (and cannot) be accomplished in these models is of independent interest.

**Relationship to Other Work.** Annotation models were first explicitly studied by Chakrabarti *et al.* in [6], and focused primarily on protocols for canonical problems in numerical streams, such as Selection, Frequency Moments, and

Frequent Items. The authors also provided protocols for some graph problems: counting triangles, connectedness, and bipartite perfect matching. The Best-Order Stream Model was put forth by Das Sarma et al. in [5]. They present protocols requiring logarithmic or polylogarithmic space (in bits) for several problems, including perfect matching and connectivity. Historical antecedents for this work are due to Lipton [7], who used fingerprinting methods to verify polynomial-time computations in logarithmic space. Recent work verifies shortest-path computations using cryptographic primitives, using polynomial space for the verifier [8].

**Our Contributions.** We identify two qualitatively different approaches to producing protocols for problems on graphs with  $n$  nodes and  $m$  edges. In the first, the helper directly proves matching upper and lower bounds on a quantity. Usually, proving one of the two bounds is trivial: the helper provides a feasible solution to the problem. But proving *optimality* of the feasible solution can be more difficult, requiring the use of structural properties of the problem. In the second, we simulate the execution of a non-streaming algorithm, using the helper to maintain the algorithm’s internal data structures to control the amount of memory used by the verifier. The helper must provide the contents of the data structures so as to limit the amount of annotation required.

Using the first approach (Section 3), we show that only constant space and annotation linear in the input size  $m$  is needed to determine whether a directed graph is a DAG and to compute the size of a maximum matching. We describe this as an  $(m, 1)$  protocol, where the first entry refers to the annotation size (which we also call the hcost) and the second to the memory required for the verifier (which we also call the vcost). Our maximum matching result significantly extends the bipartite perfect matching protocol of [6], and is tight for dense graphs, in the sense that there is a lower bound on the *product* of hcost and vcost of  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits for this problem. Second, we define a streaming version of the linear programming problem, and provide an  $(m, 1)$  protocol. By exploiting duality, we hence obtain  $(m, 1)$  protocols for many graph problems with totally unimodular integer programming formulations, including shortest  $s$ - $t$  path, max-flow, min-cut, and minimum-weight bipartite perfect matching. We also show all are tight by proving lower bounds of  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits for all four problems. A more involved protocol obtains *optimal* tradeoffs between annotation cost and working memory for dense LPs and matrix-vector multiplication; this complements recent results on approximate linear algebra in streaming models (see e.g. [9][10]).

For the second approach (Section 4), we make use of the idea of “memory checking” due to Blum et al. [11], which allows a small-space verifier to outsource data storage to an untrusted server. We present a general simulation theorem based on this checker, and obtain as corollaries tight protocols for a variety of canonical graph problems. In particular, we give an  $(m, 1)$  protocol for verifying a minimum spanning tree, an  $(m + n \log n, 1)$  protocol for single-source shortest paths, and an  $(n^3, 1)$  protocol for all-pairs shortest paths. We provide a lower bound of  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits for the latter two problems, and an identical lower bound for MST when the edge weights can be given incrementally. While

powerful, this technique has its limitations: there does not seem to be any generic way to obtain the same kind of tradeoffs observed above. Further, there are some instances where direct application of memory checking does not achieve the best bounds for a problem. We demonstrate this by presenting an  $(n^2 \log n, 1)$  protocol to find the diameter of a graph; this protocol leverages the ability to use randomized methods to check computations more efficiently than via generating or checking a deterministic witness. In this case, we rely on techniques to verify matrix-multiplication in quadratic time, and show that this is tight via a nearly matching lower bound for diameter of  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ .

In contrast to problems on numerical streams, where it is often trivial to obtain  $(m, 1)$  protocols by replaying the stream in sorted order, achieving linear-sized annotations with logarithmic space is challenging for many graph problems. Simply providing the solution (e.g. a matching or spanning tree) is insufficient, since we have the additional burden of demonstrating that this solution is *optimal*. A consequence is that we are able to provide solutions to several problems for which no solution is known in the best-order model (even though one can reorder the stream in the best-order model so that “solution” edges arrive first).

All omitted proofs may be found online at [\[2\]](#).

## 2 Model and Definitions

Consider a data stream  $\mathcal{A} = \langle a_1, a_2, \dots, a_m \rangle$  with each  $a_i$  in some universe  $\mathcal{U}$ . Consider a probabilistic *verifier*  $\mathcal{V}$  who observes  $\mathcal{A}$  and a deterministic *helper*  $\mathcal{H}$  who also observes  $\mathcal{A}$  and can send a message  $h$  to  $\mathcal{V}$  after  $\mathcal{A}$  has been observed by both parties. This message, also referred to as an *annotation*, should itself be interpreted as a data stream that is parsed by  $\mathcal{V}$ , which may permit  $\mathcal{V}$  to use space sublinear in the size of the annotation itself. That is,  $\mathcal{H}$  provides an annotation  $h(\mathcal{A}) = (h_1(\mathcal{A}), h_2(\mathcal{A}), \dots, h_\ell(\mathcal{A}))$ .

We study randomized streaming protocols for computing functions  $f(\mathcal{A}) \rightarrow \mathbb{Z}$ . Specifically, assume  $\mathcal{V}$  has access to a private random string  $\mathcal{R}$  and at most  $w(m)$  machine words of working memory, and that  $\mathcal{V}$  has one-way access to the input  $\mathcal{A} \cdot h$ , where  $\cdot$  represents concatenation. Denote the output of protocol  $\mathcal{P}$  on input  $\mathcal{A}$ , given helper  $h$  and random string  $\mathcal{R}$ , by  $\text{out}(\mathcal{P}, \mathcal{A}, \mathcal{R}, h)$ . We allow  $\mathcal{V}$  to output  $\perp$  if  $\mathcal{V}$  is not convinced that the annotation is valid. We say that  $h$  is *valid* for  $\mathcal{A}$  with respect to  $\mathcal{P}$  if  $\Pr_{\mathcal{R}}(\text{out}(\mathcal{P}, \mathcal{A}, \mathcal{R}, h) = f(\mathcal{A})) = 1$ , and we say that  $h$  is  $\delta$ -*invalid* for  $\mathcal{A}$  with respect to  $\mathcal{P}$  if  $\Pr_{\mathcal{R}}(\text{out}(\mathcal{P}, \mathcal{A}, \mathcal{R}, h) \neq \perp) \leq \delta$ . We say that  $h$  is a *valid helper* if  $h$  is valid for all  $\mathcal{A}$ . We say that  $\mathcal{P}$  is a valid protocol for  $f$  if

1. There exists at least one valid helper  $h$  with respect to  $\mathcal{P}$  and
2. For all helpers  $h'$  and all streams  $\mathcal{A}$ , either  $h'$  is valid for  $\mathcal{A}$  or  $h'$  is  $\frac{1}{3}$ -invalid for  $\mathcal{A}$ .

Conceptually,  $\mathcal{P}$  is a valid protocol for  $f$  if for each stream  $\mathcal{A}$  there is *at least* one way to convince  $\mathcal{V}$  of the true value of  $f(\mathcal{A})$ , and  $\mathcal{V}$  rejects all other annotations as invalid (this differs slightly from [6] to allow for multiple  $h$ 's that can convince  $\mathcal{V}$ ). The constant  $\frac{1}{3}$  can be any constant less than  $\frac{1}{2}$ .

Let  $h$  be a valid helper chosen to minimize the length of  $h(\mathcal{A})$  for all  $\mathcal{A}$ . We define the help cost  $\text{hcost}(\mathcal{P})$  to be the maximum length of  $h$  over all  $\mathcal{A}$  of length  $m$ , and the verification cost  $\text{vcost}(P) = w(m)$ , the amount of working memory used by the protocol  $P$ . All costs are expressed in machine words of size  $\Theta(\log m)$  bits, i.e. we assume any quantity polynomial in the input size can be stored in a constant number of words; in contrast, lower bounds are expressed in bits. We say that  $\mathcal{P}$  is an  $(h, v)$  protocol for  $f$  if  $\mathcal{P}$  is valid and  $\text{hcost}(\mathcal{A}) = O(h + 1)$ ,  $\text{vcost}(\mathcal{A}) = O(v + 1)$ . While both  $\text{hcost}$  and  $\text{vcost}$  are natural costs for such protocols, we often aim to achieve a  $\text{vcost}$  of  $O(1)$  and then minimize  $\text{hcost}$ . In other cases, we show that  $\text{hcost}$  can be decreased by increasing  $\text{vcost}$ , and study the tradeoff between these two quantities.

In some cases,  $f$  is not a function of  $\mathcal{A}$  alone; instead it depends on  $\mathcal{A}$  and  $h$ . In such cases,  $\mathcal{V}$  should simply *accept* if convinced that the annotation has the correct properties, and output  $\perp$  otherwise.

In this paper we primarily consider graph streams, which are streams whose elements are edges of a graph  $G$ . More formally, consider a stream  $\mathcal{A} = \langle e_1, e_2, \dots, e_m \rangle$  with each  $e_i \in [n] \times [n]$ . Such a stream defines a (multi)graph  $G = (V, E)$  where  $V = \{v_1, \dots, v_n\}$  and  $E$  is the (multi)set of edges that naturally corresponds to  $\mathcal{A}$ . We use the notation  $\{i : m(i)\}$  for the multiset in which  $i$  appears with multiplicity  $m(i)$ . Finally, we will sometimes consider graph streams with directed or weighted edges; in the latter case each edge  $e_i \in [n] \times [n] \times \mathbb{Z}_+$ .

## 2.1 Fingerprints

Our protocols make careful use of *fingerprints*, permutation-invariant hashes that can be efficiently computed in a streaming fashion. They determine in small space (with high probability) whether two streams have identical frequency distributions. They are the workhorse of algorithms proposed in earlier work on streaming models with an untrusted helper [5, 6, 7, 13]. We sometimes also need the fingerprint function to be linear.

**Definition 1 (Fingerprints).** A *fingerprint* of a multiset  $M = \{i : m(i)\}$ , where each  $i \in [q]$  for some known upper bound  $q$ , is defined as a computation over the finite field with  $p$  elements,  $\mathbb{F}_p$ , as  $\mathfrak{f}_{p,\alpha}(M) = \sum_{i=1}^q m(i)\alpha^i$ , with  $\alpha$  chosen uniformly at random from  $\mathbb{F}_p$ . We typically write  $\mathfrak{f}(M)$ , leaving  $p, \alpha$  implicit.

Some properties of  $\mathfrak{f}$  are immediate: it is linear in  $M$ , and can easily be computed incrementally as elements of  $[q]$  are observed in a stream one by one. The main property of  $\mathfrak{f}$  is that  $\Pr[\mathfrak{f}(M) = \mathfrak{f}(M') | M \neq M'] \leq q/p$  over the random choice of  $\alpha$  (due to standard properties of polynomials over a field). Therefore, if  $p$  is sufficiently large, say, polynomial in  $q$  and in an (assumed) upper bound on the multiplicities  $m(i)$ , then this event happens with only polynomially small probability. For cases when the domain of the multisets is not  $[q]$ , we either establish a bijection to  $[q]$  for an appropriate value of  $q$ , or use a hash function to map the domain onto a large enough  $[q]$  such that there are no collisions with high probability (whp). In all cases,  $p$  is chosen to be  $O(1)$  words.

A common subroutine of many of our protocols forces  $\mathcal{H}$  to provide a “label”  $l(u)$  for each node upfront, and then replay the edges in  $E$ , with each edge  $(u, v)$  annotated with  $l(u)$  and  $l(v)$  so that each instance of each node  $v$  appears with the *same* label  $l(v)$ .

**Definition 2.** We say a list of edges  $E'$  is *label-augmented* if (a)  $E'$  is preceded by a sorted list of all nodes  $v \in V$ , each with a value  $l(v)$  and  $\deg(v)$ , where  $l(v)$  is the label of  $v$  and  $\deg(v)$  is claimed to be the degree of  $v$ ; and (b) each edge  $e=(u, v)$  in  $E'$  is annotated with a pair of symbols  $l(e, u)$  and  $l(e, v)$ . We say a list of label-augmented edges  $E'$  is *valid* if for all edges  $e=(u, v)$ ,  $l(e, u)=l(u)$  and  $l(e, v)=l(v)$ ; and  $E'=E$ , where  $E$  is the set of edges observed in the stream  $A$ .

**Lemma 1 (Consistent Labels).** There is a valid  $(m, 1)$  protocol that accepts any valid list of label-augmented edges.

*Proof.*  $\mathcal{V}$  uses the annotation from Definition 2 (a) to make a fingerprint of the multiset  $S_1 := \{(u, l(u)) : \deg(u)\}$ .  $\mathcal{V}$  also maintains a fingerprint  $f_1$  of all  $(u, l(e, u))$  pairs seen while observing the edges of  $L$ . If  $f_1 = f(S_1)$  then (whp) each node  $u$  must be presented with label  $l(e, u) = l(u)$  every time it is reported in an edge  $e$  (and moreover  $u$  must be reported in exactly  $\deg(u)$  edges), else the multiset of observed (node, label) pairs would not match  $S_1$ . Finally,  $\mathcal{V}$  ensures that  $E' = E$  by checking that  $f(E) = f(E')$ .  $\square$

### 3 Directly Proving Matching Upper and Lower Bounds

#### 3.1 Warmup: Topological Ordering and DAGs

A (directed) graph  $G$  is a DAG if and only if  $G$  has a *topological ordering*, which is an ordering of  $V$  as  $v_1, \dots, v_n$  such that for every edge  $(v_i, v_j)$  we have  $i < j$  [14, Section 3.6]. Hence, if  $G$  is a DAG,  $\mathcal{H}$  can prove it by providing a topological ordering. If  $G$  is not a DAG,  $\mathcal{H}$  can provide a directed cycle as witness.

**Theorem 1.** There is a valid  $(m, 1)$  protocol to determine if a graph is a DAG.

*Proof.* If  $G$  is not a DAG,  $\mathcal{H}$  provides a directed cycle  $C$  as  $(v_1, v_2), (v_2, v_3) \dots (v_k, v_1)$ . To ensure  $C \subseteq E$ ,  $\mathcal{H}$  then provides  $E \setminus C$ , allowing  $\mathcal{V}$  to check that  $f(C \cup (E \setminus C)) = f(E)$ .

If  $G$  is a DAG, let  $v_1, \dots, v_n$  be a topological ordering of  $G$ . We require  $\mathcal{H}$  to replay the edges of  $G$ , with edge  $(v_i, v_j)$  annotated with the ranks of  $v_i$  and  $v_j$  i.e.  $i$  and  $j$ . We ensure  $\mathcal{H}$  provides consistent ranks via the Consistent Labels protocol of Lemma 1, with the ranks as “labels”. If any edge  $(v_i, v_j)$  is presented with  $j > i$ ,  $\mathcal{V}$  rejects immediately.  $\square$

#### 3.2 Maximum Matching

We give an  $(m, 1)$  protocol for maximum matching which leverages the combinatorial structure of the problem. Previously, matching was only studied in the bipartite case, where an  $(m, 1)$  protocol and a lower bound of  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$

bits for dense graphs were shown [6, Theorem 11]. The same lower bound applies to the more general problem of maximum matching, so our protocol is tight up to logarithmic factors.

The protocol shows matching upper and lower bounds on the size of the maximum matching. Any feasible matching presents a lower bound. For the upper bound we appeal to the Tutte-Berge formula [15, Chapter 24]: the size of a maximum matching of a graph  $G = (V, E)$  is equal to  $\frac{1}{2} \min_{V_S \subseteq V} (|V_S| - \text{occ}(G - V_S) + |V|)$ , where  $G - V_S$  is the subgraph of  $G$  obtained by deleting the vertices of  $V_S$  and all edges incident to them, and  $\text{occ}(G - V_S)$  is the number of components in the graph  $G - V_S$  that have an odd number of vertices. So for any set of nodes  $V_S$ ,  $\frac{1}{2}(|V_S| - \text{occ}(G - V_S) + |V|)$  is an upper bound on the size of the maximum matching, and there exists some  $V_S$  for which this quantity equals the size of a maximum matching  $M$ . Conceptually, providing both  $V_S$  and  $M$ ,  $\mathcal{H}$  proves that the maximum matching size is  $M$ . Additionally,  $\mathcal{H}$  has to provide a proof of the value of  $\text{occ}(G - V_S)$  to  $\mathcal{V}$ . We omit the proof.

**Theorem 2.** *There is a valid  $(m, 1)$  protocol for maximum matching. Moreover, any protocol for max-matching requires  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits.*

### 3.3 Linear Programming and TUM Integer Programs

We present protocols to solve linear programming problems in our model leveraging the theory of LP duality. This leads to non-trivial schemes for a variety of graph problems.

**Definition 3.** *Given a data stream  $\mathcal{A}$  containing entries of vectors  $\mathbf{b} \in \mathbb{R}^b$ ,  $\mathbf{c} \in \mathbb{R}^c$ , and non-zero entries of a  $b \times c$  matrix  $A$  in some arbitrary order, possibly interleaved. Each item in the stream indicates the index of the object it pertains to. The LP streaming problem on  $\mathcal{A}$  is to determine the value of the linear program  $\min\{\mathbf{c}^T \mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ .*

We present our protocol as if each entry of each object appears at most once (if an entry does not appear, it is assumed to be zero). When this is not the case, the final value for that entry is interpreted as the *sum* of all corresponding values in the stream.

**Theorem 3.** *There is a valid  $(|A|, 1)$  protocol for the LP streaming problem, where  $|A|$  is the number of non-zero entries in the constraint matrix  $A$  of  $\mathcal{A}$ .*

*Proof.* The protocol shows an upper bound by providing a primal-feasible solution  $\mathbf{x}$ , and a lower bound by providing a dual-feasible solution  $\mathbf{y}$ . When the value of both solutions match,  $\mathcal{V}$  is convinced that  $\mathbf{x}$  is optimal.

From the stream,  $\mathcal{V}$  fingerprints the sets  $S_A = \{(i, j, A_{i,j})\}$ ,  $S_B = \{(i, \mathbf{b}_i)\}$  and  $S_C = \{(i, \mathbf{c}_j)\}$ . Then  $\mathcal{H}$  provides all pairs of values  $\mathbf{c}_j, \mathbf{x}_j, 1 \leq j \leq c$ , with each  $\mathbf{x}_j$  additionally annotated with  $|A_{\cdot j}|$ , the number of non-zero entries in column  $j$  of  $A$ . This allows  $\mathcal{V}$  to fingerprint the multiset  $S_X = \{(j, \mathbf{x}_j) : |A_{\cdot j}|\}$  and calculate the solution cost  $\sum_{j=1}^b \mathbf{c}_j \mathbf{x}_j$ .

To prove feasibility, for each row  $i$  of  $A$ ,  $A_{i\cdot}$ ,  $\mathcal{H}$  sends  $\mathbf{b}_i$ , then (the non-zero entries of)  $A_i$ , so that  $A_{ij}$  is annotated with  $\mathbf{x}_j$ . This allows the  $i$ th constraint to be checked in constant space.  $\mathcal{V}$  fingerprints the values given by  $\mathcal{H}$  for  $A$ ,  $\mathbf{b}$ , and  $\mathbf{c}$ , and compares them to those for the stream. A single fingerprint of the multiset of values presented for  $\mathbf{x}$  over all rows is compared to  $f(S_X)$ .  $\mathcal{V}$  is convinced  $\mathbf{x}$  is feasible if all constraints are met and all fingerprint tests pass.

Correctness follows by observing that the agreement with  $f(A)$  guarantees (whp) that each entry of  $A$  is presented correctly and no value is omitted. Since  $\mathcal{H}$  presents each entry of  $\mathbf{b}$  and  $\mathbf{c}$  once, in index order, the fingerprints  $f(S_B)$  and  $f(S_C)$  ensure that these values are presented correctly. The claimed  $|A_{\cdot j}|$  values must be correct: if not, then the fingerprints of either  $S_X$  or  $S_A$  will not match the multisets provided by  $\mathcal{H}$ .  $f(S_X)$  also ensures that each time  $\mathbf{x}_j$  is presented, the same value is given (similar to Lemma 1).

To prove that  $\mathbf{x}$  is primal-optimal, it suffices to show a feasible solution  $\mathbf{y}$  to the dual  $A^T$  so that  $c^T \mathbf{x} = b^T \mathbf{y}$ . Essentially we repeat the above protocol on the dual, and check that the claimed values are again consistent with the fingerprints of  $S_A, S_B, S_C$ .  $\square$

For any graph problem that can be formulated as a linear program in which each entry of  $A$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  can be derived as a linear function of the nodes and edges, we may view each edge in a graph stream  $\mathcal{A}$  as providing an update to values of one or more entries of  $A$ ,  $\mathbf{b}$ , and  $\mathbf{c}$ . Therefore, we immediately obtain a protocol for problems of this form via Theorem 3. More generally, we obtain protocols for problems formulated as *totally unimodular integer programs* (TUM IPs), since optimality of a feasible solution is shown by a matching feasible solution of the dual of its LP relaxation [16].

**Corollary 1.** *There is a valid  $(|A|, 1)$  protocol for any graph problem that can be formulated as a linear program or TUM IP in which each entry of  $A$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  is a linear function of the nodes and edges of graph.*

This follows immediately from Theorem 3 and the subsequent discussion: note by the linearity of the fingerprinting,  $\mathcal{H}$  presents only the (aggregated) values of  $S_A, S_B$  and  $S_C$ , not information from the unaggregated graph stream.

**Corollary 2.** *Shortest  $s - t$  path, max-flow, min-cut, and minimum weight bipartite perfect matching (MWBPM) all have valid  $(m, 1)$  protocols. For all four problems, a lower bound of  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits holds for dense graphs.*

*Proof.* The upper bound follows from the previous corollary because all the problems listed possess formulations as TUM IPs and moreover the constraint matrix in each case has  $O(m + n)$  non-zero entries. For example, for max-flow,  $\mathbf{x}$  gives the flow on each edge, and the weight of each edge in the stream contributes (linearly) to constraints on the capacity of that edge, and the flow through incident nodes. We omit a proof of the lower bounds.  $\square$

Conceptually, the above protocols for solving the LP streaming problem are straightforward:  $\mathcal{H}$  provides a primal solution, potentially repeating it once for

each row of  $A$  to prove feasibility, and repeats the protocol for the dual. There are efficient protocols for the problems listed in the corollary since the constraint matrices of their IP formulations are sparse. For dense constraint matrices, however, the bottleneck is proving feasibility. We observe that computing  $A\mathbf{x}$  reduces to computing  $b$  inner-product computations of vectors of dimension  $c$ . There are  $(c^\alpha, c^{1-\alpha})$  protocols to verify such inner-products [6]. But we can further improve on this since one of the vectors is held constant in each of the tests. This reduces the space needed by  $\mathcal{V}$  to run these checks in parallel; moreover, we prove a lower bound of  $\text{hcost} \cdot \text{vcost} = \Omega(\min(c, b)^2)$  bits, and so obtain an *optimal* tradeoff for square matrices, up to logarithmic factors. We omit the proof for space reasons.

**Theorem 4.** *Given a  $b \times c$  matrix  $A$  and a  $c$  dimensional vector  $\mathbf{x}$ , the product  $A\mathbf{x}$  can be verified with a valid  $(bc^\alpha, c^{1-\alpha})$  protocol. Moreover, any such protocol requires  $\text{hcost} \cdot \text{vcost} = \Omega(\min(c, b)^2)$  bits for dense matrices.*

**Corollary 3.** *For  $c \geq b$  there is a valid  $(c^{1+\alpha}, c^{1-\alpha})$  protocol for the LP streaming problem.*

*Proof.* This follows by using the protocol of Theorem 4 to verify  $A\mathbf{x} \leq \mathbf{b}$  and  $A^T\mathbf{y} \geq \mathbf{c}$  within the protocol of Theorem 3. The cost is  $(bc^\alpha + cb^\alpha, c^{1-\alpha} + b^{1-\alpha})$ , so if  $c \geq b$ , this is dominated by  $(c^{1+\alpha}, c^{1-\alpha})$ .  $\square$

Our protocol for linear programming relied on only two properties: strong duality, and the ability to compute the value of a solution  $\mathbf{x}$  and check feasibility via matrix-vector multiplication. Such properties also hold for more general convex optimization problems, such as quadratic programming and a large class of second-order cone programs. Thus, similar results apply for these mathematical programs, motivated by applications in which weak peripheral devices or sensors perform error correction on signals. We defer full details from this presentation.

Theorem 4 also implies the existence of protocols for graph problems where *both* hcost and vcost are sublinear in the size of the input (for dense graphs). These include:

- An  $(n^{1+\alpha}, n^{1-\alpha})$  protocol for verifying that  $\lambda$  is an eigenvalue of the adjacency matrix  $A$  or the Laplacian  $L$  of  $G$ :  $\mathcal{H}$  provides the corresponding eigenvector  $x$ , and  $\mathcal{V}$  can use the protocol of Theorem 4 to verify that  $Ax = \lambda x$  or  $Lx = \lambda x$ .
- An  $(n^{1+\alpha}, n^{1-\alpha})$  protocol for the problem of determining the effective resistance between designated nodes  $s$  and  $t$ , where the edge weights are resistances. The problem reduces to solving an  $n \times n$  system of linear equations [17].

## 4 Simulating Non-streaming Algorithms

Next, we give protocols by appealing to known non-streaming algorithms for graph problems. At a high level, we can imagine the helper running an algorithm on the graph, and presenting a “transcript” of operations carried out by the algorithm as the proof to  $\mathcal{V}$  that the final result is correct. Equivalently, we can imagine that  $\mathcal{V}$  runs the algorithm, but since the data structures are large, they are stored by  $\mathcal{H}$ , who provides the contents of memory needed for each step.

There may be many choices of the algorithm to simulate and the implementation details: our aim is to choose ones that result in smaller annotations.

Our main technical tool is the off-line memory checker of Blum et al. [11], which we use to efficiently verify a sequence of accesses to a large memory. Consider a *memory transcript* of a sequence of read and write operations to this memory (initialized to all zeros). Such a transcript is *valid* if each read of address  $i$  returns the last value written to that address. The protocol of Blum et al. requires each read to be accompanied by the timestamp of the last write to that address; and to treat each operation (read or write) as a read of the old value followed by the write of a new value. Then it suffices to ensure that a fingerprint of all write operations (augmented with timestamps) matches a fingerprint of all read operations (using the provided timestamps), along with some simple local checks on timestamps. Consequently, any valid (timestamp-augmented) transcript is accepted by  $\mathcal{V}$ , while any invalid transcript is rejected by  $\mathcal{V}$  with high probability. We use this memory checker to obtain the following general simulation result.

**Theorem 5.** *Suppose  $P$  is a graph problem possessing a non-randomized algorithm  $\mathcal{M}$  in the random-access memory model that, when given  $G = (V, E)$  in adjacency list or adjacency matrix form, outputs  $P(G)$  in time  $t(m, n)$ , where  $m = |E|$  and  $n = |V|$ . Then there is an  $(m + t(m, n), 1)$  protocol for  $P$ .*

*Proof (sketch).*  $\mathcal{H}$  first repeats (the non-zero locations of) a valid adjacency list or matrix representation  $G$ , as writes to the memory (which is checked by  $\mathcal{V}$ );  $\mathcal{V}$  uses fingerprints to ensure the edges included in the representation precisely correspond to those that appeared in the stream, and can use local checks to ensure the representation is otherwise valid. This requires  $O(m)$  annotation and effectively initializes memory for the subsequent simulation. Thereafter,  $\mathcal{H}$  provides a valid augmented transcript  $T'$  of the read and write operations performed by algorithm  $\mathcal{M}$ ;  $\mathcal{V}$  rejects if  $T'$  is invalid, or if any read or write operation executed in  $T'$  does not agree with the prescribed action of  $\mathcal{M}$ . As only one read or write operation is performed by  $\mathcal{M}$  in each timestep, the length of  $T'$  is  $O(t(m, n))$ , resulting in an  $(m + t(m, n), 1)$  protocol for  $P$ .  $\square$

Although Theorem 5 only allows the simulation of deterministic algorithms,  $\mathcal{H}$  can non-deterministically “guess” an optimal solution  $S$  and prove optimality by invoking Theorem 5 on a (deterministic) algorithm that merely checks whether  $S$  is optimal. Unsurprisingly, it is often the case that the best-known algorithms for verifying optimality are more efficient than those finding a solution from scratch (see e.g. the MST protocol below), and this gives the simulation theorem considerable power.

**Theorem 6.** *There is a valid  $(m, 1)$  protocol to find a minimum cost spanning tree; a valid  $(m + n \log n, 1)$  protocol to verify single-source shortest paths; and a valid  $(n^3, 1)$  protocol to verify all-pairs shortest paths.*

*Proof.* We first prove the bound for MST. Given a spanning tree  $T$ , there exists a linear-time algorithm  $\mathcal{M}$  for verifying that  $T$  is minimum (see e.g. [18]). Let  $\mathcal{M}'$

be the linear-time algorithm that, given  $G$  and a subset of edges  $T$  in adjacency matrix form, first checks that  $T$  is a spanning tree by ensuring  $|T|=n-1$  and  $T$  is connected (using e.g. breadth-first search), and then executes  $\mathcal{M}$  to ensure  $T$  is minimum. We obtain an  $(m, 1)$  protocol for MST by having  $\mathcal{H}$  provide a minimum spanning tree  $T$  and using Theorem 5 to simulate  $\mathcal{M}'$ . The upper bound for single-source shortest paths follows from Theorem 5 and the fact that there exist implementations of Dijkstra's algorithm requiring time  $O(m+n \log n)$ . The upper bound for all-pairs shortest paths also follows from Theorem 5 and the fact that the Floyd-Warshall algorithm runs in time  $O(n^3)$ .  $\square$

We now provide near-matching lower bounds for all three problems. Proofs are omitted for space reasons.

**Theorem 7.** *Any protocol for verifying single-source or all pairs shortest paths requires  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits. Additionally, if edge weights may be specified incrementally, then an identical lower bound holds for MST.*

**Diameter.** The diameter of  $G$  can be verified via our all-pairs shortest path protocol, but the next protocol improves over the memory checking approach.

**Theorem 8.** *There is a valid  $(n^2 \log n, 1)$  protocol for computing graph diameter. Furthermore, any protocol for diameter requires  $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$  bits.*

*Proof.* [6, Theorem 5.2] gives an  $(n^2 \log l, 1)$  protocol for verifying that  $A^l = B$  for a matrix  $A$  presented in a data stream and for any positive integer  $l$ . Note that if  $A$  is the adjacency matrix of  $G$ ; then  $(I + A)_{ij}^l \neq 0$  if and only if there is a path of length at most  $l$  from  $i$  to  $j$ . Therefore, the diameter of  $G$  is equal to the unique  $l > 0$  such that  $(I + A)_{ij}^l \neq 0$  for all  $(i, j)$ , while  $(I + A)_{ij}^{l-1} = 0$  for some  $(i, j)$ . Our protocol requires  $\mathcal{H}$  to send  $l$  to  $\mathcal{V}$ , and then run the protocol of [6, Theorem 5.2] twice to verify  $l$  is as claimed. Since the diameter is at most  $n - 1$ , this gives an  $(n^2 \log n, 1)$  protocol. We omit the proof of the lower bound.

## 5 Conclusion and Future Directions

In this paper, we showed that a host of graph problems possess streaming protocols requiring only constant space and linear-sized annotations. For many applications of the annotation model, the priority is to minimize vcost, and these protocols achieve this goal. However, these results are qualitatively different from those involving numerical streams in the earlier work [6]: for the canonical problems of heavy hitters, frequency moments, and selection, it is trivial to achieve an  $(m, 1)$  protocol by having  $\mathcal{H}$  replay the stream in sorted (“best”) order. The contribution of [6] is in presenting protocols obtaining optimal tradeoffs between hcost and vcost in which both quantities are sublinear in the size of the input. There are good reasons to seek these tradeoffs. For example, consider a verifier with access to a few MBs or GBs of working memory. If an  $(m, 1)$  protocol requires only a few KBs of space, it would be desirable to use more of the available memory to significantly reduce the running time of the verification protocol.

In contrast to [6], it is non-trivial to obtain  $(m, 1)$  protocols for the graph problems we consider, and we obtain tradeoffs involving sublinear values of  $\text{hcost}$  and  $\text{vcost}$  for some problems with an algebraic flavor (e.g. matrix-vector multiplication, computing effective resistances, and eigenvalues of the Laplacian). We thus leave as an open question whether it is possible to obtain such tradeoffs for a wider class of graph problems, and in particular if the use of memory checking can be adapted to provide tradeoffs.

A final open problem is to ensure that the work of  $\mathcal{H}$  is scalable. In motivating settings such as Cloud computing environments, the data is very large, and  $\mathcal{H}$  may represent a *distributed* cluster of machines. It is a challenge to show that these protocols can be executed in a model such as the MapReduce framework.

**Acknowledgements.** We thank Moni Naor for suggesting the use of memory checking.

## References

1. McGregor, A.: Graph mining on streams. In: Encyc. of Database Systems. Springer, Heidelberg (2009)
2. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. *Theor. Comput. Sci.* 348(2), 207–216 (2005)
3. Demetrescu, C., Finocchi, I., Ribichini, A.: Trading off space for passes in graph streaming problems. In: SODA 2006, pp. 714–723 (2006)
4. Aggarwal, G., Datar, M., Rajagopalan, S., Ruhl, M.: On the streaming model augmented with a sorting primitive. In: FOCS 2004, pp. 540–549 (2004)
5. Das Sarma, A., Lipton, R.J., Nanongkai, D.: Best-order streaming model. In: Theory and Applications of Models of Computation, pp. 178–191 (2009)
6. Chakrabarti, A., Cormode, G., Mcgregor, A.: Annotations in data streams. In: ICALP 2009, pp. 222–234 (2009)
7. Lipton, R.J.: Efficient checking of computations. In: Choffrut, C., Lengauer, T. (eds.) STACS 1990. LNCS, vol. 415, pp. 207–215. Springer, Heidelberg (1990)
8. Yiu, M., Lin, Y., Mouratidis, K.: Efficient verification of shortest path search via authenticated hints. In: ICDE (2010)
9. Clarkson, K.L., Woodruff, D.P.: Numerical linear algebra in the streaming model. In: STOC 2009, pp. 205–214 (2009)
10. Sarlos, T.: Improved approximation algorithms for large matrices via random projections. In: IEEE FOCS 2006 (2006)
11. Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories, pp. 90–99 (1995)
12. Cormode, G., Mitzenmacher, M., Thaler, J.: Streaming graph computations with a helpful advisor. CoRR, vol. abs/1004.2899 (2010)
13. Lipton, R.J.: Fingerprinting sets. Princeton University, Tech. Rep. Cs-tr-212-89 (1989)
14. Kleinberg, J., Tardos, E.: Algorithm Design (2005)
15. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency (2003)
16. Schrijver, A.: Theory of linear and integer programming (1986)
17. Bollobas, B.: Modern Graph Theory. Springer, Heidelberg (1998)
18. King, V.: A simpler minimum spanning tree verification algorithm. *Algorithmica* 18(2), 263–270 (1997)

# Algorithms for Dominating Set in Disk Graphs: Breaking the $\log n$ Barrier<sup>\*</sup> (Extended Abstract)<sup>\*\*</sup>

Matt Gibson<sup>1</sup> and Imran A. Pirwani<sup>2</sup>

<sup>1</sup> Dept. of Electrical & Computer Engineering, University of Iowa,  
Iowa City, IA 52242, USA  
[mrgibson@engineering.uiowa.edu](mailto:mrgibson@engineering.uiowa.edu)

<sup>2</sup> Dept. of Computing Science, University of Alberta Edmonton,  
Alberta T6G 2E8, Canada  
[imran.pirwani@gmail.com](mailto:imran.pirwani@gmail.com)

**Abstract.** We consider the problem of finding a lowest cost dominating set in a given disk graph containing  $n$  disks. The problem has been extensively studied on subclasses of disk graphs, yet the best known approximation for disk graphs has remained  $O(\log n)$  – a bound that is asymptotically no better than the general case. We improve the status quo in two ways: for the unweighted case, we show how to obtain a PTAS using the framework recently proposed (independently) by Mustafa and Ray [16] and by Chan and Har-Peled [4]; for the weighted case where each input disk has an associated rational weight with the objective of finding a minimum cost dominating set, we give a randomized algorithm that obtains a dominating set whose weight is within a factor  $2^{O(\log^* n)}$  of a minimum cost solution, with high probability – the technique follows the framework proposed recently by Varadarajan [19].

## 1 Introduction

For a set  $\mathcal{D}$  of  $n$  disks in the Euclidean plane, define an intersection graph,  $G = (V, E)$ , thus:  $V = \mathcal{D}$ ;  $\{u, v\} \in E \Leftrightarrow \text{disk}(u) \cap \text{disk}(v) \neq \emptyset$ .  $G$  is called a *disk graph*; it is a *unit disk graph* when the disk radii are identical.

Given a graph the *minimum dominating set* (MDS) problem is to find a smallest subset  $\mathcal{D}' \subseteq V$  such that every vertex is either in  $\mathcal{D}'$  or is adjacent to a vertex in  $\mathcal{D}'$ . On general graphs, the problem is  $(1 - \varepsilon) \ln n$  hard to approximate for any  $\varepsilon > 0$  under standard complexity theoretic assumptions [10, 5], while a greedy algorithm yields an  $O(\log n)$  approximation [20].

Nevertheless, better approximations are possible for restricted domains. For example, the problem admits a *polynomial-time approximation scheme* (PTAS)

\* Work of the first author was supported by NSF grants CCF 0915543 and CCF 0830402. Work of the second author was supported by Alberta Ingenuity.

\*\* Several details are left out due to space constraints. For the full version of the paper, please see [11].

for unit disk graphs and *growth-bounded graphs* [12][17]. The problem is NP-hard on these domains [6]. However, for the disk graph case,  $o(\log n)$  approximations have remained elusive – perhaps, in part, because known techniques for unit disk graphs and solutions to other problems on disk graphs have either relied on packing properties [12][17][8][3], or when packing property does not hold, as in the *minimum weighted dominating set* on unit disk graphs, the fact that disk radii are uniform [11][18]. Erlebach and van Leeuwen recently studied the dominating set problem on *fat objects*, e.g., disk graphs, [9]. They note that existing techniques for disk graphs do not seem sufficient to solve MDS [9]; they also give an  $O(1)$ -approximation for fat objects of *bounded ply*. Recently, Kammer and Tholey [14] give an  $O(1)$ -approximation algorithm for MDS when the input is a disk graph with some special properties as well as for intersection graphs with  $r$ -regular polygons and other fat objects.

In their recent break-through papers, Chan and Har-Peled [4], and Mustafa and Ray [16] independently showed how a simple *local search* algorithm on certain geometric graphs yields a PTAS for some problems; Chan and Har-Peled [4] show local search yields a PTAS for maximum independent set problem on admissible objects, while Mustafa and Ray [16] show local search yields a PTAS for the minimum hitting set problem given a collection of points and half-spaces in  $\mathbb{R}^3$ , and also for points and admissible regions in  $\mathbb{R}^2$ . They both use the *planar separator theorem* to relate the cost of the local search solution with the optimum solution. In the framework, at the crux lies the analysis of a certain graph whose vertices are objects found by local search and ones that belong to an optimum solution, and whose edges (which are only between the two kinds of vertices) satisfy a property relating the two solutions. They show that there exists such a graph which is also planar. Mustafa and Ray [16] refer to the existence of such a planar graph as the *locality condition*.

*Results:* Our first result is a PTAS for the minimum dominating set problem for disk graphs via a local search algorithm, as in [4][16]. Our analysis also uses the framework introduced by these two papers. Our main new contribution is to show the existence of a planar graph satisfying the locality condition. This graph turns out to be the dual of a weighted Voronoi diagram in the plane.

The minimum dominating set problem for disk graphs can be reduced to the problem of hitting half-spaces in  $\mathbb{R}^4$  with the smallest number of a given set of points. That is, given the set  $\mathcal{D}$  of disks that form the input to the MDS problem, we can easily compute a map  $\pi$  from  $\mathcal{D}$  to a set of points in  $\mathbb{R}^4$ , and a map  $h$  from  $\mathcal{D}$  to a set of half-spaces in  $\mathbb{R}^4$ , with the following property: Two disks  $d_1$  and  $d_2$  from  $\mathcal{D}$  intersect if and only if  $\pi(d_1)$  lies in  $h(d_2)$ . Thus we can efficiently reduce the MDS problem for disks to a hitting set problem for points and half-spaces in  $\mathbb{R}^4$ . While there is a PTAS for the hitting set problem in  $\mathbb{R}^3$ , as shown by [16], there is none known for  $\mathbb{R}^4$ . It is not hard to see that a local search such as the one in [16] does not yield a PTAS in  $\mathbb{R}^4$ .

Rather than reduce to a hitting set problem, we are able to establish the locality condition by staying in the plane itself. In fact, the graph for the locality condition is the dual of the weighted Voronoi diagram of the centers of the disks

in the local search solution and the optimal solution, where the weights are the radii of the disk. This can be seen as generalizing the situation considered by [16] for the hitting set problem with points and disks in the plane. In that case, the graph for the locality condition is the Delaunay triangulation, which is the dual of the unweighted Voronoi diagram.

For the case when the disks are weighted, we give the first  $o(\log n)$  approximation algorithm; we give a  $2^{O(\log^* n)}$  approximation algorithm<sup>1</sup>. This result is based on the framework recently introduced by Varadarajan for the weighted geometric set cover problem [19]. Our contribution here is to observe that the framework is applicable to our dominating set problem as well; the weighted Voronoi diagram is the key to this result also.

We assume that the inputs for both problems satisfy non-degeneracy assumptions – no three disk centers on a line and no four disks tangent to a circle. This is without loss of generality, as these conditions can be enforced by simple perturbations. In Section 2, we present our PTAS for the unweighted dominating set problem, and in Section 3 our algorithm for weighted dominating set.

## 2 The Unweighted Case: PTAS via Local Search

In this section, we give our PTAS for minimum dominating set for disk graphs. Here, we are given a disk graph with a set  $\mathcal{D}$  of  $n$  disks in the Euclidean plane, and we are interested in computing a minimum cardinality dominating set of the disk graph. The algorithm is given in Section 2.1 and the analysis of the approximation ratio is given in Section 2.2.

### 2.1 The Algorithm

*Local Search.* Call a subset of disks,  $B \subseteq \mathcal{D}$ ,  $b$ -locally optimal if one cannot obtain a smaller dominating set by removing a subset  $X \subseteq B$  of size at most  $b$  from  $B$  and replacing that with a subset of size at most  $|X| - 1$  from  $\mathcal{D} \setminus B$ . Our algorithm will compute a  $b$ -locally optimal set of disks for  $b = \frac{c}{\epsilon^2}$  where  $c > 0$  is a large enough constant. Our algorithm begins with an arbitrary feasible set of disks and proceeds by making small local exchanges of size  $b = O(\frac{1}{\epsilon^2})$ , for a given  $\epsilon > 0$ . We stop when no further local improvements are possible.

Suppose that the solution returned is  $B$ . Finally, for reasons apparent in the analysis, we check to see if for any disk  $u \in B$  there is a disk  $v \in \mathcal{D}$  such that  $u$  is completely contained in  $v \in \mathcal{D} \setminus B$ . If such a disk exists, then simply replace  $u$  with the largest such disk  $v$ . We return this as our final solution and call it  $B$ . Our replacement step ensures that there is no disk in  $B$  that is properly contained in some other disk in  $\mathcal{D}$ .

### 2.2 Approximation Ratio

We will show that our algorithm is a PTAS, thus proving the following theorem:

---

<sup>1</sup>  $\log^* n$  is the fewest number of iterated “logarithms” applied to  $n$  to yield a constant.

**Theorem 1.** *For any  $\epsilon > 0$ , there exists a polynomial time algorithm for the minimum dominating set problem on disk graphs that returns a solution whose cost is at most  $(1 + \epsilon)OPT$  where  $OPT$  is the cost of an optimal solution.*

Let  $R$  be the disks in an optimal solution; we may assume no disk in  $R$  is properly contained in any other disk in  $\mathcal{D}$ . Thus, no disk in  $R \cup B$  is properly contained in any other disk of  $R \cup B$ . Note that by the definition of PTAS, we need to show that  $|B| \leq (1 + \epsilon) \cdot |R|$ . We will refer to  $R$  as the set of red disks and  $B$  as the set of blue disks. Without loss of generality, we will assume that  $R \cap B = \emptyset$ , i.e. there is no disk that is both red and blue. For a disk  $u \in \mathcal{D}$ , we say a disk  $v \in R \cup B$  is a *dominator* of  $u$  if  $u$  and  $v$  intersect. Similarly, we also say that  $v$  *dominates*  $u$ .

We must show the existence of an appropriate planar graph which relates the disks in  $R$  with the disks in  $B$ . Here, we state the *locality condition* as per Mustafa and Ray [16]:

**Lemma 1 (Locality Condition).** *There exists a planar graph with vertex set  $R \cup B$ , such that for every  $d \in \mathcal{D}$ , there is a disk  $u$  from amongst the red dominators of  $d$  and a disk  $v$  amongst the blue dominators of  $d$  such that  $\{u, v\}$  is an edge in the graph.*

Section 2.3 is devoted to a proof of Lemma 1. In this extended abstract, we skip the argument (from [4, 16]) that has now become standard which uses the lemma to show that  $|B| < (1 + \epsilon)|R|$ ; we also skip the running time analysis.

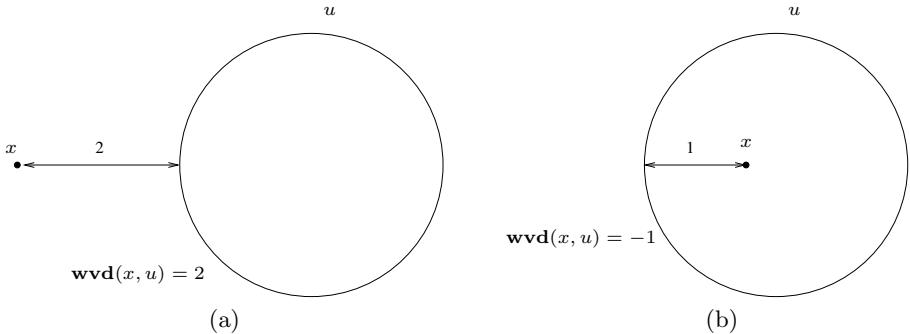
### 2.3 Establishing the Locality Condition

This section is devoted to the proof of Lemma 1, that is, the construction of an appropriate planar graph which satisfies the locality condition.

*Weighted Voronoi Diagram.* We will be using a generalization of Voronoi diagrams called a *weighted Voronoi Diagram* (WVD). Instead of defining cells with respect to a set of points, we will be defining cells with respect to red and blue disks. In order to do this generalization for disks, we must define the distance between a point in the plane and a disk.

Let  $u$  be a disk and let  $x$  be a point in the plane. We define  $\text{wvd}(x, u) = d(x, c_u) - r_u$  where  $c_u$  is the center of  $u$ ,  $r_u$  is the radius of  $u$ , and  $d(x, c_u)$  is the Euclidean distance between  $x$  and  $c_u$ . Intuitively, for a point  $x$ ,  $\text{wvd}(x, u)$  is the Euclidean distance from  $x$  to the boundary of  $u$ ; the distance to a disk is negative for points that are strictly inside the disk. Alternatively, if  $x \notin u$ , then  $\text{wvd}(x, u)$  is the amount we would need to increase the radius of  $u$  so that  $x$  lies on the boundary of  $u$ ; if  $x \in u$ , then  $\text{wvd}(x, u)$  is the negative of the amount we would need to decrease the radius of  $u$  so that  $x$  lies on the boundary of  $u$ . See Figure 1 for an illustration.

For a disk  $u$  in any collection of disks, let  $\text{cell}(u)$  be the set of points  $x$  in the plane such that  $\text{wvd}(x, u) \leq \text{wvd}(x, v)$ ,  $u \neq v$ . The cells of all the disks in the collection induce a decomposition of the plane, and this is the WVD. This is just the



**Fig. 1.** An illustration for the distances used in our WVD. (a)  $\text{wvd}(x, u)$  when  $x$  is not in  $u$ . (b)  $\text{wvd}(x, u)$  when  $x$  is in  $u$ .

standard weighted Voronoi diagram of the centers of the disks, where the weight of the center of a disk is simply the radius of the disk [2].

Consider the WVD of the disks in  $R \cup B$ . First, we will show that for every  $u \in R \cup B$ ,  $u$  has a non-empty cell in the WVD. That is, there is some point in the plane that is closer to  $u$  than it is to any other red or blue disk.

**Lemma 2.** *In the weighted Voronoi diagram of the union of red and blue disks, the cell of every disk  $u$  is nonempty. Moreover,  $c_u$  (the center of  $u$ ) belongs only to  $\text{cell}(u)$ .*

*Proof.* We will show that  $c_u$  is only in  $\text{cell}(u)$ . Suppose for the sake of contradiction that  $c_u \in \text{cell}(v)$  such that  $u \neq v$ . This means that  $\text{wvd}(c_u, v) \leq \text{wvd}(c_u, u) = d(c_u, c_u) - r_u = -r_u$ . So,  $-r_u \geq \text{wvd}(c_u, v) = d(c_u, c_v) - r_v \Rightarrow r_v \geq d(c_u, c_v) + r_u$ . This implies that  $u$  is contained in  $v$ , and since the two disks are not the same, the containment is proper. But this is a contradiction, since no disk in  $R \cup B$  contains another such disk.  $\square$

*The Graph.* Any cell in the WVD of  $R \cup B$  is star-shaped with respect to the center of the corresponding disk. That is, for every point  $y \in \text{cell}(u)$ , the segment  $\overline{c_u y}$  is contained within  $\text{cell}(u)$ .

The graph for the locality condition is simply the dual of the WVD of  $R \cup B$ . That is, for each cell in the WVD there is a vertex, and there is an edge between two vertices if and only if their corresponding cells share a boundary in the diagram (that is, if and only if there is a point in the plane equidistant from the two disks). The graph is planar – exploiting the fact that the cells are star-shaped, the edges can easily be drawn so that no two edges intersect [2].

**Corollary 1.** *The dual of the power diagram of  $R \cup B$  is a planar graph.*

Because every red and blue disk has a nonempty cell in the WVD, every such disk will also have a corresponding vertex in our planar graph. We are now ready to show that for each  $d \in \mathcal{D}$ , there is a disk  $u$  from amongst the red dominators

of  $d$  and a disk  $v$  amongst the blue dominators of  $d$  such that  $\text{cell}(u)$  and  $\text{cell}(v)$  share a boundary in the WVD. This would then imply that their corresponding vertices in the graph share an edge, completing the proof of Lemma 2. For simplicity, if there is an edge connecting the vertex corresponding to  $\text{cell}(u)$  and the vertex corresponding to  $\text{cell}(v)$ , then we will simply say there is an edge connecting  $u$  and  $v$ .

**Lemma 3.** *In the dual graph of the weighted Voronoi diagram for  $R \cup B$ , for an arbitrary input disk  $u \in \mathcal{D}$ , there is an edge between some red dominator of  $u$  and some blue dominator of  $u$ .*

*Proof.* Consider the WVD of  $R \cup B$ . Without loss of generality, assume  $c_u \in \text{cell}(r)$  for some  $r \in R$ . Now,  $r$  must be a dominator of  $u$ , because  $r$  is the closest disk in  $R \cup B$  to  $c_u$ . If  $r$  does not dominate  $u$ ,  $u$  is not dominated by any disk in  $R \cup B$  which contradicts the fact that both  $R$  and  $B$  are dominating sets.

Let  $b$  denote a closest blue disk to  $c_u$ , that is  $\text{wvd}(c_u, b) \leq \text{wvd}(c_u, b')$  for all other blue disks  $b'$ . Note that  $b$  must dominate  $u$ , because if it did not, then no blue disks would dominate  $u$ . This would contradict the fact that  $B$  is a dominating set. Also, note that for any disk  $d \in \mathcal{D}$  such that  $\text{wvd}(c_u, d) \leq \text{wvd}(c_u, b)$ ,  $d$  must intersect with  $u$ .

If  $\text{wvd}(c_u, b) = \text{wvd}(c_u, r)$ , we are done, since then there is an edge in the dual graph incident on  $r$  and  $b$ . So, let us assume that  $\text{wvd}(c_u, b) > \text{wvd}(c_u, r)$ .

We will walk from  $c_u$  to  $c_b$  along the straight line segment  $\overline{c_u c_b}$ . The proof strategy is that during this walk, we will be crossing red cells and at some point before reaching  $c_b$  we will enter a blue cell, in particular,  $\text{cell}(b)$ . We must have entered this cell from a red cell  $\text{cell}(r')$  which shares a boundary with  $\text{cell}(b)$ , and thus  $\{r', b\}$  is an edge in our planar graph. Moreover, we will argue that  $r'$  necessarily dominates  $u$ , completing the proof.

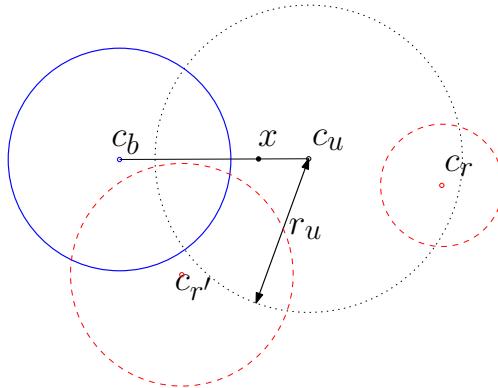
As seen in the proof of Lemma 2,  $c_b \in \text{cell}(b)$ , and thus we will enter  $\text{cell}(b)$  at some point in time along our walk from  $c_u$  to  $c_b$ . Let  $x$  be the point at which we first enter  $\text{cell}(b)$ . Then  $x$  is on the boundary of  $\text{cell}(b)$  and  $\text{cell}(r')$  for some  $r' \in R \cup B$ . If  $r' = r$ , we are done. Otherwise, we have

$$\text{wvd}(c_u, r') < d(c_u, x) + \text{wvd}(x, r') = d(c_u, x) + \text{wvd}(x, b) = \text{wvd}(c_u, b).$$

(Here the strictness of the first inequality comes from our non-degeneracy assumption which implies that  $c_{r'}$  cannot lie on the line through  $c_u$  and  $c_b$ .) Now, it must be the case that  $r' \in R$  because  $\text{wvd}(c_u, r') < \text{wvd}(c_u, b)$  and  $b$  is the closest blue disk to  $c_u$ . This also implies that  $r'$  must dominate  $u$ . See Figure 2 for an illustration.

Therefore  $\text{cell}(b)$  and  $\text{cell}(r')$  share a boundary implying that the edge  $\{b, r'\}$  is in our graph. Moreover,  $b$  is blue,  $r'$  is red, and both dominate  $u$ , which completes the proof.  $\square$

Together, Corollary 1 and Lemma 3 prove Lemma 2.



**Fig. 2.** Proof of Lemma 3. The dotted disk is  $u$  with center  $c_u$  and radius  $r_u$ . The two red disks  $r$  and  $r'$  are shown as dashed disks with centers  $c_r$  and  $c_{r'}$ , respectively. The only blue disk  $b$  is shown as a solid disk with center  $c_b$ .

### 3 The Weighted Dominating Set Case

In this section, we study a classical generalization of the dominating set problem. Each disk  $u$  now has an associated rational weight,  $w_u$ . The goal is to find a dominating set  $D$  having the lowest cost, that is,  $\text{wt}(D) = \sum_{u \in D} w_u$  be as small as possible. We will prove the following theorem:

**Theorem 2.** *Given a disk graph,  $G = (V, E)$  of  $n$  weighted disks  $D$  in the plane, there is a randomized algorithm that produces a dominating set  $V' \subseteq V$ , and  $\text{wt}(V') \leq 2^{O(\log^* n)} \cdot \text{OPT}$ , w.h.p., where  $\text{OPT}$  denotes the cost of an optimal solution.*

The high-level structure of the algorithm is as follows: we first solve a natural linear programming relaxation, followed by a randomized rounding step; this step allows us to ignore the weights of the disks in the sampling (pruning) stage. In the rounding step, we make several copies of the disks to ensure that two properties hold. First, every disk in  $D$  is covered by at least  $n$  of the copies. Second, the weight of the copies is  $O(n \cdot \lambda^*)$ , where  $\lambda^*$  is the objective function value of an optimal LP solution. Following this step, we recursively apply a randomized pruning step where we remove some of the copies according to the algorithm given in the proof of Theorem 3 while ensuring that the remaining copies are a dominating set of  $D$ . The main goal of the pruning step is to remove some of the copies while approximately preserving the ratio of the cost of the remaining copies to the “depth” of the disks in  $D$  with respect to the remaining copies. We recursively apply the pruning step until the disks in  $D$  are covered by only a constant number of the remaining copies; the depth of our recursion is  $\Theta(\log^* n)$ . We can then show that the expected weight of our final dominating set is at worst  $2^{O(\log^* n)} \cdot \lambda^*$ .

First, we define some terms that are used in the remaining part of the section. Given a disk  $v$  and a set of disks  $S$ , we say that  $v$  is  $L$ -covered by  $S$  if there are

exactly  $L$  disks in  $S$  each of which intersects  $v$ . In other words, neighborhood of  $v$  in  $S$  has size  $L$ . We will make use of the following lemma, which is our main contribution to the weighted case:

**Lemma 4.** *Let  $S$  be a set of  $m$  disks, and  $1 \leq L \leq m$  an integer. Let  $Q$  be another (possibly infinite) set of disks. There are  $O(m \cdot L^2)$  disks of  $Q$  that intersect distinct subsets of  $S$  each of size at most  $L$ .*

*Proof.* We first define a few concepts that we use in the proof. We focus on subsets  $S' \subseteq S$  of size at most  $L$  and disks of  $Q$  whose neighborhood is precisely one of these subsets; let us denote this subset of  $Q$  by  $Q'$ . For a set  $S' \subseteq S$  of size at most  $L$ , and a pair of disks  $u, v \in Q'$ , we say that  $u$  and  $v$  are related if they both intersect every disk in  $S'$  and no other disk of  $S \setminus S'$ , i.e.  $u \cap S = v \cap S = S'$ . So we have an equivalence relation on  $Q'$  where each equivalence class corresponds to a set  $S' \subseteq S$ . We wish to bound the number of these equivalence classes. Let these subsets of  $S$  be  $\{S_1, S_2, \dots, S_t\}$ , and correspondingly,  $t$  equivalence classes  $\{Q_1, Q_2, \dots, Q_t\}$ , where each disk in  $Q_i$  intersects every disk in  $S_i$ , and no other disk of  $S \setminus S_i$ . Consider any set  $Q_i$  and an arbitrary disk  $v \in Q_i$ . By scaling and/or translating  $v$  we can obtain a disk  $v'$  with the following property:  $v'$  has the same neighborhood as all the disks in  $Q_i$  and is sharing a single point with three, two, or one disk in  $S_i$  and is intersecting all the other disks in more than one point; for the cases when  $v'$  is touching a single disk in  $S_i$ , or two disks in  $S_i$ , we continue to translate and scale  $v'$  so that it touches two disks outside of  $S_i$ , or one disk outside of  $S_i$ , respectively. Without loss of generality, we assume that  $S$  has four special disks whose borders form the North, South, East, and West boundary, respectively, of the region that contains the input disks. We call these special disks  $N, S, E, W$ , respectively. Such a transformed disk,  $v'$ , that touches exactly three disks is referred to as  $v_i$ . We say that a disk  $d$  is *canonical* with respect to a set of disks  $D'$  if there are three distinct disks in  $D'$  such that  $d$  intersects the three disks at only one point each. Note that each  $v_i$  is a canonical disk with respect to the set  $S$ . We say that a canonical disk  $v$  is  $\kappa$ -*canonical* with respect to a set of disks  $D'$  if at most  $\kappa$  disks from  $D'$  intersect the interior of  $v$ . Therefore, each of the canonical disks  $v_i$  that we defined are  $L$ -canonical disks. It is easy to see that  $t$  is within a constant factor of the number of  $L$ -canonical disks with respect to  $S$ . For each  $v_i$ , the set of disks that shares exactly one point with it is called the *defining set* of  $v_i$  and every disk of  $S_i$  that shares more than one point with  $v_i$  is said to be in the *conflict set* of  $v_i$ . Note that the defining set of  $v_i$  has at least one disk from  $S_i$ , but at most two remaining disks can be from outside  $S_i$ . We will upper bound the number of  $L$ -canonical disks with respect to  $S$  (and hence upper bound  $t$ ) by choosing a random sample  $S' \subseteq S$  and calculating the expected number of 0-canonical disks with respect to  $S'$ . This technique dates back to that of Clarkson [7].

Let us choose a random subset  $S' \subseteq S$  using  $k$  independent trials in which we pick each disk from  $S$  with uniform probability, while we add  $N, S, E, W$  in  $S'$  with probability 1. Now, for a fixed  $v_i$  to be a 0-canonical disk in  $S'$ , its defining set must have been picked in  $S'$ , and its conflict set must not be in  $S'$ . The probability of this event is at least

$$\left(\frac{k}{m}\right)^3 \cdot \left(1 - \frac{L}{m}\right)^k$$

Thus, the expected number of disks among  $v_1, \dots, v_t$  ( $L$ -canonical disks for the sets) that are 0-canonical disks for  $S'$  is at least

$$t \cdot \left(\frac{k}{m}\right)^3 \cdot \left(1 - \frac{L}{m}\right)^k$$

We will show that the maximum number of 0-canonical disks for  $S'$  is  $O(k)$ .

*Claim.* For a set  $S'$  of disks of size  $k$ , the maximum number of 0-canonical disks induced is  $O(k)$ .

*Proof.* We will bound the number of 0-canonical disks by the number of Voronoi vertices of a weighted Voronoi diagram with  $k$  sites in which the sites are represented by the  $k$  centers of disks in  $S'$ , and the weight of each site is the radius of the corresponding disk. Every Voronoi vertex is equidistant from the disks of the regions sharing that vertex. So each Voronoi vertex in the Voronoi diagram corresponds to the center of a disk that touches the boundary of exactly three disks of  $S'$  (disks corresponding to the three regions defining that vertex) and does not intersect any other disk of  $S'$ . Since the number of Voronoi vertices of a Voronoi diagram having  $k$  sites is bounded linearly in  $k$ , the number of canonical disks that touch three disks of  $S'$  are thus bounded linearly in  $k$  as well. This leads to the final bound of  $O(k)$  on the maximum number of canonical disks that  $S'$  admits.  $\square$

According to the claim, the maximum number of 0-canonical disks for  $S'$  is  $O(k)$ . So,

$$t \cdot \left(\frac{k}{m}\right)^3 \cdot \left(1 - \frac{L}{m}\right)^k \leq c_1 k,$$

for some constant  $c_1 > 0$ . Choosing  $k = \frac{2m}{L}$  yields  $t \leq c'mL^2$ .  $\square$

We prove the following variant of a theorem of Varadarajan in [19].

**Theorem 3.** *Given a disk graph  $G = (V, E)$  and set of  $n$  weighted disks  $D \subseteq V$  in the plane s.t.  $D$  dominates  $V$ , there is a randomized algorithm that produces a subset  $D' \subseteq D$ , such that for any disk  $v \in V$ , if  $v$  is  $L$ -covered in  $D$ , then  $v$  is at least  $\log L$ -covered in  $D'$  and  $\Pr[d \in D'] \leq \frac{c \log L}{L}$ .*

*Proof.* We only describe a randomized process that selects a subset,  $D'$  of disks such that any disk  $v \in V$  that is covered by  $D$  in the range  $[L, 2L]$ ,  $v$  is at least  $\log L$ -covered in  $D'$ . Let  $N_m = D$ , and let  $C_m$  denote the set of equivalence classes of disks in  $V$  such that each class intersects at most  $2L$  disks of  $D$ . Note that since the disks in one equivalence class of  $V$  have the same neighborhood in  $D$ , if we obtain a set  $D'$  that at least  $\log L$ -covers one disk in that class, then all the disks in that class are also at least  $\log L$ -covered. Therefore, we

can assume we have one representative disk from each class and our goal is to at least log  $L$ -cover these disks. We use this fact crucially in our analysis. By Lemma 4,  $|\mathcal{C}_m| \leq c' \cdot n_m L^2$ ,  $n_m = |N_m|$ . So, there is a disk  $d_m$  that covers at most  $2c'L^2$  classes of  $\mathcal{C}_m$ . Find such a disk  $d_m \in N_m$ , and recursively compute a sequence for  $N_{m-1} = N_m \setminus \{d_m\}$ , and append the sequence to  $d_m$ . That is, in the arrangement of  $N_{m-1}$  we consider the classes  $\mathcal{C}_{m-1}$  whose coverage in  $N_{m-1}$  is at most  $2L$ . The recursion stops when there are fewer than  $L$  disks remaining, at which point, we compute an arbitrary sequence of the remaining set of disks.

Let  $\sigma$  be the *reverse* of this sequence, that is,  $\sigma = (d_1, d_2, \dots, d_m)$ . When considering disk  $d_j$ , we make an instant decision about including it in our cover or not. Call a disk  $d_j \in N_j$  *forced* if for some disk  $v \in \mathcal{C}_j$ , not including  $d_j$  *will not* log  $L$ -cover  $v$ , whose coverage in  $N_m$  is in  $[L, 2L]$ . Otherwise, if  $d_j$  is not forced, we add it to  $D'$  with probability  $\frac{c \cdot \log L}{L}$ . We will upper bound the probability of  $d_j$  being forced – we will show that it is at most  $O(1/L)$ .

Observe that if a disk  $d_j$  is forced because of  $v$ , then all the disks  $d_{j'}$  (with  $j' \geq j$ ) that cover  $v$  are also forced, and the number of such disks is at most  $\log L - 1$  (otherwise  $d_j$  won't be forced). So it is sufficient to upper bound the probability of a disk  $d_i$  being the first disk forced because of  $v$ . Let us denote this event by  $\mathcal{E}_i(v)$ . Since from among the disks that cover  $v$  at most the last  $\log L$  disks can be forced, the probability of one of these  $\log L$  disks being forced is at most  $\log L$  times the probability that one of the disks before it is the “first” forced disk because of  $v$ . We use  $\mathcal{E}_i$  to denote the event that  $d_i$  is the first disk forced because of some disk that it covers. We omit the proof of the following claim from this extended abstract.

*Claim.*

$$\Pr[\mathcal{E}_i(v)] \leq \frac{1}{L^4}$$

Note that any disk  $d_{i'}$  that occurs before  $d_i$  in  $\sigma$  if  $d_{i'}$  is forced for a disk  $v'$  that is not covered by  $d_i$ , which forces a disk  $d_k$  which occurs after  $d_i$  in  $\sigma$  and that  $d_k$  also covers  $v$ , then that event has no bearing on the event of  $d_i$  being a first forced disk for  $v$ . So, to upper bound the probability that some  $d_j$  is a forced disk for a fixed disk  $v$ , we sum over all valid indices  $i < j$  with  $d_i$  being the first forced disk because of  $v$ , and obviously there are at most  $\log L$  of them,

$$\Pr\left[\begin{smallmatrix} \text{some } d_j \text{ is} \\ \text{forced by} \\ \text{disk } v \end{smallmatrix}\right] \leq \sum_i \frac{1}{L^4} \leq \frac{1}{L^3}.$$

Since there are at most  $2c'L^2$  classes of  $\mathcal{C}_j$  having coverage in the range  $[L, 2L]$  that are covered by  $d_j$ ,  $d_j$  can be a forced addition for any one of the at most  $2c'L^2$  representative disks. So,

$$\Pr\left[\begin{smallmatrix} d_j \text{ is forced} \\ \text{for some} \\ \text{disk } v \in \mathcal{C}_j \end{smallmatrix}\right] \leq \frac{2c'}{L}.$$

The probabilistic algorithm finds a dominating set  $D' \subseteq D$  where the probability of a given disk being in  $D'$  is at most  $\frac{c \cdot \log L}{L}$  and each disk  $v \in V$  that is covered in the range  $[L, 2L]$  by  $D$ , is at least  $\log L$ -covered in  $D'$ . We repeat the process

for points that are between  $2L$  and  $4L$  deep, and so on. Note that the probability of a disk being in  $D'$  is still the same.  $\square$

### 3.1 Proof of Theorem 2

Let the input instance be a disk graph based on a set of disks  $D$ . For any disk  $d \in D$ , let  $N[d]$  denote the set of neighbors of  $d$  in the graph, inclusive. Consider the following natural LP relaxation for the weighted dominating set problem:

$$(LP) \min \sum_{d \in D} w_d x_d$$

subject to,

$$\begin{aligned} \sum_{d': d' \in N[d]} x_{d'} &\geq 1, \forall d \in D \\ x_d &\geq 0, \forall d \in D \end{aligned}$$

After solving the LP relaxation, we create a set  $D_0$  of disks as follows. For each disk  $d$  such that  $x_d \geq \frac{1}{2n}$ , we add  $\lfloor \frac{x_d}{1/(2n)} \rfloor$  copies of  $d$  to  $D_0$ . Each copy of  $d$  inherits its original cost. For each disk  $d$  with  $x_d < \frac{1}{2n}$ , we don't add any copy to  $D_0$ . It is easily verified that  $\text{wt}(D_0) \leq 2n \cdot \lambda^*$ , where  $\lambda^*$  is the objective function value of the optimal LP solution. Furthermore, we have that each disk  $d \in D$  is  $n$ -covered by  $D_0$ .

In the next phase, our algorithm will recursively apply Theorem 3 to obtain a successively sparse dominating set. For the  $i$ th application of the theorem, we set  $L_i = \log L_{i-1}$ , for  $i = 2, 3, \dots, t$  to obtain a set  $D_i \subseteq D_{i-1}$ . For the first application, we set  $L_1 = n$ . Details of the approximation ratio are omitted from this extended abstract.

## 4 Concluding Remarks and Open Questions

Given the negative result of Marx [15] which shows that even for the simple case of unweighted unit disk graph, an EPTAS for the problem would contradict the *exponential time hypothesis* [13]<sup>2</sup>, it is unlikely that the dependence of  $1/\varepsilon$  as an exponent of  $n$  on the running time for the PTAS can be improved to, say,  $f(1/\varepsilon) \cdot n^{O(1)}$ . However, the running time of the local search PTAS is  $n^{O(1/\varepsilon^2)}$ . Can this be improved to  $n^{O(1/\varepsilon)}$ ? In our work, we have made no attempt to improve the running time.

For the weighted case, we are only able to show a constant integrality gap for the lower bound despite numerous attempts. Thus, we believe that the right upper bound for the approximation factor is  $O(1)$ .

*Acknowledgments.* We thank Sariel Har-Peled and Kasturi Varadarajan for suggesting the use of weighted Voronoi diagrams for the unweighted case, and we thank Kasturi Varadarajan for pointing out the connection between weighted set cover and weighted dominating set. We also thank Mohammad Salavatipour for his support and many valuable discussions.

---

<sup>2</sup> Marx [15] actually shows something stronger.

## References

1. Ambühl, C., Erlebach, T., Mihalák, M., Nunkesser, M.: Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 3–14. Springer, Heidelberg (2006)
2. Aurenhammer, F.: Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23(3), 345–405 (1991)
3. Chan, T.M.: Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms* 46(2), 178–189 (2003)
4. Chan, T.M., Har-Peled, S.: Approximation algorithms for maximum independent set of pseudo-disks. In: SoCG 2009, pp. 333–340 (2009)
5. Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 192–203. Springer, Heidelberg (2004)
6. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. *Discrete Mathematics* 86(1-3), 165–177 (1990)
7. Clarkson, K.L.: Applications of random sampling in computational geometry, II. In: Symposium on Computational Geometry, pp. 1–11 (1988)
8. Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.* 34(6), 1302–1323 (2005)
9. Erlebach, T., van Leeuwen, E.J.: Domination in geometric intersection graphs. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 747–758. Springer, Heidelberg (2008)
10. Feige, U.: A threshold of  $\ln n$  for approximating set cover. *J. ACM* 45(4), 634–652 (1998)
11. Gibson, M., Pirwani, I.A.: Approximation algorithms for dominating set in disk graphs. *CoRR* abs/1004.3320 (2010)
12. Hunt III, H.B., 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)
13. Impagliazzo, R., Paturi, R.: On the complexity of k-sat. *J. Comput. Syst. Sci.* 62(2), 367–375 (2001)
14. Kammer, F., Tholey, T.: Approximation algorithms for intersection graphs. To appear in APPROX-RANDOM (2010)
15. Marx, D.: On the optimality of planar and geometric approximation schemes. In: FOCS, pp. 338–348 (2007)
16. Mustafa, N.H., Ray, S.: PTAS for geometric hitting set problems via local search. In: SoCG, pp. 17–22 (2009)
17. Nieberg, T., Hurink, J., Kern, W.: Approximation schemes for wireless networks. *ACM Transactions on Algorithms* 4(4), 1–17 (2008)
18. Pandit, S., Pemmaraju, S., Varadarajan, K.: Approximation algorithms for domatic partition. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. LNCS, vol. 5687, pp. 312–325. Springer, Heidelberg (2009)
19. Varadarajan, K.: Weighted geometric set cover via quasi-uniform sampling. In: STOC 2010, pp. 641–648 (2010)
20. Vazirani, V.V.: Approximation Algorithms. Springer, New York (2001)

# Minimum Vertex Cover in Rectangle Graphs

Reuven Bar-Yehuda<sup>1</sup>, Danny Hermelin<sup>2</sup>, and Dror Rawitz<sup>3</sup>

<sup>1</sup> Department of Computer Science, Technion IIT, Haifa 32000, Israel

[reuven@cs.technion.ac.il](mailto:reuven@cs.technion.ac.il)

<sup>2</sup> Max-Planck-Institut für Informatik, Saarbrücken, Germany

[hermelin@mpi-inf.mpg.de](mailto:hermelin@mpi-inf.mpg.de)

<sup>3</sup> School of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel

[rawitz@eng.tau.ac.il](mailto:rawitz@eng.tau.ac.il)

**Abstract.** We consider the MINIMUM VERTEX COVER problem in intersection graphs of axis-parallel rectangles on the plane. We present two algorithms: The first is an EPTAS for non-crossing rectangle families, rectangle families  $\mathcal{R}$  where  $R_1 \setminus R_2$  is connected for every pair of rectangles  $R_1, R_2 \in \mathcal{R}$ . This algorithm extends to intersection graphs of pseudo-disks. The second algorithm achieves a factor of  $(1.5 + \varepsilon)$  in general rectangle families, for any fixed  $\varepsilon > 0$ , and works also for the weighted variant of the problem. Both algorithms exploit the plane properties of axis-parallel rectangles.

## 1 Introduction

In this paper we are concerned with the MINIMUM RECTANGLE VERTEX COVER problem: Given a set  $\mathcal{R} = \{R_1, \dots, R_n\}$  of (weighted) axis-parallel rectangles in the plane, find a minimum size (weight) subset of rectangles in  $\mathcal{R}$  whose removal leaves the remaining rectangles in  $\mathcal{R}$  pairwise disjoint, *i.e.* no pair of remaining rectangles share a common point. This problem is a special case of the classical MINIMUM VERTEX COVER problem, which asks to find a minimum weight subset of vertices in a given graph, whose removal leaves the graph without edges. When the input graph is a *rectangle graph*, an intersection graph of axis parallel rectangles in the plane, and the rectangle representation of the input graph is given alongside the input, MINIMUM VERTEX COVER becomes MINIMUM RECTANGLE VERTEX COVER.

MINIMUM VERTEX COVER is one of the most extensively studied combinatorial problems in computer science, a study dating back to König's classical early 1930s result [1], and probably even prior to that. Karp proved that the problem is NP-complete in his famous list of fundamental NP-complete problems [2], while Garey and Johnson extensively used MINIMUM VERTEX COVER as an intermediate problem in many of their early NP-completeness reductions [3]. Since then, MINIMUM VERTEX COVER played a pivoting role in the development of both approximation algorithms [4, 5], and the theory of parameterized complexity [6], the two main disciplines for coping with the widespread phenomena of NP-hardness.

From the perspective of approximation algorithms, MINIMUM VERTEX COVER has many polynomial-time algorithms achieving an approximation ratio of 2 [7,8,9,10], the first of these given in Nemhauser and Trotter's fundamental paper [10]. Moreover, the problem is known to be approximable in within  $2 - \frac{\lg \lg n}{2 \lg n}$  [11,12], within  $2 - \frac{\ln \ln n}{\ln n} (1 - o(1))$  [13] and even within  $2 - \Theta(\frac{1}{\sqrt{\log n}})$  [14]. On the other hand, it is also known that MINIMUM VERTEX COVER is inapproximable within a factor of 1.36, unless P=NP [15]. There are however many natural special-case graph classes for which one can improve on this barrier. For instance, in the class of interval graphs, which can be thought of as one dimensional analogs of rectangle graphs, MINIMUM VERTEX COVER is polynomial-time solvable [16]. In planar graphs, the problem is known to admit a polynomial-time approximation scheme (PTAS) [17,18,19], and even an efficient PTAS (EPTAS) due to Baker's seminal framework for NP-hard planar graph problems [20].

The dual problem of MINIMUM RECTANGLE VERTEX COVER is the MAXIMUM RECTANGLE INDEPENDENT SET problem: Given a family of axis-parallel rectangles in the plane, find a maximum size (or weight) subset of pairwise disjoint rectangles. This problem has been extensively studied in the computational geometry community, and has several applications in data mining [21,22], automated label placement [23,24,25], and in network resource allocation with advance reservation for line topologies [21,26], which also apply to MINIMUM VERTEX COVER in rectangle graphs (see below). Fowler *et al.* [27] showed that MAXIMUM INDEPENDENT SET in rectangle graphs is NP-hard, implying the NP-hardness of MINIMUM VERTEX COVER in rectangle graphs. Asano [28] showed that MAXIMUM INDEPENDENT SET and MINIMUM VERTEX COVER remain NP-hard even in intersection graphs of unit squares. There have been several  $O(\lg n)$  approximation algorithms independently suggested for this problem [23,29,30,22]. Lewin-Eitan *et al.* [26] devised a  $4q$ -approximation algorithm for the problem, where  $q$  is the size of the maximum clique in the input graph. Recently, Chalermsook and Chuzhoy [21] were able to break the  $\lg n$  approximation barrier by devising a sophisticated  $O(\lg \lg n)$  randomized approximation algorithm. A simpler  $O(\lg n / \lg \lg n)$ -approximation algorithm was given in [31]. There are also many special cases in which MAXIMUM RECTANGLE INDEPENDENT SET admits a polynomial-approximation scheme (PTAS) [23,29,32,33].

In contrast to the vast amount of research devoted to MAXIMUM RECTANGLE INDEPENDENT SET there has been surprisingly very little focus on the MINIMUM RECTANGLE VERTEX COVER problem. Nevertheless, some of the results for MAXIMUM RECTANGLE INDEPENDENT SET carry through to MINIMUM RECTANGLE VERTEX COVER. For instance, the result of Fowler *et al.* [27] implies that MINIMUM RECTANGLE VERTEX COVER is NP-hard. Also, by applying the Nemhauser and Trotter Theorem (see Section 2) as a preprocessing step, any PTAS for MAXIMUM RECTANGLE INDEPENDENT SET can be converted into a PTAS for MINIMUM RECTANGLE VERTEX COVER. Thus, the results in [23] imply that MINIMUM RECTANGLE VERTEX COVER has a PTAS when all rectangles have equal height, while [32] gives a PTAS when all rectangles are squares. Erlebach *et al.* [33] gave an explicit PTAS for MINIMUM RECTANGLE

VERTEX COVER in bounded aspect-ratio rectangle families (without using the Nemhauser and Trotter procedure), and an EPTAS for MINIMUM RECTANGLE VERTEX COVER in unit squares and squares is implied by [34] and [35], respectively. Finally, we mention the work by Chan and Har-Peled [31] who devised a PTAS for MAXIMUM INDEPENDENT SET in families of *pseudo-disks*, which are families of regions on the plane such that the boundaries of every pair of regions intersect at most twice. This result implies a PTAS for MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families.

**Related work.** MINIMUM VERTEX COVER and its dual counterpart MAXIMUM INDEPENDENT SET have been previously studied in many geometric intersection graphs other than rectangle graphs. Gavril [16] gave a polynomial-time algorithm for both of these problems in chordal graphs, intersection graphs of subgraphs of a tree. Apostolico *et al.* [36] gave a polynomial-time algorithm for these two problems in intersection graphs of chords on a circle, which were later improved by Cenek and Stuart [37], while Golumbic and Hammer [38] gave a polynomial-time algorithm for intersection graphs of arcs on a circle which was later improved in [39]. A good survey of many generalizations of these results can be found in [40][41]. Hochbaum and Maass, and later Chlebík and Chlebíková, considered intersection graphs of  $d$ -dimensional boxes in  $\mathbb{R}^d$  [42][43], while Erlebach *et al.* [33] considered intersection graphs of general fat objects in the plane. In [44][45], approximation algorithms were suggested for MAXIMUM INDEPENDENT SET and MINIMUM VERTEX COVER in the class of multiple-interval graphs.

**Applications and motivation.** Automated label placement is a central problem in geographic information systems which has been extensively studied in various settings [23][24][25]. The basic problem is to place labels around points in a geographic map, where the labels are often assumed to be rectangles [23] which are allowed to be positioned at specific places adjacent to their corresponding points in the map. The usual criterion for a legal placement is that all rectangles are pairwise disjoint. Subject to this constraint, a natural optimization criteria is to minimize the number of labels to be removed so as the remaining labels form a legal placement. This is exactly MINIMUM RECTANGLE VERTEX COVER.

MINIMUM RECTANGLE VERTEX COVER can also be used to model shared-resource scheduling scenarios where requests are given in advance to the system. Consider the typical critical-section scheduling problem occurring in all modern operating system: A set of programs request access to a shared resource in memory for read/write purposes. The goal of the operating system is to serve as many requests as possible, so long as no two programs access the same memory entries simultaneously, to avoid obvious data-consistency hazards. In a simplified variant of this problem, one can assume that all programs have a single request to fixed array of registers in memory, and this request occurs during a fixed interval of their running time. If these requests are known beforehand, the problem of minimizing the number of programs not to be served can naturally be modeled as MINIMUM RECTANGLE VERTEX COVER by using the  $x$ -axis to measure the shared memory array, and the  $y$ -axis to measure program execution-time.

**Results and techniques.** In this paper we present two approximation algorithms for the MINIMUM RECTANGLE VERTEX COVER problem. For a pair of rectangles  $R_1$  and  $R_2$  in our input set of rectangles  $\mathcal{R}$ , we say that  $R_1$  and  $R_2$  *cross* if they intersect, but neither rectangle contains a corner of the other rectangle. This is equivalent to requiring that  $R_1 \setminus R_2$  is connected for every  $R_1, R_2 \in \mathcal{R}$ . (We assume w.l.o.g. that the rectangles are in general position.) We say that  $\mathcal{R}$  is *non-crossing* if there is no pair of crossing rectangles in  $\mathcal{R}$ . Our first algorithm is an EPTAS for MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families:

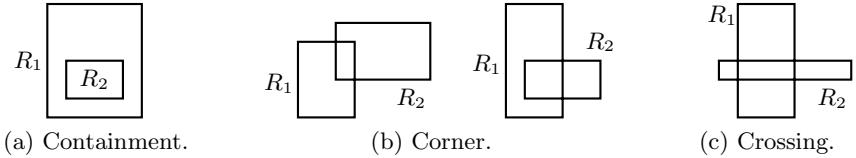
**Theorem 1.** *Given any  $\varepsilon > 0$ , MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families can be approximated within  $(1+\varepsilon)$  in  $2^{\text{poly}(1/\varepsilon)} \cdot \text{poly}(n)$  time.*

We mention that MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families is NP-hard according to [28]. Theorem 1 generalizes the PTAS result of Agarwal *et al.* [23] and Chan [32] for squares and equal height rectangles, and it also handles several families of rectangles which cannot be handled by the PTAS of Erlebach *et al.* [33]. In terms of time complexity, our algorithm dramatically improves on all these algorithms, and also on the algorithm of Chan and Har-Peled [31], since all there algorithms have running-times of the form  $n^{\text{poly}(1/\varepsilon)}$ . Furthermore, our algorithm easily extends to intersection graphs of pseudo-disks, which is the class of graphs considered in [31].

The novelty behind the algorithm in Theorem 1 lies in its usage of the *arrangement graph* [46] of the input set of rectangles  $\mathcal{R}$ . This graph is defined by considering all intersection points occurring on boundary of rectangles as vertices, and the boundary curves connecting them as edges. By its definition, the arrangement graph of a rectangle family is planar and 4-regular, and thus has a very convenient structure. However, there is no immediate way to translate approximate vertex-covers in the arrangement graph  $A_{\mathcal{R}}$  of  $\mathcal{R}$ , to vertex covers in the corresponding rectangle graph  $G_{\mathcal{R}}$ . Nevertheless, we show that we can translate tree-decompositions in  $A_{\mathcal{R}}$  to tree-decompositions in  $G_{\mathcal{R}}$  of roughly the same width, and this allows with some technical effort to simulate Baker's algorithm [20]. We believe that the arrangement graph can be a useful tool in other intersection-graph problems.

The second algorithm we present in this paper applies to general rectangle families and can handle also weights. This algorithm exploits the observation that the rectangles of a triangle-free rectangle graph can be partitioned into two classes, where no pair of rectangles cross in each class. This, in combination with Theorem 1 and the fact that we can clean all triangles from our input graph at cost of a 1.5 factor to the approximation guarantee, gives us Theorem 2 below for the unweighted case. For the weighted case, we use the additional observation that triangle-free non-crossing rectangle graphs are planar, and so we can use Baker's algorithm [20] directly.

**Theorem 2.** *Given any  $\varepsilon > 0$ , MINIMUM RECTANGLE VERTEX COVER can be approximated within a factor of  $1.5 + \varepsilon$  in  $2^{\text{poly}(1/\varepsilon)} \cdot \text{poly}(n)$  time.*



**Fig. 1.** Possible intersections between two rectangles  $R_1$  and  $R_2$

**Overview.** The remainder of the paper is organized as follows. Section 2 contains notation and terminology and a short explanation about the Nemhauser-Trotter reduction [10]. Our EPTAS for MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families is given in Section 3, and our  $(1.5 + \varepsilon)$ -approximation algorithm for MINIMUM RECTANGLE VERTEX COVER is given in Section 4. Several proofs are omitted due to space limitations.

## 2 Preliminaries

We denote our input set of axis-parallel rectangles in the plane by  $\mathcal{R} = \{R_1, \dots, R_n\}$ . We assume that each rectangle  $R$  is specified by two intervals  $R = (X, Y)$ , where  $X$  is the projection of  $R$  on the  $x$ -axis, and  $Y$  is the projection of  $R$  on the  $y$ -axis. We assume w.l.o.g. that  $\mathcal{R}$  is in *general position*, i.e. that all intervals in the specification of  $\mathcal{R}$  have different endpoints. The *boundary* of a rectangle  $R$  is the set of all points with minimum and maximum  $x$ -coordinate values, and minimum and maximum  $y$ -coordinate values. Two rectangles  $R_1 = (X_1, Y_1)$  and  $R_2 = (X_2, Y_2)$  *intersect*, denoted  $R_1 \cap R_2 \neq \emptyset$ , if they share a common point, i.e. if  $X_1 \cap X_2 \neq \emptyset$  and  $Y_1 \cap Y_2 \neq \emptyset$ . Two non-intersecting rectangles are said to be *disjoint*. There are three possible types of intersections between two rectangles  $R_1$  and  $R_2$ :

1. *Containment intersection*:  $R_1$  contains  $R_2$ . In this case  $R_1$  contains all corners of  $R_2$ , and the boundaries of  $R_1$  and  $R_2$  do not intersect (Fig. 1a).
2. *Corner intersection*:  $R_1$  contains one or two corners of  $R_2$ . In this case the boundaries of  $R_1$  and  $R_2$  intersect exactly twice (Fig. 1b).
3. *Crossing intersection*: the intersection of  $R_1$  and  $R_2$  does not involve any corners. In this case, the boundaries of  $R_1$  and  $R_2$  intersect four times (Fig. 1c).

Given a graph  $G$ , we use  $V(G)$  and  $E(G)$  to denote its vertex set and edge set, respectively. For a given vertex-subset  $V \subseteq V(G)$ , we let  $G[V]$  denote the subgraph of  $G$  induced by  $V$ , i.e. the subgraph with vertex-set  $V$  and edge-set  $\{\{u, v\} \in E(G) : u, v \in V\}$ . We write  $G - V$  to denote the induced subgraph  $G[V(G) \setminus V]$ . We will also be considering vertex-weighted graphs, i.e. graphs  $G$  equipped with a weight-function  $w : V(G) \rightarrow \mathbb{Q}$ . A *vertex cover* of  $G$  is a subset of vertices  $C \subseteq V(G)$  such that  $\{u, v\} \cap C \neq \emptyset$  for any edge  $\{u, v\} \in E(G)$ . For a non-negative real  $\alpha \in \mathbb{R}^{\geq 0}$ , an  $\alpha$ -*approximate vertex cover* of  $G$  is a vertex

cover  $C$  with  $|C| \leq \alpha \cdot \text{OPT}$  (or  $w(C) \leq \alpha \cdot \text{OPT}$  in the weighted case), where  $\text{OPT}$  is the size (weight) of a minimum vertex cover of  $G$ . The *intersection graph*  $G_{\mathcal{R}}$  corresponding to our input set of rectangles  $\mathcal{R}$  is the graph with vertex-set  $V(G_{\mathcal{R}}) = \mathcal{R}$ , and edge-set  $E(G_{\mathcal{R}}) = \{\{R_1, R_2\} : R_1 \cap R_2 \neq \emptyset\}$ .

We will be using an important tool due to Nemhauser and Trotter [10] that allows us to focus on graphs whose entire vertex-set already constitutes a good approximate vertex-cover:

**Theorem 3 (Nemhauser&Trotter [10]).** *There is a polynomial-time algorithm that given a vertex-weighted graph  $G$ , computes a vertex set  $V \subseteq V(G)$  such that: (i)  $V$  is a 2-approximate vertex-cover of  $G[V]$ , and (ii) any  $\alpha$ -approximate vertex cover of  $G[V]$  can be converted in polynomial-time to an  $\alpha$ -approximate vertex cover of  $G$ .*

Finally, we will be using the notion of treewidth and tree-decomposition of graphs, introduced in the form below by Robertson and Seymour [47].

**Definition 1 (Treewidth [47]).** A tree decomposition of a graph  $G$  is a pair  $(\mathcal{T}, \mathcal{X})$ , where  $\mathcal{X} \subseteq 2^{V(G)}$  is a family of vertex subsets of  $G$ , and  $\mathcal{T}$  is a tree over  $\mathcal{X}$ , satisfying the following conditions: (i)  $\bigcup_{X \in \mathcal{X}} G[X] = G$ , and (ii)  $\mathcal{X}_v = \{X \in \mathcal{X} : v \in X\}$  is connected in  $\mathcal{T}$  for all  $v \in V(G)$ . The width of  $\mathcal{T}$  is  $\max_{X \in \mathcal{X}} |X| - 1$ . The treewidth of  $G$ , denoted  $\text{tw}(G)$ , is the minimum width over all tree decompositions of  $G$ .

### 3 An EPTAS for Non-crossing Rectangle Graphs

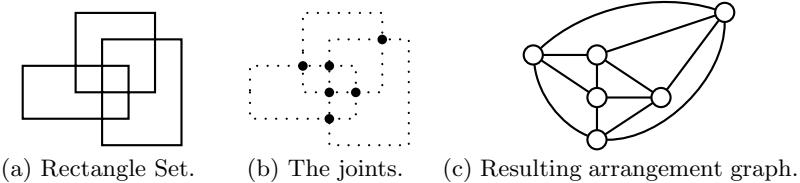
In this section we present an EPTAS for MINIMUM RECTANGLE VERTEX COVER in unweighted non-crossing rectangle families. This algorithm extends to intersection graphs of pseudo-disks.

Let  $\mathcal{R}$  denote our input set of unweighted non-crossing rectangles. The first step of our algorithm is to clean  $\mathcal{R}$  from containment intersections and pairwise intersecting subsets of size greater than some constant  $q \geq 2$  to be chosen later. This can be done using standard techniques, and allows us to gain substantial structure at a small cost to the approximation factor of our algorithm.

**Lemma 1.** *Suppose that MINIMUM RECTANGLE VERTEX COVER in corner-intersecting rectangle families with no  $q + 1$  pairwise intersecting rectangles can be approximated within a factor of  $\alpha$ . Then MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families can be approximated within a factor of  $\max\{\alpha, 1 + 1/q\}$  in polynomial-time.*

Due to Lemma 1, we can henceforth assume that  $\mathcal{R}$  contains only corner intersections, and that the maximum clique in  $G_{\mathcal{R}}$  is of size at most  $q$ . We also apply the Nemhauser&Trotter algorithm (Theorem 3) on  $G_{\mathcal{R}}$  after applying Lemma 1, and so we assume that  $\mathcal{R}$  is a 2-approximate vertex cover of  $G_{\mathcal{R}}$ .

The main idea of algorithm is as follows. We will construct the so-called *arrangement graph*  $A_{\mathcal{R}}$  of  $\mathcal{R}$  which is build by considering all intersection points



**Fig. 2.** A set  $\mathcal{R}$  of corner intersecting rectangles and its arrangement graph  $A_{\mathcal{R}}$

occurring on boundaries of rectangles as vertices, and the boundary curves connecting them as edges. By this construction,  $A_{\mathcal{R}}$  is a planar graph, and as such, it has very specific structure. The most tempting approach is to use Baker’s EP-TAS for MINIMUM VERTEX COVER in planar graphs on  $A_{\mathcal{R}}$ , and to convert the  $(1 + \varepsilon)$ -approximate vertex-cover of  $A_{\mathcal{R}}$  to a  $(1 + \varepsilon')$ -approximate vertex-cover of  $G_{\mathcal{R}}$ . Unfortunately, this attempt fails, since the natural transformation from vertices of  $A_{\mathcal{R}}$  to rectangles of  $G_{\mathcal{R}}$  produces the entire set of rectangles  $\mathcal{R}$  on any vertex-cover of  $A_{\mathcal{R}}$ . We therefore take an alternative route. The basic idea is to mimic Baker’s algorithm by using the observation that tree-decompositions of  $A_{\mathcal{R}}$  correspond to tree-decompositions of  $G_{\mathcal{R}}$  of roughly the same width. Thus, instead of applying Baker’s algorithm on  $A_{\mathcal{R}}$  as a black-box, we can simulate its steps directly on  $G_{\mathcal{R}}$ . Using then an extension of Baker’s analysis, we can show that this approach indeed gives us our the desired  $(1 + \varepsilon)$  approximation factor.

### 3.1 The Arrangement Graph

In this section we present several properties of *arrangement graphs* [46] of rectangle families. An intersection of two rectangle boundaries is called a *joint*. The *arrangement graph*  $A_{\mathcal{R}}$  of a rectangle family  $\mathcal{R}$  is the multi-graph that is defined as follows: The vertex set of  $A_{\mathcal{R}}$  is the set of joints. The edge set of  $A_{\mathcal{R}}$  consists of the rectangle boundary fragments, namely  $\{u, v\}$  is an edge in  $A_{\mathcal{R}}$  if and only if  $u$  and  $v$  are two joints located on the boundary of some rectangle such that no other joint is located on the boundary between them. It is not difficult to see that the arrangement graph defined as above is in fact planar and 4-regular (see example in Fig. 2).

For a given subset of joints  $J \subseteq V(A_{\mathcal{R}})$ , the set of rectangles that is induced by  $J$  is defined by  $\mathcal{R}(J) = \{R \in \mathcal{R} : \exists j \in J \text{ s.t. } j \text{ is on the boundary of } R\}$ . The following lemma is immediate from the fact that  $\mathcal{R}$  is in general position.

**Lemma 2.**  $|\mathcal{R}(J)| \leq 2|J|$  for any set of joints  $J \subseteq V(A_{\mathcal{R}})$ .

The following lemma states that the number of joints is  $A_{\mathcal{R}}$  is linear in  $|\mathcal{R}|$ .

**Lemma 3.**  $|V(A_{\mathcal{R}})| \leq 4q \cdot |\mathcal{R}|$ .

### 3.2 Baker’s Algorithm

Our algorithm for MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families simulates Baker’s classical algorithm for MINIMUM VERTEX

COVER in planar graphs [20] on the arrangement graph  $A_{\mathcal{R}}$  of  $\mathcal{R}$ . The main idea behind Baker's approach is the observation that given a planar graph  $G$  and any positive integer  $k$ , one can partition the vertex-set of  $G$  into  $k$  classes such that deleting each the vertices in class results in subgraph of treewidth at most  $3k$  (see Lemma 4 below). Combining this observation along with the well-known algorithm for MINIMUM VERTEX COVER in bounded treewidth graphs (see e.g. [48]), gives an EPTAS for MINIMUM VERTEX COVER in planar graphs.

**Lemma 4 (Baker [20]).** *Given a planar graph  $G$  and an integer  $k$ , one can partition in polynomial-time  $V(G)$  into  $k$  subsets  $V_1, \dots, V_k$  such that  $\text{tw}(G - V_i) \leq 3k$  for all  $i$ ,  $1 \leq i \leq k$ .*

In order to properly simulate Baker's approach on MINIMUM RECTANGLE VERTEX COVER we will need a slightly more general framework. In particular, our algorithm will not necessarily produce a partition of the vertex-set of  $G_{\mathcal{R}}$ . Also, our algorithm will produce vertex-sets whose deletion results in a subgraph of  $G_{\mathcal{R}}$  with treewidth slightly more than  $3k$ . Nevertheless, it is not difficult to show that a slight relaxation of these two requirements does not alter Baker's analysis too much:

**Lemma 5.** *Let  $G$  be a graph with  $n$  vertices, and let  $c_1$  and  $c_2$  be two fixed positive integers. Suppose that there is a polynomial-time algorithm that, given  $G$  and a positive integer  $k$ , produces vertex-sets  $U_1, \dots, U_k$  with the following properties: (1)  $\bigcup_i U_i = V(G)$ ; (2)  $\sum_i |U_i| \leq c_1 \cdot n$ ; and (3)  $\text{tw}(G - U_i) \leq c_2 \cdot k$  for every  $i$ . Then one can compute a vertex cover of  $G$  within a factor of  $(1 + \varepsilon)$  in  $2^{\text{poly}(1/\varepsilon)} \cdot \text{poly}(n)$  time, for any given  $\varepsilon > 0$ .*

### 3.3 Our Algorithm

We are now in position to describe our EPTAS. The key lemma we need is Lemma 6 below that allows us to convert tree-decompositions of  $A_{\mathcal{R}}$  to a tree-decompositions of  $G_{\mathcal{R}}$  of approximately the same width.

**Lemma 6.**  $\text{tw}(G_{\mathcal{R}}) \leq 2 \cdot \text{tw}(A_{\mathcal{R}}) + 1$ .

*Proof.* Let  $(\mathcal{T}, \mathcal{X})$  is a tree-decomposition of  $A_{\mathcal{R}}$  whose width it  $\text{tw}(A_{\mathcal{R}})$ . Now let  $\mathcal{X}' = \{\mathcal{R}(X) : X \in \mathcal{X}\}$ , and let  $\mathcal{T}'$  be a tree over  $\mathcal{X}'$  with an edge  $\{\mathcal{R}(X_1), \mathcal{R}(X_2)\}$  for every edge  $\{X_1, X_2\}$  in  $\mathcal{T}$ . We show that  $(\mathcal{T}', \mathcal{X}')$  is a tree-decomposition of  $G_{\mathcal{R}}$ , namely that  $(\mathcal{T}', \mathcal{X}')$  satisfies all requirements of Definition 1. First, observe that any rectangle has at least two corresponding joints since we assume there are no isolated rectangles in  $\mathcal{R}$ . Furthermore, if two rectangles intersect, then there is a joint  $j \in V(A_{\mathcal{R}})$  that corresponds to both these rectangles. Hence, for every edge  $\{R_1, R_2\} \in E(G_{\mathcal{R}})$ , there is at least one node in  $\mathcal{X}'$  which contains both  $R_1$  and  $R_2$ . Thus,  $\bigcup_{X \in \mathcal{X}'} G_{\mathcal{R}}[X] = G_{\mathcal{R}}$ .

Now suppose there is some rectangle  $R$  which is contained in two nodes  $\mathcal{R}(X_1)$  and  $\mathcal{R}(X_2)$  of  $\mathcal{T}'$ . Then  $R$  has two joints  $j_1$  and  $j_2$  with  $j_1 \in X_1$  and  $j_2 \in X_2$ . By construction, there is a path  $j_1, i_1, \dots, i_r, j_2$  connecting  $j_1$  to  $j_2$  in  $A_{\mathcal{R}}$ , where  $i_1, \dots, i_r$  are all joints of  $R$ . Since  $(\mathcal{T}, \mathcal{X})$  is a proper tree decomposition of  $A_{\mathcal{R}}$ , it

follows that there is a path  $X_1, Y_1, \dots, Y_{r'}, X_2$  connecting  $X_1$  and  $X_2$  in  $\mathcal{T}$ , with  $Y_i \cap \{j_1, i_1, \dots, i_r, j_2\} \neq \emptyset$  for each  $i$ ,  $1 \leq i \leq s$ . Thus, each node in the path  $\mathcal{R}(X_1), \mathcal{R}(Y_1), \dots, \mathcal{R}(Y_s), \mathcal{R}(X_2)$  connecting  $\mathcal{R}(X_1)$  and  $\mathcal{R}(X_2)$  in  $\mathcal{T}'$  contains  $R$ , and since  $R$ ,  $\mathcal{R}(X_1)$ , and  $\mathcal{R}(X_2)$  were chosen arbitrarily, this shows that for each  $R \in \mathcal{R}$ :  $\{\mathcal{R}(X) \in \mathcal{X}' : R \in X\}$  is connected in  $\mathcal{T}'$ . Thus, both requirements of Definition 11 are fulfilled by  $(\mathcal{T}', \mathcal{X}')$ .

Finally, observe that due to Lemma 2,  $\max_{X' \in \mathcal{X}'} |X'| \leq 2 \max_{X \in \mathcal{X}} |X|$ . It follows that the width of  $(\mathcal{T}', \mathcal{X}')$  is at most  $2\text{tw}(A_{\mathcal{R}}) + 1$ , and we are done.  $\square$

Our algorithm consists of the following steps:

1. Set  $q = \lceil 1/\varepsilon \rceil$  and  $k = \lceil 8q/\varepsilon \rceil = \lceil 8/\varepsilon^2 \rceil$ .
2. Apply Lemma 10 so that  $\mathcal{R}$  does not have any containment intersections and no pairwise intersecting subsets of rectangles of size greater than  $q$ .
3. Apply the Nemhauser&Trotter Theorem on  $G_{\mathcal{R}}$ , and let  $\mathcal{R}' \subseteq \mathcal{R}$  denote the resulting subset of rectangles.
4. Construct the arrangement graph  $A_{\mathcal{R}'}$  corresponding to  $\mathcal{R}'$ , and partition  $A_{\mathcal{R}'}$  into  $k$  subsets  $V_1, \dots, V_k$  using Lemma 4.
5. Use Lemma 5 on  $G_{\mathcal{R}'}$  with  $U_i = \mathcal{R}'(V_i)$ , for every  $i$ .

Observe that the arrangement graph of  $\mathcal{R}' \setminus U_i$  is a subgraph of  $A_{\mathcal{R}'} - V_i$ . Hence, according to Lemmas 2, 3 and 6 above,  $U_1, \dots, U_k$  satisfy the three conditions of Lemma 5. Thus the above algorithm outputs a  $(1 + \varepsilon)$ -approximate vertex-cover of  $G_{\mathcal{R}}$  in  $2^{\text{poly}(1/\varepsilon)} \cdot \text{poly}(n)$  time. This proves Theorem 11.

Finally, we mention that our EPTAS can be modified to deal with intersection graphs of pseudo-disks. Specifically, in Step 2, instead of removing cliques, we remove point cliques, namely subsets of pseudo-disks  $\mathcal{Q}$  such that  $\bigcap_{R \in \mathcal{Q}} R \neq \emptyset$ . This is sufficient, since the number of joints in the arrangement graph  $A_{\mathcal{D}}$  of a set  $\mathcal{D}$  of pseudo-disks, where no point is contained in more than  $q$  pseudo-disks, is  $O(q \cdot |\mathcal{D}|)$  [49].

## 4 General Rectangle Graphs

In this section we present an algorithm for MINIMUM RECTANGLE VERTEX COVER in general rectangle families. Our algorithm achieves an approximation factor of  $1.5 + \varepsilon$ , for any given  $\varepsilon > 0$ , in time  $2^{\text{poly}(1/\varepsilon)} \cdot \text{poly}(n)$ , and works also for the weighted variant of the problem.

We begin with the unweighted case, and with the following lemma which relies on an observation already made by Lewin-Eytan *et al.* [26]. A rectangle family is said to be *triangle-free* if there are no three pairwise intersecting rectangles in the family.

**Lemma 7.** *Any triangle-free rectangle family can be partitioned into two non-crossing subsets in polynomial time.*

Observe that Lemma 7 is already enough, along with the results in Section 3, to obtain our desired  $1.5 + \varepsilon$  approximation factor. The algorithm proceeds in the following six steps, given a rectangle family  $\mathcal{R}$  and  $\varepsilon > 0$ :

1. Apply Lemma 11 to obtain a triangle-free rectangle family  $\mathcal{R}' \subseteq \mathcal{R}$ .
2. Apply the Nemhauser&Trotter algorithm to obtain a subset  $\mathcal{R}'' \subseteq \mathcal{R}'$  which is a 2-approximate vertex-cover of  $G_{\mathcal{R}''}$ .
3. Use Lemma 7 to obtain a partitioning  $\{\mathcal{R}_1, \mathcal{R}_2\}$  of  $\mathcal{R}''$ , where both  $\mathcal{R}_1$  and  $\mathcal{R}_2$  are non-crossing.
4. Compute an  $\varepsilon$ -approximate vertex-cover  $\mathcal{C}_1$  of  $G_{\mathcal{R}_1}$  and an  $\varepsilon$ -approximate vertex-cover  $\mathcal{C}_2$  of  $G_{\mathcal{R}_2}$  using the EPTAS of Section 3.
5. Use the best of the two vertex-covers  $\mathcal{R}_1 \cup \mathcal{C}_2$  and  $\mathcal{R}_2 \cup \mathcal{C}_1$  for  $G_{\mathcal{R}''}$ , along with the Nemhauser&Trotter Theorem to compute a vertex-cover of  $G_{\mathcal{R}''}$ .
6. Add the removed rectangles as required by Lemma 11 to obtain a vertex-cover for  $G_{\mathcal{R}}$ .

The fact that this algorithm outputs a vertex-cover which is a factor of  $1.5 + \varepsilon$  off the optimum follows from a similar analysis used in Lemma 5. Clearly, both  $\mathcal{R}_1 \cup \mathcal{C}_2$  and  $\mathcal{R}_2 \cup \mathcal{C}_1$  are vertex covers for  $G_{\mathcal{R}''}$ . Furthermore, letting  $\text{OPT}$ ,  $\text{OPT}_1$ , and  $\text{OPT}_2$  denote the size of the minimum vertex-covers of  $G_{\mathcal{R}''}$ ,  $G_{\mathcal{R}_1}$ , and  $G_{\mathcal{R}_2}$  respectively, we get:  $|\mathcal{R}_1 \cup \mathcal{C}_2| + |\mathcal{R}_2 \cup \mathcal{C}_1| \leq |\mathcal{R}| + (1 + \varepsilon)\text{OPT}_1 + (1 + \varepsilon)\text{OPT}_2 \leq 2\text{OPT} + (1 + \varepsilon)\text{OPT} = (3 + \varepsilon)\text{OPT}$ . Thus, the minimum of both  $\mathcal{R}_1 \cup \mathcal{C}_2$  and  $\mathcal{R}_2 \cup \mathcal{C}_1$  gives a  $(1.5 + \varepsilon)$ -approximate vertex-cover for  $G_{\mathcal{R}''}$ . Applying the Nemhauser&Trotter Theorem along with Lemma 11 shows that the algorithm above outputs a  $(1.5 + \varepsilon)$ -approximate vertex-cover for  $G_{\mathcal{R}}$ .

For the weighted variant of the problem, we observe that all steps of the algorithm above, apart from Step 4, can be applied also in the weighted case. For the first step we use a weighted version of Lemma 11, which can be obtained by a standard application of the local-ratio technique [11] (see e.g. [45]). All other steps have immediate weighted counterparts. To replace Step 4, we use the following observation that the intersection graph of any triangle-free non-crossing rectangle family is planar:

**Lemma 8.** *If  $\mathcal{R}$  is triangle-free and non-crossing, then  $G_{\mathcal{R}}$  is planar.*

Thus, according to Lemma 8 above, we can apply Baker's algorithm for MINIMUM VERTEX COVER in planar graphs instead of our EPTAS in step 4 of the algorithm above. Indeed, Baker's algorithm can also handle weights. Thus, by the same analysis given above, we get a  $1.5 + \varepsilon$  for the weighted variant of MINIMUM RECTANGLE VERTEX COVER. We mention also that Lemma 8 above can also be used to obtain a 1.5-approximation algorithm for the weighted variant of MINIMUM RECTANGLE VERTEX COVER in non-crossing rectangle families.

**Acknowledgment.** We thank Micha Sharir for helpful discussions.

## References

1. König, D.: Gráfok és mátrixok. Matematikai és Fizikai Lapok 38, 116–119 (1931)
2. Karp, R.: Reducibility among combinatorial problems. In: Complexity of Computer Computation, pp. 85–103 (1972)
3. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

4. Hochbaum, D.: Approximation Algorithms for NP-hard Problems. PWS Publishing Company (1997)
5. Vazirani, V.: Approximation Algorithms. Springer (2003)
6. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
7. Bar-Yehuda, R., Even, S.: A linear time approximation algorithm for the weighted vertex cover problem. *J. Algorithm* 2, 198–203 (1981)
8. Clarkson, K.: A modification of the greedy algorithm for vertex cover. *Inform. Process. Lett.* 16, 23–25 (1983)
9. Hochbaum, D.: Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.* 11(3), 555–556 (1982)
10. Nemhauser, G.L., Trotter, L.E.: Vertex packings: structural properties and algorithms. *Mathematical Programming* 8, 232–248 (1975)
11. Bar-Yehuda, R., Even, S.: A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics* 25, 27–46 (1985)
12. Monien, B., Speckenmeyer, E.: Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica* 22(1), 115–123 (1985)
13. Halperin, E.: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.* 31(5), 1608–1623 (2002)
14. Karakostas, G.: A better approximation ratio for the vertex cover problem. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1043–1050. Springer, Heidelberg (2005)
15. Dinur, I., Safra, S.: The importance of being biased. In: 34th annual ACM Symposium on Theory of Computing, pp. 33–42 (2002)
16. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.* 1(2), 180–187 (1972)
17. Lipton, R., Tarjan, R.: Applications of a planar separator theorem. *SIAM J. Comput.* 9(3), 615–627 (1980)
18. Bar-Yehuda, R., Even, S.: On approximating a vertex cover for planar graphs. In: 14th Annual ACM STOC, pp. 303–309 (1982)
19. Chiba, N., Nishizeki, T., Saito, N.: Applications of the Lipton and Tarjan’s planar separator theorem. *Journal of information processing* 4(4), 203–207 (1981)
20. Baker, B.: Approximation algorithms for NP-complete problems on planar graphs. *J. ACM* 41(1), 153–180 (1994)
21. Chalermsook, P., Chuzhoy, J.: Maximum independent set of rectangles. In: 20th Annual ACM-SIAM SODA, pp. 892–901 (2009)
22. Khanna, S., Muthukrishnan, S., Paterson, M.: On approximating rectangle tiling and packing. In: 9th annual ACM-SIAM SODA, 384–393 (1998)
23. Agarwal, P.K., van Kreveld, M.J., Suri, S.: Label placement by maximum independent set in rectangles. *Comput. Geom.* 11(3-4), 209–218 (1998)
24. Doerschler, J., Freeman, H.: A rule-based system for dense-map name placement. *Communications of the ACM* 35(1), 68–79 (1992)
25. Freeman, H.: Computer name placement. In: Geographical Information Systems: Principles and Applications, pp. 445–456 (1991)
26. Lewin-Eytan, L., Naor, J., Orda, A.: Admission control in networks with advance reservations. *Algorithmica* 40(4), 293–304 (2004)
27. Fowler, R., Paterson, M., Tanimoto, S.: Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.* 12(3), 133–137 (1981)
28. Asano, T.: Difficulty of the maximum independent set problem on intersection graphs of geometric objects. In: 6th ICTAG (1991)

29. Berman, P., DasGupta, B., Muthukrishnan, S., Ramaswami, S.: Efficient approximation algorithms for tiling and packing problems with rectangles. *J. Algorithm* 41 (2001)
30. Chan, T.: A note on maximum independent sets in rectangle intersection graphs. *Inform. Process. Lett.* 89(1), 19–23 (2004)
31. Chan, T.M., Har-Peled, S.: Approximation algorithms for maximum independent set of pseudo-disks. In: 25th annual ACM SOCG, pp. 333–340 (2009)
32. Chan, T.M.: Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithm* 46 (2003)
33. Erlebach, T., Jansen, K., Seidel, E.: Polynomial-time approximation schemes for geometric intersection graphs. *SIAM J. Comput.* 34 (2005)
34. Marx, D.: Efficient approximation schemes for geometric problems? In: 13th annual European Symposium on Algorithms, pp. 448–459 (2005)
35. van Leeuwen, E.J.: Better approximation schemes for disk graphs. In: 10th Scandinavian Workshop on Algorithm Theory, pp. 316–327 (2006)
36. Apostolico, A., Atallah, M., Hambrusch, S.: New clique and independent set algorithms for circle graphs. *Discrete Appl. Math.* 36(1), 1–24 (1992)
37. Cenek, E., Stewart, L.: Maximum independent set and maximum clique algorithms for overlap graphs. *Discrete Appl. Math.* 131(1), 77–91 (2003)
38. Golumbic, M., Hammer, P.: Stability in circular arc graphs. *J. Algorithm* 9(3), 314–320 (1988)
39. Hsu, W.L., Tsai, K.H.: Linear time algorithms on circular-arc graphs. *Inform. Process. Lett.* 40(3), 123–129 (1991)
40. Golumbic, M.: Algorithmic graph theory and perfect graphs. Academic Press, New York (1980)
41. Golumbic, M., Trenk, A.: Tolerance Graphs. Cambridge University Press, Cambridge (1985)
42. Chlebík, M., Chlebíková, J.: Approximation hardness of optimization problems in intersection graphs of  $d$ -dimensional boxes. In: 16th Annual ACM-SIAM SODA 2005, pp. 267–276 (2005)
43. Hochbaum, D., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM* 32(1), 130–136 (1985)
44. Bar-Yehuda, R., Halldórsson, M.M., Naor, J., Shachnai, H., Shapira, I.: Scheduling split intervals. *SIAM J. Comput.* 36(1), 1–15 (2006)
45. Butman, A., Hermelin, D., Lewenstein, M., Rawitz, D.: Optimization problems in multiple-interval graphs. In: 18th annual ACM-SIAM SODA, pp. 268–277 (2007)
46. Agarwal, P.K., Sharir, M.: Arrangements and their applications. In: Handbook of Computational Geometry, pp. 49–119 (1998)
47. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithm* 7, 309–322 (1986)
48. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
49. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, Cambridge (1995)

# Feedback Vertex Sets in Tournaments\*

Serge Gaspers<sup>1</sup> and Matthias Mnich<sup>2</sup>

<sup>1</sup> CMM, Universidad de Chile, Santiago de Chile  
[sgaspers@dim.uchile.cl](mailto:sgaspers@dim.uchile.cl)

<sup>2</sup> Technische Universiteit Eindhoven, Eindhoven, The Netherlands  
[m.mnich@tue.nl](mailto:m.mnich@tue.nl)

**Abstract.** We study combinatorial and algorithmic questions around minimal feedback vertex sets in tournament graphs.

On the combinatorial side, we derive strong upper and lower bounds on the maximum number of minimal feedback vertex sets in an  $n$ -vertex tournament. We prove that every tournament on  $n$  vertices has at most  $1.6740^n$  minimal feedback vertex sets and that there is an infinite family of tournaments, all having at least  $1.5448^n$  minimal feedback vertex sets. This improves and extends the bounds of Moon (1971).

On the algorithmic side, we design the first polynomial space algorithm that enumerates the minimal feedback vertex sets of a tournament with polynomial delay. The combination of our results yields the fastest known algorithm for finding a minimum size feedback vertex set in a tournament.

## 1 Introduction

A tournament  $T = (V, A)$  is a directed graph with exactly one arc between every pair of vertices. A feedback vertex set (FVS) of  $T$  is a subset of its vertices whose deletion makes  $T$  acyclic. A minimal FVS of  $T$  is a FVS of  $T$  that is minimal with respect to vertex-inclusion. The complement of a minimal FVS  $F$  induces a maximal acyclic subtournament whose unique vertex of in-degree zero is a “Banks winner” [1]: identifying the vertices of  $T$  with candidates in a voting scheme and arcs indicating preference of one candidate over another, the *Banks winner* of  $T[V \setminus F]$  is the candidate collectively preferred to every other candidate in  $V \setminus F$ . Banks winners play an important role in social choice theory.

*Extremal Combinatorics.* We denote the number of minimal FVSs in a tournament  $T$  by  $f(T)$ , and the maximum  $f(T)$  over all  $n$ -vertex tournaments by  $M(n)$ . The letter “M” was chosen in honor of Moon who in 1971 proved [19] that

$$1.4757^n \leq M(n) \leq 1.7170^n$$

for large  $n$ . Our combinatorial main result are the stronger bounds

$$1.5448^n \leq M(n) \leq 1.6740^n .$$

---

\* Part of this research has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403.

To prove our new lower bound on  $M(n)$ , we construct an infinite family of tournaments all having  $21^{n/7} > 1.5448^n$  minimal FVSs. To prove our new upper bound on  $M(n)$ , we bound the maximum of a convex function bounding  $M(n)$  from above, and otherwise rely on case distinctions and recurrence relations.

For general directed graphs, no non-trivial upper bounds on the number of minimal FVSs are known. For undirected graphs, Fomin et al. [8] show that any undirected graph on  $n$  vertices contains at most  $1.8638^n$  minimal FVSs, and that infinitely many graphs have  $105^{n/10} > 1.5926^n$  minimal FVSs. Lower bounds of roughly  $\log n$  on the size of a maximum-size acyclic subtournament have been obtained by Reid and Parker [24] and Neumann-Lara [21]. Other bounds on minimal or maximal sets with respect to vertex-inclusion have been obtained for dominating sets [9], bicliques [12], separators [10], potential maximal cliques [11], bipartite graphs [4],  $r$ -regular subgraphs [14], and, of course, independent sets [18][20]. The increased interest in exponential time algorithms over the last few years has given new importance to such bounds, as the enumeration of the corresponding objects may be used in exponential time algorithms to solve various problems; see, for example [23][6][11][17][22].

*Enumeration.* An algorithm by Schwikowski and Speckenmeyer [25] lists the minimal FVSs of a tournament  $T$  with polynomial delay, by traversing a hypergraph whose vertices are bijectively mapped to minimal FVSs of  $T$ . Unfortunately the Schwikowski-Speckenmeyer-algorithm may use exponential space, and it is not known whether the minimal FVS problem allows a polynomial delay enumeration algorithm with polynomially bounded space complexity in *general* graphs. Our algorithmic main result provides such an enumeration algorithm for the family of *tournaments*. Our algorithm is inspired from that by Tsukiyama et al. for the (conceptually simpler) enumeration of maximal independent sets [26]. It is based on iterative compression, a technique for parameterized [23] and exact algorithms [7]. We thereby positively answer Fomin et al.'s [7] question if the technique could be applied to other algorithmic areas.

*Exact Algorithms.* In the third [29] in a series [27][28][29] of very influential surveys on exact exponential time algorithms, Woeginger observes that Moon's upper bound on  $M(n)$  provides an upper bound on the overall running time of the enumeration algorithm of Schwikowski and Speckenmeyer. He explicitly asks for a faster algorithm finding a feedback vertex set of a tournament of minimum size. Our new bound yields a time complexity of  $O(1.6740^n)$ . Unlike upper bound proofs on other [48][9][10][11][12][14][18][20] minimal or maximal sets with respect to vertex inclusion, for minimal FVSs in tournaments no known (non trivial) proof readily translates into a polynomial-space branching algorithm. Due to its space complexity, which differs from its time complexity by only a polynomial factor, the Schwikowski-Speckenmeyer-algorithm has only limited practicability [29]. With our new enumeration algorithm, we achieve however a polynomial-space  $O(1.6740^n)$ -time algorithm to find a minimum sized feedback vertex set in tournaments, and to even enumerate all minimal ones. Dom et al. [5] independently answered Woeginger's question by constructing an iterative-compression

algorithm solving only the optimization version of the problem. However, the running time of their algorithm grows at least with  $1.708^n$  and hence their result is inherently weaker than ours.

*Organization of the paper.* Preliminaries are provided in Section 2. Section 3 proves the lower bound on  $M(n)$ , and Section 4 gives the upper bound. We conclude with the polynomial-space polynomial-delay enumeration algorithm in Section 5. The main result of the paper is formulated in Corollary 4.

## 2 Preliminaries

Let  $T = (V, A)$  be a tournament. For a vertex subset  $V' \subseteq V$ , the tournament  $T[V']$  induced by  $V'$  is called a *subtournament* of  $T$ . For each vertex  $v \in V$ , its *in-neighborhood* and *out-neighborhood* are defined as  $N^-(v) = \{u \in V \mid (u, v) \in A\}$  and  $N^+(v) = \{u \in V \mid (v, u) \in A\}$ , respectively. If there is an arc  $(u, v) \in A$  then we say that  $u$  *beats*  $v$  and write  $u \rightarrow v$ . A tournament  $T$  is *strong* if there exists a directed path between any two vertices. A non-strong tournament  $T$  has a unique factorization  $T = S_1 + \dots + S_r$  into strong subtournaments  $S_1, \dots, S_r$ , where every vertex  $u \in V(S_k)$  beats all vertices  $v \in V(S_\ell)$ , for  $1 \leq k < \ell \leq r$ . For  $n \in \mathbb{N}$  let  $\mathcal{T}_n$  denote the set of tournaments with  $n$  vertices and let  $\mathcal{T}_n^*$  denote the set of strong tournaments on  $n$  vertices.

The *score* of a vertex  $v \in V$  is the size of its out-neighborhood, and denoted by  $s_v(T)$  or  $s_v$  for short. Consider a labeling  $1, \dots, n$  of the vertices of  $T$  such that their scores are non-decreasing, and associate with  $T$  the *score sequence*  $s(T) = (s_1, \dots, s_n)$ . If  $T$  is strong then  $s(T)$  satisfies the *Landau inequalities* [15, 16]:

$$\sum_{v=1}^k s_v \geq \binom{k}{2} + 1 \quad \text{for all } k = 1, \dots, n-1, \text{ and} \tag{1}$$

$$\sum_{v=1}^n s_v = \binom{n}{2}. \tag{2}$$

For every non-decreasing sequence  $s$  of positive integers satisfying conditions (1)–(2), there exists a tournament whose score sequence is  $s$  [16].

Let  $L$  be a set of non-zero elements from the ring  $\mathbb{Z}_n$  of integers modulo  $n$  such that for all  $i \in \mathbb{Z}_n$  exactly one of  $+i$  and  $-i$  belongs to  $L$ . The tournament  $T_L = (V_L, A_L)$  with  $V_L = \{1, \dots, n\}$  and  $A_L = \{(i, j) \in V_L \times V_L \mid (j - i) \bmod n \in L\}$  is the *circular  $n$ -tournament induced by  $L$* . A *triangle* is a tournament of order 3. The cyclic triangle is denoted  $C_3$ .

A *FVS*  $F$  of a tournament  $T = (V, A)$  is a subset of vertices, such that  $T[V \setminus F]$  has no directed cycle. It is *minimal* if it does not contain a FVS of  $T$  as a proper subset. Let  $\mathcal{F}(T)$  be the collection of minimal FVSs of  $T$ ; its cardinality is denoted by  $f(T)$ . A *minimum FVS* is a FVS with a minimum number of vertices.

Acyclic tournaments are sometimes called *transitive*; the (up to isomorphism unique) transitive tournament on  $n$  vertices is denoted  $TT_n$ . Let  $\tau$  be the unique

topological order of the vertices of  $TT_n$  such that  $\tau(u) < \tau(v)$  if and only if  $u$  beats  $v$ . For such an order  $\tau$  and integer  $i \in \{1, \dots, n\}$  the subsequence of the first  $i$  values of  $\tau$  is denoted  $\tau_i(V(TT_n)) = (\tau^{-1}(1), \dots, \tau^{-1}(i))$ ; call  $\tau_1(V(TT_n))$  the *source* of  $TT_n$ . For a minimal FVS  $F$  of a tournament  $T$  the subtournament  $T[V \setminus F]$  is a *maximal transitive subtournament* of  $T$  and  $V \setminus F$  is a *maximal transitive vertex set*.

### 3 Lower Bound on the Maximum Number of Minimal FVSs

We prove a lower bound of  $21^{n/7} > 1.5448^n$  on the maximum number of minimal FVSs of tournaments with  $n$  vertices.

Formally, we will bound from below the values of the function  $M(n)$  mapping integers  $n$  to  $\max_{T \in \mathcal{T}_n} f(T)$ . By convention, set  $M(0) = 1$ . Note that  $M$  is monotonically non-decreasing on its domain: given any tournament  $T \in \mathcal{T}_n$  and any vertex  $v \in V(T)$ , for every minimal FVS  $F \in \mathcal{F}(T[V(T) \setminus \{v\}])$  either  $F \in \mathcal{F}(T)$  or  $F \cup \{v\} \in \mathcal{F}(T)$ . As  $T$  and  $v$  are arbitrarily it follows that  $M(n) \geq M(n - 1)$ .

We will now show that there is an infinite family of tournaments on  $n = 7k$  vertices, for any  $k \in \mathbb{N}$ , with  $21^{n/7} > 1.5448^n$  minimal FVSs, improving upon Moon's [19] bound of  $1.4757^n$ . Let us use the following observation.

**Observation 1** ([19]). *If  $T = S_1 + \dots + S_r$  is the factorization of a tournament  $T$  into strong subtournaments  $S_1, \dots, S_r$ , then  $f(T) = f(S_1) \cdot \dots \cdot f(S_r)$ .*

Let  $ST_7$  denote the Paley digraph of order 7, i.e. the circular 7-tournament induced by the set  $L = \{1, 2, 4\}$  of quadratic residues modulo 7. All maximal transitive subtournaments of  $ST_7$  are transitive triangles, of which there are exactly 21, as each vertex is the source of 3 distinct transitive triangles. Thus, all minimal FVSs for  $ST_7$  are minimum FVSs. We remark that  $ST_7$  is the unique 7-vertex tournament without any  $TT_4$  as subtournament [24].

**Lemma 1.** *There exists an infinite family of tournaments with  $21^{n/7}$  minimal FVSs.*

*Proof.* Let  $k \in \mathbb{N}$  and form the tournament  $T_0 = ST_7 + \dots + ST_7$  from  $k$  copies of  $ST_7 \in \mathcal{T}_7^*$ . Then  $T_0 \in \mathcal{T}_n$  for  $n = 7k$ , and the number of minimal FVSs in  $T_0$  is  $f(T_0) = f(ST_7)^k = 21^k = 21^{n/7}$ .  $\square$

### 4 Upper Bound on the Maximum Number of Minimal FVSs

We give an upper bound of  $\beta^n$ , where  $\beta = 1.6740$ , on the maximum number of minimal FVSs in any tournament  $T \in \mathcal{T}_n$ , for any positive integer  $n$ . This improves the bound of  $1.7170^n$  by Moon [19]. Instead of minimal FVSs we count

maximal transitive subtournaments, and with respect to Observation 1 we count the maximal transitive subtournaments of *strong* tournaments.

We start with three properties of maximal transitive subtournaments. First, for a strong tournament  $T = (V, A)$  with score sequence  $s = (s_1, \dots, s_n)$  the following holds: if  $TT_k = (V', A')$  is a maximal transitive subtournament of  $T$  with  $\tau_1(V') = (t)$  then  $T[V' \setminus \{t\}]$  is a maximal transitive subtournament of  $T[N^+(t)]$ . Hence  $f(T) \leq \sum_{v=1}^n M(s_v)$ , where  $s_v \leq n - 2$  for all  $v \in V$ . This allows us to effectively bound  $f(T)$  via a recurrence relation.

Second, there cannot be too many vertices with large score.

**Lemma 2.** *For  $n \geq 8$  and  $k \in \{0, 1, 2\}$ , any strong tournament  $T \in \mathcal{T}_n^*$  has at most  $2(k+1)$  vertices of score at least  $n - 2 - k$ .*

*Proof.* Fix some strong tournament  $T \in \mathcal{T}_n^*$  and  $k \in \{0, 1, 2\}$ . Suppose for contradiction that  $T$  contains  $2k+3$  vertices with score at least  $n - 2 - k$ . Then the Landau inequalities (1) and (2) imply the contradiction

$$\begin{aligned} 2\binom{n}{2} &= 2 \left( \sum_{v=1}^{n-(2k+3)} s_v + \sum_{v=n-(2k+2)}^n s_v \right) \\ &\geq 2 \left( \binom{n-(2k+3)}{2} + 1 + (2k+3)(n-2-k) \right) = n^2 - n + 2. \end{aligned}$$

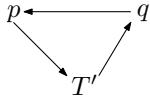
□

For  $n \leq 7$ , we can explicitly list the strong  $n$ -vertex tournaments for which the Lemma fails: the cyclic triangle for  $k = 0$ , the tournaments  $RT_5, ST_6$  for  $k = 1$  and  $ST_7$  for  $k = 2$ .  $RT_5$  is the regular tournament of order 5 and  $ST_6$  is the tournament obtained by arbitrarily removing some vertex from  $ST_7$  (defined in the previous section) and all incident arcs.

Third, let  $T'$  be a tournament obtained from a tournament  $T$  by reversing all arcs of  $T$ . Then,  $f(T) = f(T')$ , whereas the score  $s_v(T)$  of each vertex  $v$  turns into  $s_v(T') = n - 1 - s_v(T)$ . This implies that analyzing score sequences with maximum score  $s_n \geq n - 1 - c$  for some constant  $c$  is symmetric to analyzing score sequences with minimum score  $s_1 \leq c$ .

Our proof that any tournament on  $n$  vertices has at most  $\beta^n$  maximal transitive subtournaments consists of several parts. We start by proving the bound for tournaments with few vertices. The inductive part of the proof first considers tournaments with large maximum score (and symmetrically small minimum score), and then all other tournaments.

We begin the proof by considering tournaments with up to 10 vertices. For  $n \leq 4$  exact values for  $M(n)$  were known before [19]. For  $n = 5, \dots, 9$  we obtained exact values for  $M(n)$  with the help of a computer. For these values the extremal tournaments obey the following structure: pick a strong tournament  $T' \in \mathcal{T}_{n-2}^*$  and construct the strong tournament  $pq(T') \in \mathcal{T}_n^*$  by attaching two vertices to  $T'$  as in Fig. 1, namely add vertices  $p$  and  $q$  to  $T'$ , and arcs  $q \rightarrow p$ , and  $p \rightarrow t$ ,  $t \rightarrow q$  for each vertex  $t$  in  $T'$ . Then  $f(pq(T')) = 2f(T') + 1$ .

**Fig. 1.** A tournament  $pq(T') \in \mathcal{T}_n^*$  with  $f(pq(T')) = 2f(T') + 1$ 

For  $n = 5$ , there are exactly two non-isomorphic strong tournaments  $QT_5 \cong pq(C_3), RT_5 \in \mathcal{T}_5^*$ . For these,  $f(QT_5) = f(RT_5) = M(5) = 2 \cdot 3 + 1 = 7$ . For  $n = 6$ ,  $ST_6$  is the unique tournament from  $\mathcal{T}_6$  with  $f(ST_6) = M(6) = 12$  minimal FVSs. For  $n = 7$  the previous section showed  $f(ST_7) = 21$ , and in fact  $ST_7$  is the unique 7-vertex tournament with  $M(7) = 21$  minimal FVSs. For  $n \in \{8, 9\}$ ,  $ST_n \cong pq(ST_{n-2})$ ; then  $f(ST_n) = M(n)$ . Table I summarizes that for  $n \leq 9$ ,  $M(n) \leq \beta^n$ .

**Table 1.** Extremal tournaments of up to 9 vertices

$n$	$M(n)$	$M(n)^{1/n} \approx$	$T \in \mathcal{T}_n : f(T) = M(n)$
1	1	1.00000	$T \in \mathcal{T}_1$
2	1	1.00000	$T \in \mathcal{T}_2$
3	3	1.44225	$T \in \mathcal{T}_3 \setminus \{TT_3\}$
4	3	1.31607	$T \in \mathcal{T}_4 \setminus \{TT_4\}$
5	7	1.47577	$QT_5 \cong pq(C_3), RT_5$
6	12	1.51309	$ST_6 \cong ST_7 - \{1\}$
7	21	1.54486	$ST_7$
8	25	1.49535	$ST_8 \cong pq(ST_6)$
9	43	1.51879	$ST_9 \cong pq(ST_7)$

Next, we bound  $M(10)$  by means of  $M(n)$  for  $n \leq 9$ . Let  $W$  be a maximal transitive vertex set of  $T \in \mathcal{T}_{10}^*$ . Then either  $v^* \in W$  or  $v^* \notin W$ , where  $v^*$  is a vertex with score  $s_{10}$ . There are at most  $M(s_{10}) \leq M(9)$  maximal transitive vertex sets  $W$  such that  $v^* \in W$  and at most  $M(9)$  such sets  $W$  for which  $v^* \notin W$ . As  $(2M(9))^{1/10} = 86^{1/10} < 1.5612$ , the proof follows for all tournaments with at most 10 vertices.

For the rest of this section we consider tournaments with  $n \geq 11$  vertices. Let  $T = (V, A)$  be a strong tournament on  $n \geq 11$  vertices; we will show that  $f(T) \leq \beta^n$ . The proof considers four main cases and several subcases with respect to the minimum and maximum score of the tournament. Due to space constraints, we only give an outline of the proof; the full proof is given in [13].

The idea of the proof is as follows. By  $W$  we denote a maximal transitive vertex set of  $T$ . If there is a vertex  $v$  in  $T$  of large score at least  $n - 3$ , then either  $v$  is the source of  $W$  or only few other vertices can be the source of  $W$ . We can then look at the subtournament induced by these few vertices, and branch on their inclusion with respect to  $W$ . In this way, we fix the first few elements of the acyclic ordering of  $W$ . Moreover, there cannot be too many vertices

of large score by Lemma 2. Suppose that in one branch,  $\tau_k(W) = (a_1, a_2, \dots, a_k)$  and for some  $i \in \{1, \dots, k\}$ ,  $|N^+(a_i) \setminus W| \geq c$ , then we can upper bound the number of such maximal transitive vertex sets  $W$  by  $M(a_i - (k - i) - c)$ . The case when some vertex  $v$  in  $T$  has small score at most 2 is symmetric.

The tightest case of our proof is the following:  $s_n = n - 3$ ,  $s_{b_1} = n - 3$ ,  $s_{b_2} = n - 4$ , where  $b_1 \rightarrow b_2$  are the two in-neighbors of  $n$ , and  $N^-(b_1) \neq N^-(b_2) \setminus \{b_1\}$ . Denote  $c_1 \rightarrow c_2$  the in-neighbors of  $b_1$  and  $d_1 \rightarrow d_2$  the in-neighbors of  $b_2$ . We count the different maximal transitive vertex sets  $W$  depending on the membership or non-membership of  $b_1, b_2$ , and  $n$  in  $W$ .

(1) If  $b_1, b_2 \notin W$ , then  $n \in W$  by maximality of  $W$  and  $\tau_1(W) = (n)$  as no vertex in  $W$  beats  $n$ . There are at most  $M(s_n) = M(n - 3)$  such  $W$ .

(2) If  $b_1, n \notin W$  and  $b_2 \in W$ , then some in-neighbor of  $b_2$  is in  $W$ , otherwise  $W$  were not maximal as  $W \cup \{n\}$  would be a transitive vertex set. There are at most  $M(s_{b_2} - 1) = M(n - 5)$  possibilities for  $\tau_2(W) = (d_2, b_2)$ , at most  $M(s_{d_1} - 2) \leq M(n - 5)$  for  $\tau_2(W) = (d_1, b_2)$ , and at most  $M(s_{d_1} - 2) \leq M(n - 5)$  for  $\tau_3(W) = (d_1, d_2, b_2)$ .

(3) If  $b_1 \notin W$  and  $b_2, n \in W$ , then  $\tau_2(W) = (b_2, n)$ . There are at most  $M(s_{b_2} - 1) = M(n - 5)$  such  $W$ .

(4) If  $n \notin W$  and  $b_1 \in W$ , then we consider two subcases. If  $N^-(b_1) \cap W \neq \emptyset$ , then some in-neighbor of  $b_1$  is the source of  $W$ . There are at most  $M(s_{c_2} - 1) \leq M(n - 4)$  possibilities for  $\tau_2(W) = (c_2, b_1)$ , at most  $M(s_{c_1} - 2) \leq M(n - 5)$  for  $\tau_3(W) = (c_1, c_2, b_1)$ , and at most  $M(s_{c_1} - 2) \leq M(n - 5)$  for  $\tau_2(W) = (c_1, b_1)$ . Otherwise, no in-neighbor of  $b_1$  is in  $W$ , and thus,  $\tau_1(W) = (b_1)$ . Moreover,  $b_2 \in W$  and some in-neighbor of  $b_2$  is the source of  $T[W \setminus \{b_1\}]$ , otherwise  $n$  could be added. This leaves us with a total of at most  $3M(s_{b_1} - 4) = 3M(n - 7)$  possibilities for which  $\tau_4(W) = (b_1, d_1, d_2, b_2)$ ,  $\tau_3(W) = (b_1, d_2, b_2)$ , or  $\tau_3(W) = (b_1, d_1, b_2)$ .

(5) If  $b_2 \notin W$  and  $b_1, n \in W$ , then  $\tau_2(W) = (b_1, n)$ . There are at most  $M(s_{b_1} - 2) = M(n - 5)$  such  $W$ .

(6) If  $b_1, b_2, n \in W$ , then  $\tau_3(W) = (b_1, b_2, n)$ . As at least one out-neighbor of  $b_2$  is an in-neighbor of  $b_1$ , there are at most  $M(s_{b_2} - 2) = M(n - 6)$  such  $W$ .

Altogether, in this case,

$$\begin{aligned} f(T) &\leq M(n - 3) + 3M(n - 5) + M(n - 5) + (M(n - 4) + 2M(n - 5) \\ &\quad + 3M(n - 7)) + M(n - 5) + M(n - 6) \\ &\leq 3\beta^{n-7} + \beta^{n-6} + 7\beta^{n-5} + \beta^{n-4} + \beta^{n-3}, \end{aligned}$$

which is at most  $\beta^n$  because  $\beta \geq 1.6740$ .

Now suppose that every vertex in  $T$  has score at least three and at most  $n - 4$ . In that case we define a linear function  $G_n$  mapping feasible score sequences  $s = (s_1, \dots, s_n)$  to  $\sum_{v=1}^n \beta^{s_v}$  for  $\beta = 1.6740$ . We then define special score sequences

$\sigma(n)$  and show that these sequences maximize  $G_n$ , based on the strict convexity of  $G_n$ . For example,

$$\sigma(17) = (3, 3, 3, 3, 3, 3, 4, 7, 8, 9, 12, 13, 13, 13, 13, 13, 13) .$$

The proof is completed by bounding  $f(n)$  in terms of  $G(\sigma(n))$ .

All cases taken together imply the following upper bound on the number of maximal transitive subtournaments.

**Theorem 1.** *Any strong tournament  $T \in \mathcal{T}_n^*$  has at most  $1.6740^n$  maximal transitive subtournaments.*

Moon [19] already observed that the following limit exists.

**Corollary 2.** *It holds  $1.5448 \leq \lim_{n \rightarrow \infty} (M(n))^{1/n} \leq 1.6740$ .*

We conjecture that the Paley digraph of order 7,  $ST_7$ , plays the same role for FVSs in tournaments as triangles play for independent sets in graphs, i.e. that the tournaments  $T$  maximizing  $(f(T))^{1/|V(T)|}$  are exactly those whose factors are copies of  $ST_7$ .

## 5 Polynomial-Delay Enumeration in Polynomial Space

In this section, we give a polynomial-space algorithm for the enumeration of the minimal FVSs in a tournament with polynomial delay.

Let  $T = (V, A)$  be a tournament with  $V = \{v_1, \dots, v_n\}$ , and for each  $i = 1, \dots, n$  let  $T_i = T[\{v_1, \dots, v_i\}]$ . For a vertex set  $X$ , we write  $\chi_X(i) = 1$  if  $v_i \in X$  and  $\chi_X(i) = 0$  otherwise. Let  $<$  denote the total order on  $V$  induced by the labels of the vertices. For vertex sets  $X, Y \subseteq V$ , say that  $X$  is *lexicographically smaller* than  $Y$  and write  $X \prec Y$  if for the minimum index  $i$  for which  $\chi_X(i) \neq \chi_Y(i)$  it holds that  $v_i \in X$ . Because  $X$  and  $Y$  are totally ordered by the restriction of  $<$  to  $X$  and  $Y$ , respectively,  $\prec$  is also a total order and each collection of subsets of  $V$  has a unique *lexicographically smallest* element.

The algorithm enumerates the maximal acyclic vertex sets of  $T$ . It performs a depth-first search in a tree  $\mathcal{T}$  with the maximal acyclic vertex sets of  $T$  as leaves, whose forward and backward edges are constructed “on the fly”. The depth of  $\mathcal{T}$  is  $|V|$ , and we refer to the vertices of  $\mathcal{T}$  as *nodes*. The algorithm only needs to keep in memory the path from the root to the current node in the tree and all the children of the nodes on this path. Each node at level  $j$  is labeled by a maximal acyclic vertex set  $J$  of  $T_j$ . As for its children, there are two cases. In case  $J \cup \{v_{j+1}\}$  is acyclic then  $J$ ’s only child is  $J \cup \{v_{j+1}\}$ . In case  $J \cup \{v_{j+1}\}$  is not acyclic then  $J$  has at least one and at most  $\lfloor j/2 \rfloor + 1$  children. Let  $L_J = (v^1, v^2, \dots, v^{|J|})$  be a labeling of the vertices in  $J$  such that  $(v^r, v^s) \in A$  for all  $1 \leq r < s \leq j$ ; we view  $L_J$  as a sequence of vertices. The children of  $J$  are as follows. The first child  $J^0$  is a copy of  $J$ , and is always present. The potential other children are, for  $1 \leq z \leq |J| + 1$ ,

$$J^z = \{v^i \in J \mid i < z \wedge v^i \rightarrow v_{j+1}\} \cup \{v_{j+1}\} \cup \{v^i \in J \mid i \geq z \wedge v_{j+1} \rightarrow v^i\}$$

where set  $J^z$  is a potential child of  $J$  only if  $J^z$  is a maximal acyclic vertex set in  $T_{j+1}$  (the maximality of  $J^z$  can clearly be checked in polynomial time). Note how we try to insert  $v_{j+1}$  at every possible position in  $J$ . However, only at most  $\lfloor j/2 \rfloor + 1$  positions make sense for  $v_{j+1}$ : before  $v^1$  if  $v_{j+1} \rightarrow v^1$ , between  $v^i$  and  $v^{i+1}$  if  $v^i \rightarrow v_{j+1} \rightarrow v^{i+1}$ , where  $1 \leq i \leq |J| - 1$ , and after  $v^{|J|}$  if  $v^{|J|} \rightarrow v_{j+1}$ ; all other positions do not give maximal acyclic vertex sets and should not be generated in an actual implementation. Note that  $J^z$  may be a potential child of several sets on the same level in  $\mathcal{T}$ . Of all these sets,  $J^z$  is made the child only of the lexicographically smallest such set. To determine whether  $J$  is the lexicographically smallest such set, we compute by a greedy algorithm the lexicographically smallest maximal acyclic vertex set  $H = H(J^z)$  of  $T_j$  which contains  $J^z \setminus \{v_{j+1}\}$  as a subset. That is, we iteratively build the set  $H$  by setting

$$\begin{aligned} H_0 &= J^z \setminus \{v_{j+1}\}, \\ H_i &= \begin{cases} H_{i-1} \cup \{v_i\}, & \text{if } H_{i-1} \cup \{v_i\} \text{ is acyclic,} \\ H_{i-1}, & \text{otherwise,} \end{cases} \quad i = 1, \dots, j, \\ H &= H_j. \end{aligned}$$

Then we make  $J^z$  a child of the node labeled  $J$  only if  $H = J$ . This completes the description of the algorithm.

To show that the algorithm is correct, we prove that for every maximal acyclic vertex set  $W$  of  $T$  there is exactly one leaf in  $\mathcal{T}$  labeled with  $W$ . By construction of the algorithm, it suffices to show that at least one leaf is labeled by  $W$ . The proof is by induction on the number  $n = |V|$  of vertices in  $T$ . For  $n = 1$  the claim clearly holds, so suppose that  $n > 1$  and that the claim is true for all tournaments with fewer vertices. Then from the induction hypothesis we can conclude that for the induced subtournament  $T' := T_{n-1}$  there is a tree  $\mathcal{T}'$  constructed by the above algorithm and a bijection  $f'$  from the maximal acyclic vertex sets of  $T'$  to the leaves of  $\mathcal{T}'$ .

Let  $W$  be a maximal acyclic vertex set of  $T$ . If  $v_n \notin W$  then  $W$  is an acyclic vertex set of  $T'$  as removing a vertex from a digraph does not introduce cycles. In fact,  $W$  is a maximal acyclic vertex set of  $T'$ : for any vertex  $v_\ell \in V \setminus (W \cup \{v_n\})$ ,  $T'[W \cup \{v_\ell\}]$  has a cycle as  $W$  is a maximal acyclic vertex set for  $T$  and  $T'[W \cup \{v_\ell\}] = T[W \cup \{v_\ell\}]$ . Hence there exists a leaf  $f'(W)$  in  $\mathcal{T}'$  labeled by  $W$ . Since  $W \cup \{v_n\}$  is not acyclic, by maximality of  $W$  for  $T$ , the algorithm constructs the child  $W^0$  of  $f'(W)$  labeled by  $W$ , and that child will be a leaf in the final tree constructed by the algorithm.

If  $v_n \in W$ , then let  $W' = W \setminus \{v_n\}$ . So,  $W'$  is an acyclic vertex set of  $T'$ . In case  $W'$  is maximal for  $T'$ , there is a leaf  $f'(W')$  in  $\mathcal{T}'$  that is labeled by  $W'$ . Since  $W' \cup \{v_n\}$  is acyclic, the algorithm will create a single child of  $f'(W')$  labeled by  $W' \cup \{v_n\} = W$ , and that child will be a leaf in the final tree constructed by the algorithm. In case  $W'$  is not maximal for  $T'$ , let  $N$  be the lexicographically smallest extension of  $W'$  to a maximal acyclic vertex set of  $T'$ . Hence there exists a leaf  $f'(N)$  in the tree  $\mathcal{T}'$  labeled by  $N$ . Observe that the sequence  $L_W$

is a subsequence of  $L_N$ , and that  $N \cup \{v_n\}$  is not acyclic. Hence the algorithm creates children  $N^1, N^2, \dots$ , one of which will be labeled by  $W$ .

To see that the algorithm runs with polynomial delay, note that the children and parent of a given node in  $\mathcal{T}$  can all be computed in polynomial time. It follows that  $\mathcal{T}$  can be traversed in a depth-first manner with polynomial delay per step of the traversal, and thus the leaves of  $\mathcal{T}$  can be output with only a polynomial delay.

We show that the algorithm requires only polynomial space. We already observed that each node in  $\mathcal{T}$  at level  $j$  has at most  $\lfloor j/2 \rfloor + 1$  children. For each node we store the maximal acyclic vertex set by which it is labeled. Because we are traversing  $\mathcal{T}$  in a depth-first-search manner, in each step of the algorithm we only need to save data of  $O(n^2)$  nodes: those of the  $O(n)$  nodes on the path from the root to the currently active node labeled by  $J$ , and the  $O(n)$  children for each node on this path.

**Theorem 3.** *The described algorithm enumerates all FVSs of a tournament with polynomial delay and uses polynomial space.*

**Corollary 4.** *In a tournament with  $n$  vertices a minimum directed feedback vertex set can be found in  $O(1.6740^n)$  time and polynomial space.*

**Acknowledgments.** We thank Gerhard J. Woeginger for help with the presentation of the results.

## References

1. Banks, J.S.: Sophisticated voting outcomes and agenda control. *Soc. Choice Welfare* 1(4), 295–306 (1985)
2. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. *SIAM J. Comput.* 39(2), 546–563 (2009)
3. Byskov, J.M.: Enumerating maximal independent sets with applications to graph colouring. *Oper. Res. Lett.* 32(6), 547–556 (2004)
4. Byskov, J.M., Madsen, B.A., Skjernaa, B.: On the number of maximal bipartite subgraphs of a graph. *J. Graph Theory* 48(2), 127–132 (2005)
5. Dom, M., Guo, J., Hüffner, F., Niedermeier, R., Truss, A.: Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms* 8(1), 320–331 (2010)
6. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms Appl.* 7(2), 131–140 (2003)
7. Fomin, F.V., Gaspers, S., Kratsch, D., Liedloff, M., Saurabh, S.: Iterative compression and exact algorithms. *Theor. Comput. Sci.* 411(7-9), 1045–1053 (2010)
8. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: exact and enumeration algorithms. *Algorithmica* 52(2), 293–307 (2008)
9. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. *ACM Trans. Algorithms* 5(1), 1–17 (2008)

10. Fomin, F.V., Villanger, Y.: Treewidth computation and extremal combinatorics. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssen, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 210–221. Springer, Heidelberg (2008)
11. Fomin, F.V., Villanger, Y.: Finding induced subgraphs via minimal triangulations. In: Proc. of STACS 2010, LIPIcs. Schloss Dagstuhl - Leibniz Center of Informatics (2010)
12. Gaspers, S., Kratsch, D., Liedloff, M.: On independent sets and bicliques in graphs. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 171–182. Springer, Heidelberg (2008)
13. Gaspers, S., Mnich, M.: On feedback vertex sets in tournaments. arXiv Technical Report (May 2009), <http://arxiv.org/abs/0905.0567>
14. Gupta, S., Raman, V., Saurabh, S.: Fast exponential algorithms for maximum  $r$ -regular induced subgraph problems. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 139–151. Springer, Heidelberg (2006)
15. Harary, F., Moser, L.: The theory of round robin tournaments. Amer. Math. Monthly 73(3), 231–246 (1966)
16. Landau, H.G.: On dominance relations and the structure of animal societies. III. The condition for a score structure. Bull. Math. Biophys. 15, 143–148 (1953)
17. Lawler, E.L.: A note on the complexity of the chromatic number problem. Information Processing Letters 5(3), 66–67 (1976)
18. Miller, R.E., Muller, D.E.: A problem of maximum consistent subsets. IBM Research Report RC-240, J. T. Watson Research Center, Yorktown Heights, NY (1960)
19. Moon, J.W.: On maximal transitive subtournaments. Proc. Edinburgh Math. Soc. 17(4), 345–349 (1971)
20. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
21. Neumann-Lara, V.: A short proof of a theorem of Reid and Parker on tournaments. Graphs Combin. 10, 363–366 (1994)
22. Raman, V., Saurabh, S., Sikdar, S.: Efficient exact algorithms through enumerating maximal independent sets and other techniques. Theor. Comput. Syst. 41(3), 563–587 (2007)
23. Reed, B.A., Smith, K., Vetta, A.: Finding odd cycle transversals. Oper. Res. Lett. 32(4), 299–301 (2004)
24. Reid, K.B., Parker, E.T.: Disproof of a conjecture of Erdős and Moser on tournaments. J. Combin. Theory 9(3), 225–238 (1970)
25. Schwikowski, B., Speckenmeyer, E.: On enumerating all minimal solutions of feedback problems. Discrete Appl. Math. 117, 253–265 (2002)
26. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. SIAM J. Comput. 6(3), 505–517 (1977)
27. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Combinatorial Optimization - Eureka, you shrink!, vol. 2570, pp. 185–207. Springer, Berlin (2003)
28. Woeginger, G.J.: Space and time complexity of exact algorithms: Some open problems. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 281–290. Springer, Heidelberg (2004)
29. Woeginger, G.J.: Open problems around exact algorithms. Discrete Appl. Math. 156(3), 397–405 (2008)

# ***n*-Level Graph Partitioning\***

Vitaly Osipov and Peter Sanders

Karlsruher Institut für Technologie  
`{osipov,sanders}@kit.edu`

**Abstract.** We present a multi-level graph partitioning algorithm based on the extreme idea to contract only a single edge on each level of the hierarchy. This obviates the need for a matching algorithm and promises very good partitioning quality since there are very few changes between two levels. Using an efficient data structure and new flexible ways to break local search improvements early, we obtain an algorithm that scales to large inputs and produces the best known partitioning results for many inputs. For example, in Walshaw’s well known benchmark tables we achieve 155 improvements dominating the entries for large graphs.

## 1 Introduction

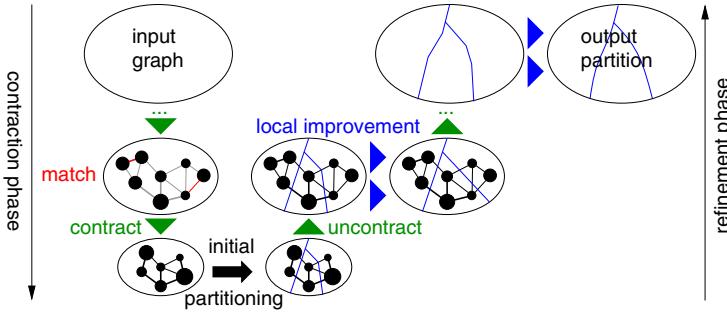
Many important applications of computer science involve processing large graphs, e.g., stemming from finite element methods, digital circuit design, route planning, social networks, etc. Very often these graphs need to be partitioned or clustered such that there are few edges between the blocks (pieces).

A successful heuristic for partitioning large graphs is the *multilevel graph partitioning* approach (MGP) depicted in Figure 1 where the graph is recursively contracted to a smaller graph with the same basic structure. After applying an *initial partitioning* algorithm to this small graph, the contraction is undone and, at each level, a *local refinement* method improves the partition induced by the coarser level. Section 2 explains the method in more detail. Most systems instantiate MGP in a very similar way: Maximal matchings are contracted between two levels that try to include as many heavy edges as possible. Local refinement uses a linear time variant of local search. MGP has two crucial advantages over most other approaches to graph partitioning: We get near linear execution time since the graph shrinks geometrically and we get good partitioning quality since a good solution on some level yields a good initial solution on the next finer level, i.e., local search needs little work to further improve the solution.

Our central idea is to get even better partitions by making subsequent levels as similar as possible – we (un)contract only a *single* edge between two levels. We call this *n*-GP since we have (almost) *n* levels of hierarchy. More details are described in Section 3. *n*-GP has the additional advantage that there is no longer a need for an algorithm finding heavy matchings. This is remarkable insofar as a considerable amount of work on approximate maximum weight matching

---

\* Partially supported by DFG grant SA 933/3-2.



**Fig. 1.** Multilevel graph partitioning

was motivated by the MGP application [24][5][22][18]. Still, at first glance,  $n$ -GP seems to have substantial disadvantages also. Firstly, storing each level explicitly would lead to quadratic space consumption. We avoid this by using a dynamic graph data structure with little space overhead. Secondly, choosing maximal matchings instead of just a single edge for contraction has the side effect that the graph is contracted everywhere, leading to a more uniform distribution of node weights. We solve this problem by explicitly factoring node weights into the *edge rating* function prioritizing the edges to be contracted. Already in [13][4] edge ratings have proven to lead to better results for graph partitioning. Perhaps the most serious problem is that the most common approach to local search is to let it run for a number of steps proportional to the current number of nodes. In the context of  $n$ -GP this could lead to a quadratic overall number of local search steps. Therefore, we develop a new, more adaptive stopping criteria for the local search that drastically accelerates  $n$ -GP without significantly reducing partitioning quality.

We have implemented  $n$ -GP in the graph partitioner KaSPar (Karlsruhe Sequential Partitioner). Experiments reported in Section 5 indicate that KaSPar scales well to large networks, computes the best known partitions for many instances of a “standard benchmark” and needs time comparable to system that previously computed the best results for large networks. Section 6 summarizes the results and discusses future directions.

## More Related Work

There has been a huge amount of research on graph partitioning so that we refer to introductory and overview papers such as [7][8][26][30] for more material. Well-known software packages based on MGP are Chaco [12], DiBaP [19], Jostle [29][30], Metis [16][17], Party [23][25], and Scotch [20][21].

KaSPar was developed partly in parallel with KaPPa (Karlsruhe Parallel Partitioner) [14]. KaPPa is a “classical” matching based MGP algorithm designed for scalable parallel execution and its local search only considers independent pairs of blocks at a time. Still, for  $k = 2$ , its interesting to compare KaSPar and KaPPa since KaPPa achieves the previously best partitioning results for many

large graphs, since both systems use a similar edge ratings, and since running times for a two processor parallel code and a sequential code could be expected to be roughly comparable.

There is a long tradition of  $n$ -level algorithms in geometric data structures based on randomized incremental construction (e.g., [111]). Our motivation for studying  $n$ -level are *contraction hierarchies* [10], a preprocessing technique for route planning that is at the same time simpler and an order of magnitude more efficient than previous techniques using a small number of levels.

## 2 Preliminaries

Consider an undirected graph  $G = (V, E, c, \omega)$  with edge weights  $\omega : E \rightarrow \mathbf{R}_{>0}$ , node weights  $c : V \rightarrow \mathbf{R}_{\geq 0}$ ,  $n = |V|$ , and  $m = |E|$ . We extend  $c$  and  $\omega$  to sets, i.e.,  $c(V') := \sum_{v \in V'} c(v)$  and  $\omega(E') := \sum_{e \in E'} \omega(e)$ .  $\Gamma(v) := \{u : \{v, u\} \in E\}$  denotes the neighbors of  $v$ .

We are looking for *blocks* of nodes  $V_1, \dots, V_k$  that partition  $V$ , i.e.,  $V_1 \cup \dots \cup V_k = V$  and  $V_i \cap V_j = \emptyset$  for  $i \neq j$ . The *balancing constraint* demands that  $\forall i \in 1..k : c(V_i) \leq L_{\max} := (1 + \epsilon)c(V)/k + \max_{v \in V} c(v)$  for some parameter  $\epsilon$ . The last term in this equation arises because each node is atomic and therefore a deviation of the heaviest node has to be allowed. The objective is to minimize the total *cut*  $\sum_{i < j} \omega(E_{ij})$  where  $E_{ij} := \{\{u, v\} \in E : u \in V_i, v \in V_j\}$ . By default, our initial inputs will have unit edge and node weights. However, even those will be translated into weighted problems in the course of the algorithm.

*Contracting* an edge  $\{u, v\}$  means replacing the nodes  $u$  and  $v$  by a new node  $x$  connected to the former neighbors of  $u$  and  $v$ . We set  $c(x) = c(u) + c(v)$ . If replacing edges of the form  $\{u, w\}, \{v, w\}$  would generate two parallel edges  $\{x, w\}$ , we insert a single edge with  $\omega(\{x, w\}) = \omega(\{u, w\}) + \omega(\{v, w\})$ . *Uncontracting* an edge  $e$  undoes its contraction. Partitions computed for the contracted graph are extrapolated to the uncontracted graph in the obvious way, i.e.,  $u$  and  $v$  are put into the same block as  $x$ .

*Local Search* is done by moving single nodes between blocks. The gain  $g_B(v)$  of moving node  $v$  to block  $B$  is the decrease in total cut size caused by this move. For example, if  $v$  has 5 incident edges of unit weight, two of which are inside  $v$ 's block and three of which lead to block  $B$  then  $g_B(v) = 3 - 2 = 1$ .

## 3 $n$ -Level Graph Partitioning

Figure 2 gives a high-level recursive summary of  $n$ -GP. The base case is some other partitioner used when the graph is sufficiently small. In KaSPar, contraction is stopped when either only  $20k$  nodes remain, no further nodes are eligible for contraction, or there are less edges than nodes left. The latter happens when the graph consists of many independent components. As observed in [14] Scotch [20] produces better initial partitions than Metis, and therefore we also use it in KaSPar .

**Function**  $n\text{-GP}(G, k, \epsilon)$ 

```

if  $G$  is small then return initialPartition( $G, k, \epsilon$ )
pick the edge  $e = \{u, v\}$  with the highest rating
contract  $e$ ;  $\mathcal{P} := n\text{-GP}(G, k, \epsilon)$ ; uncontract  $e$ 
activate( $u$ ); activate( $v$ ); localSearch()
return  $\mathcal{P}$ 

```

**Fig. 2.**  $n\text{-GP}$ 

The edges to be contracted are chosen according to an edge rating function. KaSPar adopts the rating function

$$\text{expansion}^*(\{u, v\}) := \frac{\omega(\{u, v\})}{c(u)c(v)}$$

which fared best in [14]. Additionally, in order to avoid unbalanced inputs to the initial partitioner, KaSPar never allows a node  $v$  to participate in a contraction if the weight of  $v$  exceeds  $1.5n/(20k)$ . Selecting contracted edges can be implemented efficiently by keeping the contractable *nodes* in a priority queue sorted by the rating of their most highly rated incident edge.

In order to make contraction and uncontraction efficient, we use a “semidynamic” graph data structure: When contracting an edge  $\{u, v\}$ , we mark both  $u$  and  $v$  as deleted, introduce a new node  $w$ , and redirect the edges incident to  $u$  and  $v$  to  $w$ . The advantage of this implementation is that edges adjacent to a node are still stored in adjacency arrays which are more efficient than linked lists needed for a full fledged dynamic graph data structure. A disadvantages of our approach is a certain space overhead. However, it is relatively easy to show that this space overhead is bounded by a logarithmic factor even if we contract edges in some random fashion (see [4]). In Section 5 we will demonstrate experimentally that the overhead is actually often a small constant factor. Indeed, this is not very surprising since the edge rating function is not random, but designed to keep the contracted graph sparse. Overall, with respect to asymptotic memory overhead,  $n\text{-GP}$  is no worse than methods with a logarithmic number of levels.

### 3.1 Local Search Strategy

Our local search strategy is similar to the FM-algorithm [6] that is also used in many other MGP systems. We now outline our variant and then discuss differences.

Originally, all nodes are unmarked. Only unmarked nodes are allowed to be activated or moved from one block to another. Activating a node  $v \in B'$  means that for blocks  $\{B \neq B' : \exists \{v, u\} \in E \wedge u \in B\}$  we compute the gain

$$g_B(v) = \sum \{\omega(\{v, u\}) : \{v, u\} \in E, v \in B\} - \sum \{\omega(\{v, u\}) : \{v, u\} \in E, v \in B'\}$$

of moving  $v$  to block  $B$  for blocks where  $v$  can be moved. Note that gains are allowed to be negative. Node  $v$  is then inserted into the priority queue  $P_B$  using

$g_B(v)$  as the priority. We call a queue  $P_B$  eligible if the highest gain node in  $P_B$  can be moved to block  $B$  without violating the balance constraint for block  $B$ . Local search repeatedly looks for the highest gain node  $v$  in any eligible priority queue  $P_B$  and moves  $v$  to block  $B$ . When this happens, node  $v$  becomes nonactive and marked, the unmarked neighbors of  $v$  get activated and the gains of the active neighbors are updated. The local search is stopped if either no eligible nonempty queues remain, or one of the stopping criteria described below applies. After the local search stops, it is rolled back to the lowest cut state reached during the search (which is the starting state if no improvement was achieved). Subsequently all previously marked nodes are unmarked. The local search is repeated until no improvement is achieved.

The main difference to the usual FM-algorithm is that our routine performs a highly localized search starting just at the uncontracted edge. Indeed, our local search does nothing if none of the uncontracted nodes is a *border node*, i.e., has a neighbor in another block. Other FM-algorithms initialize the search with all border nodes. In  $n$ -GP the local search may find an improvement quickly after moving a small number of nodes. However, in order to exploit this case, we need a way to stop the search much earlier than previous algorithms which limit the number of steps to a constant fraction of the current number of nodes  $|V|$ .

*Stopping Using a Random Walk Model.* It makes sense to make a stopping rule more adaptive by making it dependent on the past history of the search, e.g., on the difference between the current cut and the best cut achieved before.

We model the gain values in each step as identically distributed, independent random variables whose expectation  $\mu$  and Variance  $\sigma^2$  is obtained from the previously observed  $p$  steps. In the full paper we show how from these assumptions we can (heuristically) derive that it is unlikely that the local search will produce a better cut if

$$p\mu^2 > \alpha\sigma^2 + \beta \quad (1)$$

where  $\alpha$  and  $\beta$  are tuning parameters and  $\mu$  is the average gain since the last improvement. For the variance  $\sigma^2$ , we can also use the variance observed throughout the current local search. Parameter  $\beta$  is a base value that avoids stopping just after a small constant number of steps that happen to have small variance. Currently we set it to  $\ln n$ .

## 4 Trial Trees

It is a standard technique in optimization heuristics to improve results by repeating various parts of the algorithm. We generalize several approaches used in MGP by adapting an idea initially used in a fast randomized min-cut algorithm [15]: After reducing the number of nodes by a factor  $c$ , we perform two independent trials using different random seeds for tie breaking during contraction, initial partitioning, and local search. Among these trials the one with the smaller cut is used for continuing upwards. This way, we perform independent trials at many levels of contraction controlled by a single tuning parameter  $c$ . As long as  $c > 2$ , the total number of contraction steps performed stays  $\mathcal{O}(n)$ .

**Table 1.** Basic properties of the graphs from our benchmark set. The large instances are split into five groups: geometric graphs, FEM graphs, street networks, sparse matrices, and social networks. Within their groups, the graphs are sorted by size.

Medium sized instances			Large instances		
graph	<i>n</i>	<i>m</i>	graph	<i>n</i>	<i>m</i>
rgg17	$2^{17}$	1 457 506	fetooth	78 136	905 182
rgg18	$2^{18}$	3 094 566	598a	110 971	1 483 868
Delaunay17	$2^{17}$	786 352	ocean	143 437	819 186
Delaunay18	$2^{18}$	1 572 792	144	144 649	2 148 786
bcsstk29	13 992	605 496	wave	156 317	2 118 662
4elt	15 606	91 756	m14b	214 765	3 358 036
fesphere	16 386	98 304	auto	448 695	6 629 222
cti	16 840	96 464	deu	4 378 446	10 967 174
memplus	17 758	108 384	eur	18 029 721	44 435 372
cs4	33 499	87 716	af_shell10	1 508 065	51 164 260
pwt	36 519	289 588	Social networks		
bcsstk32	44 609	1 970 092	coAuthorCiteseer	227 320	1 628 268
body	45 087	327 468	coAuthorhDBLP	299 067	1 955 352
t60k	60 005	178 880	cnr2000	325 557	3 216 152
wing	62 032	243 088	citationCiteseer	434 102	32 073 440
finan512	74 752	522 240	coPaperDBLP	540 486	30 491 458
ferotor	99 617	662 431			
bel	463 514	1 183 764			
nld	893 041	2 279 080			
af_shell9	504 855	17 084 020			

## 5 Experiments

*Implementation.* We implemented KaSPar in C++ using gcc-4.3.2. We use priority queues based on paring heaps [28] available in the policy-based elementary data structures library (pb\\_ds) for implementing contraction and refinement procedures. In the following experimental study we compared KaSPar to Scotch 5.1, kMetis 4.0 and the same version of KaPPa as in [14].

*System.* We performed our experiments on a single core of an Intel Xeon Quad-core Processor featuring 2x4 MB of L2 cache and clocked at 2.667 GHz of a 2 processor Intel Xeon X5355 node with 16 GB of RAM running Suse Linux Enterprise 10.

*Instances.* We report results on two suites of instances summarized in Table II. *rggX* is a *random geometric graph* with  $2^X$  nodes that represent random points in the unit square and edges connect nodes whose Euclidean distance is below  $0.55\sqrt{\ln n/n}$ . This threshold was chosen in order to ensure that the graph is almost connected. *DelaunayX* is the Delaunay triangulation of  $2^X$  random points in the unit square. Graphs *bcsstk29..ferotor* and *fetooth..auto* come from Chris Walshaw's benchmark archive [27]. Graphs *bel*, *nld*, *deu* and *eur* are undirected versions of the road networks of Belgium, the Netherlands, Germany, and

Western Europe respectively, used in [3]. Instances `af_shell9` and `af_shell10` come from the Florida Sparse Matrix Collection [2]. `coAuthorsDBLP`, `coPapersDBLP`, `citationCiteseer`, `coAuthorsCiteseer` and `cnr2000` are examples of social networks taken from [9]. All node and edge weights are one.

For the number of partitions  $k$  we choose the values used in [27]: 2, 4, 8, 16, 32, 64. Our default value for the allowed imbalance is 3 % since this is one of the values used in [27] and the default value in Metis.

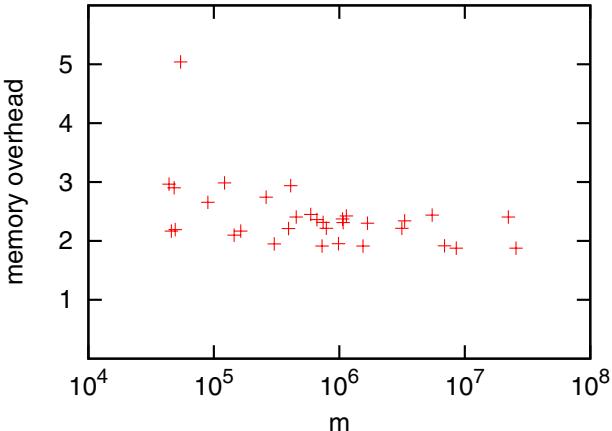
When not otherwise mentioned, we perform 10 repetitions for the small networks and 5 repetitions for the other. We report the arithmetic average of computed cut size, running time and the best cut found. When further averaging over multiple instances, we use the geometric mean in order to give every instance the same influence on the final figure.

*Configuring the Algorithm.* We use two sets of parameter settings *fast* and *strong*. These methods only differ in the constant factor  $\alpha$  in the local search stopping rule, see Equation (1), in the contraction factor  $c$  for the trial tree (Section 4), and in the number of initial partitioning attempts  $a$  performed at the coarsest level of contraction:

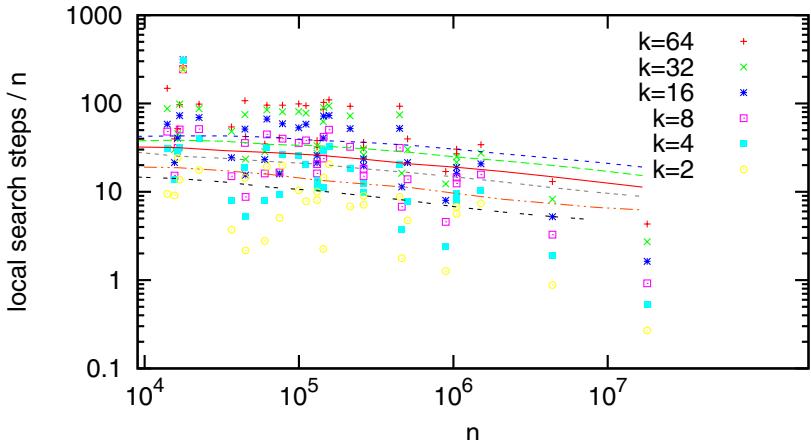
strategy	$\alpha$	$c$	$a$
fast	1	8	$25/\log_2 k$
strong	4	2.5	$100/\log_2 k$

Note that this are considerably less parameters compared to KaPPa. In particular, there is no need for selecting a matching algorithm, an edge coloring algorithm, or global and local iterations for refinement.

*Scalability.* Figure 3 shows the number of edges touched during contraction (KaSPar strong, small and large instances) relative to the input size. We see that this scales linearly with the number of input edges and with a fairly small



**Fig. 3.** Number of edges created during contraction



**Fig. 4.** Total number of local search steps. The nearly straight lines represent series for the graphs rgg15..rgg24 and Delaunay15..Delaunay24 for different  $k$ .

constant factor between 2 and 3. Interestingly, the number of local search steps during local improvement (Figure 4) *decreases* with increasing input size. This can be explained by the sublinear number of border vertices that we have in graphs that have small cuts and by small average search space sizes for the local search. Indeed, experiments shown in the full paper indicate that the average length of local searches grows only logarithmically with  $n$ . All this translates into fairly complicated running time behavior. Still, experiments discussed in the full paper warrant the conclusion that running time scales “near linearly” with the input size. The term in the running time depending on  $k$  grows sublinearly with the input size so that for very large graphs the number of blocks does not matter much.

*Does the Random Walk Model Work?* We have compared KaSPar fast with a variant where the stopping rule is disabled (i.e.,  $\alpha = \infty$ ). For the small instances this yields about 1 % better cut sizes at the cost of an order of magnitude larger running time. This is a small improvement both compared to the improvement KaSPar achieves over other systems and compared to just repeating KaSPar fast 10 times (see Table 2).

*Do Trial trees help?* We use the following evaluation: We run KaSPar strong and measure its elapsed time. Then for different values of initial partitionings  $a$  we repeat KaSPar strong without trial trees ( $c = 0$ ), until the sum of the run times of all repetitions exceeds the run time of KaSPar strong. Than for different values  $a$  we compare the best edge cut achieved during repeated runs to the one produced by KaSPar strong. Finally, we average the obtained results over 5 repetitions of this procedure. If we then compare the computed partitions, we usually get almost identical results (a fraction of a percent difference). However, most of the time trial trees are a bit better and for *road networks* we

get considerable improvements. For example, for the European network we get an improvement of 10 % on average over all  $k$ .

*Comparison with other Systems.* Table 2 summarizes the results by computing geometric means over 10 runs for the small instances and over 5 runs for the large instances and social networks. We exclude the European road network for  $k = 2$  because KaPPa runs out of memory in that case. Detailed per instance results can be found in the full paper. KaPPa strong produces 5.9 % larger cuts than KasPar strong for small instances (average value) and 8.1 % larger cuts for the large instances. This comparison might seem a bit unfair because KaPPa is about five times faster. However, KaPPa is using  $k$  processors in parallel. Indeed, for  $k = 2$  KaSPar strong needs only about twice as much time. Also note that KaPPa strong needs about twice as much time as KaSPar fast while still producing 6 % larger cuts despite running in parallel. The case  $k = 2$  is also interesting because here KaPPa and KaSPar are most similar – parallelism does not play a big role (2 processors) and both local search strategies work only on two blocks at all time. Therefore 6 % improvement of KaSPar over KaPPa we can attribute mostly to the larger number of levels.

Scotch and kMetis are much faster than KaSPar but also produce considerably larger cuts – e.g., 32 % larger for large instances (kMetis, average). For the European road network (not in the table, see above), the difference in cut size even exceeds a factor of two. Such gaps usually cannot be breached by just running the faster solver a larger number of times. For example, for large instances, Scotch is only a factor around 4 faster than KaSPar fast, yet its best cut values obtained from 5 runs are still 12.7 % larger than the average values of KaSPar fast.

For social networks all systems have problems. KaSPar lags further behind in terms of speed but extends its lead with respect to the cut size. We mostly attribute the larger run time to the larger cut sizes relative to the number of nodes which greatly increase the number of local searches necessary. A further effect may be that the time for a local search step is proportional to the number of partitions adjacent to the nodes participating in the local search. For “well behaved” graphs this is mostly two, but for social networks which get denser on the coarser levels this value can get larger.

*The Walshaw Benchmark* [27] considers 34 graphs using  $k \in \{2, 4, 8, 16, 32, 64\}$  and balance parameter  $\epsilon \in \{0, 0.01, 0.03, 0.05\}$  giving a total of 816 table entries. Only cut sizes count – running time is not reported. We tried all combinations except the case  $\epsilon = 0$  which KaSPar cannot handle yet. We ran KaSPar strong with a time limit of one hour and report the best result obtained in the full paper. KaSPar improved 155 values in the benchmark table: 42 for 1%, 49 for 3% and 64 for 5% allowed imbalance. Moreover, it reproduced equally sized cuts in 83 cases. If we count only results for graphs having over  $44k$  nodes and  $\epsilon > 0$ , KaSPar improved 131 and reproduced 27 cuts, thus summing up to 63% of large graph table slots. We should note, that 51 of the new improvements are over partitioners different from KaPPa. Most of the improvements lie in the lower triangular part of the table, meaning that KaSPar is particularly good for

**Table 2.** Geometric means (times, cut values) over all instances

code	small graphs			large graphs			social networks		
	best	avg.	t[s]	best	avg.	t[s]	best	avg.	t[s]
KaSPar strong	2 675	2 729	7.37	12 450	12 584	87.12	-	-	-
KaSPar fast	2 717	2 809	1.43	12 655	12 842	14.43	93657	99062	297.34
KaSPar fast, $\alpha = \infty$	2 697	2 780	23.21	-	-	-	-	-	-
KaPPa strong	2 807	2 890	2.10	13 323	13 600	28.16	117701	123613	78.00
KaPPa fast	2 819	2 910	1.29	13 442	13 727	16.67	117927	126914	46.40
kMetis	3 097	3 348	0.07	15 540	16 656	0.71	117959	134803	1.42
Scotch	2 926	3 065	0.48	14 475	15 074	3.83	168764	168764	17.69

k	Large Instances					
	KaSPar strong			KaPPa strong		
	best	avg.	t[s]	best	avg.	t[s]
2	2 842	2 873	36.89	2 977	3 054	15.03
4	5 642	5 707	60.66	6 190	6 384	30.31
8	10 464	10 580	75.92	11 375	11 652	37.86
16	17 345	17 567	102.52	18 678	19 061	39.13
32	27 416	27 707	137.08	29 156	29 562	31.35
64	41 284	41 570	170.54	43 237	43 644	22.36

either large graphs, or smaller graphs with small  $k$ . On the other hand, for small graphs, large  $k$ , and  $\epsilon = 1\%$  KaSPar was often not able to obtain a feasible solution. A primary reason for this seems to be that initial partitioning yields highly infeasible solutions that KaSPar is not able to improve considerably during refinement. This is not astonishing, since Scotch targets  $\epsilon = 3\%$  and does not even guarantee that.

## 6 Conclusion

$n$ -GP is a graph partitioning approach that scales to large inputs and currently computes the best known partitions for many large graphs, at least when a certain imbalance is allowed. It is in some sense simpler than previous methods since no matching algorithm is needed. Although our current implementation of KaSPar is a considerable constant factor slower than the fastest available MGP partitioners, we see potential for further tuning. In particular, thanks to our adaptive stopping rule, KaSPar needs to do very little local search, in particular for large graphs and small  $k$ . Thus it suffices to tune the relatively simple contraction routine to obtain significant improvements. On the other hand, the adaptive stopping rule might also turn out to be useful for matching based MGP algorithms.

A lot of opportunities remain to further improve KaSPar. In particular, we did not yet attempt to handle the case  $\epsilon = 0$  since this may require different local search strategies. We also want to try other initial partitioning algorithms and ways to integrate  $n$ -GP into other metaheuristics like evolutionary search.

We expect that  $n$ -GP could be generalized for other objective functions, for hypergraphs, and for graph clustering. More generally, the success of  $n$ -GP also suggests to look for more applications of the  $n$ -level paradigm.

An apparent drawback of  $n$ -GP is that it looks inherently sequential. However, we could probably obtain a good parallel algorithm by contracting *small* sets of highly rated, independent edges in parallel. Since this obviates the need for parallel matching algorithms, a parallelization of  $n$ -GP might be fast and simple.

*Acknowledgements.* We would like to thank Christian Schulz for supplying data for KaPPa, Scotch and Metis.

## References

1. Birn, M., Holtgrewe, M., Sanders, P., Singler, J.: Simple and fast nearest neighbor search. In: 11th Workshop on Algorithm Engineering and Experiments (2010)
2. Davis, T.: The University of Florida Sparse Matrix Collection (2008),  
<http://www.cise.ufl.edu/research/sparse/matrices>
3. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
4. Dementiev, R., Sanders, P., Schultes, D., Sibeyn, J.: Engineering an external memory minimum spanning tree algorithm. In: IFIP TCS, Toulouse (2004)
5. Drake, D., Hougardy, S.: Improved linear time approximation algorithms for weighted matchings. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 14–23. Springer, Heidelberg (2003)
6. Fiduccia, C.M., Mattheyses, R.M.: A Linear-Time Heuristic for Improving Network Partitions. In: 19th Conf. on Design Automation, pp. 175–181 (1982)
7. Fjallstrom, P.: Algorithms for graph partitioning: A survey. Linkoping Electronic Articles in Computer and Information Science 3(10) (1998)
8. Karypis, V.K.G.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20(1), 359–392 (1998)
9. Geisberger, R., Sanders, P., Schultes, D.: Better approximation of betweenness centrality. In: 10th Workshop on Algorithm Engineering and Experimentation, San Francisco, pp. 90–108. SIAM, Philadelphia (2008)
10. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 319–333. Springer, Heidelberg (2008)
11. Guibas, L.J., Knuth, D.E., Sharir, M.: Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica 7(4), 381–413 (1992)
12. Hendrickson, B.: Chaco: Software for partitioning graphs,  
<http://www.sandia.gov/~bahendr/chaco.html>
13. Holtgrewe, M.: A scalable coarsening phase for a multi-level partitioning algorithm. Diploma thesis, Universität Karlsruhe (2009)
14. Holtgrewe, M., Sanders, P., Schulz, C.: Engineering a scalable high quality graph partitioner. In: 24th IEEE International Parallel and Distributed Processing Symposium (to appear, 2010), arXiv:0910.2004
15. Karger, D.R., Stein, C.: A new approach to the minimum cut problem. J. ACM 43(4), 601–640 (1996)

16. Karypis, G., Kumar, V.: MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, <http://www.cs.umn.edu/~metis>
17. Karypis, G., Kumar, V.: MeTiS, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0 (1998)
18. Maué, J., Sanders, P.: Engineering algorithms for approximate weighted matching. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 242–255. Springer, Heidelberg (2007)
19. Meyerhenke, H., Monien, B., Sauerwald, T.: A new diffusion-based multilevel algorithm for computing graph partitions. Journal of Parallel and Distributed Computing 69(9), 750–761 (2009)
20. Pellegrini, F.: SCOTCH: Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package (2007), <http://www.labri.fr/perso/pelegin/scotch/>
21. Pellegrini, F.: SCOTCH 5.1 User's guide. Technical report, Laboratoire Bordelais de Recherche en Informatique, Bordeaux, France (2008)
22. Pettie, S., Sanders, P.: A simpler linear time  $2/3 - \epsilon$  approximation for maximum weight matching. Information Processing Letters 91(6), 271–276 (2004)
23. Preis, R.: PARTY Partitioning Library (1996), <http://www2.cs.uni-paderborn.de/cs/robsy/party.html>
24. Preis, R.: Linear time  $1/2$ -approximation algorithm for maximum weighted matching in general graphs. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 259–269. Springer, Heidelberg (1999)
25. Preis, R., Diekmann, R.: The PARTY Partitioning Library, User Guide. Technical report, University of Paderborn, Germany, Tr-rsfb-96-02 (1996)
26. Schloegel, K., Karypis, G., Kumar, V.: Graph partitioning for high performance scientific simulations. Technical Report 00-018, University of Minnesota (2000)
27. Soper, A.J., Walshaw, C., Cross, M.: A combined evolutionary search and multilevel optimisation approach to graph partitioning. J. Global Optimization 29(2), 225–241 (2004), <http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/>
28. Tavory, A., Dreizin, V., Kosnik, B.: Policy-based data structures. IBM Haifa and Redhat (2004), [http://gcc.gnu.org/onlinedocs/libstdc++/ext/pb\\_ds/](http://gcc.gnu.org/onlinedocs/libstdc++/ext/pb_ds/)
29. Walshaw, C.: JOSTLE –graph partitioning software (2005), <http://staffweb.cms.gre.ac.uk/~wc06/jostle/>
30. Walshaw, C., Cross, M.: JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview. In: Magoules, F. (ed.) Mesh Partitioning Techniques and Domain Decomposition Techniques, pp. 27–58. Civil-Comp Ltd. (2007) (invited chapter)

# Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns

Hannah Bast<sup>1,2</sup>, Erik Carlsson<sup>2</sup>, Arno Eigenwillig<sup>2</sup>, Robert Geisberger<sup>3,2</sup>,  
Chris Harrelson<sup>2</sup>, Veselin Raychev<sup>2</sup>, and Fabien Viger<sup>2</sup>

<sup>1</sup> Albert-Ludwigs-Universität Freiburg, 79110 Freiburg, Germany

<sup>2</sup> Google, Brandschenkestrasse 110, 8002 Zürich, Switzerland

<sup>3</sup> Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany

**Abstract.** We show how to route on very large public transportation networks (up to half a billion arcs) with average query times of a few milliseconds. We take into account many realistic features like: traffic days, walking between stations, queries between geographic locations instead of a source and a target station, and multi-criteria cost functions. Our algorithm is based on two key observations: (1) many shortest paths share the same *transfer pattern*, i.e., the sequence of stations where a change of vehicle occurs; (2) *direct connections* without change of vehicle can be looked up quickly. We precompute the respective data; in practice, this can be done in time linear in the network size, at the expense of a small fraction of non-optimal results. We have accelerated public transportation routing on Google Maps with a system based on our ideas. We report experimental results for three data sets of various kinds and sizes.

## 1 Introduction

In recent years, several algorithms have been developed that, after a precomputation, find shortest paths on the road network of a whole continent in a few microseconds, which is a million times faster than Dijkstra's algorithm. However, none of the tricks behind these algorithms yields similar speed-ups for public transportation networks of comparable sizes, especially when they are realistically modeled and show poor structure, like bus-only networks in big metropolitan areas. In this paper we present a new algorithm for routing on public transportation networks that is fast even when the network is realistically modeled, very large and poorly structured. These are the challenges faced by public transportation routing on Google Maps (<http://www.google.com/transit>), and our algorithm has successfully addressed them. It is based on the following new idea.

Think of the query  $A@t \rightarrow B$ , with source station  $A =$  Freiburg, target station  $B =$  Zürich, and departure time  $t = 10:00$ . Without assuming anything about the nature of the network and without any precomputation, we would have to do a Dijkstra-like search and explore many nodes to compute an optimal path. Now let us assume that we are given the following additional information: each

and every optimal path from Freiburg to Zürich, no matter on which day and at which time of the day, either is a direct connection (with no transfer in between) or it is a trip with exactly one transfer at Basel. We call *Freiburg – Zürich* and *Freiburg – Basel – Zürich* the optimal *transfer patterns* between Freiburg and Zürich (for each optimal path, take the source station, the sequence of transfers, and the target station). Note how little information the set of optimal transfer patterns for this station pair is. Additionally, let us assume that we have timetable information for each station that allows us to very quickly determine the next *direct* connection from a given station to some other station.

With this information, it becomes very easy to answer the query  $A@t \rightarrow B$  for an arbitrary given time  $t$ . Say  $t = 10:00$ . Find the next direct connection from Freiburg to Zürich after  $t$ . Say it leaves Freiburg at 12:55 and arrives in Zürich at 14:52. (There are only few direct trains between these two stations over the day.) Also find the next direct connection from Freiburg to Basel after  $t$ . Say it leaves Freiburg at 10:02 and arrives in Basel at 10:47. Then find the next direct connection from Basel to Zürich after 10:47. Say it leaves Basel at 11:07 and arrives in Zürich at 12:00. In our cost model (see Section 3) these two connections are incomparable (one is faster, and the other has less transfers), and thus we would report both. Since the two given transfer patterns were the only optimal ones, we can be sure to have found all optimal connections. And we needed only three direct-connection queries to compute them.

Conceptually, our whole scheme goes as follows. The set of all optimal transfer patterns between all station pairs is too large to precompute and store. We therefore precompute a set of *parts of* transfer patterns such that all optimal transfer patterns can be combined from these parts. For our three datasets, we can precompute such parts in 20–40 core hours per 1 million departure/arrival events and store them in 10–50 MB per 1000 stations. From these parts, also non-optimal transfer patterns can be combined, but this only means additional work at query time; it will not let us miss any optimal connections. Think of storing parts of transfer patterns, to be recombined at query time, as a lossy compression of the set of all optimal transfer patterns. We also precompute a data structure for fast direct-connection queries, which, for our three datasets, needs 3–10 MB per 1 000 stations and has a query time of 2–10  $\mu$ s.

Having this information precomputed, we then proceed as follows for a given query  $A@t \rightarrow B$ . From the precomputed parts, we compute all combinations that yield a transfer pattern between  $A$  and  $B$ . We overlay all these patterns to form what we call the *query graph*. Finding the optimal connection(s) amounts to a shortest-path computation on the query graph with source  $A$  and target  $B$ , where each arc evaluation is a direct-connection query. The query graph for our simple example from above has three nodes ( $A = \text{Freiburg}$ ,  $B = \text{Zürich}$ , and  $C = \text{Basel}$ ) and three arcs ( $A \rightarrow B$ ,  $A \rightarrow C$ ,  $C \rightarrow B$ ). Due to the non-optimal transfer patterns that come from the recombination of the precomputed parts, our actual query graphs typically have several hundreds of arcs. However, since direct-connection queries can be computed in about 10  $\mu$ s, this will still give us query times on the order of a few milliseconds, and by the way our approach works, these times are independent of the size of the network.

## 2 Related Work

The most successful “tricks of the trade” for fast routing on transportation networks can be summarized under the following five headings: *bidirectional search*, *exploiting hierarchy*, *graph contraction*, *goal direction*, and *distance tables*. The recent overview article [2] describes each of these and provides evidence and explanations why they give excellent speed-ups on road networks, but fail to do so on public transportation networks, especially such with poor structure. Two recent surveys on fast routing on road networks and on public transportation networks, respectively, are [5] and [10].

A fully realistic model like ours was recently considered in [6] and [3]. However, the network considered in those papers is relatively small (about 8900 stations) and very well-structured (German trains, almost no local transport). Also, there are only very few walking arcs, as walking between stations is rarely an issue for pure train networks. Reported query times are about one second for [6] and a few hundred milliseconds for [3]. The title of the latter paper aptly states that obtaining speed-ups for routing on public transportation networks in a realistic model “is harder than expected”.

The best query times so far, of around 1 ms, were achieved in [4] and [7]. However, their model does support neither walking between stations, nor traffic days, nor multi-criteria costs, and, especially for the latter, it looks unlikely that their algorithms can be suitably extended. These two papers considered a graph of the European long-distance trains, and also two local networks (Berlin and Frankfurt), of size about 10 000 stations each.

Our algorithm is the first to yield fast query times (on the order of a few milliseconds) on a fully realistic model for public transportation networks also with poor structure (like bus-only networks) and of sizes more than an order of magnitude larger than what was considered so far.

## 3 Problem Formalization

A timetable describes the available trips of vehicles (buses, trains, ferries etc.) along sequences of stations (bus stops, train stations, ports etc.), including the times of day at which they depart and arrive. For routing, it is represented as a graph, see [11] for a comparison of various graph models. We use a *time-expanded graph* with three kinds of nodes, each carries a time and belongs to a station. For every *elementary connection* from station  $A$  to the next station  $B$  on the same trip, we put a *departure node*  $Ad@t_1$  at  $A$  with the departure time  $t_1$ , an *arrival node*  $Ba@t_2$  at  $B$  with the arrival time  $t_2$  and an arc  $Ad@t_1 \rightarrow Ba@t_2$  to model riding this vehicle from  $A$  to  $B$ . If the vehicle continues from  $B$  at time  $t_3$ , we put an arc  $Ba@t_2 \rightarrow Bd@t_3$  that represents staying on the vehicle at  $B$ . This is possible no matter how small the difference  $t_3 - t_2$  is.

Transfers at  $B$  shall be possible only to departures after a *minimum transfer duration*  $\Delta t_B$ . For each departure node  $Bd@t$  we put a *transfer node*  $Bt@t$  at the same time and an arc  $Bt@t \rightarrow Bd@t$  between them. Also, we put an arc  $Bt@t \rightarrow Bt@t'$  to the transfer node at  $B$  that comes next in the ascending order

of departure times (with ties broken arbitrarily); these arcs form the *waiting chain* at  $B$ . Now, to allow a transfer after having reached  $Ba@t_2$ , we put an arc to the first transfer node  $Bt@t$  with  $t \geq t_2 + \Delta t_B$ . This gives the opportunity to transfer to that and all later departures from  $B$ .

For exposition, we regard the graph as fully time-expanded, meaning times increase unbounded from time 0 (midnight of day 0). In practice, we exploit the periodicity of timetables by using times modulo 24 hours and bit masks to indicate a trip's traffic days. Also, we allow additional transfers by walking to nearby stations. See Section 6 for details. In our implementation, the resulting graphs can be stored in about 35 MB of memory per 1 million elementary connections.

Our scheme supports a fairly general class of multi-criteria cost functions and optimality notions. In our implementation, a cost is a pair  $(d, p)$  of non-negative duration and penalty. Duration of an arc is the difference in time between its endpoints. Penalty applies mostly to transfers: each station  $B$  defines a fixed penalty score for transferring, and that is the penalty component of the cost of arcs  $Ba@t \rightarrow Bt@t'$ . The arcs from departure to arrival nodes may be given a small penalty score for using that elementary connection. Other arcs, in particular waiting arcs, have penalty zero. The cost of a path in the graph is the component-wise sum of the costs of the arcs.

We say cost  $(d_1, p_1)$  *dominates* or *is better than* cost  $(d_2, p_2)$  in the Pareto sense if  $d_1 \leq d_2$  and  $p_1 \leq p_2$  and one of the inequalities is strict. Each finite set of costs has a unique subset of (Pareto-)optimal costs that are pairwise non-dominating but dominate any other cost in the set (in the Pareto sense).

**Definition 1.** Consider a station-to-station query  $A@t \rightarrow B$ . Take the first transfer node  $At@t'$  with  $t' \geq t$ . For this query, we extend the graph by a source node  $S$  with an arc of duration  $t' - t$  and penalty 0 that leads to  $At@t'$  and by a target node  $T$  with incoming arcs of zero cost from all arrival nodes of  $B$ .

The paths from  $S$  to  $T$  are the feasible connections for the query. If the cost of a feasible connection is not dominated by any other, we call them optimal cost and optimal connection, respectively, for the query.

The query's result is an optimal set of connections, that is, a set of optimal connections containing exactly one for each optimal cost.

Note that the waiting chain at  $A$  makes paths from  $S$  through all departure nodes after time  $t$  feasible. We exclude multiple connections for the same optimal cost. They do occur (even for a single-criterion cost function) but add little value.<sup>1</sup>

## 4 Basic Algorithm

In this section we present a first simple algorithm that illustrates the key ideas. It has very fast query times but a quadratic precomputation complexity.

<sup>1</sup> Connections of equal cost, relative to query time  $t$ , arrive at the same time. It is preferable to choose one that departs as late as possible from  $A$ ; we will return to that in Section 6. Those that depart latest often differ in trivial ways (e.g., using this or that tram between two train stations), so returning just one is fine.

#### 4.1 Fast Direct-Connection Queries

**Definition 2.** For a direct-connection query  $A@t \rightarrow B$ , the feasible connections are defined as in Definition 1, except that only connections without transfers are permitted. The result of the query are the optimal costs in this restricted set.

The following data structure answers direct-connection queries in about  $10\ \mu\text{s}$ .

1. Precompute all *trips* (maximal paths in the graph without transfer nodes) and group them into *lines*  $L_1, L_2, \dots$  such that all trips on a line share the same sequence of stations, do not overtake each other (FIFO property, like the *train routes* in [1]), and have the same penalty score between any two stations.

The trips of a line are sorted by time and stored in a 2D array like this:

line L17	S154	S097	S987	S111	...
trip 1	8:15	8:22 8:23	8:27 8:29	8:38 8:39	...
trip 2	9:14	9:21 9:22	9:28 9:28	9:37 9:38	...
...	...	...	...	...	...

2. For each station, precompute the sorted list of lines incident to it and its position(s) on each line. For example:

$$\begin{aligned} S097: & (L8, 4) (L17, 2) (L34, 5) (L87, 17) \dots \\ S111: & (L9, 1) (L13, 5) (L17, 4) (L55, 16) \dots \end{aligned}$$

3. To answer a direct-connection query  $A@t \rightarrow B$ , intersect the incidence lists of  $A$  and  $B$ . For each occurrence of  $A$  before  $B$  on a line, read off the cost of the earliest feasible trip, then choose the optimal costs among all these.

In our example, the query  $S097@9:03 \rightarrow S111$  finds positions 2 and 4 on  $L_{17}$  and the trip that reaches  $S111$  at 9:37.

**Lemma 1.** A query  $A@t \rightarrow B$  to the direct-connection data structure returns all optimal costs of direct connections.

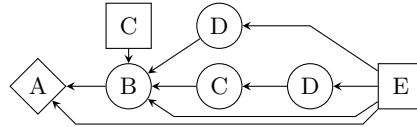
*Proof.* The straightforward proof can be found in the extended version [1].

#### 4.2 Transfer patterns precomputation

**Definition 3.** For any path, consider the subsequence of nodes formed by the first node, each arrival node whose successor is a transfer node, and the last node. The sequence of stations of these nodes is the transfer pattern of the path.

An optimal set of transfer patterns for a pair  $(A, B)$  of stations is a set  $S$  of transfer patterns such that for all queries  $A@t \rightarrow B$  there is an optimal set of connections whose transfer patterns are contained in  $S$ , and such that each element in  $S$  is the transfer pattern of an optimal connection for a query  $A@t \rightarrow B$  at some time  $t$ .

For every source station  $A$ , we compute optimal sets of transfer patterns to all stations  $B$  reachable from it and store them in one DAG for  $A$ . This DAG has three different types of nodes: one *root node* labeled  $A$ , for each reachable station  $B$  a *target node* labeled  $B$ , and for each transfer pattern prefix



**Fig. 1.** DAG for the transfer patterns ‘AE’, ‘ABE’, ‘ABC’, ‘ABDE’ and ‘ABCDE’. The root node is the diamond, prefix nodes are circles and target stations are rectangles. There are potentially several prefix nodes with the same label: In our example, ‘D’ occurs twice, the top one representing the prefix ‘ABD’ and the bottom one ‘ABCD’.

$AC_1 \dots C_i$ , occurring in at least one transfer pattern  $AC_1 \dots C_i \dots B$ , a *prefix node* labeled  $C_i$ . They are connected in the natural way such that precisely the transfer patterns we want to store are represented by a path from their target stations to the root node, labeled in reverse order; Figure 1 shows an example. We use the following algorithm *transfer\_patterns*( $A$ ).

1. Run a multi-criteria variant of Dijkstra’s algorithm [9,13,8] starting from labels of cost zero at all *transfer* nodes of station  $A$ .
2. For every station  $B$ , choose optimal connections with the *arrival chain algorithm*: For all distinct arrival times  $t_1 < t_2 < \dots$  at  $B$ , select a dominant subset in the set of labels consisting of (i) those settled at the arrival node(s) at time  $t_i$  and (ii) those selected for time  $t_{i-1}$ , with duration increased by  $t_i - t_{i-1}$ ; ties to be broken in preference of (ii).
3. Trace back the paths of all labels selected in Step 2. Create the DAG of transfer patterns of these paths by traversing them from the source  $A$ .

**Lemma 2.** *If  $c$  is an optimal cost for the query  $A@t_0 \rightarrow B$ , *transfer\_patterns*( $A$ ) computes the transfer pattern of a feasible connection for the query that realizes cost  $c$ .*

*Proof.* Let  $c = (d, p)$ . The label set for time  $t_0 + d$  keeps a label with penalty  $p$  that departs at or after  $t_0$ . This needs that duration and penalty are optimized independently (i.e., Pareto-style). For details, see the extended version [1].

Running *transfer\_patterns*( $A$ ) for all stations  $A$  is easy to parallelize by splitting the set of source stations  $A$  between machines, but even so, the total running time remains an issue. We can estimate it as follows. Let  $s$  be the number of stations, let  $n$  be the average number of nodes per station (< 569 for all our graphs, with the optimizations of Section 6), and let  $\ell$  be the average number of settled labels per node and per run of *transfer\_patterns*( $A$ ) (< 16 in all our experiments, in the setting of Sections 6 and 7). Then the total number of labels settled by *transfer\_patterns*( $A$ ) for all stations  $A$  is  $L = \ell \cdot n \cdot s^2$ .

Steps 1–3 have running time essentially linear in  $L$ , with logarithmic factors for maintaining various sets. (For Step 1, this includes the priority queue of Dijkstra’s algorithm. The bounded out-degree of our graphs bounds the number of unsettled labels linearly in  $L$ .) Since  $L$  is quadratic in  $s$ , this precomputation is

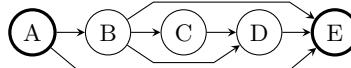
infeasible in practice for large networks, despite parallelization. We will address this issue in Sections 5 and 7.

### 4.3 Query Graph Construction and Evaluation

For a query  $A@t \rightarrow B$ , we build the *query graph* as follows, independent of  $t$ :

1. Fetch the precomputed transfer patterns DAG for station  $A$ .
2. Search target node  $B$  in the DAG. Assume it has  $\ell$  successor nodes with labels  $C_1, \dots, C_\ell$ . Add the arcs  $(C_1, B), \dots, (C_\ell, B)$  to the query graph.
3. Recursively perform Step 2 for each successor node with a label  $C_i \neq A$ .

Figure 2 shows the query graph from A to E built from the DAG in Figure 1.



**Fig. 2.** Query graph  $A \rightarrow E$  from transfer patterns ‘AE’, ‘ABE’, ‘ABDE’ and ‘ABCDE’.

**Lemma 3.** *For each transfer pattern  $AC_1 \dots C_k B$  in the DAG there exists the path  $\langle A, C_1, \dots, C_k, B \rangle$  in the constructed query graph.*

*Proof.* The straightforward proof can be found in the extended version [1].

Given the query graph, evaluating the query is simply a matter of a time-dependent multi-criteria Dijkstra search [6] on that graph. Labels in the queue are tuples of station and cost (time and penalty). Relaxing an arc  $C \rightarrow D$  for a label with time  $t$  amounts to a direct-connection query  $C@t \rightarrow D$ .

By storing parent pointers from each label to its predecessor on the shortest path, we eventually obtain, for an optimal label at the target station, the sequence of transfers on an optimal path from the source to the target, as well as the times at which we arrive at each of these transfers. More details on the optimal paths can be provided by augmenting the direct-connection data structure.

**Theorem 1.** *For a given query  $A@t \rightarrow B$ , the described search on the query graph from A to B returns the set of optimal costs and for each such cost a corresponding path.*

*Proof.* We precomputed transfer patterns for each optimal cost of the query (Lemma 2) and the paths connecting these transfer stations are in our query graph (Lemma 3). From the correctly answered direct-connection queries (Lemma 1), the time-dependent Dijkstra algorithm on the query graph computes all optimal costs including matching paths.

## 5 Hub Stations

The preprocessing described in Section 4.2 uses quadratic time and produces a result of quadratic size. To reduce this, we do these expensive *global* searches only from a suitably preselected set of *hubs* and compute transfer patterns from hubs to all other stations.<sup>2</sup> For all non-hub stations, we do *local* searches computing

<sup>2</sup> Computing transfer patterns only to other hubs is not faster.

only those transfer patterns without hubs or their parts up to the first hub. More precisely, let  $AC_1 \dots C_k B$  be a transfer pattern from a non-hub  $A$  that we would have stored in Section 4.2. If any of the  $C_i$  is a hub, we do not store this pattern any more. The hub  $C_i$  with minimal  $i$  is called an *access station* of  $A$ . We still store transfer patterns  $A \dots C_i$  and  $C_i \dots B$  into and out of the access station. This shrinks transfer patterns enough to allow query processing entirely from main memory on a single machine, even for large networks (see Section 8).<sup>3</sup>

*Selecting the hubs.* We create a time-independent graph by overlaying the nodes and arcs of each line (as computed in Section 4.1), using the minimum of arc costs. Then, we perform cost-limited Dijkstra searches from a random sample of source stations. The stations being on the largest number of shortest paths are chosen as hubs.<sup>4</sup>

*Transfer patterns computation.* The *global* search remains as described in Section 4.2. The *local* search additionally marks labels stemming from labels at transfer nodes of hubs as *inactive*, and stops as soon as all unsettled labels are inactive [12]. Inactive labels are ignored when the transfer patterns are read off.<sup>5</sup>

*Query graph.* Processing a query  $A@t \rightarrow B$  looks up the set  $\mathcal{X}$  of access stations of  $A$  and constructs the query graph from the transfer patterns between the station pairs  $\{(A, B)\} \cup (\{A\} \times \mathcal{X}) \cup (\mathcal{X} \times \{B\})$ . The evaluation of the query graph remains unchanged.

**Lemma 4.** *If  $c$  is an optimal cost for the query  $A@t_0 \rightarrow B$ , then the query graph from  $A$  to  $B$  contains the transfer pattern of a feasible connection for the query that realizes cost  $c$ .*

*Proof.* If no suitable transfer pattern  $A \dots B$  was computed, then  $A$  is a non-hub with an access station  $X$  such that a suitable transfer pattern has been computed in two parts  $A \dots X$  and  $X \dots B$ , as we show in the extended version [1]. This needs that duration and penalty are optimized independently (i.e., Pareto-style).

## 6 Further Refinements

Above, we simplified the presentation of our algorithm. Our actual implementation includes the following refinements.

*Location-to-location.* Our implementation answers location-to-location queries. In the model from Definition 1, the source and target node now stand for the source and target locations, and they are connected to a selection of source and

<sup>3</sup> The number of global searches could be reduced further by introducing several levels of hubs, but in our implementation the total cost for the global searches is below the total cost for the local searches already with one level of hubs; see Section 8.

<sup>4</sup> We experimented with a variety of hub selection strategies, but they showed only little difference with respect to preprocessing time and query graph sizes, and so we stuck with the simplest strategy.

<sup>5</sup> Before that, inactive labels are needed to dominate non-optimal paths around hubs.

target stations nearby with arcs that take the walking cost into consideration. The query graph is built from transfer patterns for all pairs of source and target stations. This feature is of high practical value for dense metropolitan networks.

*Walking arcs for transfers.* Likewise, transfers by walking from stations to nearby stations are very important in metropolitan networks. We add arcs from arrival nodes to transfer nodes of useful nearby stations whose costs reflect the additional walking. This results in about twice more arcs in the graph and duplicates certain entities in the algorithm: transfer patterns now alternate between riding a vehicle and walking, the query graph requires two nodes per station, and two global searches from each hub are necessary, as the hub can either be the arrival or departure station of a transfer.

*More compact graph model.* In the precomputation, we optimize the representation of the graph from Section 3 in two ways. Departure nodes are removed and their predecessors (transfer node and maybe arrival node) are linked directly to their successor (the next arrival node), cf. [III, §8.1.2]. To exploit the periodicity of timetables, we *roll up* the graph modulo one day, that is, we label nodes with times modulo 24 hours and use bit masks to indicate each trip’s traffic days.

*Query graph search.* After we have determined the earliest arrival time at the target station, we execute a backward search to find the optimal connection that *departs latest*, see footnote [1] on page 293. Furthermore, we apply the  $A^*$  heuristic to goal-direct the searches, using minimal costs between station pairs (computed along with the direct-connection data structure) as lower bounds.

## 7 Heuristic Optimizations

The system described so far gives exact results, that is, for each query we get an optimal connection for every optimal cost. However, despite the use of hubs (Section 5), the precomputation is not significantly faster than the quadratic precomputation described in Section 4. The reason is that, although the results of the local searches (the local transfer patterns) are reasonably small, almost every local search has a local path of very large cost and hence has to visit a large portion of the whole network before it can stop. Indeed, this *15 hours to the nearby village problem* is at the core of what makes routing in public transportation networks so hard [2].

The good news is that with our transfer patterns approach we don’t have this problem at query time but only in the precomputation. Note here that our approach is unique in that it precomputes information for *all* queries, not just for queries where source and target are sufficiently “far apart”. The bad news is that, despite intensive thought, we did not find a solution that is both fast and exact. We eventually resorted to the following simple but approximate solution: limit the local searches to at most two transfers, that is, using at most three vehicles. We call this the *3-legs heuristic*, and as we will see in Section 8 it indeed makes the local searches reasonably fast. Theoretically, we may now miss some optimal transfer patterns, but we found this to play no role in the

practical use of our algorithm. For example, on our CH graph (Section 8), on 10 000 random queries the 3-leg heuristic gave only three non-optimal results, and all three of these were only a few percent off the optimum. We remark that errors in the input data are a much bigger issue in practice. More details on the quality of our approximation are given in the extended version [1].

Having accepted a small fraction of non-optimal results, we also developed and apply various other heuristics, which may lead to a non-optimal solution at query time, but whose measured effect in practice is again tolerable. The following heuristics in combination speed up our query times by a factor of 3–5.

1. In local searches, mark labels as inactive that travel through hubs without transfer beyond a distance threshold. (Requires fixup in query graph building.)
2. Do only one global search per hub, starting at transfer *and* arrival nodes.
3. Optimize duration relaxed by penalty [10], thus discarding Pareto-optimal trips whose improvement in penalty is small in relation to the longer duration.
4. Drop rare transfer patterns if optima on other patterns are almost as good.

In precomputation time, these additional heuristics (esp. reducing the number of labels with 3.) save roughly another factor of 2. Unlike the 3-leg heuristic, they are not essential for the feasibility of our approach.

## 8 Experiments

The experimental results we provide in this section are for a fully-fledged C++ implementation, with all the refinements from Section 6 and all the tricks from Section 7 included. For precomputation, our experiments were run on a compute cluster of Opteron and Xeon-based 64-bit servers. Queries were answered by a single machine of the cluster, with all data in main memory.

*Graphs.* We ran our experiments on three different graphs: the train + local transport network of most of Switzerland (CH), the complete transport network of the larger New York area (NY), and the train + local transport network of much of North America (NA). Table 1 summarizes the different sizes and types.

**Table 1.** The three public transportation graphs from our experiments

name	#stations	#nodes	#arcs	space	type
CH	20.6 K	3.5 M	11.9 M	64 MB	trains + local, well-structured
NY	29.4 K	16.7 M	79.8 M	301 MB	mostly local, poor structure
NA	338.1 K	113.2 M	449.1 M	2 038 MB	trains + local, poor structure

*Direct-connection queries.* Table 2 shows that the preprocessing time for the direct-connection data structure is negligible compared to the transfer patterns precomputation time. The space requirement is from 3 MB per 1000 stations for CH to 10 MB per 1000 stations for NY and NA. A query takes from 2  $\mu$ s for CH to around 10  $\mu$ s for NY and NA. Note that the larger direct-connection query time for NY and NA is a yardstick for their poor structure (not for their size).

**Table 2.** Direct-connection data structure: construction time and size. The query time range is from getting the fastest to all Pareto-optimal connections.

name	precomp. time	output size	query time
CH	< 1 min	68 MB	2 $\mu$ s
NY	4 min	335 MB	5–9 $\mu$ s
NA	49 min	3 399 MB	9–14 $\mu$ s

*Transfer patterns precomputation.* Our precomputation time (Table 3) is 20–40 (CPU core) hours per 1 million nodes and the resulting (parts of) transfer patterns can be stored in 10–50 MB per 1000 stations. Again, these ratios depend mostly on the structure of the network (best for CH, worst for NY and NA), and not on its size.

**Table 3.** Transfer patterns precomputation times and results

name	precomp. time		output size		#TP/station pair	
	local	global	local	global	local	global
CH (w/o hubs)	–	635 h	–	18 562 MB	–	11.0
CH (w/ hubs)	562 h	24 h	229 MB	590 MB	2.6	25.8
CH (heuristic)	57 h	4 h	60 MB	154 MB	2.0	6.8
NY (heuristic)	724 h	64 h	787 MB	786 MB	3.7	16.4
NA (heuristic)	2 632 h	571 h	6 849 MB	7 151 MB	3.4	10.5

*Query graph construction and evaluation.* Table 4 shows that, on average, query graph construction and evaluation take 5  $\mu$ s and 15  $\mu$ s per arc, respectively. The typical number of arcs in a query graph for a station-to-station query (1:1) is below 1000 and the typical query time is below 10 ms. Location-to-location queries with 50 source and 50 target stations (50:50) take about 50 ms.

**Table 4.** Average query graph construction time, size, and evaluation time. The third column also provides the median, 90%-ile and 99%-ile. Column ‘D’ is the domination, either Pareto or Single-criterion (sum of duration and penalty).

name		constr.	#arcs (50/mean/90/99)				D	eval.	#arc ev.	
CH	w/o hubs	1:1	< 1 ms	32	34	56	86	P	< 1 ms	89
CH	w/ hubs	1:1	1 ms	189	264	569	1286	P	3 ms	540
CH	heuristic	1:1	< 1 ms	80	102	184	560	P	< 1 ms	194
NY	heuristic	1:1	2 ms	433	741	1 917	3 597	P	6 ms	721
NY	heuristic	1:1	2 ms	433	741	1 917	3 597	S	3 ms	248
NY	heuristic	50:50	32 ms	3 214	6 060	15 878	35 382	S	18 ms	1 413
NA	heuristic	1:1	2 ms	261	536	1 277	3 934	P	10 ms	705
NA	heuristic	1:1	2 ms	261	536	1 277	3 934	S	5 ms	321
NA	heuristic	50:50	22 ms	2 005	3 484	7 240	25 775	S	21 ms	1 596

## 9 Conclusions

We believe that the transfer patterns idea has great potential, and we have shown some of its potential in this paper. Obvious directions for future research are: (1) get exact local searches with a feasible precomputation time; (2) make the precomputation faster; (3) reduce the size of the query graphs; (4) speed up the (already very fast) direct-connection queries. Transfer patterns are, by their very nature, also well-suited for so-called *profile queries* (compute all paths from  $A$  to  $B$  over a large time window). We want to explore this potential further.

**Acknowledgements.** The authors acknowledge inspiring conversations with Alex Hall as well as helpful comments by Alex Gontmakher and an anonymous reviewer of the conference submission.

## References

1. Extended version of this paper, <http://ad.informatik.uni-freiburg.de/papers>
2. Bast, H.: Car or public transport – two worlds. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms. LNCS, vol. 5760, pp. 355–367. Springer, Heidelberg (2009)
3. Berger, A., Delling, D., Gebhardt, A., Müller-Hannemann, M.: Accelerating time-dependent multi-criteria timetable information is harder than expected. In: AT-MOS 2009. Dagstuhl Seminar Proceedings (2009)
4. Delling, D.: Time-dependent SHARC-routing. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 332–343. Springer, Heidelberg (2008)
5. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering route planning algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
6. Disser, Y., Müller-Hannemann, M., Schnee, M.: Multi-criteria shortest paths in time-dependent train networks. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 347–361. Springer, Heidelberg (2008)
7. Geisberger, R.: Contraction of timetable networks with realistic transfers. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 71–82. Springer, Heidelberg (2010)
8. Loui, R.P.: Optimal paths in graphs with stochastic or multidimensional weights. Commun. ACM 26, 670–676 (1983)
9. Möhring, R.H.: Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen. In: Horster, P. (ed.) Angewandte Mathematik – insbesondere Informatik, pp. 192–220. Vieweg (1999)
10. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 67–90. Springer, Heidelberg (2007)
11. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.D.: Efficient models for timetable information in public transportation systems. J. Exp. Algorithmics 12 (2007)
12. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 66–79. Springer, Heidelberg (2007)
13. Theune, D.: Robuste und effiziente Methoden zur Lösung von Wegproblemen. Teubner (1995)

# Finding the Diameter in Real-World Graphs

## Experimentally Turning a Lower Bound into an Upper Bound

Pierluigi Crescenzi<sup>1</sup>, Roberto Grossi<sup>2</sup>, Claudio Imbrenda<sup>2</sup>,  
Leonardo Lanzi<sup>1</sup>, and Andrea Marino<sup>1</sup>

<sup>1</sup> Dipartimento di Sistemi e Informatica, Università di Firenze

<sup>2</sup> Dipartimento di Informatica, Università di Pisa

**Abstract.** The diameter of an unweighted graph is the maximum pairwise distance among its connected vertices. It is one of the main measures in real-world graphs and complex networks. The *double sweep* is a simple method to find a lower bound for the diameter. It chooses a random vertex and performs two breadth-first searches (BFSes), returning the maximum length among the shortest paths thus found. We propose an algorithm called *fringe*, which uses few BFSes to find a *matching* upper bound for almost all the graphs in our dataset of 44 real-world graphs. In the few graphs it cannot, we perform an exhaustive search of the diameter using a cluster of machines for a total of 40 cores. In all cases, the diameter is surprisingly *equal* to the lower bound found after very few executions of the double sweep method. The lesson learned is that the latter can be used to find the diameter of real-world graphs in many more cases than expected, and our fringe algorithm can quickly validate this finding for most of them.

## 1 Introduction

The diameter  $D$  of an unweighted undirected graph  $G = (V, E)$  is the maximum pairwise distance  $D = \max_{u,v \in V} d(u, v)$  between connected vertices, where  $d(u, v)$  denotes the number of edges found along the shortest path from  $u$  to  $v$  (or vice versa). We assume that  $G$  is connected, otherwise we consider its largest connected component to define its diameter. A large body of experimental study on real-world graphs and complex (e.g. social) networks (see, for example, [4]) uses the diameter of the underlying graph as one of the relevant measures to analyze, along with the degree distribution, the size evolution, the local density, the clustering coefficient, and so on.

The textbook algorithm to compute the diameter requires  $n$  breadth-first searches (BFSes), each taking  $O(n + m)$  time, where  $n = |V|$  is the number of vertices and  $m = |E|$  is the number of edges in  $G$ . This method is too expensive since there might be many graphs to process, with most of them having large size. This situation is well known and several approaches have been proposed (see, for example, [16] and the introduction of [12]). These algorithms require  $\Omega(n^2/\text{polylog}(n))$  time (and, sometimes,  $\Omega(n^2/\text{polylog}(n))$  space).

A useful practical alternative is to execute a suitable small number of BFSes to empirically establish some lower and upper bounds, denoted as  $L$  and  $U$  respectively, such that  $L \leq D \leq U$  holds. This is not merely an arbitrary heuristics, since we obtain the actual value of  $D$  for the input graph  $G$  when  $L = U$ . For this reason, we call it a *self-checking heuristics*: although there is no guarantee of success for every feasible input, a self-checking heuristics allows the user to certificate that the output is the desired one, namely, the diameter in our case. The drawback that there is no guarantee of success is compensated by the following advantages:

- (a) The self-checking heuristics requires few BFSes in practice, and thus its complexity is linear [12].
- (b) An empirical upper bound is possible, whereas probabilistic or average-case practical approaches can provide just a lower bound [7] since the diameter is a *worst-case* measure.
- (c) Large graphs can be analyzed, since BFS has a good external-memory implementation [14] and works on graphs stored in compressed format [3].

What if the self-checking heuristics terminates returning two values  $L$  and  $U$  such that  $L < U$  holds? The requirement is that this case should be quite rare, so that one can resort to one of the costly algorithms that are always guaranteed to find the diameter.

The *double sweep* is a simple method described in [12,6] that is very effective in finding a lower bound for the diameter: choose a random vertex  $r$ , run a BFS at  $r$ , and find a vertex  $a$  at maximum distance from  $r$ ; then, run a BFS at  $a$  and find a vertex  $b$  at maximum distance from  $a$ ; return the length of the shortest path from  $a$  to  $b$ .

Let  $\text{ecc}(u) = \max_{v \neq u} d(u, v)$  be *eccentricity* of  $u \in V$ ; equivalently,  $\text{ecc}(u)$  is the height of the BFS-tree rooted at  $u$ . Hence, the double sweep method computes  $\text{ecc}(a)$ . It is iterated over a small number of experiments, so that it returns  $L$  as the maximum  $\text{ecc}(a)$  over the vertices  $a$  that are identified by each iteration. A good heuristics is to mark the already identified  $a$ s, so that they are always chosen to be distinct one from the other.

Some authors noted that the above method empirically finds the diameter for some classes of graphs (see, for instance, [8]). In this paper, we also show that the value found by double sweep is always better than the lower bound provided with the graphs available in [10]. Note that there are cases in which double sweep can badly fail as shown in Section 4. However, having more than one iteration seems to help in practice.

To our knowledge, the best upper bound for a self-checking heuristics is described in [12]. It can be very close to  $L$ , but it rarely matches: in this paper, we will see that in the case of 44 networks, it matches only 12 times. We will discuss more about these methods in Section 4.

*Our results.* In this paper, we introduce a simple but powerful algorithm, called *fringe*, that experimentally provides a *matching* upper bound  $U$  (i.e.  $U = L$ ) in many cases. The idea behind our self-checking heuristics is that of using the double sweep as a starting point that identifies the three vertices  $r$ ,  $a$  and  $b$ . Not

only we use its value  $L$  as a lower bound, but we then consider the vertex  $u$  that is halfway along the shortest path connecting  $a$  and  $b$  to find an upper bound  $U$ . We use  $u$  as if it were the “center” of the graph  $G$  (whatever “center” means). Clearly, there are situations in which  $u$  is far away from the real center of  $G$ , and we provide a refinement of the above idea to deal with these situations. We refer the reader to the notion of fringe introduced in Section 2, where the description of our algorithm is given.

What is interesting is that using a dataset of 44 real-world graphs described in Section 3, our experiments in Section 4 show that the fringe algorithm provably finds a matching upper bound in all graphs except few of them (just 5 in number). For the latter graphs, the upper bounds are tighter or no worse than the best known in the literature. We measure the performance of the methods by counting how many BFSes are executed, which is a realistic indicator of the running time.

Even more interesting, we ran long-term experiments with five 24GB-RAM machines for a total of 40 cores, to compute (using the textbook algorithm) the diameter for the few graphs for which the fringe algorithm did not find a matching upper bound. To our surprise, the diameter of these graphs is  $D = L$  in all cases, where  $L$  is the lower bound found by the double sweep [126].

The lesson learned in our large-scale experiments is the following. Run the double sweep method to have an educated guess of what is the diameter of a large graph. Use our fringe algorithm to validate this fact. In the few cases in which  $L < U$ , run any  $\Omega(n^2/\text{polylog}(n))$ -time algorithm with guaranteed result.

Our study indicates two further lines of research. The former is related to the weighted graphs, and seeks for simple practical methods to find (nearly) tight lower and upper bounds for the *weighted* diameter, in which the distance  $d(\cdot, \cdot)$  takes into account the edge weights. The latter poses the question of finding a theoretical  $o(n^2/\text{polylog}(n))$ -time algorithm that can check if a given value  $D$  is the diameter of the input graph  $G$ . Note that we only require the algorithm to produce a binary answer. This is enough, since we can employ it for  $D = L, L + 1, \dots, U$  since  $L$  and  $U$  are usually quite close each other.

## 2 The Fringe Algorithm

Here we describe our *fringe* method to improve the upper bound  $U$  and possibly match the lower bound  $L$  obtained by the *double sweep* method of [126]. The full code that combines the latter two methods, improving either  $L$  or  $U$  (or both) whenever possible, is reported in [diameter.algoritmica.org](http://diameter.algoritmica.org).

We consider an unweighted undirected graph  $G = (V, E)$ . We also assume that  $G$  is connected (if not,  $G$  denotes its largest connected component). For any vertex  $u \in V$ , let  $T_u$  denote an unordered BFS-tree obtained by starting the BFS traversal of  $G$  from  $u$ . We recall that the eccentricity  $\text{ecc}(u)$  is the height of  $T_u$ , and that  $2 \cdot \text{ecc}(u) \geq \text{diam}(G)$ .

We also denote the diameter of a subgraph  $G' \subseteq G$  by  $\text{diam}(G')$ , where  $G' = (V', E')$ ,  $V' \subseteq V$ , and  $E' \subseteq E$ . If  $V' = V$  and  $G'$  is connected,  $G'$  satisfies the property that  $\text{diam}(G') \geq \text{diam}(G)$  since the edges in  $E \setminus E'$  are not taken into account. We will use this fact for  $G' \equiv T_u$ , namely,  $\text{diam}(T_u) \geq \text{diam}(G)$ .

We define the *fringe* of  $u$ , denoted  $F(u)$ , as the set of vertices  $v \in V$  such that  $d(u, v) = \text{ecc}(u)$ . We observe that these vertices correspond to the leaves of  $T_u$  having maximum depth. This notion is central to our algorithm since we are interested in vertices  $u$  of real-word graphs for which  $|F(u)|$  is not so large. Let  $B(u) = \max_{z \in F(u)} \text{ecc}(z)$ . We now define the upper bound  $U(u)$  computed by the fringe algorithm starting from a certain node  $u$ .

$$U(u) = \begin{cases} 2 \cdot \text{ecc}(u) - 1 & \text{if } |F(u)| > 1 \text{ and } B(u) = 2 \cdot \text{ecc}(u) - 1 \\ 2 \cdot \text{ecc}(u) - 2 & \text{if } |F(u)| > 1 \text{ and } B(u) < 2 \cdot \text{ecc}(u) - 1 \\ \text{diam}(T_u) & \text{otherwise} \end{cases} \quad (1)$$

**Lemma 1.**  $U(u) \geq D$ , where  $D$  is the diameter of  $G$ .

*Proof.* Since  $D \equiv \text{diam}(G)$ , it is always upper bounded by  $\text{diam}(T_u)$ . The *default* case is therefore  $U(u) = \text{diam}(T_u)$ . We now consider some cases explicitly, and see if they can improve over the default case. If the fringe  $F(u)$  contains just one vertex, that vertex belongs to  $\text{diam}(T_u)$ , and the default case holds. Hence, we assume that  $F(u)$  contains more than one vertex, and consider  $B(u)$ .

When  $B(u) = 2 \cdot \text{ecc}(u)$ , we actually have  $D = B(u)$ : There must exist two vertices in  $F(u)$  that (a) are among the leaves of  $T_u$ , and (b) have the root of  $T_u$  as their lowest common ancestor, and (c) are hit by the double sweep at  $u$ , which finds that  $L = 2 \cdot \text{ecc}(u)$ . Since  $B(u) = \text{diam}(T_u)$  here, the default case covers this situation.

When  $B(u) = 2 \cdot \text{ecc}(u) - 1$ , we also find  $D = B(u)$ : In this case, we have  $d(x, y) \leq 2 \cdot \text{ecc}(u) - 1$  for any two distinct vertices  $x, y$  such that  $x \in F(u)$  or  $y \in F(u)$  (possibly both). Suppose by contradiction that  $D \geq 2 \cdot \text{ecc}(u)$ . This implies that there must exist two distinct vertices  $x', y' \in V \setminus F(u)$  such that  $d(x', y') \geq 2 \cdot \text{ecc}(u)$ . But since they are both not in the fringe of  $u$ , their connecting path inside  $T_u$  is of length at most  $2 \cdot \text{ecc}(u) - 2$ , thus contradicting the fact that their distance  $d(x', y')$  is longer. Also, the double sweep at  $u$  finds that  $L = 2 \cdot \text{ecc}(u) - 1$ , so it cannot be  $D < 2 \cdot \text{ecc}(u) - 1$ . Hence, we set  $U(u) = B(u)$ .

When  $B(u) < 2 \cdot \text{ecc}(u) - 1$ , we can set  $U(u) = 2 \cdot \text{ecc}(u) - 2$  since the diameter can be equal or smaller. The argument is analogous to the previous case: Suppose by contradiction that  $D \geq 2 \cdot \text{ecc}(u) - 1$ . This implies that there must exist two distinct vertices  $x', y' \in V \setminus F(u)$  such that  $d(x', y') \geq 2 \cdot \text{ecc}(u) - 1$ . But this is a contradiction since  $d(x', y') \leq 2 \cdot \text{ecc}(u) - 2$ .  $\square$

We have the ingredients of the fringe algorithm for finding an upper bound  $U$ :

1. Let  $r, a$ , and  $b$  be the vertices identified by double sweep (using two BFSes).
2. Find the vertex  $u$  that is halfway along the path connecting  $a$  and  $b$  inside the BFS-tree  $T_a$  (with ties broken arbitrarily).
3. Compute the BFS-tree  $T_u$  and its eccentricity  $\text{ecc}(u)$ .
4. If  $|F(u)| > 1$ , find the BFS-tree  $T_z$  for each  $z \in F(u)$ , and compute  $B(u)$ :
  - If  $B(u) = 2 \cdot \text{ecc}(u) - 1$ , return  $2 \cdot \text{ecc}(u) - 1$ .
  - If  $B(u) < 2 \cdot \text{ecc}(u) - 1$ , return  $2 \cdot \text{ecc}(u) - 2$ .
5. Return the diameter  $\text{diam}(T_u)$ .

**Theorem 1.** *The fringe algorithm correctly computes an upper bound for the diameter of the input graph  $G$ , using at most  $|F(u)| + 3$  breadth-first searches.*

The proof of Theorem 1 immediately follows from Lemma 1 and by inspection of the fringe algorithm. We can use a slack parameter  $F_{\max}$  in practice: if  $|F(u)| > F_{\max}$ , we skip all the  $|F(u)|$  BFSes and simply return  $\text{diam}(T_u)$ . Since we iterate the algorithm over several choices of  $r, a, b, u$ , this guarantees that we never exceed  $F_{\max} + 3$  BFSes per iteration. Additionally, we avoid calculating  $U(u)$  when  $2 \cdot \text{ecc}(u) - 2$  is not an improvement over the best upper bound found in previous iterations. We refer the reader to the full code in our website.

### 3 The Datasets

We chose our set of real-word graphs, so as to cover the largest taxonomy as possibile. The list of graphs, their features, and their source are reported in Table 1 (Note that some graphs have been made undirected, even though they were originally directed.) Here, we simply list them: a detailed description of each graph can be found in our website ([diameter.algoritmica.org](http://diameter.algoritmica.org)).

*Social networks.* In on-line social networks, nodes represent people and edges represent interactions between people: Epinions social network (**soc-Epinions1**), LiveJournal social network (**soc-LiveJournal1**), Slashdot social network, November 2008 (**soc-Slashdot0811**), Wikipedia vote network (**wiki-Vote**), Slashdot social network, February 2009 (**soc-Slashdot0902**).

*Communication networks.* In communication networks, nodes represent people and edges represent communication among them: Enron email network (**email-Enron**), EU email communication network (**email-EuAll**), Wikipedia Talk network (**wiki-Talk**).

*Citation networks.* In this kind of network, nodes represent papers and edges represent citations: Patent citation network (**cit-Patents**), CiteSeer (**CiteSeer**), Hep-th Citation Graph(**hep-th-citations-MAX**).

*Collaboration networks.* In the collaboration networks, nodes represent people and edges represent collaborations: Astro Physics collaboration network (**ca-AstroPh**), Condense Matter collaboration network (**ca-CondMat**), General Relativity and Quantum Cosmology collaboration network (**ca-GrQc**), High Energy Physics - Phenomenology collaboration network (**ca-HepPh**), High Energy Physics - Theory collaboration network (**ca-HepTh**), Actor collaboration network (**IMDB**), DBLP co-author network (**dblp20080824-MAX**), free software development collaboration network (**advogato**).

*Web graphs.* In Web graphs, nodes represent webpages and edges are hyperlinks: (**web**) (**uk-2005**) [5], and [2] (**arabic-2005**), (**cnr-2000**), (**eu-2005**), (**in-2004**), (**indochina-2004**), (**it-2004**), (**sk-2005**).

*Product co-purchasing networks.* In these cases, nodes represent products and edges link commonly co-purchased products: Amazon product co-purchasing network, March 02 2003 (**amazon0302**), Amazon product co-purchasing network,

**Table 1.** The data set (the fifth and the sixth column refer to the largest connected component)

Network	Acronym	Nodes	Edges	Nodes l.c.c.	Edges l.c.c.	Source
advogato	ADVO	7418	96074	5272	91806	[13]
amazon0302	AMA1	262111	1799582	262111	1799582	[10]
amazon0312	AMA2	400727	4699736	400727	4699736	[10]
Amazon0505	AMA3	410236	4878872	410236	4878872	[10]
arabic-2005	ARAB	22743881	1107806146	22634275	1104463734	[2]
as-skitter	ASSK	1696415	22190596	1694616	22188418	[10]
ca-AstroPh	CAAS	18771	396100	17903	393944	[10]
ca-CondMat	CACO	23133	186878	21363	182572	[10]
ca-GrQc	CAGR	5241	28968	4158	26844	[10]
ca-HepPh	CAH1	12006	236978	11204	235238	[10]
ca-HepTh	CAH2	9875	51946	8638	49612	[10]
cit-HepPh	CIT1	34546	841840	34401	841654	[10]
cit-HepTh	CIT2	27770	704646	27400	704116	[10]
cit-Patents	CITP	3774768	33037894	3764117	33023480	[10]
citeseer	CITE	259217	1064080	220997	1010654	[11]
cnr-2000	CNR2	325557	5477938	325557	5477938	[2]
dblp20080824-MAX	DBLP	511163	3742140	511163	3742140	[15]
dip20090126-MAX	DIP2	19928	82404	19928	82404	[15]
email-Enron	EMA1	36691	367660	33695	361620	[10]
email-EuAll	EMA2	265214	731138	224832	681588	[10]
eu-2005	EU20	862664	32276936	862664	32276936	[2]
HC-BIOGRID	HCBI	4039	20642	4039	20642	[15]
hep-th-citations-MAX	HEPT	27400	704042	27400	704042	[15]
imdb	IMDB	908830	75177226	880455	74989272	[9]
in-2004	IN20	1353703	26252344	1353703	26252344	[2]
indochina-2004	INDO	7414758	301969638	7320539	298109708	[2]
it-2004	IT20	41290577	2054949790	41290577	2054949790	[2]
itdk0304-rlinks-undirected	ITDK	192244	1218132	190914	1215220	[15]
p2p-Gnutella31	P2PG	62586	295782	62561	295754	[10]
p2p	P2P	5380578	284076802	5380491	284076702	[5]
roadNet-CA	ROA1	1965206	5533214	1957027	5520776	[10]
roadNet-PA	ROA2	1088092	3083796	1087562	3083028	[10]
roadNet-TX	ROA3	1379917	3843320	1351137	3758402	[10]
sk-2005	SK20	50634118	3620101486	50634118	3620101486	[2]
soc-Epinions1	SOCE	75879	811478	75877	811476	[10]
soc-LiveJournal1	SOCL	4847571	86739236	4843953	86725498	[10]
soc-sign-epinions	SOC1	131827	1423564	119130	1409144	[10]
soc-sign-Slashdot090221	SOC2	82140	1000960	82140	1000960	[10]
soc-Slashdot0811	SOC3	77360	1092972	77360	1092972	[10]
soc-Slashdot0902	SOC4	82168	1165064	82168	1165064	[10]
trust	TRUS	49288	762434	49288	762434	[13]
web	WEB	39454463	1566054250	39252879	1562879784	[5]
wiki-Talk	WIK1	2394385	9319128	2388953	9313362	[10]
wiki-Vote	WIK2	7115	201522	7066	201470	[10]

March 12 2003 ([amazon0312](#)), Amazon product co-purchasing net, May 05 2003 ([amazon0505](#)).

*Internet peer-to-peer networks.* In peer-to-peer networks, nodes represent computers and edges represent communication among them: Gnutella peer-to-peer network, August 31 2002 ([p2p-Gnutella31](#)), ([p2p](#)).

*Road networks.* In this case, intersections and endpoints are represented by nodes and the roads connecting these intersections or road endpoints are represented by undirected edges: California road network ([roadNet-CA](#)), network of Pennsylvania ([roadNet-PA](#)).

*Autonomous systems graphs.* These are graphs of the Internet: Autonomous systems by Skitter ([as-Skitter](#)), Router topology ([itdk0304-rlinks-undirected](#)).

*Signed networks.* These are networks with positive and negative edges such as friend/foe, trust/distrust, and so on: Epinions social network ([soc-sign-epinions](#)), Slashdot social network, February 2009 ([soc-sign-Slashdot090221](#)).

*Biological networks.* These graphs refer to databases of physical and genetic and biological interactions: ([HC-BIOGRID](#)), Database of Interacting Proteins ([dip20090126-MAX](#)).

## 4 Experiments

In this section we describe both the experimental setting and the obtained results. All the code, the data set and the logs of the experiments are available at [diameter.algoritmica.org](#). The data set is distributed according to a unique format, which is the same adopted by [\[12\]](#): to this aim, we have implemented several conversion utilities that are also available at the previously specified URL.

*Setting.* We used 5 machines with 8 cores each (Intel Xeon CPU X5570 at 2.93GHz), where each core has 8 MB shared cache with a 24 GB shared memory. The operating system is CentOS 5.3 with a Linux kernel version 2.6.18 and gcc version 4.1.2. This huge computational power was mainly aimed at allowing us to run the textbook algorithms for computing the diameter when our fringe algorithm does not provide a tight upper bound.

*Upper bound computation methods.* We compared an implementation of our fringe algorithm, called **fub**, with three methods for computing upper bounds on the diameter. The first two methods, called **rtub** and **htub** respectively, are taken from [\[12\]](#). In particular, **rtub** selects a random node  $r$  and returns the diameter of  $T_r$ , while **htub** chooses  $r$  as one of the nodes with the highest degree. The third method, called **mtub**, is a simplification of **fub**, in which we simply returns the diameter of  $T_u$ .

*Experiment description.* For each of the 44 networks described in Section 3 and for each of the above four upper bound computation methods, we have performed 10 experiments. Each experiment consisted of 10 executions of the algorithm at hand: for each execution,  $F_{\max}$  is set to 50000 (in other words, no practical bound is set to the number of BFSes to be performed).

*Results.* The results of our experiments are summarized in Table 2, where each row corresponds to one network. The first column indicates the acronym of the network, while the second column shows the corresponding diameter  $D$ . The third and the fourth columns indicate the lower bound  $L$  computed by the double sweep method and the number  $X_L$  of its executions returning  $L$  as output. Moreover, we report the value  $S_L$  which has been computed as follows. For any  $i$  with  $1 \leq i \leq X_L$ , let  $s_i$  be the minimum number of breadth-first searches required by the  $i$ -th experiment returning  $L$  as output for the first time:  $S_L$  is equal to  $\sum_{i=1}^{X_L} s_i / X_L$ .<sup>1</sup> In other words,  $S_L$  indicates how quickly the double sweep algorithm converges to its best lower bound.

For each of the four upper bound algorithms, we report in the table the best upper bound  $U$  and the number  $X_U$  of executions returning  $U$  as output. Moreover, we report the value  $S_U$  which has been computed analogously to  $S_L$ . Namely, for any  $i$  with  $1 \leq i \leq X_U$ , let  $s_i$  be the minimum number of breadth-first searches required by the  $i$ -th experiment returning  $U$  as output for the first time:  $S_U$  is equal to  $\sum_{i=1}^{X_U} s_i / X_U$ . In other words,  $S_U$  indicates how quickly the algorithm converges to its best upper bound. Finally, in the case of the **fub** method, we also report  $B$ , which is the maximum size  $|F(u)|$  of the fringe: by Theorem 1, the value  $B + 3$  indicates the maximum number of breadth-first searches that are required by a single execution of the algorithm.

In the table, for each row we highlight the best upper bound computed by any of the four methods for the corresponding network.

*Discussion.* The first impressive observation is that the two columns  $D$  and  $L$  are identical, that is, the value returned by the double sweep method is actually an upper bound too. By looking at columns  $X_L$  and  $S_L$ , moreover, we can observe that this bound is frequently and quickly achieved. Indeed, almost for every network the tight bound is computed at every experiment (that is, 10 times) and most of the times (that is, 31 times) two breadth-first searches are sufficient to compute this bound. In any case, at most 15 breadth-first searches are required (in the case of a road network).

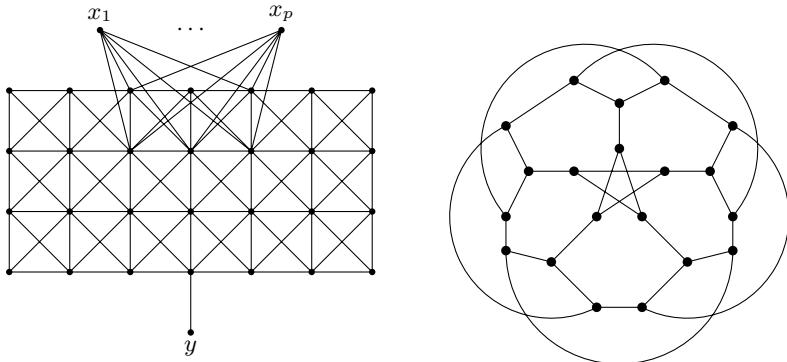
The second important observation is that, in almost all networks, the tightness of  $L$  can be experimentally proved in a very efficient way by using our fringe algorithm. Indeed, by looking at columns  $U$ ,  $X_U$ , and  $S_U$  corresponding to **fub**, we can observe that (1) only 7 values of  $U$  are not highlighted (that is, **fub** almost always computes the best upper bound), (2) the empirical probability of **fub** returning the best upper bound is high (on the average 0.93), and (3) the number  $S_L$  of breadth-first searches required by **fub** in order to prove the tightness of

---

<sup>1</sup> Notice that since, for each experiment, we perform 10 executions of the double sweep algorithm, then  $2 \leq S_L \leq 20$ .

**Table 2.** Summary of experimental results (in the case of the starred upper bound values, by performing a slightly greater number of executions, we have been able to compute either a tight upper bound (in the case of CAH2 and of DBLP) or an almost tight upper bound (12 in the case of P2PG)))

	D	dslb			fub			mtub			hdtub			rtub			
		L	X <sub>L</sub>	S <sub>L</sub>	U	X <sub>U</sub>	S <sub>U</sub>	B	U	X <sub>U</sub>	S <sub>U</sub>	U	X <sub>U</sub>	S <sub>U</sub>	U	X <sub>U</sub>	S <sub>U</sub>
ADVO	9	9	10	2	<b>9</b>	10	6	9	<b>9</b>	10	6	<b>9</b>	10	4	<b>9</b>	2	24
AMA1	38	38	10	2	<b>38</b>	10	3	7	<b>38</b>	10	4	<b>38</b>	10	12	<b>38</b>	2	8
AMA2	20	20	10	2	<b>20</b>	10	18	9	22	10	12	21	10	4	21	2	30
AMA3	22	22	10	2	<b>22</b>	10	3	11	<b>22</b>	10	4	<b>22</b>	10	4	<b>22</b>	3	21
ARAB	47	47	10	2	<b>47</b>	10	9	6	48	10	4	51	10	8	50	2	14
ASSK	31	31	10	12	<b>31</b>	10	5	3	32	10	4	34	10	28	34	5	33
CAAS	14	14	10	2	<b>14</b>	10	11	4	15	10	9	15	10	12	16	2	28
CACO	15	15	10	2	<b>15</b>	7	31	9	16	8	30	17	10	4	17	2	26
CAGR	17	17	10	2	<b>17</b>	10	23	15	18	10	18	19	10	4	18	2	10
CAH1	13	13	10	2	<b>13</b>	10	50	63	14	10	16	15	10	4	15	3	25
CAH2	18	18	10	2	20*	10	4	2	20	10	4	20	10	16	20	2	30
CIT1	14	14	10	5	<b>14</b>	10	13	3	15	10	8	15	10	20	16	9	14
CIT2	15	15	10	2	<b>15</b>	10	20	8	16	10	17	16	10	20	16	1	24
CITP	26	26	10	2	28	10	12	4	30	10	4	29	10	8	31	1	40
CITE	52	52	10	2	<b>52</b>	10	3	11	<b>52</b>	10	4	54	10	4	<b>52</b>	2	16
CNR2	34	34	10	2	<b>34</b>	9	11	3	<b>34</b>	9	19	39	10	4	35	10	10
DBLP	22	22	10	2	24*	2	27	25	24	1	21	24	10	8	25	4	13
DIP2	30	30	10	8	<b>30</b>	10	9	3	<b>30</b>	10	27	33	10	12	31	1	16
EMA1	13	13	10	2	<b>13</b>	10	12	9	<b>13</b>	10	18	<b>13</b>	10	24	<b>13</b>	1	32
EMA2	14	14	10	2	<b>14</b>	10	3	48	<b>14</b>	10	4	15	10	4	15	7	9
EU20	21	21	10	2	<b>21</b>	10	14	15	22	10	4	25	10	4	23	3	6
HCBI	23	23	10	4	<b>23</b>	7	26	3	<b>23</b>	6	28	<b>23</b>	10	32	<b>23</b>	1	28
HEPT	15	15	10	2	<b>15</b>	10	19	4	16	10	11	16	10	20	17	2	28
IMDB	14	14	10	2	<b>14</b>	10	17	18	15	8	25	16	10	4	16	3	29
IN20	43	43	10	2	<b>43</b>	10	5	11	44	10	4	44	10	8	<b>43</b>	1	8
INDO	43	43	10	2	<b>43</b>	8	39	21	44	10	20	44	10	8	45	4	14
IT20	45	45	10	2	<b>45</b>	10	30	15	46	10	6	47	10	4	46	3	29
ITDK	26	26	10	2	<b>26</b>	10	7	5	28	10	4	29	10	4	28	4	17
P2PG	11	11	10	3	14*	10	14	132	15	9	20	14	10	12	15	6	19
P2P	9	9	10	4	<b>9</b>	8	1125	1266	10	7	22	10	10	4	11	6	28
ROA1	865	865	9	15	987	9	8	2	987	9	12	1047	10	16	988	1	32
ROA2	794	794	8	12	803	8	22	5	803	6	13	873	10	32	832	1	36
ROA3	1064	1064	10	8	1079	3	18	9	1079	1	16	1166	10	24	1128	1	24
SK20	40	40	10	5	<b>40</b>	10	18	18	41	10	21	41	10	4	41	7	12
SOCE	15	15	10	4	<b>15</b>	10	19	11	16	10	9	16	10	4	16	8	17
SOCL	20	20	10	2	<b>20</b>	10	7	12	<b>20</b>	10	8	<b>20</b>	10	28	22	7	19
SOC1	16	16	10	2	<b>16</b>	10	3	5	<b>16</b>	10	4	<b>16</b>	10	8	17	8	11
SOC2	13	13	10	2	<b>13</b>	10	7	17	<b>13</b>	10	4	<b>13</b>	10	20	14	7	18
SOC3	12	12	10	6	<b>12</b>	10	13	13	13	10	6	<b>12</b>	10	20	13	2	32
SOC4	13	13	10	3	<b>13</b>	10	7	17	14	10	4	<b>13</b>	10	20	14	6	18
TRUS	14	14	10	2	<b>14</b>	10	3	2	<b>14</b>	10	4	15	10	4	15	8	19
WEB	32	32	10	2	<b>32</b>	10	52	63	34	10	4	38	10	4	34	6	20
WIK1	11	11	10	2	<b>11</b>	10	9	10	12	10	4	12	10	8	12	10	12
WIK2	7	7	10	2	<b>7</b>	10	64	134	8	10	8	8	10	4	8	4	22



**Fig. 1.** A bad network ( $k = 4$ ) for the double sweep algorithm (left) and for the fringe algorithm (right)

$L$  is incredibly low with respect to the number of vertices in the network (apart from the p2p network in which case the number of searches is 1125). It is also worth observing that in the case of 22 networks, our algorithm is the only one able to compute the tight upper bound.

Note that, as one could expect, the maximum size of the fringe is most of the times greater than the average number of performed breadth-first searches: however, it is worth noting that this value is quite small with respect to the number of vertices in the network.

As we already said, our fringe algorithm fails to prove the tightness of the lower bound only in 7 cases. In the case of the **ca-HepTh**, however, if we execute the algorithm 20 times (instead of 10), the correct upper bound is computed and the tightness of the lower bound can be proved. The same holds in the case of the **dblp20080824-MAX** network. In the case of the **p2p-Gnutella31** network, by running the algorithm 200 times, we have been able to lower the upper bound down to 12: however, this bound is not tight. In the case of the **cit-Patents** network and of the three road networks, finally, we have not been able to obtain better results, even by increasing the number of algorithm executions.

One might object that in some cases the number of breadth-first searches performed by our fringe algorithm is higher than the number of searches performed by other algorithms. For instance, in the case of the **ca-HepPh** network, the value  $S_U$  corresponding to **fub** is 50, while the value corresponding to **rtub** is 25. We then executed this latter algorithm with a much greater number of breadth-first searches (that is, 400). In this case, the bound returned by **rtub** improved to 14, but it did not reach the lower bound 13 (which is tight).

We have also tried to modify our algorithm by introducing a different selection of the vertex  $u$ , that is, by choosing as  $u$  the highest degree vertex. It turns out, however, that this modification almost always returns bounds which are higher than the bounds obtained by **fub**.

Finally, going back to the impressive performance of the double sweep algorithm, we would like to point out that it is not difficult to design a network

for which the behavior of this algorithm is not very good (see the left part of Figure 1). To this aim consider a unitary grid with  $k$  rows and  $1+3k/2$  columns. Each vertex of the grid is connected to all vertices whose Euclidean distance is at most  $\sqrt{2}$ . Moreover, there are  $p$  additional vertices  $x_1, x_2, \dots, x_p$  which are connected to the neighbors of the middle point of the upper row of the grid and one additional vertex  $y$  which is connected to the middle point of the lower row of the grid. If  $p$  is sufficiently large with respect to  $k$ , then the `dslb` algorithm will choose one of these points as the vertex  $r$ . As a consequence, the vertex  $y$  will be chosen as the vertex  $a$  of the algorithm. The breadth-first search starting from  $y$  will have height equal to  $k+1$ : this will be the value reported by the algorithm. It is easy to see that the diameter of the network is instead  $3k/2$ .

Note that, in the case of the graph shown in the left part of Figure 1 with  $p = 100000$ , the fringe algorithm, for every experiment, computes an upper bound which is equal to the diameter of the graph (that, is 6) by executing (as expected) 7 BFSes on the average. However, it is easy to find combinations of the values of  $p$  and  $k$  for which the computed upper bound is not equal to the diameter: for instance, if  $k = 8$  and  $p = 100000$ , the best answer we obtained was 15. In the case of the fringe algorithm (and of the other upper bound methods), moreover, we can define a family of graphs for which the probability of computing the correct value is 0: these graphs have the property that no BFS tree has the diameter equal to the diameter of the graph (an example of such graphs is shown in the right part of Figure 1). Note that in the case of these latter graphs the lower bound computed by the double sweep method is tight with probability 1, since the height of each BFS tree is equal to the diameter of the graph.

## 5 Conclusion

In this paper, we have proposed an algorithm which uses few BFSes to find an upper bound which matches, for almost all the graphs in our dataset, the lower bound computed by the double sweep method to find a lower bound for the diameter. By using this algorithm and, in few cases, the textbook algorithm we have been able to prove that, in all networks, the diameter is equal to the lower bound found by the double sweep algorithm. The lesson learned is that this latter method can be used to find the diameter of real-world graphs in many more cases than expected, and our fringe algorithm can quickly validate this finding for most of them.

It is an interesting open question to understand why, in the case of some real-world graphs, the fringe method fails to validate the lower bound computed by the double sweep algorithm: to this aim, it might be useful to investigate the similarities between these real-world graphs and the synthetic graphs introduced at the end of the previous section. It would also be interesting to explore the behavior of both the double sweep and the fringe algorithms when applied to synthetic networks produced by well-known models, such as the Erdős-Rényi or the preferential attachment models.

Finally, as already specified in the introduction, our study indicates two further lines of research: the former is related to the weighted graphs, while the

latter is related to the question of finding a theoretical  $o(n^2/\text{polylog}(n))$ -time algorithm that can check if a given value  $D$  is the diameter of the input graph  $G$ .

**Acknowledgments.** The first author would like to thank Michel Habib for introducing him to the double sweep algorithm. We all wish to thank Maurizio Davini for his effort to provide us with powerful machines to run the experiments. We are grateful to Andrea Clementi, Francesco Pasquale, and Riccardo Silvestri for suggesting us the counter-example for the double sweep algorithm. Finally, we thank the anonymous referees for their comments.

## References

1. Alegre, I., Fiol, M., Yebra, J.: Some large graphs with given degree and diameter. *J. Graph Theory* 10, 219–224 (1986)
2. Boldi, P., Vigna, S.: Webgraph (2001), <http://webgraph.dsi.unimi.it/>
3. Boldi, P., Vigna, S.: The WebGraph framework I: Compression techniques. In: Proc. of the 13th International World Wide Web Conference, pp. 595–601 (2004)
4. Brandes, U., Erlebach, T.: Network Analysis: Methodological Foundations. Springer, Heidelberg (2005)
5. Complexnetworks Team: Complex networks and real-world graphs (2008), <http://complexnetworks.fr/>
6. Corneil, D.G., Dragan, F.E., Habib, M., Paul, C.: Diameter determination on restricted graph families. *Discrete Appl. Math.* 113(2-3), 143–166 (2001)
7. Faloutsos, C.: Graph mining: Patterns, generators and tools. In: Combinatorial Pattern Matching, p. 274 (2009)
8. Handler, G.: Minimax location of a facility in an undirected tree graph. *Transportation Science* 7(287–293) (1973)
9. IMDB: The internet movie database (1990), <http://www.imdb.com/>
10. Leskovec, J.: Stanford Network Analysis Package (SNAP) Website (2009), <http://snap.stanford.edu>
11. Library, S.L.D.: Citeseer Website (1997), <http://citeseer.ist.psu.edu/citeseer.html>
12. Magnien, C., Latapy, M., Habib, M.: Fast computation of empirically tight bounds for the diameter of massive graphs. *J. Exp. Algorithmics* 13 (2009)
13. Massa, P., Souren, K.: Trustlet Website (2007), <http://www.trustlet.org>
14. Mehlhorn, K., Meyer, U.: External-memory breadth-first search with sublinear i/o. In: Proceedings of the 10th Annual European Symposium on Algorithms, pp. 723–735 (2002)
15. Sommer, C.: Christian sommer's homepage (2009), <http://www.sommer.jp/graphs/>
16. Zwick, U.: Exact and approximate distances in graphs - a survey. In: Meyer auf der Heide, F. (ed.) *ESA 2001. LNCS*, vol. 2161, pp. 33–48. Springer, Heidelberg (2001)

# Budgeted Red-Blue Median and Its Generalizations\*

MohammadTaghi Hajiaghayi<sup>1</sup>, Rohit Khandekar<sup>2</sup>, and Guy Kortsarz<sup>3, \*\*</sup>

<sup>1</sup> AT&T Labs—Research & University of Maryland  
[hajiagh@research.att.com](mailto:hajiagh@research.att.com)

<sup>2</sup> IBM T.J. Watson research center  
[rohitk@us.ibm.com](mailto:rohitk@us.ibm.com)

<sup>3</sup> Rutgers University—Camden  
[guyk@camden.rutgers.edu](mailto:guyk@camden.rutgers.edu)

**Abstract.** In a Content Distribution Network application, we have a set of servers and a set of clients to be connected to the servers. Often there are a few server types and a hard budget constraint on the number of deployed servers of each type. The simplest goal here is to deploy a set of servers subject to these budget constraints in order to minimize the sum of client connection costs. These connection costs often satisfy metricity, since they are typically proportional to the distance between a client and a server within a single autonomous system. A special case of the problem where there is only one server type is the well-studied *k-median* problem.

In this paper, we consider the problem with two server types and call it the *budgeted red-blue median* problem. We show, somewhat surprisingly, that running a single-swap local search for each server type *simultaneously*, yields a constant factor approximation for this case. Its analysis is however quite non-trivial compared to that of the *k-median* problem (Arya et al., 2004; Gupta and Tangwongsan, 2008).

Later we show that the same algorithm yields a constant approximation for the *prize-collecting* version of the budgeted red-blue median problem where each client can potentially be served with an alternative cost via a different vendor. In the process, we also improve the approximation factor for the *prize-collecting k-median* problem from 4 (Charikar et al., 2001) to  $3 + \epsilon$ , which matches the current best approximation factor for the *k-median* problem.

## 1 Introduction

Consider the following problem called the *budgeted red-blue median problem*. The input is a set of facilities  $\mathcal{F}$  and a set of clients  $\mathcal{C}$  in a metric space. The distance between two points in this metric space  $i, j \in \mathcal{F} \cup \mathcal{C}$  is denoted by  $d(i, j)$ . The

\* Part of this work was done while the authors were meeting at DIMACS. We would like to thank DIMACS for hospitality.

\*\* Research partially supported by NSF grant 0819959.

facilities are partitioned into two sets: *red* facilities  $\mathcal{R}$  and *blue* facilities  $\mathcal{B}$ . The input also includes two integers  $k_r, k_b > 0$ . Given a subset of *open* facilities, a client  $j$  gets served by the nearest open facility. The goal of the problem is to open a subset of red facilities  $R \subseteq \mathcal{R}$  and a subset of blue facilities  $B \subseteq \mathcal{B}$  such that

- $|R| \leq k_r$  and  $|B| \leq k_b$ ,
- the total connection cost  $\text{cost}(R, B) := \sum_{j \in \mathcal{C}} d(j, R \cup B)$  is minimized.

Here  $d(j, S) = \min_{i \in S} d(j, i)$  denotes the shortest distance from  $j$  to any point in  $S$ . A special case in which all facilities have the same color is the well-studied  *$k$ -median problem*. In content distribution network applications and several other applications in telecommunication networks and clustering, it is vital to obtain solutions for these problems without violating any budget (see e.g., [53]).

Another related and well-motivated problem is the *weighted  $W$ -median* problem in which given a non-negative opening cost  $w_i$  for each facility  $i$ , we want to open a set of facilities whose opening cost is within our budget  $W$  and minimize the total connection cost. This budget constraint is a Knapsack constraint and thus for general opening costs, we do not hope to get any approximation algorithm if we insist not to violate our budget  $W$ . Indeed as in Knapsack, if we are allowed to violate the budget within a factor  $1 + \epsilon$ , we can obtain a constant factor approximation algorithm using the filtering method of Lin and Vitter [21]. In addition, for polynomially bounded opening costs (for which Knapsack is solvable), we can solve the problem on trees without violating budget  $W$ . Using probabilistic embeddings of general metrics into tree metrics [4, 11], this immediately results in an  $O(\log n)$  approximation algorithm for weighted  $W$ -median in general metrics without violating budget  $W$  when opening costs are polynomially bounded. When there are only two different facility opening costs, one can guess the number of facilities of each type in the optimum solution. Thus this special case can be reduced to the budgeted red-blue median problem.

Last but not least, in several of the above applications, each client can be satisfied with an alternative cost often via a different vendor. Indeed, this cost is called the *penalty* of this client that we pay in case it is not connected to one of our deployed servers. More formally, in all problem formulations above we can assume that each client  $j \in \mathcal{C}$  has a *penalty*  $p_j \in \mathbf{Q}_+$ . The client pays the service cost, i.e., its distance to the nearest open facility, if it is at most its penalty  $p_j$ ; otherwise the client remains unserved and pays the penalty  $p_j$ . The goal then is to minimize the sum of connection costs and paid penalties.

## 1.1 Related Results

Aforementioned the most special case of our problems in this paper is the well-known  *$k$ -median problem*. The first constant factor approximation for the  *$k$ -median problem* was given by Charikar et al. [7], which was subsequently improved by Jain-Vazirani [17], Charikar-Guha [6], and Arya et al. [3]. The latter presents the current best approximation factor of  $3 + \epsilon$  for  *$k$ -median* via a local search heuristic. Their analysis was recently simplified by Gupta and

Tangwongsan [14]. The problem cannot be approximated within a factor strictly less than  $1 + 2/e$ , unless  $\mathbf{NP} \subseteq \mathbf{DTIME}[n^{O(\log \log n)}]$  [16]. It is known that the integrality gap of the natural LP relaxation of the problem is at most 3, but currently there is no algorithm that achieves a 3-approximation in polynomial time [1]. An extension of  $k$ -median to the case in which we can open at most  $k$  facilities, but also have to pay their facility opening cost was studied by [10], who gave a 5-approximation. The  $k$ -median problem with penalties was also considered; the current best approximation factor for prize-collecting  $k$ -median is 4 due to Charikar et al. [8]. The problem in which the underlying metric is Euclidean, although NP-hard [23], admits a PTAS due to the results of Arora, Raghavan, and Rao [2], and then Koliopoulos and Rao [18] (who provided an almost-linear time algorithm).

The Lagrangian relaxation approach was used by Jain and Vazirani [17] for the  $k$ -median problem. When we apply this approach to the budgeted red-blue median problem, we can get two solutions whose convex combination has cost at most a constant factor times the optimum cost. These two solutions have  $k_r^1$  (resp.  $k_r^2$ ) red and  $k_b^1$  (resp.  $k_b^2$ ) blue facilities where  $k_r^1 + k_b^1 = k_r^2 + k_b^2 = k_r + k_b$ . It may happen, for example, that  $k_r^1 > k_r$  and  $k_b^2 > k_b$ , i.e., the bound on red facilities is violated in the first solution and the bound on blue facilities is violated in the second solution. Unlike the case for the  $k$ -median, both of these solutions may be infeasible. Therefore, it seems very hard to combine them to get a solution that has no violation while having cost within a constant factor of the optimum cost. The observation that the Lagrangian relaxation approach fails for the budgeted red-blue median problem was also shared and verified by Jain [15].

*Local search based approaches.* From a practical point of view, a simple combinatorial algorithm is much more desirable than the one that requires to solve a linear programming relaxation. To this end, our main approach in this paper is to extend the local search technique which is a popular heuristic for hard combinatorial optimization problems. A relatively few instances of approximation guarantees via local search are known. Korupolu, Plaxton, and Rajaraman [19] gave the first approximation guarantees of this type for the facility location and  $k$ -median problems based on a simple local search heuristic proposed by Kuehn and Hamburger [20]. For the  $k$ -median problem, however, they violate the constraint on the number of open facilities by a factor  $1 + \epsilon$ . Later Arya et al. [3] could approximate the problem without violating this constraint. The local search later has been used for other facility location type problems [22][25][9][24] and recently even for maximum generalized assignment [13] and maximizing submodular functions [12].

## 1.2 Our Results

The main result of this paper is a constant factor approximation algorithm for the budgeted red-blue median problem via novel analysis of a natural local search algorithm. More formally, we analyze the following local search algorithm.

1. Let  $R \subset \mathcal{R}$  and  $B \subset \mathcal{B}$  be arbitrary subsets with  $|R| = k_r$  and  $|B| = k_b$ .
2. While there exist  $r \in R$ ,  $r' \in \mathcal{R}$  and  $b \in B$ ,  $b' \in \mathcal{B}$  such that  $\text{cost}(R - r + r', B - b + b') < \text{cost}(R, B)$  do:  $R \leftarrow R - r + r'$  and  $B \leftarrow B - b + b'$ .
3. Output  $R$  and  $B$ .

Here  $S - s_1 + s_2$  denotes  $(S \setminus \{s_1\}) \cup \{s_2\}$ . Since  $r$  and  $r'$  (or  $b$  and  $b'$ ) may be identical, our algorithm outputs a locally optimum solution w.r.t. three local operations: (1) delete a red facility and add a red facility, (2) delete a blue facility and add blue facility, and (3) delete a red and a blue facilities and add a red and a blue facilities. In Section 2, we prove the following theorem.

**Theorem 1.** *The above local search algorithm yields a constant approximation to the budgeted red-blue median problem.*

In fact, it is somewhat surprising that this natural local search algorithm even works. We point out why in the next section by explaining the main challenges in the analysis. We omit the standard details regarding how to make this algorithm run in polynomial time. In Section 3, we show how the local search analysis can be extended to the prize-collecting version. More specifically, we improve the current best approximation factor 4 of the LP-rounding algorithm for prize-collecting  $k$ -median due to Charikar et al. [8] as follows.

**Theorem 2.** *The multi-swap local search algorithm of Arya et al. [3] yields  $(3 + \epsilon)$ -approximation for the prize-collecting  $k$ -median problem.*

Last but not least, we show how we can combine the techniques for the budgeted red-blue median problem and prize-collecting variants to obtain the most general theorem of this paper.

**Theorem 3.** *There is a local search algorithm that yields a constant approximation for the prize-collecting budgeted red-blue median problem.*

The proof of this theorem is omitted from this extended abstract due to lack of space.

### 1.3 An Overview of Our Techniques

*The budgeted red-blue median problem.* Let us first understand why the standard local search analysis of  $k$ -median [3][14] does not extend easily to the budgeted red-blue median problem. In the  $k$ -median analysis, we consider several test swaps for the locally optimum solution  $S$ . Each of these swaps includes deleting a facility from  $S$  and adding a facility from the optimum solution  $O$  and rerouting the clients. These swaps are chosen carefully to bound the cost of  $S$ . In case of the budgeted red-blue median problem, however, this choice may conflict with the budget constraints on the number of red and blue facilities allowed. For example, after deleting, say, a red facility, to keep the cost bounded, one may need to add a blue facility to serve the clients previously served by the dropped red facility.

This happens, for example, when there is no other red facility close-by. In such a case, we are forced to delete another blue facility and possibly add another red facility in order to balance the number of red and blue facilities. As a result, bounding the cost of the solution after the swap becomes much trickier.

Our analysis begins by partitioning the solutions  $S$  and  $O$  into *blocks* (see Section 2.1) with some useful structural properties. Intuitively speaking, a block is a subset of  $S \cup O$  for which the test swaps can be analyzed “independently” of other blocks, even when a test swap involves rerouting clients served by facilities from multiple blocks. These blocks are defined based on the distances and the colors of the facilities. For example, let  $s_i \in S$  be the closest facility in  $S$  for exactly one facility  $o_i \in O$  for  $i = 1, 2$ . If  $s_i$  has the same color as  $o_i$ , then  $\{s_i, o_i\}$  defines a block. On the other hand if  $\{s_1, s_2, o_1, o_2\}$  has two red and two blue facilities, this set defines a block. In general, a block contains an equal number of red facilities and an equal number of blue facilities from the two solutions  $S$  and  $O$  such that for any facility  $o \in O$  in a block, the closest facility to it in  $S$  is also in the same block. A typical block also satisfies a key property: it contains a large number of facilities in  $S$  that are not the closest facilities in  $S$  to any facility in  $O$ . It turns out that such facilities, called *very good facilities*, are compatible to be swapped with any facility in  $O$  [14] and their abundance is crucial to the overall analysis. We use a careful counting argument to show that a partitioning into blocks satisfying these properties exists.

In Section 2.2, we describe the test swaps for any single block. If  $s_i$  and  $o_i$  described above have the same color, we can consider the swap: add  $o_i$  and delete  $s_i$ . However, if  $s = s_i$  is the closest one to several facilities  $\{o_1, \dots, o_l\}$  in the optimum solution, then deleting  $s$  may be bad for our solution. The previous  $k$ -median analyses, therefore, avoided swaps in which  $s$  is deleted.

Unfortunately, it turns out that we do not have a luxury of avoiding such swaps. Consider, for example, the case where  $k_r = 1$  and  $s$  is the only red facility in  $S$ . Suppose that  $o$  is the unique red facility in  $O$ . To bound the cost of clients served by  $o$  in solution  $O$ , we need to consider a test swap in which  $o$  is added. Note however that if  $o$  is added,  $s$  must be deleted to satisfy the budget  $k_r = 1$ . Our analysis considers a test swap in which we delete  $s$  and open the facility  $o_i \in \{o_1, \dots, o_l\}$  that is closest to  $s$ . If  $s$  and  $o_i$  are of different colors, we combine this swap with another carefully chosen red-blue swap to balance the number of red and blue facilities. The cost after such a swap may potentially be significantly higher than that of the optimum solution. To “cancel” this high cost, we consider several other test swaps in which facilities  $\{o_1, \dots, o_l\}$  are added one-by-one. Using the properties of a block mentioned above, we show how to bound the overall cost for all the swaps considered.

In our opinion, these new swaps and a method to bound their costs is the main technical contribution of our paper. We encourage the reader to read the exposition in paragraphs titled ‘Intuition’ and ‘Example in Figure 2’ in Section 2.3 for further intuition behind our approach.

*The prize-collecting version.* We show that the multi-swap local search algorithm of the  $k$ -median problem [3] yields  $(3 + \epsilon)$ -approximation for the prize-collecting

$k$ -median problem. The proof is based on the techniques of Arya et al. [3] or Gupta and Tangwongsan [14] applied to the clients that do not pay penalty in either solution  $S$  or  $O$ . The other clients contribute the same amount to either solutions and thus are easy to handle. Essentially the same line of argument holds for the prize-collecting version of budgeted red-blue problem.

## 2 Proof of Theorem 1

We begin with some notation and preliminaries. We call the local search operations in our algorithm as *valid swaps*. Let  $O = R^* \cup B^*$  denote the optimum solution where  $R^* \subset \mathcal{R}$  and  $B^* \subset \mathcal{B}$  and let  $S = R \cup B$  denote the locally optimum (also called local) solution. For a facility  $o \in O$ , let  $N^*(o)$  denote the clients that are served by  $o$  in solution  $O$ , i.e., these clients have  $o$  as the closest facility among facilities in  $O$ . Similarly, for  $s \in S$ , let  $N(s)$  denote the clients that are served by  $s$  in solution  $S$ . For  $A \subset O$ , let  $N^*(A) = \cup_{o \in A} N^*(o)$  and for  $A \subset S$ , let  $N(A) = \cup_{s \in A} N(s)$ . For a client  $j \in \mathcal{C}$ , let  $O_j = d(j, O)$  and  $S_j = d(j, S)$  be its contribution to the optimum and local solutions respectively.

**Definition 1 (functions  $\eta$  and  $\mu$ ).** Define a function  $\eta : O \rightarrow S$  as follows. For  $o \in O$ , let  $\eta(o)$  be the facility in  $S$  that it closest to  $o$ , where ties are broken arbitrarily. Thus we have  $d(o, \eta(o)) = d(o, S)$ .

For a facility  $s \in S$  with  $\eta^{-1}(s) \neq \emptyset$ , define  $\mu(s)$  to be the facility in  $\eta^{-1}(s)$  that it closest to  $s$  where ties are broken arbitrarily, i.e., we have  $d(s, \mu(s)) = d(s, \eta^{-1}(s))$ .

See Figure 2 for an example. Note that if  $o \in O \cap S$ , then we have  $\eta(o) = o$ . The definition of function  $\eta$  is motivated by the paper of Gupta and Tangwongsan [14] who offer a simplified proof of the  $k$ -median local search algorithm of Arya et al. [3].

**Definition 2 (very good, good, and bad facilities).** We call a facility  $s \in S$  very good, if  $\eta^{-1}(s) = \emptyset$ ; good, if  $\eta^{-1}(s) \neq \emptyset$  and no facility in  $\eta^{-1}(s)$  has the same color as  $s$ ; and bad, if some facility in  $\eta^{-1}(s)$  has the same color as  $s$ .

### 2.1 The Blocks

We now present a procedure (see Figure 1) to partition the set  $R^*$  into  $R_1^*, \dots, R_t^*$ , the set  $B^*$  into  $B_1^*, \dots, B_t^*$ , the set  $R$  into  $R_1, \dots, R_t$ , and the set  $B$  into  $B_1, \dots, B_t$  for some integer  $t$ . The parts  $R_i^*, B_i^*, R_i, B_i$  are said to form **block- $i$**  for  $i = 1, \dots, t$ . Note that this procedure is used only for the sake of analysis. It shows how to first compute **block-1** and then recursively compute **block- $i$**  for  $i = 2, \dots, t$ .

**Lemma 1.** The partitions of  $R^*$ ,  $B^*$ ,  $R$ , and  $B$  computed in Figure 1 satisfy the following properties.

Compute **block-1**, i.e.,  $R_1^*$ ,  $B_1^*$ ,  $R_1$ , and  $B_1$ :

1. Start with  $R_1^* = B_1^* = R_1 = B_1 = \emptyset$ .
2. If there is a bad facility  $r \in R$  such that  $|\eta^{-1}(r)| = 1$ , then let  $R_1 = \{r\}$ ,  $B_1 = \emptyset$ ,  $R_1^* = \eta^{-1}(r)$ ,  $B_1^* = \emptyset$ , and stop. If there is a bad facility  $b \in B$  such that  $|\eta^{-1}(b)| = 1$ , then let  $R_1 = \emptyset$ ,  $B_1 = \{b\}$ ,  $R_1^* = \emptyset$ ,  $B_1^* = \eta^{-1}(b)$ , and stop.
3. If there are good facilities  $r \in R$  and  $b \in B$  such that  $|\eta^{-1}(r)| = |\eta^{-1}(b)| = 1$ , then let  $R_1 = \{r\}$ ,  $B_1 = \{b\}$ ,  $R_1^* = \eta^{-1}(b)$ ,  $B_1^* = \eta^{-1}(r)$ , and stop.
4. If there is no bad facility in  $S$ , let  $R_1^* = R^*$ ,  $B_1^* = B^*$ ,  $R_1 = R$ ,  $B_1 = B$ , and stop. Otherwise let  $s \in S$  be a bad facility such that  $|\eta^{-1}(s)|$  is maximum.
5. Add  $s$  to either  $R_1$  or  $B_1$  according to whether it is a red or a blue facility. Add facilities in  $\eta^{-1}(s)$  to  $R_1^*$  and  $B_1^*$  according to their color. If  $|R_1^*| = |R_1|$  and  $|B_1^*| = |B_1|$ , then stop.
6. If  $|R_1^*| > |R_1|$ , then add  $|R_1^*| - |R_1|$  very good or good red facilities to  $R_1$ . While doing so, give a preference to very good red facilities. For each facility  $s$  thus added to  $R_1$ , add facilities in  $\eta^{-1}(s)$  to  $B_1^*$ .
7. If on the other hand  $|B_1^*| > |B_1|$ , then add  $|B_1^*| - |B_1|$  very good or good blue facilities to  $B_1$ . While doing so, give a preference to very good blue facilities. For each facility  $s$  thus added to  $B_1$ , add facilities in  $\eta^{-1}(s)$  to  $R_1^*$ .
8. Repeat steps 6 and 7 until we have  $|R_1^*| = |R_1|$  and  $|B_1^*| = |B_1|$ , and then stop.

Recurse on  $R^* \setminus R_1^*$ ,  $B^* \setminus B_1^*$ ,  $R \setminus R_1$ , and  $B \setminus B_1$  to compute **block- $i$**  for  $i \geq 2$ .

**Fig. 1.** A procedure to compute partitions of  $R^*$ ,  $B^*$ ,  $R$ , and  $B$

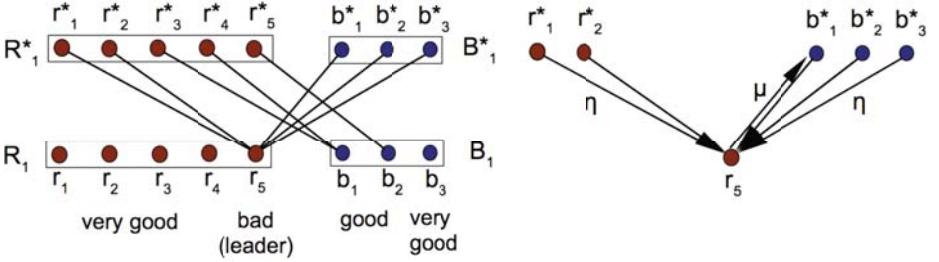
1.  $|R_i^*| = |R_i|$  and  $|B_i^*| = |B_i|$  for all  $i = 1, \dots, t$ .
2. For each  $o \in R_i^* \cup B_i^*$ , we have  $\eta(o) \in R_i \cup B_i$  for  $i = 1, \dots, t$ .
3. For  $i = 1, \dots, t$ , at most one facility in  $R_i \cup B_i$  is bad. We call such a facility leader.
4. For  $i = 1, \dots, t$ , if there is a leader in  $R_i \cup B_i$ , we have
  - (a) either all facilities in  $R_i$ , except the leader, are very good,
  - (b) or all facilities in  $B_i$ , except the leader, are very good.

The proof of this lemma is omitted due to lack of space.

## 2.2 The Swaps

Since  $S = R \cup B$  is a local solution, any swap of a red facility and a blue facility does not decrease the cost of the solution, i.e.,  $\text{cost}(R - r^- + r^+, B - b^- + b^+) \geq \text{cost}(R, B)$ . We use  $\text{swap}(r^-, r^+ | b^-, b^+)$  to denote this swap. When  $r^- = r^+$ , we also use  $\text{swap}(b^-, b^+)$  to denote this swap. Similarly, when  $b^- = b^+$ , we also use  $\text{swap}(r^-, r^+)$  to denote this swap. We now consider several inequalities of this type and add them to get the desired result. For each such swap considered below, we upper bound  $\text{cost}(R - r^- + r^+, B - b^- + b^+) - \text{cost}(R, B)$  by giving a feasible assignment of clients to facilities.

Recall the definition of valid swaps; we call a swap valid if it does not change the number of red and blue facilities in the solution.



**Fig. 2.** On the left is an example of **block-1**: the facilities  $R_1^*, B_1^* \subset O$  are shown at the top while  $R_1, B_1 \subset S$  are shown at the bottom. We draw an edge between each  $o \in R_1^* \cup B_1^*$  and  $\eta(o) \in R_1 \cup B_1$ . A single bad facility in  $R_1 \cup B_1$ , called leader, is  $r_5$ . The facilities  $r_1, \dots, r_4, b_3$  are very good while the facilities  $b_1, b_2$  are good. Here case 4(a) holds. On the right is an example of functions  $\eta$  and  $\mu$ . We have  $\mu(r_5) = b_1^* \in \eta^{-1}(r_5)$ .

**Lemma 2.** 1. Let  $s \in R_1$  (resp.  $s \in B_1$ ) be a very good facility and  $o \in R_1^*$  (resp.  $o \in B_1^*$ ) be any facility. Then

$$\sum_{j \in N^*(o)} (O_j - S_j) + \sum_{j \in N(s) \setminus N^*(o)} 2O_j \geq 0. \quad (1)$$

2. Let  $s \in R_1$  (resp.  $s \in B_1$ ) be either good or bad facility such that  $o = \mu(s) \in R_1^*$  (resp.  $o \in B_1^*$ ). Then

$$\sum_{j \in N^*(o)} (O_j - S_j) + \sum_{j \in N(s) \cap N^*(\eta^{-1}(s) \setminus \{o\})} (O_j + S_j) + \sum_{j \in N(s) \setminus N^*(\eta^{-1}(s))} 2O_j \geq 0. \quad (2)$$

3. Let  $s_1 \in R_1 \cup B_1$  be either good or bad facility,  $s_2 \in R_1 \cup B_1$  be a very good facility, and  $o_2 \in R_1^* \cup B_1^*$  be any facility such that deleting  $s_1, s_2$  and adding  $o_1 = \mu(s_1), o_2$  is a valid swap. Then

$$\sum_{j \in N^*(o_1) \cup N^*(o_2)} (O_j - S_j) + \sum_{j \in [N(s_1) \cup N(s_2)] \cap [N^*(\eta^{-1}(s_1) \setminus \{o_1, o_2\})]} (3O_j + S_j) + \sum_{j \in [N(s_1) \cup N(s_2)] \setminus [N^*(\eta^{-1}(s_1) \cup \{o_2\})]} 2O_j \geq 0. \quad (3)$$

4. Let  $s_1, s_2 \in R_1 \cup B_1$  be either good or bad facilities such that deleting  $s_1, s_2$  and adding  $o_1 = \mu(s_1), o_2 = \mu(s_2)$  is a valid swap. Then

$$\sum_{j \in N^*(o_1) \cup N^*(o_2)} (O_j - S_j) + \sum_{j \in [N(s_1) \cup N(s_2)] \cap [N^*(\eta^{-1}(s_1) \setminus \{o_1\}) \cup N^*(\eta^{-1}(s_2) \setminus \{o_2\})]} (3O_j + S_j) + \sum_{j \in [N(s_1) \cup N(s_2)] \setminus [N^*(\eta^{-1}(s_1)) \cup N^*(\eta^{-1}(s_2))]} 2O_j \geq 0. \quad (4)$$

*Proof.* For a client  $j$ , let  $s(j)$  denote the facility that serves  $j$  in solution  $S$  and let  $o(j)$  denote the facility that serves  $j$  in solution  $O$ .

For item 1, consider  $\text{swap}(s, o)$ . We reroute clients as follows. A client  $j \in N^*(o)$  is rerouted to  $o$  and thus the increase in its service cost is  $O_j - S_j$ . A client  $j \in N(s) \setminus N^*(o)$  is rerouted to  $\eta(o(j))$ . Note that  $\eta(o(j)) \neq s$  since  $s$

is very good. The increase in its service cost is thus  $d(j, \eta(o(j))) - S_j \leq O_j + d(o(j), \eta(o(j))) - S_j \leq O_j + d(o(j), s(j)) - S_j \leq O_j + O_j + S_j - S_j = 2O_j$ . This sequence of inequalities follows from repeated use of triangle inequality. The clients not in  $N^*(o) \cup N(s)$  are not rerouted. This proves item 1.

For item 2, consider  $\text{swap}(s, o)$ . A client  $j \in N^*(o)$  is rerouted to  $o$  and thus the increase in its service cost is  $O_j - S_j$ . Consider a client  $j \in N(s) \setminus N^*(\eta^{-1}(s))$ . Since  $o(j) \notin \eta^{-1}(s)$ , we have  $\eta(o(j)) \neq s$ . Such a client is therefore rerouted to  $\eta(o(j))$  and thus the increase in its service cost is  $d(j, \eta(o(j))) - S_j \leq 2O_j$  as shown in item 1. A client  $j \in N(s) \cap N^*(\eta^{-1}(s) \setminus \{o\})$  is rerouted to  $o$  and thus the increase in its service cost is  $d(j, o) - S_j \leq d(j, s(j)) + d(s(j), o) - S_j \leq S_j + d(s(j), o(j)) - S_j \leq O_j + S_j$ . Here  $d(s(j), o) \leq d(s(j), o(j))$  follows from  $o(j) \in \eta^{-1}(s)$ ,  $o = \mu(s)$ , and the definition of  $\mu$ . The clients not in  $N^*(o) \cup N(s)$  are not rerouted. This proves item 2.

The proofs of items 3 and 4 are very similar. Therefore we prove item 4 and omit the proof of item 3. For item 4, consider the swap: delete  $s_1, s_2$  and add  $o_1, o_2$ . In this swap, we reroute the clients as follows. A client  $j \in N^*(o_1)$  is rerouted to  $o_1$  and a client  $j \in N^*(o_2)$  is rerouted to  $o_2$ . Clearly the increase in service cost of clients  $j \in N^*(o_1) \cup N^*(o_2)$  is  $O_j - S_j$ .

Now consider a client  $j \in [N(s_1) \cup N(s_2)] \setminus [N^*(o_1) \cup N^*(o_2)]$ . Assume without loss of generality that  $j \in N(s_1)$ ; a similar argument also holds for the case  $j \in N(s_2)$ . Let  $o(j)$  be the facility that serves  $j$  in  $O$ . If  $\eta(o(j)) = s_1$ , then  $j$  is rerouted to  $o_1$  and the increase in service cost is  $d(j, o_1) - d(j, s_1) \leq d(s_1, o_1) \leq d(s_1, o(j)) \leq S_j + O_j$ . This sequence of inequalities follows from repeated use of triangle inequality and from the fact  $o_1 = \mu(s_1)$ . If  $\eta(o(j)) = s_2$ , then it is rerouted to  $o_2$  and the increase in service cost is  $d(j, o_2) - S_j \leq d(j, o(j)) + d(o(j), s_2) + d(s_2, o_2) - S_j \leq d(j, o(j)) + d(o(j), s_2) + d(s_2, o(j)) - S_j \leq d(j, o(j)) + d(o(j), s_1) + d(s_1, o(j)) - S_j \leq O_j + 2(O_j + S_j) - S_j = 3O_j + S_j$ . This sequence of inequalities follows from repeated use of triangle inequality and from the fact  $o_2 = \mu(s_2)$  and  $\eta(o(j)) = s_2$ . Now consider the case that  $\eta(o(j))$  is neither  $s_1$  or  $s_2$ . Let  $s(j)$  denote the facility that serves  $j$  in  $S$ . We reroute  $j$  to  $\eta(o(j))$  and the increase in service cost is thus  $d(j, \eta(o(j))) - S_j \leq O_j + d(o(j), \eta(o(j))) - S_j \leq O_j + d(o(j), s(j)) - S_j \leq O_j + O_j + S_j - S_j = 2O_j$ . This proves item 4.

### 2.3 Putting Together

**Intuition.** Note that inequality (1) has “ $-S_j$ ” terms for some clients and “ $+O_j$ ” terms for some clients. The analysis of Arya et al. [3] or Gupta and Tangwongsan [14] is based on adding several inequalities of this type so that the “ $-S_j$ ” term is included for each client  $j$  once and “ $+O_j$ ” term is included for each client  $j$  at most 5 times. Thus overall, they get  $-\sum_j S_j + 5 \sum_j O_j \geq 0$ . This directly gives a 5-approximation. Unfortunately, such an analysis does not work in our setting. We also have to add several inequalities (2)-(4), thus incurring “ $+S_j$ ” terms for some clients. We then use inequality (1) repeatedly to “cancel” the “ $+S_j$ ” terms in order to prove a constant approximation. All the swaps

to be considered are contained in a block. For block- $i$ , we prove the following inequality:

$$\sum_{j \in N^*(R_i^* \cup B_i^*)} S_j \leq O(1) \cdot \left[ \sum_{j \in N^*(R_i^* \cup B_i^*)} O_j + \sum_{j \in N(R_i \cup B_i)} O_j \right]. \quad (5)$$

Adding these inequalities over all the blocks, we get a constant approximation:

$$\begin{aligned} \mathbf{cost}(S) &= \sum_{i=1}^t \sum_{j \in N^*(R_i^* \cup B_i^*)} S_j \leq O(1) \cdot \sum_{i=1}^t \left[ \sum_{j \in N^*(R_i^* \cup B_i^*)} O_j + \sum_{j \in N(R_i \cup B_i)} O_j \right] \\ &\leq O(1) \cdot 2 \cdot \mathbf{cost}(O). \end{aligned}$$

The proof of inequality (5) is omitted due to lack of space. However here we illustrate how to prove it using the example in Figure 2.

We start with some notation. If  $R_1$  has at least one good or very good facility, we fix a function  $g : R_1^* \rightarrow R_1$  such that each facility in  $g(R_1^*)$  is either good or very good and  $|g^{-1}(r)| \leq 2$  for all  $r \in R_1$ . It is easy to see that such a function exists. Similarly, if  $B_1$  has at least one good or very good facility, we fix a function  $g : B_1^* \rightarrow B_1$  such that each facility in  $g(B_1^*)$  is either good or very good and  $|g^{-1}(b)| \leq 2$  for all  $b \in B_1$ .

**Example in Figure 2.** To convey our intuition, we prove inequality (5) for the example of block-1 in Figure 2. For concreteness, assume that the function  $\mu$  is given by  $r_5 \mapsto b_1^*, b_1 \mapsto r_4^*, b_2 \mapsto r_5^*$ . Also assume that  $g$  is given by  $r_1^* \mapsto r_1, r_2^* \mapsto r_2, r_3^* \mapsto r_3, r_4^* \mapsto r_4, r_5^* \mapsto r_4, b_1^* \mapsto b_1, b_2^* \mapsto b_2, b_3^* \mapsto b_3$ . To obtain “ $-S_j$ ” terms for clients in  $N^*(B_1^*)$ , we consider the following swaps and the corresponding inequalities:

- $\mathbf{swap}(g(\mu(g(b_1^*))), \mu(g(b_1^*)) \mid g(b_1^*), b_1^*)$  which is same as  $\mathbf{swap}(r_4, r_4^* \mid b_1, b_1^*)$  (consider inequality (3)),
- $\mathbf{swap}(g(\mu(g(b_2^*))), \mu(g(b_2^*)) \mid g(b_2^*), b_2^*)$  which is same as  $\mathbf{swap}(r_4, r_5^* \mid b_2, b_2^*)$  (consider inequality (3)),
- $\mathbf{swap}(g(b_3^*), b_3^*)$  which is same as  $\mathbf{swap}(b_3, b_3^*)$  (consider inequality (1)).

If we add these three inequalities, we get

$$\sum_{j \in N^*(\{r_4^*, b_1^*, r_2^*, b_2^*, b_3^*\})} (O_j - S_j) + \sum_{j \in N^*(r_3^*)} (3O_j + S_j) + \sum_{j \in N(\{b_3, r_4, b_1\})} 2O_j + \sum_{j \in N(\{r_4, b_2\})} 2O_j \geq 0. \quad (6)$$

We next consider the following swaps and the corresponding inequalities:

- $\mathbf{swap}(g(r_1^*), r_1^*)$  which is same as  $\mathbf{swap}(r_1, r_1^*)$  (consider inequality (1)),
- $\mathbf{swap}(g(r_2^*), r_2^*)$  which is same as  $\mathbf{swap}(r_2, r_2^*)$  (consider inequality (1)),
- $\mathbf{swap}(g(r_3^*), r_3^*)$  which is same as  $\mathbf{swap}(r_3, r_3^*)$  (consider inequality (1)). We in fact multiply this inequality by factor 2 in order to cancel the “ $+S_j$ ” term in the second term of (6) above.

Adding these three inequalities, we get

$$\sum_{j \in N^*(\{r_1^*, r_2^*\})} (O_j - S_j) + 2 \sum_{j \in N^*(r_3^*)} (O_j - S_j) + \sum_{j \in N(\{r_1, r_2\})} 2O_j + 2 \sum_{j \in N(r_3)} 2O_j \geq 0. \quad (7)$$

Adding (6) and (7), we get our desired inequality

$$\sum_{j \in N^*(R_1^* \cup B_1^*)} S_j \leq 5 \sum_{j \in N^*(R_1^* \cup B_1^*)} O_j + 4 \sum_{j \in N(R_1 \cup B_1)} O_j.$$

### 3 Proof of Theorem 2

In this section, we outline the proof of Theorem 2. We consider the multi-swap local search algorithm of Arya et al. [3]: start with any  $k$  facilities in the solution  $S$  and output a local optimum solution w.r.t. the following  $q$ -swap operation: delete  $q$  facilities from  $S$  and add  $q$  facilities in  $\mathcal{F} \setminus S$  to  $S$ . We use a notation similar to the previous section. In addition, let  $P \subseteq \mathcal{C}$  denote the set of clients that pay penalty in the locally optimum solution  $S$  and let  $P^* \subseteq \mathcal{C}$  denote the set of clients that pay penalty in the optimum solution  $O$ . We prove the following theorem which implies that  $S$  is a  $(3 + 2/q)$ -approximation.

#### Theorem 4

$$\sum_{j \notin P} S_j + \sum_{j \in P} p_j \leq \left(3 + \frac{2}{q}\right) \sum_{j \notin P^*} O_j + \left(1 + \frac{1}{q}\right) \sum_{j \in P^*} p_j.$$

Note that even if the multiplier of  $\sum_{j \in P^*} p_j$  on the right is  $(1 + 1/q)$  instead of 1, one may use the above result, as a subroutine, in the algorithm for the robust  $k$ -median problem [8]. This is a version of the  $k$ -median problem in which at most  $l$  clients may be left unserved. We obtain a solution which has number of outliers at most  $l(1 + \epsilon)(1 + \gamma)$  and has cost at most  $(3 + \epsilon)(1 + 1/\gamma)$  for any fixed  $\epsilon, \gamma > 0$ . We omit further details from here.

The proof of Theorem 4 is omitted due to lack of space.

### References

1. Archer, A., Rajagopalan, R., Shmoys, D.B.: Lagrangian relaxation for the  $k$ -median problem: New insights and continuity properties. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 31–42. Springer, Heidelberg (2003)
2. Arora, S., Raghavan, P., Rao, S.: Approximation schemes for Euclidean  $k$ -medians and related problems. In: STOC 1998 (1998)
3. 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)
4. Bartal, Y.: On approximating arbitrary metrics by tree metrics. In: STOC 1998 (1998)

5. Bateni, M., Hajiaghayi, M.: Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In: SODA 2009 (2009)
6. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM J. Comput. 34(4), 803–824 (2005)
7. Charikar, M., Guha, S., Tardos, E., Shmoys, D.: A constant-factor approximation algorithm for the  $k$ -median problem. J. Comp. Sys. Sci. 65(1), 129–149 (2002)
8. Charikar, M., Khuller, S., Mount, D.M., Narasimhan, G.: Algorithms for facility location problems with outliers. In: SODA 2001 (2001)
9. Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Math. Program. 102(2), 207–222 (2005)
10. Devanur, N.R., Garg, N., Khandekar, R., Pandit, V., Rohit, K., Vinayaka, P., Saberi, A., Vazirani, V.V.: Price of anarchy, locality gap, and a network service provider game. In: Deng, X., Ye, Y. (eds.) WINE 2005. LNCS, vol. 3828, pp. 1046–1055. Springer, Heidelberg (2005)
11. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. System Sci. 69(3), 485–497 (2004)
12. Feige, U., Mirrokni, V.S., Vondrak, J.: Maximizing non-monotone submodular functions. In: FOCS 2007 (2007)
13. Fleischer, L., Goemans, M.X., Mirrokni, V.S., Sviridenko, M.: Tight approximation algorithms for maximum general assignment problems. In: SODA 2006 (2006)
14. Gupta, A., Tangwongsan, K.: Simpler analyses of local search algorithms for facility location. ArXiv e-prints, arXiv:0809.2554 (2008)
15. Jain, K.: Private communication (2009)
16. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: STOC 2002 (2002)
17. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and  $k$ -median problems using the primal-dual schema and Lagrangian relaxation. Journal of the ACM 48(2), 274–296 (2001)
18. Kolliopoulos, S.G., Rao, S.: A nearly linear-time approximation scheme for the Euclidean  $\$k\$$ -median problem. SIAM J. Comput. 37(3), 757–782 (2007)
19. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. Journal of Algorithms 37(1), 146–188 (2000)
20. Kuehn, A., Hamburger, M.: A heuristic program for locating warehouses. Management Science 9, 643–666 (1963)
21. Lin, J.-H., Vitter, J.S.: Approximation algorithms for geometric median problems. Inf. Process. Lett. 44(5), 245–249 (1992)
22. Mahdian, M., Pál, M.: Universal facility location. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 409–421. Springer, Heidelberg (2003)
23. Megiddo, N., Supowit, K.J.: On the complexity of some common geometric location problems. SIAM J. Comput. 13(1), 182–196 (1984)
24. Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: FOCS 2001 (2001)
25. Zhang, J., Chen, B., Ye, Y.: A multi-exchange local search algorithm for the capacitated facility location problem. In: Bienstock, D., Nemhauser, G.L. (eds.) IPCO 2004. LNCS, vol. 3064, pp. 219–233. Springer, Heidelberg (2004)

# All Ternary Permutation Constraint Satisfaction Problems Parameterized above Average Have Kernels with Quadratic Numbers of Variables\*

Gregory Gutin<sup>1</sup>, Leo van Iersel<sup>2</sup>, Matthias Mnich<sup>3</sup>, and Anders Yeo<sup>1</sup>

<sup>1</sup> Royal Holloway, University of London, United Kingdom  
`{gutin, anders}@cs.rhul.ac.uk`

<sup>2</sup> University of Canterbury, Christchurch, New Zealand  
`l.j.j.v.iersel@gmail.com`

<sup>3</sup> Technische Universiteit Eindhoven, Eindhoven, The Netherlands  
`m.mnich@tue.nl`

**Abstract.** A ternary Permutation-CSP is specified by a subset  $\Pi$  of the symmetric group  $S_3$ . An instance of such a problem consists of a set of variables  $V$  and a multiset of constraints, which are ordered triples of distinct variables of  $V$ . The objective is to find a linear ordering  $\alpha$  of  $V$  that maximizes the number of triples whose rearrangement (under  $\alpha$ ) follows a permutation in  $\Pi$ . We prove that all ternary Permutation-CSPs parameterized above average have kernels with quadratic numbers of variables.

## 1 Introduction

Parameterized complexity theory is a multivariate framework for a refined analysis of hard ( $\text{NP}$ -hard) problems. A *parameterized problem* is a subset  $L \subseteq \Sigma^* \times \mathbb{N}$  over a finite alphabet  $\Sigma$ .  $L$  is *fixed-parameter tractable* if the membership of an instance  $(I, k)$  in  $\Sigma^* \times \mathbb{N}$  can be decided in time  $f(k) \cdot |I|^{O(1)}$  where  $f$  is a computable function of the *parameter*  $k$  only [9,10,26]. (We would like  $f(k)$  to grow as slowly as possible.)

Given a pair  $L, L'$  of parameterized problems, a *bikernelization from  $L$  to  $L'$*  is a polynomial-time algorithm that maps an instance  $(x, k)$  to an instance  $(x', k')$  (the *bikernel*) such that (i)  $(x, k) \in L$  if and only if  $(x', k') \in L'$ , (ii)  $k' \leq h(k)$ , and (iii)  $|x'| \leq g(k)$  for some functions  $h$  and  $g$ . The function  $g(k)$  is called the *size* of the bikernel. A *kernelization* of a parameterized problem  $L$  is simply a bikernelization from  $L$  to itself, i.e., a *kernel* is a bikernel when  $L = L'$ .

The notion of a bikernelization was introduced by Alon et al. [1], who observed that a decidable parameterized problem  $L$  is fixed-parameter tractable if and only if it admits a bikernelization to a decidable parameterized problem  $L'$ . Not every fixed-parameter tractable problem has a kernel of polynomial size unless

\* Part of this research has been supported by the EPSRC, grant EP/E034985/1, the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403, and the Allan Wilson Centre for Molecular Ecology and Evolution.

$\text{coNP} \subseteq \text{NP/poly}$  [23]; low degree polynomial size kernels are of main interest due to applications [15].

For maximization problems whose lower bound on the solution value is a monotonically increasing unbounded function of the instance size, the standard parameterization by solution value is trivially fixed-parameter tractable. Mahajan and Raman [24] were the first to recognize both practical and theoretical importance of parameterizing maximization problems differently: above tight lower bounds. They considered MAX SAT with the tight lower bound  $m/2$ , where  $m$  is the number of clauses, and the problem is to decide whether we can satisfy at least  $m/2 + k$  clauses, where  $k$  is the parameter. Mahajan and Raman proved that this parameterization of MAX SAT is fixed-parameter tractable by obtaining a kernel with  $O(k)$  variables. Despite clear importance of parameterizations above tight lower bounds, until recently only a few sporadic non-trivial results on the topic were obtained [17, 20, 21, 24, 28].

Massive interest in parameterizations above tight lower bound came with the paper of Mahajan et al. [25], who stated several questions on fixed-parameter tractability of maximization problems parameterized above tight lower bounds, some of which are still open. Several of those questions were answered by newly-developed methods [17, 8, 18, 19], using algebraic, probabilistic and harmonic analysis tools. In particular, an advanced probabilistic approach allowed Gutin et al. [18] to prove the existence of a quadratic kernel for the parameterized BETWEENNESS ABOVE AVERAGE (BETWEENNESS-AA) problem, thus, answering an open question of Benny Chor [26].

BETWEENNESS is just one representative of a rich family of *ternary Permutation Constant Satisfaction Problems (CSPs)*. A ternary Permutation-CSP is specified by a subset  $\Pi$  of the symmetric group  $S_3$ . An instance of such a problem consists of a set of variables  $V$  and a multiset of constraints, which are ordered triples of distinct variables of  $V$ . The objective is to find a linear ordering  $\alpha$  of  $V$  that maximizes the number of triples whose rearrangement (under  $\alpha$ ) follows a permutation in  $\Pi$ . Important special cases are BETWEENNESS [5, 12, 18, 27] and CIRCULAR ORDERING [11, 13], which find applications in circuit design and computational biology [6, 27], and in qualitative spatial reasoning [23], respectively.

In this paper, we prove that all ternary Permutation-CSPs have kernels with quadratic numbers of variables, when parameterized above average (AA), which is a tight lower bound. This result is obtained by first reducing all the problems to just one, LINEAR ORDERING-AA, then showing that LINEAR ORDERING-AA has a kernel with quadratic numbers of variables and constraints and, thus, concluding that there is a bikernel with a quadratic number of variables from each of the problems AA to LINEAR ORDERING-AA. Using the last result, we prove that there are bikernels with a quadratic number of variables from all ternary Permutation-CSPs to most ternary Permutation-CSPs. This implies the existence of kernels with a quadratic number of variables for most ternary Permutation-CSPs. The remaining ternary Permutation-CSPs are proved to be equivalent to ACYCLIC SUBDIGRAPH-AA (a *binary Permutation-CSP* defined in Section 2) and since ACYCLIC SUBDIGRAPH-AA, as shown in [19], has a kernel

with a quadratic number of variables, the remaining ternary Permutation-CSPs have a kernel with a quadratic number of variables.

The most difficult part of this set of arguments is the proof that LINEAR ORDERING-AA has a kernel with quadratic numbers of variables and constraints. We can show that if we want to prove this in a similar way as for Betweenness-AA (that is, eliminate all instances of Linear Ordering-AA whose optimal solution coincides with the lower bound) we need an infinite number of reduction rules. See [16] for further details. So, determining fixed-parameter tractability of LINEAR ORDERING-AA turns out to be much harder than for BETWEENNESS-AA. Fortunately, we found a nontrivial way of reducing LINEAR ORDERING-AA to a combination of BETWEENNESS-AA and ACYCLIC SUBDIGRAPH-AA. Using further probabilistic and deterministic arguments for the mixed problem, we prove that LINEAR ORDERING-AA has a kernel with quadratic numbers of variables and constraints.

The rest of the paper is organized as follows. In Section 2, we define and discuss ternary Permutation-CSPs; we also reduce all nontrivial ternary Permutation-CSPs AA to LINEAR ORDERING-AA. In Section 3, we describe probabilistic and harmonic analysis tools used in the paper. In Section 4, we obtain some results on BETWEENNESS-AA and ACYCLIC SUBDIGRAPH-AA needed in the following section, where we prove that LINEAR ORDERING-AA has a quadratic kernel. In Section 5, we also prove our main result, Theorem 4, that all ternary Permutation-CSPs parameterized above average have kernels with a quadratic number of variables. Due to the space limit, many proofs are omitted; they can be found in [16].

## 2 Permutation CSPs Parameterized above Average

Let  $V$  be a set of  $n$  variables. A *linear ordering* of  $V$  is a bijection  $\alpha : V \rightarrow [n]$ , where  $[n] = \{1, 2, \dots, n\}$ . The symmetric group on three elements is  $\mathcal{S}_3 = \{(123), (132), (213), (231), (312), (321)\}$ . A *constraint set over  $V$*  is a multiset  $\mathcal{C}$  of *constraints*, which are permutations of three distinct elements of  $V$ . For each subset  $\Pi \subseteq \mathcal{S}_3$  and a linear ordering  $\alpha$  of  $V$ , a constraint  $(v_1, v_2, v_3) \in \mathcal{C}$  is  $\Pi$ -*satisfied by*  $\alpha$  if there is a permutation  $\pi \in \Pi$  such that  $\alpha(v_{\pi(1)}) < \alpha(v_{\pi(2)}) < \alpha(v_{\pi(3)})$ . If  $\Pi$  is fixed, we will simply say that  $(v_1, v_2, v_3) \in \mathcal{C}$  is *satisfied by*  $\alpha$ .

For each subset  $\Pi \subseteq \mathcal{S}_3$ , the problem  $\Pi$ -CSP is to decide whether for a given pair  $(V, \mathcal{C})$  of variables and constraints there is a linear ordering  $\alpha$  of  $V$  that  $\Pi$ -satisfies all constraints in  $\mathcal{C}$ . A complete dichotomy of the  $\Pi$ -CSP problems with respect to their computational complexity was given by Guttmann and Maucher [22]. For that, they reduced  $2^{|\mathcal{S}_3|} = 64$  problems by two types of symmetry. First, two problems differing just by a consistent renaming of the elements of their permutations are of the same complexity. Second, two problems differing just by reversing their permutations are of the same complexity. The symmetric reductions leave 13 problems  $\Pi_i$ -CSP,  $i = 0, 1, \dots, 12$ , whose time complexity is polynomial for  $\Pi_{11} = \emptyset$  and  $\Pi_{12} = \mathcal{S}_3$  and was otherwise established by Guttmann and Maucher [22], see Table II.

**Table 1.** Ternary Permutation-CSPs (after symmetry considerations)

$\Pi \subseteq \mathcal{S}_3$	Common Problem Name	Complexity to Satisfy All Constraints
$\Pi_0 = \{(123)\}$	LINEAR ORDERING	polynomial
$\Pi_1 = \{(123), (132)\}$		polynomial
$\Pi_2 = \{(123), (213), (231)\}$		polynomial
$\Pi_3 = \{(132), (231), (312), (321)\}$		polynomial
$\Pi_4 = \{(123), (231)\}$		NP-complete
$\Pi_5 = \{(123), (321)\}$	BETWEENNESS	NP-complete
$\Pi_6 = \{(123), (132), (231)\}$		NP-complete
$\Pi_7 = \{(123), (231), (312)\}$	CIRCULAR ORDERING	NP-complete
$\Pi_8 = \mathcal{S}_3 \setminus \{(123), (231)\}$		NP-complete
$\Pi_9 = \mathcal{S}_3 \setminus \{(123), (321)\}$	NON-BETWEENNESS	NP-complete
$\Pi_{10} = \mathcal{S}_3 \setminus \{(123)\}$		NP-complete

The maximization version of  $\Pi_i$ -CSP is the problem MAX- $\Pi_i$ -CSP of finding a linear ordering  $\alpha$  of  $V$  that  $\Pi_i$ -satisfies a maximum number of constraints in  $\mathcal{C}$ . Clearly, for  $i = 4, \dots, 10$  the problem MAX- $\Pi_i$ -CSP is NP-hard. In [16] we prove that MAX- $\Pi_i$ -CSP is NP-hard also for  $i = 0, 1, 2, 3$ .

Now observe that given a variable set  $V$  and a constraint multiset  $\mathcal{C}$  over  $V$ , for a random linear ordering  $\alpha$  of  $V$ , the probability of a constraint in  $\mathcal{C}$  being  $\Pi$ -satisfied by  $\alpha$  equals  $\frac{|\Pi|}{6}$ . Hence, the expected number of satisfied constraints from  $\mathcal{C}$  is  $\frac{|\Pi|}{6}|\mathcal{C}|$ , and thus there is a linear ordering  $\alpha$  of  $V$  satisfying at least  $\frac{|\Pi|}{6}|\mathcal{C}|$  constraints (and this bound is tight). A derandomization argument leads to  $\frac{|\Pi|}{6}$ -approximation algorithms for the problems MAX- $\Pi_i$ -CSP [5]. No better constant factor approximation is possible assuming the Unique Games Conjecture [5].

We study the parameterization of MAX- $\Pi_i$ -CSP above tight lower bound:

#### $\Pi$ -ABOVE AVERAGE ( $\Pi$ -AA)

*Input:* A finite set  $V$  of variables, a multiset  $\mathcal{C}$  of ordered triples of distinct variables from  $V$  and an integer  $k \geq 0$ .

*Parameter:*  $k$ .

*Question:* Is there a linear ordering  $\alpha$  of  $V$  such that at least  $\frac{|\Pi|}{6}|\mathcal{C}| + k$  constraints of  $\mathcal{C}$  are  $\Pi$ -satisfied by  $\alpha$ ?

For example, choose  $\Pi = \{(123), (321)\}$  for BETWEENNESS-AA.  $\Pi_0$ -AA is called the LINEAR ORDERING-AA problem.

Let  $\Pi$  be a subset of  $\mathcal{S}_3$ . Clearly, if  $\Pi$  is the empty set or equal to  $\mathcal{S}_3$  then the corresponding problem  $\Pi$ -AA can be solved in polynomial time. The following simple result allows us to study the  $\Pi$ -AA problems using  $\Pi_0$ -AA.

**Proposition 1.** *Let  $\Pi$  be a subset of  $\mathcal{S}_3$  such that  $\Pi \notin \{\emptyset, \mathcal{S}_3\}$ . There is a polynomial time transformation  $f$  from  $\Pi$ -AA to  $\Pi_0$ -AA such that an instance*

$(V, \mathcal{C}, k)$  of  $\Pi$ -AA is a “yes”-instance if and only if  $(V, \mathcal{C}_0, k) = f(V, \mathcal{C}, k)$  is a “yes”-instance of  $\Pi_0$ -AA.

*Proof.* From an instance  $(V, \mathcal{C}, k)$  of  $\Pi$ -AA, construct an instance  $(V, \mathcal{C}_0, k)$  of  $\Pi_0$ -AA as follows. For each triple  $(v_1, v_2, v_3) \in \mathcal{C}$ , add  $|\Pi|$  triples  $(v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)})$ ,  $\pi \in \Pi$ , to  $\mathcal{C}_0$ .

Observe that a triple  $(v_1, v_2, v_3) \in \mathcal{C}$  is  $\Pi$ -satisfied if and only if exactly one of the triples  $(v_{\pi(1)}, v_{\pi(2)}, v_{\pi(3)})$ ,  $\pi \in \Pi$ , is  $\Pi_0$ -satisfied. Thus,  $\frac{|\Pi|}{6}|\mathcal{C}| + k$  constraints from  $\mathcal{C}$  are  $\Pi$ -satisfied if and only if the same number of constraints from  $\mathcal{C}_0$  are  $\Pi_0$ -satisfied. It remains to observe that  $\frac{|\Pi|}{6}|\mathcal{C}| + k = \frac{1}{6}|\mathcal{C}_0| + k$  as  $|\mathcal{C}_0| = |\Pi| \cdot |\mathcal{C}|$ .  $\square$

For a variable set  $V$ , a constraint multiset  $\mathcal{C}$  over  $V$  and a linear ordering  $\alpha$  of  $V$ , the  $\alpha$ -deviation of  $(V, \mathcal{C})$  is the number  $\text{dev}(V, \mathcal{C}, \alpha)$  of constraints of  $\mathcal{C}$  that are  $\Pi$ -satisfied by  $\alpha$  minus  $\frac{|\Pi|}{6}|\mathcal{C}|$ . The maximum deviation of  $(V, \mathcal{C})$ , denoted  $\text{dev}(V, \mathcal{C})$ , is the maximum of  $\text{dev}(V, \mathcal{C}, \alpha)$  over all linear orderings  $\alpha$  of  $V$ . Now the problem  $\Pi$ -AA can be reformulated as the problem of deciding whether  $\text{dev}(V, \mathcal{C}) \geq k$ .

### 3 Probabilistic and Harmonic Analysis Tools

We build on the probabilistic *Strictly Above Expectation* method by Gutin et al. [19] to prove non-trivial lower bounds on the minimum fraction of satisfiable constraints in instances belonging to a restricted subclass. For such an instance with parameter  $k$ , we introduce a random variable  $X$  such that the instance is a “yes”-instance if and only if  $X$  takes with positive probability a value greater than or equal to  $k$ . If  $X$  happens to be a symmetric random variable with finite second moment then  $\mathbb{P}(X \geq \sqrt{\mathbb{E}[X^2]}) > 0$ ; it hence suffices to prove  $\mathbb{E}[X^2] = h(k)$  for some monotonically increasing unbounded function  $h$ . (Here,  $\mathbb{P}(\cdot)$  and  $\mathbb{E}[\cdot]$  denote probability and expectation, respectively.) If  $X$  is not symmetric then the following lemma can be used instead.

**Lemma 1 (Alon et al. [1]).** *Let  $X$  be a real random variable and suppose that its first, second and forth moments satisfy  $\mathbb{E}[X] = 0$ ,  $\mathbb{E}[X^2] = \sigma^2 > 0$  and  $\mathbb{E}[X^4] \leq c\sigma^4$ , respectively, for some constant  $c$ . Then  $\mathbb{P}(X > \frac{\sigma}{2\sqrt{c}}) > 0$ .*

We combine this result with the following result from harmonic analysis.

**Lemma 2 (Hypercontractive Inequality [4,14]).** *Let  $f = f(x_1, \dots, x_n)$  be a polynomial of degree  $r$  in  $n$  variables  $x_1, \dots, x_n$  with domain  $\{-1, 1\}$ . Define a random variable  $X$  by choosing a vector  $(\epsilon_1, \dots, \epsilon_n) \in \{-1, 1\}^n$  uniformly at random and setting  $X = f(\epsilon_1, \dots, \epsilon_n)$ . Then  $\mathbb{E}[X^4] \leq 9^r \mathbb{E}[X^2]^2$ .*

### 4 Facts on the Betweenness and Acyclic Subdigraph Problems

Let  $u, v, w$  be variables. We denote a betweenness constraint “ $v$  is between  $u$  and  $w$ ” by  $(v, \{u, w\})$ , and call a 3-set  $S$  of betweenness constraints over  $\{u, v, w\}$

complete if  $S = \{(u, \{v, w\}), (v, \{u, w\}), (w, \{u, v\})\}$ . Since every linear ordering of  $\{u, v, w\}$  satisfies exactly one constraint in  $S$ , we obtain the following reduction.

**Lemma 3.** *Let  $(V, \mathcal{B})$  be an instance of BETWEENNESS and let  $\alpha$  be a linear ordering of  $V$ . Let  $\mathcal{B}'$  be the set of constraints obtained from  $\mathcal{B}$  by deleting all complete subsets. Then  $\text{dev}(V, \mathcal{B}, \alpha) = \text{dev}(V, \mathcal{B}', \alpha)$ .*

An instance of BETWEENNESS without complete subsets of constraints is called reduced.

Let  $(V, \mathcal{B})$  be an instance of BETWEENNESS, with  $\mathcal{B} = \{B_1, \dots, B_m\}$ , and let  $\phi$  be a fixed function from  $V$  to  $\{0, 1, 2, 3\}$ . A linear ordering  $\alpha$  of  $V$  is called  $\phi$ -compatible if for each pair  $u, v \in V$  with  $\alpha(u) < \alpha(v)$  it holds  $\phi(u) \leq \phi(v)$ . For a random  $\phi$ -compatible linear ordering  $\pi$  of  $V$ , define a binary random variable  $y_p$  that takes value one if and only if  $B_p \in \mathcal{B}$  is satisfied by  $\pi$  (if  $B_p$  is falsified by  $\pi$ , then  $y_p = 0$ ). Let  $Y_p = \mathbb{E}[y_p] - 1/3$  for each  $p \in [m]$ , and let  $Y = \sum_{p=1}^m Y_p$ .

Now let  $\phi$  be a random function from  $V$  to  $\{0, 1, 2, 3\}$ . Then  $Y, Y_1, \dots, Y_m$  are random variables. For a constraint  $B_p = (v, \{u, w\})$ , the distribution of  $Y_p$  as it is given in Table 2 implies that  $\mathbb{E}[Y_p] = 0$ . Thus, by linearity of expectation,  $\mathbb{E}[Y] = 0$ .

**Table 2.** Distribution of  $Y_p$  for constraint  $B_p = (v, \{u, w\})$

$ \{\phi(u), \phi(v), \phi(w)\} $	Relation	Value of $Y_p$	Prob.
1	$\phi(u) = \phi(v) = \phi(w)$	0	1/16
2	$\phi(v) \neq \phi(u) = \phi(w)$	-1/3	3/16
2	$\phi(v) \in \{\phi(u), \phi(w)\}$	1/6	6/16
3	$\phi(v)$ is between $\phi(u)$ and $\phi(w)$	2/3	2/16
3	$\phi(v)$ is not between $\phi(u)$ and $\phi(w)$	-1/3	4/16

The following lemma was proved by Gutin et al. [18] for BETWEENNESS in which  $\mathcal{B}$  is a set, not a multiset, but a simple modification of its proof gives us the following (see [16] for details):

**Lemma 4.** *For a reduced instance  $(V, \mathcal{B})$  of BETWEENNESS,  $\mathbb{E}[Y^2] \geq \frac{11}{768}m$ .*

In the ACYCLIC SUBDIGRAPH problem we are given a directed multigraph  $D = (U, A)$ , with parallel arcs allowed, and ask for a linear ordering  $\pi$  of  $V$  which maximizes the number of satisfied arcs, where an arc  $(u, v) \in A$  is *satisfied by*  $\pi$  if  $\pi(u) < \pi(v)$ . If  $\pi$  is a uniformly-at-random linear ordering of  $V$  then the probability of an arc of  $D$  being satisfied is  $1/2$ . Thus, there is a linear ordering  $\pi$  of  $V$  in which the number of satisfied arcs is at least  $|A|/2$ . We therefore define, for a digraph  $D = (U, A)$  and a linear ordering  $\pi$  of  $U$ , the  $\pi$ -deviation of  $D$  as the number of arcs satisfied by  $\pi$  minus  $|A|/2$ , and denote it by  $\text{dev}(V, A, \pi)$ . In the ACYCLIC SUBDIGRAPH-AA problem we are given a directed multigraph

$D = (U, A)$  and asked to decide whether there is a linear ordering  $\pi$  of  $U$  with  $\pi$ -deviation at least  $k$ , where  $k$  is a parameter.

As every linear ordering of  $U$  satisfies exactly one of two mutually opposite arcs  $(u, v)$  and  $(v, u)$ , we obtain the following reduction.

**Lemma 5.** *Let  $D = (U, A)$  be a directed multigraph and let  $\pi$  be a linear ordering of  $V$ . Let  $A'$  be the set of arcs obtained from  $A$  by deleting all pairs of mutually opposite arcs. Then  $\text{dev}(V, A, \pi) = \text{dev}(V, A', \pi)$ .*

A directed multigraph without mutually opposite arcs is called *reduced*.

Let  $D = (U, A)$  be a directed multigraph with  $A = \{a_1, \dots, a_m\}$  as multiset of arcs, and let  $\phi$  be a fixed function from  $U$  to  $\{0, 1, 2, 3\}$ . For a random  $\phi$ -compatible linear ordering  $\pi$  of  $U$ , define a binary random variable  $x_p$  that takes value one if and only if  $a_p$  is satisfied by  $\pi$ . Let  $X_p = \mathbb{E}[x_p] - 1/2$  for each  $p \in [m]$  and let  $X = \sum_{p=1}^m X_p$ .

Now let  $\phi$  be a random function from  $U$  to  $\{0, 1, 2, 3\}$ . Then  $X, X_1, \dots, X_m$  are random variables. For an arc  $(u, v)$ , the distribution of  $X_p$  as it is given in Table 3 implies that  $\mathbb{E}[X_p] = 0$ . Thus, by linearity of expectation,  $\mathbb{E}[X] = 0$ .

**Table 3.** Distribution of  $X_p$  for an arc  $(u, v)$

Relation between $\phi(u)$ and $\phi(v)$	Value of $X_p$	Prob.
$\phi(u) = \phi(v)$	0	1/4
$\phi(u) < \phi(v)$	1/2	3/8
$\phi(u) > \phi(v)$	-1/2	3/8

We have the following analogue of Lemma 4 proved in [16].

**Lemma 6.** *For reduced directed multigraphs  $D$  it holds that  $\mathbb{E}[X^2] \geq \frac{1}{32}m$ .*

The following theorem was proved in [19].

**Theorem 1.** ACYCLIC SUBDIGRAPH-AA has a kernel with a quadratic number of vertices and arcs.

## 5 Kernels for $\Pi$ -AA Problems

We start from the following key construction of this paper. With an instance  $(V, \mathcal{C})$  of LINEAR ORDERING, we associate an instance  $(V, \mathcal{B})$  of BETWEENNESS and two instances  $(V, A')$  and  $(V, A'')$  of ACYCLIC SUBDIGRAPH as follows: If  $C_p = (u, v, w) \in \mathcal{C}$ , then  $B_p = (v, \{u, w\}) \in \mathcal{B}$ ,  $a'_p = (u, v) \in A'$ , and  $a''_p = (v, w) \in A''$ . The following lemma is proved in [16].

**Lemma 7.** *Let  $(V, C, k)$  be an instance of LINEAR ORDERING-AA and let  $\alpha$  be a linear ordering of  $V$ . Then*

$$\text{dev}(V, \mathcal{C}, \alpha) = \frac{1}{2} [\text{dev}(V, A', \alpha) + \text{dev}(V, A'', \alpha) + \text{dev}(V, \mathcal{B}, \alpha)].$$

Let  $(V, \mathcal{C}, k)$  be an instance of LINEAR ORDERING-AA, and let  $\phi$  be a function from  $V$  to  $\{0, 1, 2, 3\}$ . For a random  $\phi$ -compatible linear ordering  $\pi$  of  $V$ , define a binary random variable  $z_p$  that takes value one if and only if  $C_p$  is satisfied by  $\pi$ . Let  $Z_p = \mathbb{E}[z_p] - 1/6$  for each  $p \in [m]$ , and let  $Z = \sum_{p=1}^m Z_p$ .

**Lemma 8.** *If  $Z \geq k$  then  $(V, \mathcal{C}, k)$  is a “yes”-instance of LINEAR ORDERING-AA.*

*Proof.* By linearity of expectation,  $Z \geq k$  implies  $\mathbb{E}[\sum_{p=1}^m z_p] \geq m/6 + k$ . Thus, if  $Z \geq k$  then there is a  $\phi$ -compatible permutation  $\pi$  that satisfies at least  $m/6 + k$  constraints.  $\square$

Fix a function  $\phi : V \rightarrow \{0, 1, 2, 3\}$  and assign variables  $Y_p, X'_p, X''_p$ , respectively, to the three instances of BETWEENNESS and ACYCLIC SUBDGRAPH above.

**Lemma 9.** *For each  $p \in [m]$ , we have  $Z_p = \frac{1}{2} [X'_p + X''_p + Y_p]$ .*

*Proof.* Let  $C_p = (u, v, w) \in \mathcal{C}$ . Table 4 shows the values of  $X'_p, X''_p, Y_p, Z_p$  for some relations between  $\phi(u), \phi(v)$  and  $\phi(w)$ . The values of  $X'_p, X''_p$  and  $Y_p$  can be computed using Tables 2 and 3. In all cases of Table 4 it holds  $Z_p = \frac{1}{2}(X'_p + X''_p + Y_p)$ . Thus,  $Z_p = \frac{1}{2}[X'_p + X''_p + Y_p]$  for each possible relation between  $\phi(u), \phi(v)$  and  $\phi(w)$ .  $\square$

Let  $X = \sum_{p=1}^m [X'_p + X''_p]$ , let  $Y = \sum_{p=1}^m Y_p$  and let  $\phi$  be a random function from  $V$  to  $\{0, 1, 2, 3\}$ . Then  $X, X'_1, \dots, X'_m, X''_1, \dots, X''_m, Y, Y_1, \dots, Y_m, Z, Z_1, \dots, Z_m$  are random variables. From  $\mathbb{E}[X'] = \mathbb{E}[X''] = \mathbb{E}[Y] = 0$  it follows that  $\mathbb{E}[Z] = 0$ .

**Table 4.** Values of  $X'_p, X''_p, Y_p, Z_p$

Relation between $\phi(u), \phi(v)$ and $\phi(w)$	$X'_p$	$X''_p$	$Y_p$	$Z_p$
$\phi(u) = \phi(v) = \phi(w)$	0	0	0	0
$\phi(v) < \phi(u) = \phi(w)$	-1/2	1/2	-1/3	-1/6
$\phi(v) > \phi(u) = \phi(w)$	1/2	-1/2	-1/3	-1/6
$\phi(v) = \phi(u) < \phi(w)$	0	1/2	1/6	1/3
$\phi(v) = \phi(u) > \phi(w)$	0	-1/2	1/6	-1/6
$\phi(u) < \phi(v) = \phi(w)$	1/2	0	1/6	1/3
$\phi(u) > \phi(v) = \phi(w)$	-1/2	0	1/6	-1/6
$\phi(u) < \phi(v) < \phi(w)$	1/2	1/2	2/3	5/6
$\phi(u) < \phi(w) < \phi(v)$	1/2	-1/2	-1/3	-1/6
$\phi(v) < \phi(u) < \phi(w)$	-1/2	1/2	-1/3	-1/6
$\phi(v) < \phi(w) < \phi(u)$	-1/2	1/2	-1/3	-1/6
$\phi(w) < \phi(u) < \phi(v)$	1/2	-1/2	-1/3	-1/6
$\phi(w) < \phi(v) < \phi(u)$	-1/2	-1/2	2/3	-1/6

We will be able to use Lemma 2 in the proof of Lemma 12 due to the following:

**Lemma 10.** *The random variable  $Z$  can be expressed as a polynomial of degree 6 in independent uniformly distributed random variables with values  $-1$  and  $1$ .*

*Proof.* Consider  $C_p = (u, v, w) \in \mathcal{C}$ . Let  $\epsilon_1^u = -1$  if  $\phi(u) = 0$  or  $1$  and  $\epsilon_1^u = 1$ , otherwise. Let  $\epsilon_2^u = -1$  if  $\phi(u) = 0$  or  $2$  and  $\epsilon_2^u = 1$ , otherwise. Similarly, we can define  $\epsilon_1^v, \epsilon_2^v, \epsilon_1^w, \epsilon_2^w$ . Now  $\epsilon_1^u \epsilon_2^u$  can be seen as a binary representation of a number from the set  $\{0, 1, 2, 3\}$  and  $\epsilon_1^u \epsilon_2^u \epsilon_1^v \epsilon_2^v \epsilon_1^w \epsilon_2^w$  can be viewed as a binary representation of a number from the set  $\{0, 1, \dots, 63\}$ , where  $-1$  plays the role of  $0$ . Then we can write  $Z_p$  as the polynomial

$$\frac{1}{64} \sum_{q=0}^{63} (-1)^{s_q} W_q \cdot (\epsilon_1^u + c_1^{uq})(\epsilon_2^u + c_2^{uq})(\epsilon_1^v + c_1^{vq})(\epsilon_2^v + c_2^{vq})(\epsilon_1^w + c_1^{wq})(\epsilon_2^w + c_2^{wq}),$$

where  $c_1^{uq} c_2^{uq} c_1^{vq} c_2^{vq} c_1^{wq} c_2^{wq}$  is the binary representation of  $q$ ,  $s_q$  is the number of digits equal  $-1$  in this representation, and  $W_q$  equals the value of  $Z_p$  for the case when the binary representations of  $\phi(u), \phi(v)$  and  $\phi(w)$  are  $c_1^{uq} c_2^{uq}, c_1^{vq} c_2^{vq}$  and  $c_1^{wq} c_2^{wq}$ , respectively. The actual values for  $Z_p$  for each case are given in the proof of Lemma 9. The above polynomial is of degree 6. It remains to recall that  $Z = \sum_{p=1}^m Z_p$ .  $\square$

Let us consider the following natural transformation of our key construction introduced in the beginning of this section. Let  $(V, \mathcal{C})$  be an instance of LINEAR ORDERING and  $(V, \mathcal{B})$ ,  $(V, A')$  and  $(V, A'')$  be the associated instances of BETWEENNESS and ACYCLIC SUBDIGRAPH. Let  $b$  be the number of pairs of mutually opposite arcs in the directed multigraph  $D = (V, A' \cup A'')$  that are deleted by our reduction rule, and let  $r = 2(m - b)$ . Let  $t$  be the number of complete 3-sets of constraints in  $\mathcal{B}$  whose deletion from  $\mathcal{B}$  eliminates all complete 3-sets of constraints in  $\mathcal{B}$  and let  $s = m - 3t$ .

**Lemma 11.** *We have  $\mathbb{E}[Z^2] \geq \frac{11}{3072}(r + s)$ .*

*Proof.* Let  $A = A' \cup A'' = \{a_1, \dots, a_{2m}\}$  and  $D = (V, A)$ . Fix a function  $\phi : V \rightarrow \{0, 1, 2, 3\}$ . For a random  $\phi$ -compatible linear ordering  $\pi$  of  $V$ , define a binary random variable  $x_i$  that takes value one if and only if  $a_i$  is satisfied by  $\pi$ . Analogously, define a binary random variable  $y_i$  that takes value one if and only if  $B_i$  is satisfied by  $\pi$ . Let  $X_i = \mathbb{E}[x_i] - 1/2$  for all  $i = 1, \dots, 2m$ , let  $Y_j = \mathbb{E}[y_j] - 1/3$  for all  $j = 1, \dots, m$  and let  $X = \sum_{i=1}^{2m} X_i, Y = \sum_{i=1}^m Y_i$ . Recall that  $b$  is the number of deleted pairs of mutually opposite arcs from  $D$ , and  $t$  is the number of complete 3-sets deleted from  $\mathcal{B}$ . Assume, without loss of generality, that the remaining arcs are  $a_1, \dots, a_r$  and the remaining betweenness constraints are  $B_1, \dots, B_s$ . Then  $X = \sum_{i=1}^{2m} X_i = \sum_{i=1}^r X_i, Y = \sum_{i=1}^m Y_i = \sum_{i=1}^s Y_i$  and, by

Lemma 9,  $Z = X + Y/2$ . Now let  $\phi$  be a random function from  $V$  to  $\{0, 1, 2, 3\}$ . We have the following:

$$\begin{aligned}\mathbb{E}[Z^2] &= \mathbb{E}[X^2 + XY + Y^2/4] = \mathbb{E}[X^2] + \mathbb{E}[Y^2]/4 + \mathbb{E}\left[\left(\sum_{i=1}^r X_i\right)\left(\sum_{j=1}^s Y_j\right)\right] \\ &= \mathbb{E}[X^2] + \mathbb{E}[Y^2]/4 + \sum_{i=1}^r \sum_{j=1}^s \mathbb{E}[X_i Y_j].\end{aligned}$$

We will show that  $\mathbb{E}[X_i Y_j] = 0$  for any pair  $(i, j)$ . Let  $\phi' : V \rightarrow \{0, 1, 2, 3\}$  be defined as  $\phi'(x) = 3 - \phi(x)$  for all  $x$ . Let  $X_i(\phi)$  be the value of  $X_i$  when considering  $\phi$ -compatible orderings and define  $X_i(\phi')$ ,  $Y_i(\phi)$  and  $Y_i(\phi')$  analogously. From Table 2 we note that  $Y_j(\phi) = Y_i(\phi')$ , and from Table 3 we note that  $X_j(\phi) = -X_i(\phi')$ . From  $\mathbb{E}[X_i Y_j] = \frac{1}{4^{|V|}} \sum_{\phi} X_i(\phi) Y_j(\phi)$  it follows that

$$2\mathbb{E}[X_i Y_j] = 2 \left[ \frac{1}{4^{|V|}} \sum_{\phi} X_i(\phi) Y_j(\phi) \right] = \frac{1}{4^{|V|}} \sum_{\phi} [X_i(\phi) Y_j(\phi) + X_i(\phi') Y_j(\phi')] = 0.$$

Therefore,  $\mathbb{E}[Z^2] = \mathbb{E}[X^2] + \mathbb{E}[Y^2]/4$ . It follows from Lemmas 4 and 6 that  $\mathbb{E}[X^2] \geq r/32$  and  $\mathbb{E}[Y^2] \geq \frac{11}{768}s$ . We conclude that  $\mathbb{E}[Z^2] \geq \frac{11}{3072}(r+s)$ .  $\square$

**Lemma 12.** *There is a constant  $c > 0$  such that if  $r+s \geq ck^2$ , then  $(V, \mathcal{C}, k)$  is a “yes”-instance of LINEAR ORDERING-AA.*

*Proof.* By Lemmas 10 and 2, we have  $\mathbb{E}[Z^4] \leq 9^6 (\mathbb{E}[Z^2])^2$ . As  $\mathbb{E}[Z] = 0$ , it follows from Lemma 1 that  $\mathbb{P}\left(Z > \frac{\sqrt{\mathbb{E}[Z^2]}}{2 \cdot 9^3}\right) > 0$ . By Lemma 11,  $\mathbb{E}[Z^2] \geq \frac{11}{3072}(r+s)$ . Hence,  $\mathbb{P}\left(Z > \frac{\sqrt{\frac{11}{3072}(r+s)}}{2 \cdot 9^3}\right) > 0$ . Therefore if  $r+s \geq ck^2$ , where  $c = 4 \cdot 9^6 \cdot 3072/11$ , then by Lemma 8  $(V, \mathcal{C}, k)$  is a “yes”-instance of LINEAR ORDERING-AA.  $\square$

After we have deleted mutually opposite arcs from  $D$  and complete 3-sets of constraints from  $\mathcal{B}$  we may assume, by Lemma 12, that  $D$  has an arc multiset  $A = \{a_1, \dots, a_r\}$  left, with  $r = O(k^2)$ , and  $\mathcal{B}$  now contains  $s = O(k^2)$  constraints  $B_1, \dots, B_s$ . By Lemma 7,  $\text{dev}(V, \mathcal{C}) = \max_{\pi}[(\text{dev}(V, A, \pi) + \text{dev}(V, B, \pi))/2]$ , where the maximum is taken over all linear orderings  $\pi$  of  $V$ .

We now create a new instance  $(V', \mathcal{C}', k)$  of LINEAR ORDERING-AA as follows. Let  $\omega$  be a new variable not in  $V$ . For every  $a_i = (u_i, v_i)$  add the constraints  $(\omega, u_i, v_i)$ ,  $(u_i, \omega, v_i)$  and  $(u_i, v_i, \omega)$  to  $\mathcal{C}'$ . For every  $B_i = (a_i, \{b_i, c_i\})$  add the constraints  $(b_i, a_i, c_i)$  and  $(c_i, a_i, b_i)$  to  $\mathcal{C}'$ . Let  $V'$  be the set of variables that appear in some constraint in  $\mathcal{C}'$ . Then  $(V', \mathcal{C}')$  is an instance of LINEAR ORDERING with  $O(k^2)$  variables and constraints. Now the number of constraints in  $\mathcal{C}'$  satisfied by any linear ordering  $\alpha$  of  $V'$  equals the number of arcs in  $D$  satisfied by  $\alpha$  plus the number of constraints in  $\mathcal{B}$  satisfied by  $\alpha$ . As the average

number of constraints satisfied in  $(V', \mathcal{C}')$  equals  $(3r + 2s)/6 = r/2 + s/3$ , it follows that  $\text{dev}(V, \mathcal{C}) = \max_{\pi}[(\text{dev}(V, A, \pi) + \text{dev}(V, B, \pi))/2] = \text{dev}(V', \mathcal{C}')/2$ . Hence,  $(V', \mathcal{C}', k)$  is a kernel of LINEAR ORDERING-AA with  $O(k^2)$  variables and constraints. We have established the following theorem.

**Theorem 2.** LINEAR ORDERING-AA has a kernel with  $O(k^2)$  variables and constraints.

Using Proposition 1 and Theorem 2 we prove the following in [16].

**Theorem 3.** There is a bikernel with  $O(k^2)$  variables from  $\Pi_i$ -AA to  $\Pi_j$ -AA for each pair  $(i, j)$  such that  $0 \leq i \leq 10$  and  $0 \leq j \leq 10$  but  $j \notin \{2, 7\}$ .

Using Theorems 1 and 3 we prove the following in [16].

**Theorem 4.** All ternary Permutation-CSPs parameterized above average have kernels with  $O(k^2)$  variables.

**Acknowledgements.** The authors thank Mark Jones for carefully reading the paper and finding several minor mistakes that we have corrected.

## References

1. Alon, N., Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: Solving MAX-r-SAT above a tight lower bound. Tech. Report arXiv:0907.4573, <http://arxiv.org/abs/0907.4573>; A preliminary version was published in Proc. SODA, pp. 511–517 (2010)
2. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. Syst. Sci. 75(8), 423–434 (2009)
3. Bodlaender, H.L., Thomassé, S., Yeo, A.: Kernel bounds for disjoint cycles and disjoint paths. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 635–646. Springer, Heidelberg (2009)
4. Bonami, A.: Étude des coefficients de Fourier des fonctions de  $L^p(G)$ . Ann. Inst. Fourier 20(2), 335–402 (1970)
5. Charikar, M., Guruvswami, V., Manokaran, R.: Every permutation CSP of arity 3 is approximation resistant. In: Proc. of CCC 2009, pp. 62–73 (2009)
6. Chor, B., Sudan, M.: A geometric approach to betweenness. SIAM J. Discrete Mathematics 11(4), 511–523 (1998)
7. Crowston, R., Gutin, G., Jones, M.: Note on Max Lin-2 above average. Inform. Proc. Lett. 110, 451–454 (2010)
8. Crowston, R., Gutin, G., Jones, M., Kim, E.J., Ruzsa, I.Z.: Systems of linear equations over  $\mathbb{F}_2$  and problems parameterized above average. To appear in Proc. SWAT 2010 (2010)
9. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
10. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
11. Galil, Z., Megiddo, N.: Cyclic ordering is NP-complete. Theor. Comput. Sci. 5(2), 179–182 (1977)

12. Goerdt, A.: On random betweenness constraints. In: Gębala, M. (ed.) FCT 2009. LNCS, vol. 5699, pp. 157–168. Springer, Heidelberg (2009)
13. Goerdt, A.: On random ordering constraints. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 105–116. Springer, Heidelberg (2009)
14. Gross, L.: Logarithmic Sobolev inequalities. Amer. J. Math. 97, 1061–1083 (1975)
15. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38, 31–45 (2007)
16. Gutin, G., van Iersel, L., Mnich, M., Yeo, A.: All Ternary Permutation Constraint Satisfaction Problems Parameterized Above Average Have Kernels with Quadratic Number of Variables. Technical Report arXiv:1004.1956v2 [cs.DS]
17. Gutin, G., Kim, E.J., Lampis, M., Mitsou, V.: Vertex Cover Problem Parameterized Above and Below Tight Bounds. Theory Comput. Syst. (in press)
18. Gutin, G., Kim, E.J., Mnich, M., Yeo, A.: Betweenness Parameterized Above Tight Lower Bound. J. Comput. Sys. Sci. (in press)
19. Gutin, G., Kim, E.J., Szeider, S., Yeo, A.: A probabilistic approach to problems parameterized above or below tight bounds. J. Comput. Sys. Sci. (in press); A preliminary version is in Proc. IWPEC 2009. LNCS. 5917, pp. 234–245 (2009)
20. Gutin, G., Rafiey, A., Szeider, S., Yeo, A.: The linear arrangement problem parameterized above guaranteed value. Theory Comput. Syst. 41, 521–538 (2007)
21. Gutin, G., Szeider, S., Yeo, A.: Fixed-parameter complexity of minimum profile problems. Algorithmica 52(2), 133–152 (2008)
22. Guttmann, W., Maucher, M.: Variations on an ordering theme with constraints. In: Navarro, G., Bertossi, L., Kohayakawa, Y. (eds.) Proc. 4th IFIP International Conference on Theoretical Computer Science-TCS 2006, pp. 77–90. Springer, Heidelberg (2006)
23. Isli, A., Cohn, A.G.: A new approach to cyclic ordering of 2D orientations using ternary relation algebras. Artificial Intelligence 122(1-2), 137–187 (2000)
24. Mahajan, M., Raman, V.: Parameterizing above guaranteed values: MaxSat and MaxCut. J. Algorithms 31(2), 335–354 (1999)
25. Mahajan, M., Raman, V., Sikdar, S.: Parameterizing above or below guaranteed values. J. Comput. Syst. Sci. 75(2), 137–153 (2009)
26. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford Univ. Press, Oxford (2006)
27. Opatrný, J.: Total ordering problem. SIAM J. Comput. 8(1), 111–114 (1979)
28. Villanger, Y., Heggernes, P., Paul, C., Telle, J.A.: Interval completion is fixed parameter tractable. SIAM J. Comput. 38(5), 2007–2020 (2009)

# Strong Formulations for the Multi-module PESP and a Quadratic Algorithm for Graphical Diophantine Equation Systems

Laura Galli<sup>1</sup> and Sebastian Stiller<sup>2</sup>

<sup>1</sup> DEIS, Alma Mater University of Bologna, Italy

<sup>2</sup> Institut für Mathematik, Technische Universität Berlin, Germany

**Abstract.** The Periodic Event Scheduling Problem (PESP) is the method of choice for real-world periodic timetabling in public transport. Its MIP formulation has been studied intensely for the case of uniform modules, i.e., when all events have the same period. In practice, multiple periods are equally important. Yet, the powerful methods developed for uniform modules generally fail for the multi-module case. We analyze a certain type of Diophantine equation systems closely related to the multi-module PESP. Thereby, we identify a structure, so-called *sharp trees*, that allows to solve the system in  $\mathcal{O}(n^2)$  time if the modules form a linear lattice. Based on this we develop the machinery to solve multi-module PESPs on real-world scale. In our computational results the new MIP-formulations considerably improve the solvability of multi-module PESPs.

## 1 Introduction

The Periodic Event Scheduling Problem (PESP) is a combinatorial optimization problem of great practical importance introduced by [16]. It is the model of choice for periodic timetabling in public transport [13], [14], has been used for periodic job shop [16] and traffic signal scheduling [7], [5], and has successfully and repeatedly been applied in practice.

The task is to schedule periodic recurring events, e.g., the arrivals and departures of trains, such that between pairs of periodic events periodic constraints are fulfilled. A periodic constraint between periodic events  $i$  and  $j$  means that for every realization of  $i$  there is at least one realization of  $j$  with time difference greater or equal some lower respectively less or equal some upper bound modulo the period of the application. Such constraints can model for periodic timetables the headway constraints, the stopping times, or that passengers can quickly transfer between trains of different lines. In fact, the high modeling power of the PESP (cf. [8]) even allows to include rolling stock minimization and crew scheduling into the timetabling. This facilitated the construction of the first mathematically optimized railway timetable, namely for the Berlin underground in 2005.

The PESP is most powerfully solved as a (mixed) integer linear program ((M)IP). Apart from their practical importance, the IPs arising from the PESP

are of independent theoretical interest. Any such IP is naturally associated to a digraph, whose nodes and arcs represent respectively periodic events and periodic constraints among them. There are node variables  $\pi_i$  for each node  $i$  and offset variables  $k_a$  for each arc  $a$ . Two constraints correspond to each arc  $a = (i, j)$ :  $\ell_a \leq \pi_j - \pi_i + k_a P \leq u_a$ . Thus, for each arc the difference of the node variables must be in the interval  $[\ell_a, u_a]$  modulo a constant  $P$ . This constant is the period of the application, e.g., the time that elapses until the next train of the same line arrives. This special class of IPs has attracted a lot of research attention from combinatorics and integer programming perspectives. It has been shown to be NP-complete in [11, 12], even MAXSNP-hard in [8], and to have an unbounded Chvatal rank [9]. Nevertheless, research efforts exploiting the graph structure have led to a rich understanding of these IPs. These insights facilitate methods that are eventually capable to solve instances of considerable and practically relevant size—e.g., the timetable optimization of a complete national railway system. Some of these instances are challenging enough to be found in the MIPLIB. (These are the instances on which we base the computational validation of our method.)

The strong methods for uniform-module PESP have three ingredients (cf. [10]): first, if the instance is feasible, then for any tree there exist optimal solutions that have offsets equal to zero on that tree; second, a strong MIP formulation can be constructed using an integral cycle basis; and finally, a class of rounding cuts (the Odijk inequalities), also based on a well chosen set of cycles, can further improve the strength of the model. All of this fails if the modules are not uniform. In general, solutions need non-zero offsets on all arcs. In general, an integral cycle basis does not give an equivalent formulation, and the Odijk inequalities become arbitrarily loose for multiple modules. Yet multi-module PESPs are justified and even desired from the applications perspective: A public transport system is often comprised of bus, subway, and commuter lines with different periods. Also traffic lights in the same urban area may have different periods.

For solving PESPs a special class of linear Diophantine equation systems (DES) naturally related to the PESP is important. Again, given a digraph one associates variables  $\pi$  to the nodes and  $k$  to the arcs, and each arc represents an equation of the form:  $\pi_j - \pi_i + k_a P_a = x_a$ . Note, the coefficients of the node variables are equal to 1. Setting the arc coefficients  $P_a = 1$  would allow for a trivial solution  $k \equiv x, \pi \equiv 0$ . In this sense the proposed class is the simplest non-trivial class of DES one can associate to a digraph. Therefore, we call them Graphical Diophantine Equation Systems (GDES). GDESs are not only similar to PESPs, they also play a role in the state-of-the-art solving methods for PESPs. In the special case, when all modules  $P_a$  are equal, the GDES can be solved in linear time by a straight forward algorithm. For the general case so far one has to resort to constructing the Hermite Normal Form (HNF) of the GDES matrix, which can be done in (high) polynomial time [15].

In this work, starting from an analysis of and a new algorithm for GDES, we develop a method capable of solving real-world multi-module PESPs. We

test this on multi-period instances which we get by changing the periods in the uniform PESP instances that can be found in the MIPLIB, namely, `timtab1` and `timtab2`. These MIPLIB instances are real-world railway timetabling instances.

**Related work:** A concise exposition of the PESP and the state-of-the-art theory and solving methods can be found in [8]. In [12] a set of particularly useful rounding cuts have been proposed for the first time. In [3] optimization over the first Chvatal rank of the PESP has been studied.

Diophantine equations systems can be solved as any other linear equation system, once they are presented by their Hermite Normal Form (HNF). There is a standard polytime algorithm for constructing the HNF [15]. While further research on the HNF focuses on algorithms that use less space, we are not aware of a specific algorithm for GDES. The GDES are a special, and particularly simple class of DES. Similar flavor the Mixing Set Problem defines a particularly simple class of Diophantine inequalities that has been studied intensively [2], [1] and also has practical applications. Another very important class of DES that are closely related to GDES are the so-called *unique games* [6].

For practical instances which feature two different modules  $P \mid P'$  (e.g., a system of trains of which some run every hour and others every two hours) the state-of-the-art [8] approach is to use a PESP with uniform module  $P'$  and double the events that have higher frequency. The duplications of such an event have to be mutually fixed by additional constraints. This increases the size of the PESP.

**Our Contribution and Outline:** In Section 2 we present the so-called *sharp tree* structure and show that if that exists then the offsets can be chosen zero on it. These trees exist and can be found in  $\mathcal{O}(n^2)$  time, if the modules are nested, i.e., form a linear lattice with respect to division. In Section 3 we show that the existence of a sharp tree allows for a cycle basis formulation, which is equivalent to the original arc formulation and gives a stronger IP. Moreover, we show that we can prune a multi-module PESP, such that the fundamental cycles of a sharp tree give particularly good inequalities. The same algorithm that finds a sharp tree in case of nested modules, also solves a GDES in time quadratic in the number of nodes (cf. Section 2). We also show that without the existence of a sharp tree similar approaches to strengthen the IP formulation or to quickly solve a GDES do not extend in general. In the final section we report on twelve instances derived from the afore mentioned MIPLIB timetabling problems.

**Definitions and Basics:** Firstly, we define the two main mathematical objects under consideration.

**Definition 1.** *A digraph  $G(V, A)$  together with a natural valued function on the arc set,  $P : A \rightarrow \mathbb{N}$ , and an integer valued function on the arc set  $x : A \rightarrow \mathbb{Z}$  is called a graphical representation of the following system of Diophantine equations on variable vectors  $(\pi, k) \in \mathbb{Z}^{|V|, |A|}$ :*

$$\pi_j - \pi_i + k_a P_a = x_a \quad \forall a (= (i, j)) \in A. \quad (1)$$

*A system for which a graphical representation exists is called a graphical Diophantine equation system (GDES).*

The results we derive also hold if  $x$  maps to the rationals and  $P$  can be negative, but we can restrict w.l.o.g. (cf. [8]) to natural numbers for simplicity and also for its significance in a practical context. From a GDES one can straight forward construct its representation and this representation is unique up to isomorphism. So we speak of *the representation* of a GDES.

**Definition 2.** *Given a digraph  $G(V, A)$  together with two rational functions on the arc set,  $\ell : A \rightarrow \mathbb{Q}$  and  $u : A \rightarrow \mathbb{Q}$ , a third natural valued function on the arcs,  $P : A \rightarrow \mathbb{N}$ , and a cost vector  $c \in \mathbb{Q}^{|A|}$ , the following mixed integer program is called a classical formulation for the periodic event scheduling problem (PESP):*

$$\min \sum_{(i,j)=a \in A} c_a(\pi_j - \pi_i + k_a P_a) \quad (2)$$

$$\ell_a \leq \pi_j - \pi_i + k_a P_a \leq u_a \quad \forall a (= (i, j)) \in A \quad (3)$$

$$\pi \in \mathbb{Q}^{|V|}, k \in \mathbb{Z}^{|A|}. \quad (4)$$

In most applications the events, i.e., the nodes—not primarily the arcs—are periodic. So the constraints should rather read:  $\ell_a \leq (\pi_j + k_j P_j) - (\pi_i + k_i P_i) \leq u_a$ . It is easily checked that this is equivalent to the formulation above, when we choose  $P_a = \gcd\{P_i, P_j\}$ .

We will use subscripts for the arguments of the functions  $x, u, \ell$  and  $P$  in the remainder. We abbreviate  $n := |V|$  and  $m := |A|$ . We will use  $V(G)$  and  $A(G)$  to denote node and arc sets of a graph. Generally the values of  $P$  are called periods or modules, those of  $x$  tensions, those of  $\pi$  potentials, and those of  $k$  offsets. In the remainder we will assume w.l.o.g. the graphs to be connected. It is easy to see, that if the image of  $u$  and  $\ell$  are in the integers there is always an optimal solution with all  $\pi$  integral. So basically, the PESP is an IP, although it is constantly referred to in the literature as a MIP.

**We Summarize Some Basics on the PESP:** Notice, the objective function only refers to pairwise differences of potentials. This is partly due to the pertinent applications, partly to the mathematical structure. The  $k$  variables model the modulo operator. It would be strange to count them in a practical objective. Moreover, for any feasible solution  $(\pi, k)$  and any  $q \in \mathbb{Q}$  also  $(q + \pi, k)$  is feasible and has the same objective value.

Assume all  $P_a$  are equal, and let  $x$  be the vector of arc differences of a solution  $(\pi, k)$ , i.e.,  $x_{(i,j)} = \pi_j - \pi_i + k_{(i,j)} P$ . A vector  $x$  arising from a solution in this way is called its *tension*. It is easily checked—and we will re-prove it as a by-product of a more general theorem—that for any tree  $T$  there is a vector  $(\pi, k)'$  with the same tension  $x$  but  $k'_a = 0$  for all  $a \in T$ . Note, that  $(\pi, k)'$  is also a feasible solution and has the same objective value as  $(\pi, k)$ , because it has the same tension. This gives rise to two important features of the PESP with uniform modules.

First, if we know the tension  $x$ , we can construct a feasible solution in a simple way: Set  $\pi_i = 0$  for an arbitrary node  $i$ . Choose an arbitrary spanning tree  $T$ , and *propagate  $\pi$  starting from  $i$  along  $T$  with respect to  $x$* . Propagation means, that we solve the equality system  $\pi_j - \pi_i = x_a$  for all  $a \in A(T)$  iteratively fixing the node values as we traverse the tree.

It is helpful to notice, that any arc  $a$  with  $u_a - \ell_a \geq P$  states a redundant condition. Also, we can replace a directed arc by its antiparallel arc, simply by multiplying both constraint by  $(-1)$ .

The second important concept are *cycle bases*. A cycle basis is a basis for the linear subspace spanned by the incidence vectors of cycles in the vector space  $\mathbb{Q}^m$  spanned by the arc incidence vectors. Note that a cycle may have forward and backward arcs. For the latter the incidence vector of the cycle has a  $(-1)$  entry. A cycle basis is called *integral*, if all cycles are integer linear combination of the elements of the basis. Given a spanning tree  $T$  in the graph, the fundamental cycles  $C(a, T)$  of all non-tree arcs  $a \notin A(T)$  form a cycle basis. A *fundamental cycle*  $C(a, T)$  of a non-tree arc  $a$  with respect to a spanning tree  $T$  is composed of the arc itself and the unique path in  $T$  connecting its endnodes. Such a basis is called a *fundamental cycle basis* (sometimes also: strictly fundamental). Every fundamental cycle basis is integral.

Finally, in the case of uniform modules, we can sum the constraints along a cycle  $C$ , yielding a new, valid constraint. Replacing  $\pi_j - \pi_i$  by  $x_{(i,j)}$  this constraint reads:  $\sum_{a \in C} x_a = k_C P$ , where  $k_C = \sum_{a \in C} k_a$ , and we assume w.l.o.g. all arcs to be directed in the orientation of the cycle. If a tension vector  $x$  fulfills this cycle constraint for all cycles of an integral cycle basis, then it is the tension of a solution  $(\pi, k)$ . If in addition  $x_a \in [\ell_a, u_a]$ , then it is the tension of a feasible solution  $(\pi, k)$ . In other words, for uniform modules an integral cycle basis gives rise to an equivalent MIP formulation. This *cycle basis formulation* has proven [10] significantly stronger than the original *arc formulation*.

For a cycle  $C$  we abbreviate  $\gcd(C) := \gcd\{P_a : a \in C\}$ .

## 2 Graphical Diophantine Equations Systems

Both, the Diophantine and the MIP results in this paper are based on Lemma [1] that guarantees the existence of well structured solutions under certain conditions. To state these conditions we define the following.

**Definition 3.** Let  $\mathcal{G}$  be a GDES and  $G$  the graph of its representation. A spanning tree  $T$  in  $G$  is called a sharp tree, if each of its fundamental cycles  $C(a, T)$  has greatest common divisor equal to  $P_a$ , the module of the cycle's non-tree arc  $a \in A(G) \setminus A(T)$ .

**Lemma 1.** Let  $\mathcal{G}$  be a GDES and  $T$  a sharp tree in the graph of its representation. If  $\mathcal{G}$  has a solution, then there is a solution of  $\mathcal{G}$  with  $k_a = 0$  for all  $a \in T$ .

*Proof.* Reorder the matrix of the GDES such that the following holds: The new matrix  $M$  starts with the  $n - 1$  rows corresponding to the arcs in  $T$ . Restricted to the columns affecting the  $\pi$  variables, these rows form a lower triangular matrix (the first column omitted). The columns affecting  $k$  form a diagonal matrix.

Index the nodes according to their column and arcs according to their rows in  $M$ . For two nodes  $v$  and  $w$  denote by  $\mathcal{P}(v, w, T)$  the unique path from  $v$  to  $w$  in  $T$ .

Let  $(\pi, k)^0$  be some solution. We construct a solution  $(\pi, k)^{n-1}$  over  $n - 1$  stages denoted  $(\pi, k)^i, i \in \{1 \dots n - 1\}$ . For each  $i \in \{1 \dots n - 1\}$  successively with increasing row index we take four steps:

1. Set  $k_i^i = 0$ .
2. Re-establish the correctness of the  $i$ -th equation by changing the node value  $\pi_i$  corresponding to  $M_{i,i}$ , the right most non-zero entry in the first  $n - 1$  columns, i.e.,  $\pi_i^i := \pi_i^{i-1} + M_{i,i}k_i^0P_i$ . Let  $\ell(i)$  be minimal with  $M_{i,\ell(i)} \neq 0$ , i.e., the other node of arc  $i$ .
3. Propagate the new node value downwards along the tree. Formally: For all remaining tree rows  $t \in \{i + 1 \dots n - 1\}$  successively with increasing row index set  $\pi_t^i := \pi_t^{i-1} + M_{i,t}k_i^0P_i$  in case  $\ell(i) \notin \mathcal{P}(i, t, T)$ .
4. Re-establish correctness (in arbitrary order) for the equations of non-tree arcs by adjusting their arc variables. Formally: For all  $j \in \{n, \dots, m\}$  let  $1 \leq r < s \leq n - 1$  be the nodes of arc  $j$ , i.e.,  $M_{j,r}$  and  $M_{j,s} \neq 0$ . Set  $k_j^i := k_j^{i-1} - \frac{M_{j,r}(\pi_r^i - \pi_r^{i-1}) + M_{j,s}(\pi_s^i - \pi_s^{i-1})}{P_j}$ .

(We were a bit sloppy dropping exceptional handling of first row and column, and omitting when  $\pi_j^i := \pi_j^{i-1}$  and likewise for  $k$ .)

Obviously, each  $(\pi, k)^i$  fulfills all equalities. Observe, we touch the arc variable  $k_t$  of any tree row only in stage  $t$ . Therefore,  $(\pi, k)^i$  the solution of any stage  $i \in \{1 \dots n - 1\}$  has  $k_t^i = 0$  for all  $t \in \{1 \dots i\}$ . It remains to show for the non-tree arcs  $j \in \{n \dots m\}$  that every  $k_j^i$  is an integer, in particular, that

$$P_j \mid [M_{j,r}(\pi_r^i - \pi_r^{i-1}) + M_{j,s}(\pi_s^i - \pi_s^{i-1})].$$

For all nodes  $s$  we have  $\pi_s^i - \pi_s^{i-1} = |k_s^0P_i|$ . Now, distinguish whether  $\ell(i) \in \mathcal{P}(r, s, T)$  or not. If  $\ell(i)$  is in, so is  $i$  and the  $i$ -th arc is on the fundamental cycle of  $j$  in  $T$ . Thus,  $P_j \mid P_i$  by condition of the lemma and we are done. In case,  $\ell(i) \notin \mathcal{P}(r, s, T)$  both nodes are changed by the same value. Therefore,  $M_{j,r}(\pi_r^i - \pi_r^{i-1}) + M_{j,s}(\pi_s^i - \pi_s^{i-1}) = 0$ , which completes the proof.  $\square$

To guarantee the existence of sharp trees we need the following property:

**Definition 4.** We say a GDES (or a PESP) has nested modules, if for each pair of its modules  $P_a \leq P_b$  we have  $P_a \mid P_b$ .

We will show that GDES with nested modules have sharp trees, whereas sharp trees do not exist in general.

**Nested Modules:** The DENDI-algorithm (Diophantine Equations with Nested DIvisors) (cf. 4 for pseudo-code) solves GDES with nested modules. In addition it constructs a sharp tree. The algorithm considers the arcs in subsequent levels  $\ell$  according to their module. On the first level, it constructs spanning trees in the connected components of arcs with maximal modules. Then it shrinks these components to super-nodes and carries on with the next smaller level of modules. This way DENDI-algorithm constructs a subgraph, actually a tree, along which one can propagate the potentials  $\pi$  according to the tensions. Finally, DENDI verifies whether the equations of the non-tree arcs can

also be fulfilled for the chosen  $\pi$ . For the correctness of the algorithm we show two lemmata.

**Lemma 2.** *The subgraph  $T$  returned by the DENDI-algorithm is a sharp tree.*

*Proof.* Obviously,  $T$  is a spanning tree. Consider a non-tree arc  $a$  and its module  $P_a$ . Every other arc  $b$  in the fundamental cycle  $C(a, T)$  either belongs to the same component as  $a$  on level  $\ell$  or to a tree of a component shrunk into a super-node on an earlier level. In both cases  $P^\ell \leq P_b$ . The modules being nested, this implies  $P_a \mid P_b$ , and thus  $T$  is sharp.  $\square$

Thus, we have:

**Theorem 1.** *A GDES with nested modules has a sharp tree.*

**Lemma 3.** *The DENDI-algorithm returns failure, iff the GDES is infeasible. Moreover, the offset  $k_a = 0$  for all tree arcs  $a \in A(T)$ .*

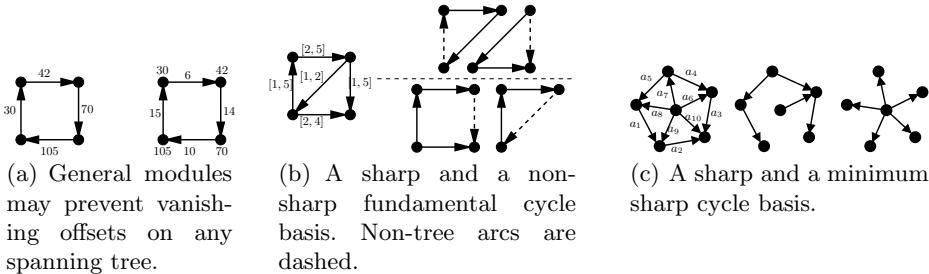
*Proof.* The offsets vanish on the tree arcs by construction, and because of the final test in DENDI-algorithm,  $(\pi, k)$  is a solution to the GDES, if they are returned. By Lemma 2  $T$  is sharp. Hence, by Lemma II the GDES has a solution iff it has a solution with  $k_a = 0$  for all  $a \in A(T)$ . If such a solution exists, it is fully determined by the potential of one node  $i$ , propagated along the spanning tree  $T$ . If  $\pi^*$  is such a solution, then  $\pi^* + z$  is also one for all  $z \in \mathbb{Z}$ . In particular, the  $\pi$  constructed in DENDI-algorithm is one. Thus, if the  $\pi$  constructed by DENDI is infeasible, then the GDES is infeasible.  $\square$

Observe that one can force any minimum spanning tree algorithm to return the same tree as the (weighted) DENDI-algorithm by introducing the following weights: The weight of an arc is (the sum of its original weight  $w$  and) a multiple of a large constant  $M (> \sum w)$ , where arcs with larger period get smaller multiples of  $M$ . Therefore, we can substitute DENDI-algorithm by any MST algorithm and conclude:

**Theorem 2.** *The DENDI-algorithm is correct, and has a running time in  $\mathcal{O}(n^2)$ , where  $n$  is the dimension of the solution vector.*

**General Modules:** Requiring nested modules may be suitable for the application but constitutes a strong mathematical restriction. Still, the example on the left of Figure 1(a) shows that we cannot hope for similar results in the general case.

*Example 1.* In Figure 1(a) the numbers next to the arcs and nodes give the modules of the arcs respectively the nodes. On the right, the arc modules result as gcd of the node modules. Assume the tensions along the cycle sum up to 1. This is feasible, because the gcd of all arcs is 1. Yet, a feasible solution must have  $k \neq 0$  on all arcs for the left graph, and either on the two vertical or the two horizontal arcs for the right graph.



In general, a solution for any cycle  $C$  must have non-zero offsets  $k$  on a subset  $S$  of  $C$ 's arcs, such that  $\gcd(S) = \gcd(C)$ . As in the example this may require all offsets to be non-zero.

Looking at GDESs from the perspective of the PESP and its applications, the following objection is valid: In the application we are given periods for the events, i.e., the nodes. The period (module) of an arc  $a = (i, j)$  arises in an equivalent formulation as  $P_a = \gcd\{P_i, P_j\}$ . Thus, the situation on the left of Figure 1(a) cannot occur. Still, (cf. the example on the right in Figure 1(a)) node modules can be such, that a solution must have non-zero offsets on a subset of the arcs, that forms a maximal matching on any cycle.

### 3 Solving PESP with Multiple Modules

The methods for the PESP with uniform modules rest on a strong formulation based on an integral cycle basis and on certain rounding cuts, that also stem from cycles. We will show that in general an integral cycle basis for a multi-module PESP does *not* give an equivalent formulation. Whereas, if a sharp tree exists, we will show that, its fundamental cycle basis provides for the desired strong formulation. In general, a multi-module PESP need not have a sharp tree. But we have seen in the previous section that a sharp tree can be found in case of nested modules.

The proper generalization of integral cycle bases for multi-module PESPs is the following type of basis:

**Definition 5.** *A fundamental cycle basis stemming from a sharp tree is called a sharp cycle basis.*

In particular, this means for each cycle  $C$  in a sharp basis, that the  $\gcd(C)$  is attained by the non-tree arc of  $C$ . Recall, that any fundamental cycle basis, and thereby any sharp cycle basis is integral.

**Lemma 4.** *Let  $\mathcal{B}$  be a sharp cycle basis in a PESP model. For an arc vector  $x$  the following three statements are equivalent:*

1. *The vector  $x$  is the arc tension of a node potential  $\pi$ .*
2. *The vector  $x$  fulfills the cycle equality for every cycle  $C$ , i.e., there is  $k_C \in \mathbb{Z}$  such that  $\sum_{a \in C} x_a = k_C \cdot \gcd(C)$ .*
3. *The vector  $x$  fulfills the cycle equality for every cycle  $C \in \mathcal{B}$ .*

*Proof.* The inclusion  $2 \Rightarrow 3$  being trivial, we show  $1 \Rightarrow 2$  and  $3 \Rightarrow 1$ .

$1 \Rightarrow 2$ : According to (1) we have  $x_{(i,j)} = \pi_j - \pi_i + k_{(i,j)} P_{(i,j)}$  for all arcs  $(i,j) \in A$ . Summing along a cycle  $C$  (multiplying the equation of  $a$  with  $(-1)$  for arcs  $a$  that lie in  $C$  contrary to its orientation) we get  $\sum_{(i,j) \in C} x_{(i,j)} = k_{(i,j)} P_{(i,j)}$ .

$3 \Rightarrow 1$ : Define a node potential  $\pi$  by setting  $\pi_s = 0$  for some node  $s$  and propagate the  $x$  value along the sharp tree  $T$ . It remains to show that for every non-tree arc  $(i,j) \in A \setminus T$  there is  $k_{(i,j)} \in \mathbb{Z}$  such that

$$\pi_j - \pi_i + x_{(i,j)} = k_{(i,j)} P_{(i,j)}. \quad (5)$$

The fundamental cycle  $C := (\mathcal{P}(i,j,T), (i,j))$  is in  $\mathcal{B}$  and  $P_{(i,j)} = \gcd(C)$ , and Equation (5) follows.  $\square$

Together with Theorem 2 we get:

**Theorem 3.** *Let  $\mathcal{G}$  be a PESP in the arc formulation. If a sharp cycle basis formulation for  $\mathcal{G}$  exists, it is equivalent. If  $\mathcal{G}$  has nested modules, then a sharp cycle basis formulation exists and can be found in time in  $\mathcal{O}(n^2)$ .*

For multiple modules the cycle basis formulation is not equivalent to the arc formulation even if the modules are nested. In the following example we show that non-sharp trees do not guarantee an equivalent cycle basis formulation.

*Example 2.* Consider the graph on the left of Figure 1(b). Set the module  $P_a = 6$  for all arcs except the diagonal one. For this set the module to 3. The interval next to an arc shows its upper and lower bounds. As cost vector choose the unit vector.

On the right of Figure 1(b) we show two fundamental cycle bases corresponding to two different trees. The non-tree arcs of a fundamental cycle are drawn dashed. On top, the cycle basis consists of two triangles. The corresponding tree is not a sharp tree, because for both cycles the non-tree arc has module 6 and the cycle's gcd is 3. Again, the example cannot occur if the periods stem from the nodes. Still, if one replaces the diagonal arc by two arcs of period 3 one can choose the node periods accordingly.

Index the arcs clockwise starting left and put the diagonal arc last. Then the optimal solution for this cycle basis formulation is  $x = (2, 2, 1, 2, 2)$ , for which DENDI-algorithm returns failure, i.e., it is not a tension of a feasible solution and thus the cycle basis formulation is *not* equivalent to the original arc formulation.

Yet, if we consider the sharp tree consisting of the left, upper, and lower arc, the corresponding cycle basis, shown at the bottom right of Figure 1(b), gives a formulation which is equivalent to the arc formulation, as stated in Theorem 3. In particular, the optimal solution we get is  $x = (3, 2, 3, 2, 1)$ , for which DENDI-algorithm is able to find a feasible set of potentials.

**Algebraic Pruning:** Assume again nested modules. Consider an arc  $a$  that is in no cycle or only in cycles  $C$  with  $\gcd(C)$  strictly smaller than  $P_a$ . Assume we

have a solution  $(x, k)^*$  to a sharp cycle basis formulation of the PESP. The arc  $a$  must lie in the sharp tree  $T$  of any such formulation. Therefore, recovering node potentials from  $x^*$  will result in a solution  $(\pi, k)'$  with  $k'_a P_a = k_{C(a,T)} \gcd(C)$ . Thus, the inequality of arc  $a$  is also fulfilled modulo  $\gcd(C)$ —which is strictly smaller than  $P_a$ . This observation allows to simplify a PESP with nested divisors:

**Theorem 4.** *Given a PESP  $\mathcal{G}$  with nested modules containing an arc  $a$  with  $\gcd(C) < P_a$  for all cycles  $C \ni a$ . The PESP  $\mathcal{G}'$ , resulting from  $\mathcal{G}$  by replacing  $P_a$  by its largest divisor  $P'_a \neq P_a$ , is equivalent to  $\mathcal{G}$ .*

One may repeatedly apply Theorem 4 to simplify the PESP in a pre-processing. In case the considered arc is in no cycle, its module is ultimately set to 1, which is equivalent to removing the arc. Any solution for the (after the removal) disconnected graph can be amalgamated to a solution of the original PESP in a linear time post-processing. As one can remove arcs, for which the difference between upper and lower bound is greater or equal to the module, even reducing to a non-trivial module can result in the arc being obsolete. This reduces the dimension of the MIP and allow to assume the following property for the sharp basis found by the DENDI:

**Observation 1.** *W.l.o.g. for every arc  $a$  a sharp cycle basis contains a cycle  $C$ , such that  $\gcd(C) = P_a$ .*

The basis of the DENDI-algorithm is sharp and thus has a tight cycle for each arc. This will be exploited in the last section, where we seek to give a small set of strong cuts derived from cycle inequalities.

**Cuts and Sharp Trees:** Solving a uniform-module PESP cycles are also used to produce a special class of rounding cuts, the so-called *Odijk* inequalities:

$$\left\lceil \frac{\sum_{a_+ \in C} \ell_a - \sum_{a_- \in C} u_a}{P} \right\rceil \leq k_C \quad \text{and} \quad \left\lfloor \frac{\sum_{a_+ \in C} u_a - \sum_{a_- \in C} \ell_a}{P} \right\rfloor \geq k_C \quad (6)$$

The key question is, for which cycles one should add the corresponding Odijk inequalities to the MIP formulation. For the case of uniform modules there is a well established heuristic reasoning: The right-hand side is rounded down (or up) by a value between 0 and  $P - 1$ . If the total value of the right-hand side is large in comparison to  $P$  the effect of rounding cannot be large.

Therefore, one is interested in shortest integral cycle bases. There is no polynomial time algorithm known for this problem. Yet, there are many heuristics to find short integral cycle bases. A standard approach is to construct a fundamental cycle basis from a minimum spanning tree (MST). Here the heuristic idea is, that the non-tree arcs feature in exactly one cycle, whereas the minimized tree arcs can occur in several cycles of the basis. Thus, the sum of all cycles will be rather small, and the Odijk inequalities likely tight. For multiple modules the rounding on a cycle  $C$  is between 0 and  $\gcd(C)$ . Assume the cycle  $C$  contains an arc  $a$  with a significantly smaller module  $P_a$  than that of all other arcs in  $C$ .

One can assume the difference between upper and lower bound on an arc to be less than its module. But, the contribution to the right-hand side by each other arc  $b \in C$  can be much larger than  $P_a$ , because  $P_b >> P_a$ . Thus, if an arc  $b$  is in no cycle  $C$  with  $\gcd(C)$  close to its own module  $P_b$ , the set of cuts will not have a relevant effect on the number of choices for  $x_b$ .

Theorem 4 allows to assume that every arc  $b \in A$  is in at least one cycle  $C$  with  $\gcd(C) = P_b$ . A sharp cycle basis for every arc  $b$  contains such a cycle  $C$ . So, for multiple-modules PESPs we propose to choose cycle basis  $\mathcal{B}$  such that (1)  $\mathcal{B}$  is a sharp cycle basis and (2)  $\mathcal{B}$  arises from a sharp tree with minimal sum of arc weights (with respect to  $(u - \ell)$ ) among all sharp trees. This can be found by the weighted version of DENDI-algorithm.

## 4 Computational Results

We study twelve instances derived from `timtab1` and `timtab2`, the MIPLIB PESP instances.<sup>1</sup> These two MIPLIB instances are anonymized real-world timetabling problems of a major European railway provider. We changed the original periods of 60 on the *nodes* randomly to the nested periods 120, 60, 30, 15 and 5, giving lower probability to the small periods as they dominate in the transition from node to arc periods. After this transition, the bounds on the arcs were adjusted relative to the change in period.

On these instances we compare the standard formulation to a sharp cycle base formulation with the basis' Odijk inequalities. For each we use CPLEX 10.0 with a timelimit (TL) of 2 hours on a 2.4Ghz processor. The sharp cycle basis formulation is strikingly faster in detecting infeasibility and solves all except one of the feasible instances with a better gap (cf. Table II).

**Table 1.** multi-period miplib PESP statistics

instance	<i>Classical</i>		<i>Sharp Tree + Odijks</i>			
	status	GAP%	time (sec.)	status	GAP%	time (sec.)
mpesp1	feasible	5.99	TL	feasible	4.19	TL
mpesp2	feasible	6.13	TL	feasible	5.73	TL
mpesp3	feasible	5.58	TL	feasible	3.83	TL
mpesp4	feasible	2.94	TL	feasible	2.50	TL
mpesp5	feasible	5.33	TL	feasible	5.29	TL
mpesp6*	feasible	9.81	TL	feasible	10.26	TL
mpesp7	feasible	12.09	TL	feasible	9.72	TL
mpesp8	feasible	12.87	TL	feasible	9.71	TL
mpesp9	-	-	TL	infeasible	-	0
mpesp10	-	-	TL	infeasible	-	3431
mpesp11	infeasible	-	6934	infeasible	-	0
mpesp12	infeasible	-	657	infeasible	-	0

<sup>1</sup> Elmar Swarat kindly provided us with the raw data of `timtab`.

Finally, note that nested periods in timetable optimization are also recommendable in the light of quality of service: They yield that more passenger actually experience the optimized transfer time, because, e.g., for co-prime periods *every* transfer time will be experienced by some passengers.

## References

1. Conforti, M., Di Summa, M., Wolsey, L.: The mixing set with divisible capacities. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 435–449. Springer, Heidelberg (2008)
2. Conforti, M., Zambelli, G.: The mixing set with divisible capacities: A simple approach. *Operations Research Letters* 37, 379–383 (2009)
3. Fischetti, M., Lodi, A.: Optimizing over the first Chvatal closure. *Mathematical Programming* 110(1), 3–20 (2006)
4. Galli, L., Stiller, S.: Strong Formulations for the Multi-module PESP and a Quadratic Algorithm for Graphical Diophantine Equation Systems. COGA Technical Report 009–2010 (2010)
5. Hassin, R.: A flow algorithm for network synchronization. *Operations Research* 44, 570–579 (1996)
6. Khot, S.: On the power of unique 2-prover 1-round games. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 767–775. ACM Press, New York (2002)
7. Köhler, E., Möhring, R., Nökel, K., Wünsch, G.: Optimization of Signalized Traffic Networks. In: Mathematics – Key Technology for the Future, pp. 179–180. Springer, Heidelberg (2008)
8. Liebchen, C.: Periodic Timetable Optimization in Public Transport. Ph.D. thesis, Technische Universität Berlin (2006)
9. Liebchen, C., Swarat, E.: The Second Chvatal Closure Can Yield Better Railway Timetables. In: Proceedings of 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems. Online Schloss Dagstuhl (2008)
10. Liebchen, C., Proksch, M., Wagner, F.H.: Performance of Algorithms for Periodic Timetable Optimization. In: Computer-aided Systems in Public Transport, pp. 151–180. Springer, Heidelberg (2008)
11. Nachtigall, K.: Cutting planes for a polyhedron associated with a periodic network. DLR Technical Report 112-96/17
12. Odijk, M.: Construction of periodic timetables, Part1: a cutting plane algorithm. TU Delft Technical Report 94-61 (1994)
13. Odijk, M.: A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research B* 30(6), 455–464 (1996)
14. Peeters, L.: Cyclic Railway Timetable Optimization. Ph.D. thesis, Erasmus University of Rotterdam (2003)
15. Schrijver, A.: Theory of Linear and Integer Programming. Wiley & Sons, Chichester (1986)
16. Serafini, P., Ukovich, W.: A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics* 2(4), 550–581 (1989)

# Robust Algorithms for Sorting Railway Cars

Christina Büsing<sup>1</sup> and Jens Maué<sup>2</sup>

<sup>1</sup> Institut für Mathematik, Technische Universität Berlin, Germany  
`cbusing@math.tu-berlin.de`

<sup>2</sup> Institute of Theoretical Computer Science, ETH Zürich, Switzerland  
`jens.mae@inf.ethz.ch`

**Abstract.** We consider a sorting problem from railway optimization called train classification: incoming trains are split up into their single cars and reassembled to form new outgoing trains. Trains are subject to delay, which may turn a prepared sorting schedule infeasible for the disturbed situation. The classification methods applied today deal with this issue by completely disregarding the input order of cars, which provides robustness against any amount of disturbance but also wastes the potential contained in the a priori knowledge about the input.

We introduce a new method that provides a feasible sorting schedule for the expected input and allows to flexibly insert additional sorting steps if the schedule has become infeasible after revealing the disturbed input. By excluding disruptions that almost never occur from our consideration, we obtain a classification process that is quicker than the current railway practice but still provides robustness against realistic delays. In fact, our algorithm allows flexibly trading off fast classification against high degrees of robustness depending on the respective need. We further explore this flexibility in experiments on real-world traffic data, underlining our algorithm improves on the methods currently applied in practice.

## 1 Introduction

An essential process in railway optimization is *train classification*, which refers to the rearrangement of cars to form new trains. With increasing world-wide freight traffic, operating freight trains efficiently becomes more and more important, and reducing the dwell time of cars in railway yards is one of the key factors to improve freight service profitability.

**General Classification Process.** Exclusively for the purpose of train classification, there are installations of railway tracks and switches called *classification yards* (see Fig. I). Such a yard features a *hump track* on which inbound trains arrive and their cars are decoupled to be pushed over a sloping ramp called *hump* at the end of the hump track. Hence, the cars accelerate by gravity and roll through a tree of switches by which each car can be individually guided to some *classification track*. This is called a *roll-in* operation. In a *pull-out* operation an engine pulls all the cars on some classification track back to the hump track in order to perform a further roll-in. A pair of pull-out and roll-in operations is called a *(sorting) step*, and an initial roll-in followed by a sequence of  $h$  sorting

steps is called a *classification schedule* of *length h*. The number of steps  $h$  essentially determines the time required to conclude the sorting procedure. There are  $\ell$  inbound trains that, concatenated in the order they arrive at the yard, form the *inbound train sequence*. Moreover, there are order specifications for the  $m$  *outbound trains*, and a classification schedule is called *feasible* if its application to the inbound train sequence yields the correctly ordered outbound trains, each on a separate classification track.

**Robust Train Classification.** Often the inbound trains are subject to delay, so we might be faced with an unexpected inbound order of trains. In our model all disturbances, i.e. every combination of number of delayed trains and amount of delay for each train, that are to be covered are given by a *set of scenarios*  $\mathcal{S}$ . In this set, each scenario  $S \in \mathcal{S}$  defines a permutation of the inbound train sequence called *modified instance*. A schedule for the original instance is called a *first-stage solution*, and it may be infeasible for the modified instance corresponding to some scenario. In response to disturbed input, we are prepared to insert up to  $k$  additional sorting steps after the  $p$ th step of the first-stage solution, providing a *recovered solution*. A first-stage solution for which, for every scenario  $S \in \mathcal{S}$ , there is a recovered solution that is feasible w.r.t.  $S$  is called *recovery robust*. Given a sequence of  $\ell$  inbound trains,  $m$  order specifications of outbound trains, and a set of scenarios  $\mathcal{S}$ , the recovery-robust train classification problem is to find a recovery robust, feasible first-stage solution of minimum length.

**Related Work.** There are many publications in the field of railway engineering that describe different train classification methods, e.g. [7][16][3][17][5]. These methods are *strictly robust*, i.e. robust w.r.t. any set of scenarios, since they apply a predefined classification schedule that is independent of the order of railway cars entering the classification process. The method of *geometric sorting* (see [7][16][3][17]) minimizes the number of sorting steps for a worst case (or unknown) input order, which is proved in [11]. The still most-commonly used method in practice is *triangular sorting* [7][16][3][17][5], which is optimal for restricting the number of roll-ins per car to three for unknown input order [11]. However, neither method exploits the situation of a partially ordered input sequence, so they apply more sorting steps than necessary in general.

This issue was explored in [11], which develops a classification method that minimizes the number of sorting steps based on complete knowledge of the input data. Moreover, for the problem variant of classification tracks of bounded length, [11] shows that minimizing the number of sorting steps is an NP-hard problem. A 2-approximation for the same setting is derived in [12], several improvements of which are experimentally evaluated in [10] and compared to an exact integer programming approach, which was earlier introduced in [15]. A related algorithmic sorting problem is considered by Dahlhaus et al. [6]. Recent overviews of train classification can be found in [9] and [8].

Since changes during the process of scheduling are time consuming, a certain amount of robustness is crucial for classification methods to work in practice. Providing strict robustness, however, wastes a lot of potential to disruption scenarios that almost never occur in practice. As described above w.r.t. train

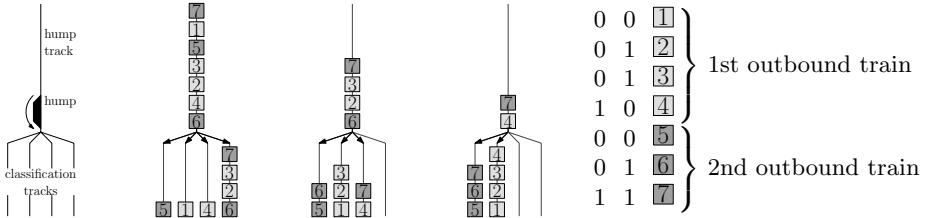
classification, this dilemma is tackled by the concept of recoverable robustness [14] by regarding realistic scenarios of delay and providing optimal robust solutions w.r.t. a limited amount of recovery in case of disturbance. This concept is applied to several railway-related optimization problems such as rolling stock scheduling [1] or timetabling [34]. A first and—to the best of our knowledge—only attempt to study this method for train classification is made by Cicerone et. al [2] for a single inbound and outbound train. (Their results are summarized in [3].) Besides the situations of strict robustness and complete recomputation from scratch, which are more of theoretical interest, they consider a recovery action that allows completely changing the classification instruction for one set of cars that have the same instruction. The most relevant scenarios in [2] are one additional car in the input and one car occurring at a different position than expected. The latter corresponds to our problem setting for the special case of trains consisting of single cars with a delay scenario of up to one train. We generalize this setting to scenarios with more delays (mainly Sect. 4) and the problem setting with complex trains. Besides, [2] deals with the scenario of a single classification track becoming unavailable before the classification starts. In this paper we focus on the most relevant reason for disruptions, which are delayed trains.

**Our Contribution.** For the mentioned recovery action of adding up to  $k$  sorting steps after an offset of  $p$  steps, we first introduce a generic algorithm in Sect. 3. We prove that, for every constant  $k \geq 1$ , finding a robust schedule of minimum length is an NP-complete problem for general sets of scenarios. For the practically relevant scenario of delaying up to  $j$  trains by an arbitrary amount each, the problem can be solved in polynomial time (see Sect. 4). Furthermore, we evaluate our new algorithm on real-world traffic data for various parameter values  $k$ ,  $p$ , and  $j$ . It turns out that, on the one hand, our algorithm yields very short schedules while providing a fair degree of robustness. On the other hand, it is capable of providing highly robust schedules that still improve on the current classification practice, emphasizing the flexibility of our approach to modulate between these conflicting objectives.

## 2 Encoding Classification Schedules

In addition to the concepts of Sect. 1, we introduce some further notation required for representing and deriving classification schedules.

**Terminology and Notation.** Corresponding to the notation of [1], we represent every car  $\tau$  by a positive integer  $\tau \in \mathbb{N}$  and a train  $T$  by a sequence of cars  $T = (\tau_1, \dots, \tau_k)$ , where  $k$  is called the *length* of  $T$ . There are  $\ell$  *inbound* trains  $T_1, \dots, T_\ell$ , whose concatenation we assume to be a permutation of  $(1, \dots, n)$ , and  $n$  is called the *volume* of cars. There are  $m$  *outbound* trains of respective length  $n_i$ ,  $i = 1, \dots, m$ , and we assume the specification of the first outbound train is  $(1, \dots, n_1)$ , the second  $(n_1+1, \dots, n_1+n_2)$ , etc. In contrast to the expected order of inbound trains, there is no order implied for the outbound trains.



**Fig. 1.** Classification yard (left). Classification process for  $h=2$ ,  $n=7$ , and  $m=2$ : initial roll-in (2nd picture), first step (3rd), second step (4th); in both steps, the *rightmost* occupied track is pulled out. Corresponding schedule encoding (right).

Regardless of which are their outbound trains, all cars are sorted simultaneously on the same set of classification tracks, called the *sorting tracks*. Their lengths and available number are unrestricted, and the number actually used corresponds to the number of sorting steps, where the track pulled in the  $k$ th step is referred to by  $\theta_k$ ,  $k = 0, \dots, h-1$ . The cars are finally collected on a separate track for each outbound train, which are called *destination tracks*.

**Schedule Representation.** We will refer to the binary representation of a decimal integer  $j \geq 0$  by  $[j]_2$ . Given any bitstring  $b = b_{h-1} \dots b_0$  of length  $h$ , let  $\text{num}(b)$  denote the integer number represented by  $b$ , i.e.,  $\text{num}(b) = \sum_{i=0}^{h-1} 2^i b_i$ . For two bitstrings  $b_1, b_2$  we define  $b_1 < b_2$  iff  $\text{num}(b_1) < \text{num}(b_2)$ . We represent classification schedules of length  $h$  by assignments of cars to bitstrings of length  $h$  [11]:  $b^j = b_{h-1}^j \dots b_0^j$  encodes the journey of the  $j$ th car with  $b_k^j = 1$  iff it visits  $\theta_k$  pulled out in the  $k$ th step. After such a pull-out, the car is sent to  $\theta_\ell$  with  $\ell = \min\{i | k < i < h, b_i^j = 1\}$ ; if  $b_i^j = 0$  for all  $i > k$ , it goes to the destination track of its outbound train. The  $n$  bitstrings  $b^1, \dots, b^n$  form an  $(n \times h)$ -matrix, and  $b_0, \dots, b_{h-1}$  denote its columns from “right” to “left”.

In order to derive a feasible schedule  $B$  of length  $h$ , two cars  $\tau$  and  $\tau+1$  of the same outgoing train must be assigned bitstrings  $b^\tau \leq b^{\tau+1}$ . If these cars occur in reversed order in the inbound sequence, we require  $b^\tau < b^{\tau+1}$ ; then, the pair  $\beta = (\tau, \tau+1)$  is called a *break*. If  $b^\tau$  and  $b^{\tau+1}$  occur in different outbound trains, there is no constraint between the two cars. As a result the length of a classification schedule depends on the maximum number of breaks of all outbound trains [12]. Figure 1 shows an example classification process and the corresponding encoding with four steps and tracks, where the rightmost track presents  $\theta_0$  in the notation above: cars 1 and 2 arrive in reversed order, so  $b_1 < b_2$ , whereas cars 2 and 3 arrive in correct order and have the same bitstring. Note that  $b^7 > b^6$  is fine though cars 6 and 7 arrive in correct order, and there is no constraint between  $b_4$  and  $b_5$  since the fourth and fifth car belong to different outbound trains.

### 3 Recovery through Additional Sorting Steps

In this section we investigate the recovery strategy of inserting a limited number of additional sorting steps to a first-stage schedule when a scenario occurs.

**Further Notation.** A pair of consecutive cars  $\beta = (\tau, \tau+1)$  is called *original break* if  $\beta$  is a break for the expected order of inbound trains. Given some  $S \in \mathcal{S}$ , we call  $\beta$  *induced by S* if  $\beta$  is a break in the modified instance corresponding to  $S$ . If  $\beta$  is not an original break but induced by any  $S \in \mathcal{S}$ ,  $\beta$  is called a *potential break*. W.l.o.g., we assume that every pair  $\beta = (\tau, \tau+1)$ ,  $\tau \in \{1, \dots, n-1\}$ , of successive cars is either an original or a potential break: for any problem instance with  $\beta$  not being a break, car  $\tau+1$  can be ignored while deriving a schedule and assigned the same bitstring as  $\tau$  in the final solution. For any first-stage solution  $B$ , a break  $\beta = (\tau, \tau+1)$  is called *unresolved w.r.t. S* if  $\beta$  is induced by  $S$  and  $b^\tau = b^{\tau+1}$ . For any scenario  $S \in \mathcal{S}$ ,  $X^S$  denotes the set of potential breaks induced by  $S$ . Note that this set  $X^S$  is uniquely defined for every scenario  $S \in \mathcal{S}$ , but there may be different scenarios  $S \neq S'$  with  $X^S = X^{S'}$ . We will repeatedly regard sets of potential breaks without considering the actual underlying scenario. In particular, we will often describe sets of scenarios (e.g. as a parameter in problem definitions) implicitly by providing the set of induced breaks of every scenario. Let  $T_1, \dots, T_\ell$  be a sequence of  $\ell$  inbound trains with  $n$  cars, and let  $X$  be the set of all original and potential breaks. For any pair of cars  $\tau_1, \tau_2$ ,  $1 \leq \tau_1 < \tau_2 \leq n$ , we define  $X_{(\tau_1, \tau_2)}$  as the set of all original and potential breaks occurring between  $\tau_1$  and  $\tau_2$ , i.e.,  $X_{(\tau_1, \tau_2)} = X \cap \{(\tau_1, \tau_1+1), (\tau_1+1, \tau_1+2), \dots, (\tau_2-1, \tau_2)\}$ .

**Recovery Model.** In many yards, there are certain classification tracks reserved for the sorting procedure considered here, while other tracks are used for different sorting activities. The initial roll-in to the reserved tracks is then scattered over the day, and, when the last train arrives, the other activities are stopped and the first pull-out performed. At this point a scenario is revealed for which the original schedule may be infeasible. With the recovery action of inserting up to  $k$  additional sorting steps to the first stage solution, we seek to obtain a feasible schedule for the modified instance. Distributing the recovered solution, i.e. the changed schedule, to all people involved in the operation takes some time depending on the available communication channels. For these reasons, inserting additional sorting steps is only allowed after an offset of  $p$  steps.

In terms of classification schedules, which present solutions to our optimization problem, this means the following: given two parameters  $p \geq 0$  and  $k \geq 0$  and a first-stage solution schedule  $B$  of length  $h$ ,  $B$  is to be recovered by inserting up to  $k$  additional columns with indices greater than  $p$ . This concept is formalized in the following definition.

**Definition 1.** Let  $B = (b_{h-1}, \dots, b_0)$  and  $B' = (b'_{h-1+j}, \dots, b'_0)$  be two classification schedules for  $n$  cars of length  $h$  and  $h+j$ ,  $j \geq 0$ , respectively. Let further  $p \geq 0$  and  $k \geq 0$ . The schedule  $B'$  is called a  $(p, k)$ -extension of  $B$  if  $j \leq k$ ,  $b_i = b'_i$  for all  $0 \leq i < p$  and  $b_{i-j} = b'_i$  for all  $p+j-1 \leq i \leq h+j-1$ .

Note that in the definition above the additional columns are all added between the  $(p-1)$ th and  $p$ th step of the original schedule. It can be shown easily that, if inserting  $k$  columns at the  $i$ th position yields a feasible recovered schedule, inserting these  $k$  columns at the  $(i-1)$ th position instead also yields a feasible schedule. Hence, inserting at the “right-most” allowed position always presents

the most powerful recovery. The notion of  $(p, k)$ -extensions yields a natural concept of recoverable robustness as stated in the following definition.

**Definition 2.** Let  $T_1, \dots, T_\ell$  be a sequence of  $\ell$  inbound trains and  $\mathcal{S}$  a set of scenarios with  $X^S$  denoting the corresponding induced set of breaks for every  $S \in \mathcal{S}$ . A classification schedule  $B$  is called  $(p, k)$ -recovery robust if, for every scenario  $S \in \mathcal{S}$ , there is a  $(p, k)$ -extension of  $B$  that is feasible w.r.t.  $S$ .

Most likely, no delay occurs and the inbound trains arrive in the expected order, in which case we usually do not want to apply any recovery for organizational reasons. For our objective this means we look for *feasible*  $(p, k)$ -recovery robust classification schedules of minimum length.

In order to specify when a given schedule is  $(p, k)$ -recovery robust for a given set of scenarios, we introduce the notion of a *block* of a schedule. A block basically is a maximal set of bitstrings representing integers between two powers of two.

**Definition 3.** Let  $B$  be a schedule of length  $h$  for an inbound train sequence of  $n$  cars, and  $p \geq 1$ . For any bitstring  $b^j$  of  $B$ ,  $b_{h-1}^j \dots b_p^j$  is called the leading part of  $b^j$ , denoted by  $b_{>p}^j$ , and  $b_{p-1}^j \dots b_0^j$  the trailing part of  $B$ , denoted by  $b_{\leq p}^j$ . A subset of  $\lambda$  consecutive bitstrings  $b^j, \dots, b^{j+\lambda-1}$  of  $B$  is called a block of  $B$  if their leading parts satisfy  $b_{>p}^{j-1} < b_{>p}^j$ ,  $b_{>p}^j = b_{>p}^{j+x}$  for all  $1 \leq x \leq \lambda - 1$ , and  $b_{>p}^{j+\lambda-1} < b_{>p}^{j+\lambda}$ , while  $\lambda$  is called the size of the block. Furthermore, the  $j$ th car of the inbound train sequence is called the head of the block.

The following lemma states the necessary and sufficient conditions for the existence of  $(p, k)$ -extensions. The recovery is performed independently for every block, where unresolved breaks are successively fixed by raising the bitstring of the second car of the break and all cars following it up to the end of the block.

**Lemma 1.** Let  $T_1, \dots, T_\ell$  be a sequence of  $\ell$  inbound trains,  $B$  a feasible classification schedule,  $S$  a scenario, and  $p, k \geq 0$ . Then, there exists a  $(p, k)$ -extension of  $B$  that is feasible for  $S$  iff the number of unresolved breaks w.r.t.  $S$  does not exceed  $2^k - 1$  for any block of  $B$ .

**General Algorithm.** Applying the observations of the previous section, we introduce a generic algorithm for computing  $(p, k)$ -recovery robust train classification schedules. Basically, the algorithms successively grows the size of a block to its maximum size. The maximum size of a block is determined by two factors: First, a schedule  $B$  assigns at most  $2^p$  different bitstrings to the trailing part of cars in the same block, i.e., at most  $2^p - 1$  breaks can be resolved. Secondly, the number of unresolved breaks in a block is limited by  $2^k - 1$  potential breaks induced by one scenario. We formalize the second condition in the following way.

**Definition 4.** Let  $T_1, \dots, T_\ell$  be a sequence of  $\ell$  inbound trains with a total of  $n$  cars,  $\tau_1, \tau_2 \in \{1, \dots, n\}$  two cars, and  $k \geq 0$ . Given a set of scenarios  $\mathcal{S}$ , a set of breaks  $X' \subseteq X_{(\tau_1, \tau_2)}$  is called  $k$ -recoverable according to  $X_{(\tau_1, \tau_2)}$  if  $|X' \cap X^S| \leq 2^k - 1$  holds for all  $S \in \mathcal{S}$ .

**Algorithm 1.** *k*-recovery robust train classification

---

**Data:** number of cars  $n$ , set of original breaks  $X_{\text{org}}$ , set of scenarios  $\mathcal{S}$ ,  $k, p \geq 0$   
**Result:**  $k$ -recovery robust classification schedule  $B$

```

1 Put  $i = 0$ ,  $\tau_i = 1$ ,  $\tau_{\max} = 0$ ,  $X = X' = \emptyset$ 
2 while  $\tau_i \leq n$  do
3   while  $\tau_{\max} < \tau_i + 2^p + |X'|$  and  $\tau_i + 2^p + |X'| \leq n$  do
4     Set  $\tau_{\max} = \tau_i + 2^p + |X'|$ 
5     Set  $X = X_{(\tau_i, \tau_{\max})} \cap (\cup_{S \in \mathcal{S}} X^S)$ 
6     Compute a maximum  $k$ -recoverable set of breaks  $X' \subseteq X$ 
7   end
8   Set  $\tau_{\max} = \tau_{i+1} = \min(\tau_i + 2^p + |X'|, n+1)$ 
9   Compute subschedule of length  $p$  for  $\tau_i, \dots, \tau_{i+1} - 1$  feasible w.r.t.  $X_{(\tau_i, \tau_{i+1} - 1)} \setminus X'$ 
10  Set  $i = i+1$ 
11 end
12 Set  $h' = \lceil \log_2 i - 1 \rceil$ 
13 for  $j = 0, \dots, i - 1$  do
14   | Set  $b_{p+h'-1}^\tau \dots b_p^\tau = [j]_2$  for all  $\tau_j \leq \tau \leq \tau_{j+1} - 1$ 
15 end
16 return  $B$ 

```

---

Algorithm 1 determines the maximum size of a block by repeatedly solving the problem of finding a maximum  $k$ -recoverable break set and thus constructs an optimal  $(p, k)$ -recovery robust schedule.

**Theorem 1.** *For any  $p \geq 0$  and  $k \geq 0$ , Alg. 1 computes an optimal  $(p, k)$ -recovery robust train classification schedule.*

In Alg. 1 the step of computing a maximum  $k$ -recoverable break set in line 6 is not specified. One way of solving this problem is integer programming. As we will show in the following, there is in general no polynomial time algorithm to solve this problem unless  $\mathbf{P} = \mathbf{NP}$ .

**Computational Complexity.** In this section we assume w.l.o.g. that we are looking for a maximum  $k$ -recoverable break set for the cars  $1, \dots, n$ , i.e., let  $\mathcal{S}$  be a set of scenarios, find a maximum  $k$ -recoverable break set  $X'$  of  $X = \cup_{S \in \mathcal{S}} X^S$ . By a reduction from the independent set problem, the decision version of this problem is strongly **NP**-hard for  $k = 1$ . A different reduction from  $2^k$ SAT leads to the **NP**-completeness for any constant  $k \geq 2$ .

**Theorem 2.** *Let  $T_1, \dots, T_\ell$  be a sequence of  $\ell$  inbound trains,  $\mathcal{S}$  a set of scenarios, and  $K \geq 0$ . For any constant  $k \geq 1$ , it is strongly **NP**-complete to decide whether there exists a  $k$ -recoverable break set of size  $K$ .*

This theorem not only states that Alg. 1 will only run in polynomial time if  $\mathbf{P} = \mathbf{NP}$  but also enable us to prove the **NP**-completeness of the  $(p, k)$ -recovery robust classification problem.

**Corollary 1.** Let  $T_1, \dots, T_\ell$  be a sequence of  $\ell$  inbound trains,  $\mathcal{S}$  a set of scenarios,  $h, p \geq 0$ , and  $k \geq 1$  const. Deciding whether there is a feasible  $(p, k)$ -recovery robust classification schedule of length at most  $h$  is an **NP**-complete problem.

**Infeasible Initial Solutions.** In our model the first-stage solution is a feasible classification schedule for the original order of trains. A special case of this setting is to allow recovery even in case of no disturbance. In this case the original breaks can be modeled by a scenario  $S_{\text{org}}$  with  $X^{S_{\text{org}}} = X_{\text{org}}$  and no original breaks are there, i.e., we assume that the cars arrive in perfect order.

## 4 Limited Number of Delayed Trains

As mentioned before, providing strict robustness wastes a lot of potential to extreme scenarios that rarely occur. For this reason we introduce a simple yet general class of scenarios in this section.

**Scenario Model.** Given some parameter  $j$ , up to  $j$  trains are delayed each by an arbitrary amount: let  $\Theta = T_1, \dots, T_\ell$  be an inbound train sequence and  $\Theta^\sigma = T_{\sigma^{-1}(1)}, \dots, T_{\sigma^{-1}(\ell)}$  be an order of trains induced by some permutation  $\sigma: [\ell] \rightarrow [\ell]$ . Then, a sequence  $\bar{\Theta} = T_{\bar{\sigma}^{-1}(1)}, \dots, T_{\bar{\sigma}^{-1}(\ell)}$ , where  $\bar{\sigma}$  is some permutation, is called an  $(\alpha, k)$ -delayed sequence of  $\Theta^\sigma$  if  $\sigma(\alpha) < k$  and the following conditions hold:  $\bar{\sigma}(x) = \sigma(x)$  if  $\sigma(x) < \sigma(\alpha)$  or  $\sigma(x) > k$ ,  $\bar{\sigma}(x) = \sigma(x) - 1$  for  $\sigma(\alpha) < \sigma(x) < k$ , and  $\bar{\sigma}(\alpha) = k$ . Less formally, train  $T_\alpha$  is delayed from the  $\sigma(\alpha)$ th to the  $k$ th position. The set of scenarios  $\mathcal{S}_j$ ,  $0 \leq j \leq \ell$ , is now defined to contain a scenario  $S$  (inducing some sequence  $\Theta^S$ ) iff there is a sequence  $\Theta^0, \dots, \Theta^j$  of train sequences  $\Theta^i$  such that  $\Theta^0 = \Theta$ ,  $\Theta^i$  is an  $(\alpha_i, k_i)$ -delayed sequence of  $\Theta^{i-1}$  for all  $i = 1, \dots, j$ , and  $\Theta^j = \Theta^S$ . Every train  $T_{\alpha_i}$  will furthermore be called to be *delayed* by  $S$ .

**Dominating Set of Scenarios.** We will see in Thrm. 3 that our considerations can be restricted to the dominating subset  $\bar{\mathcal{S}}_j \subseteq \mathcal{S}_j$  of scenarios defined as follows: a scenario  $S$  is a member of  $\bar{\mathcal{S}}_j$  iff there is a sequence  $\Theta^0, \dots, \Theta^j$  of train sequences  $\Theta^i$  such that  $\Theta^0 = \Theta$ ,  $\Theta^i$  is an  $(\alpha_i, \ell)$ -delayed sequence of  $\Theta^{i-1}$  for all  $i = 1, \dots, j$ ,  $\alpha_i < \alpha_{i-1}$  for all  $i = 1, \dots, j$ , and  $\Theta^j = \Theta^S$ . In other words, if two trains are delayed by  $S \in \bar{\mathcal{S}}_j$ , they swap their relative order and arrive later than all punctual trains. Note that for uniquely defining a scenario  $S \in \bar{\mathcal{S}}_j$  it suffices to list the  $j$  delayed trains since the order and amount of their delay is determined by the definition of  $\bar{\mathcal{S}}_j$ .

**Theorem 3.** Given any  $p, k, j \geq 0$ , let  $B$  be a feasible  $(p, k)$ -recovery robust schedule for  $\bar{\mathcal{S}}_j$ . Then,  $B$  is a feasible  $(p, k)$ -recovery robust schedule for  $\mathcal{S}_j$ .

Any potential break  $(\tau, \tau+1)$  can only be induced by  $S$  if the train containing  $\tau$  is delayed, but also the converse implication holds for  $\bar{\mathcal{S}}_j$  as stated in the following lemma.

**Lemma 2.** Let  $T_1, \dots, T_\ell$  be a sequence of inbound trains and  $S \in \bar{\mathcal{S}}_j$  some scenario. For any potential break  $\beta = (\tau, \tau+1)$  with  $\tau \in T_x$ ,  $x \in \{1, \dots, \ell\}$ ,  $\beta \in X^S$  iff  $T_x$  is delayed by  $S$ .

**Algorithm 2.** Max.  $k$ -Recoverable Set of Breaks for  $\mathcal{S}_j$  with Unique Cars

---

**Input:** Parameters  $j, k \in \mathbb{N}$  and sets of induced breaks  $X_1, \dots, X_\ell$   
**Output:** Maximum recoverable set of breaks

```

1 Descendingly sort  $X_1, \dots, X_\ell$  such that  $|X_{i_1}| \geq |X_{i_2}| \geq \dots \geq |X_{i_\ell}|$ 
2 Put  $\alpha := \max\{i_t : |X_{i_t}| = |X_{i_1}|\}$ 
3 while  $\sum_{t=1}^j |X_{i_t}| \geq 2^k$  do
4   Remove an arbitrary break from  $X_{i_\alpha}$ 
5   Put  $\alpha := \max\{i_t : |X_{i_t}| = |X_{i_1}|\}$ 
6 end
7 return  $\bigcup_{i=1}^\ell X_i$ 
```

---

As an immediate consequence, the set of potential breaks  $X^S$  of any scenario  $S \in \bar{\mathcal{S}}_j$  can be partitioned into disjoint subsets w.r.t. the respective delayed train causing the break, a fact which is applied in the algorithm of the following section. We will call the set  $X_i := \{(\tau, \tau+1) | \tau \in T_i, \exists y > i : \tau+1 \in T_y\}$  the *set of breaks induced by train  $T_i$* .

**Maximum Recoverable Sets of Breaks.** For  $\bar{\mathcal{S}}_j$  a maximum recoverable set of breaks is computed with Alg. 2: we repeatedly resolve potential breaks of the train that induces the highest number of unresolved breaks until the worst case scenario does not exceed the recovery capability given through the parameter  $k$ . Correctness, optimality, and the running time of Alg. 2 are summarized in the following theorem.

**Theorem 4.** *Given a set of potential breaks  $X$  for some classification instance with inbound trains  $T_1, \dots, T_\ell$ , a maximum  $k$ -recoverable set of breaks  $X' \subseteq X$  w.r.t.  $\bar{\mathcal{S}}_j$  can be computed in polynomial time.*

As an immediate consequence of Thrm. 4, the problem of train classification can be solved in polynomial time by combining Alg. 2 into Alg. 1. The resulting algorithm is implemented in the following section and tested for a number of real-world classification instances.

**Experimental Evaluation.** For the evaluation of the algorithm just described, we took the five real-world instances used in [10], which unfortunately are the only real-world instances available to us. They correspond to five days of traffic in the Swiss classification yard Lausanne Triage, with volumes ranging from 310 to 486, numbers of inbound trains between 44 and 49, outbound trains between 24 and 27, and numbers of breaks between 24 and 28. In order to obtain unique types of cars, we converted all cars of the same type between two consecutive original breaks to distinct types ascending in the order the cars appear between the breaks. The algorithm was implemented in C++, compiled with GNU g++ 4.4, and run on an 1.8 GHz Intel Core Duo CPU with 2 GB main memory.

Essentially, through adjusting the parameters  $p$ ,  $k$ , and  $j$ , the algorithm allows flexibly trading off shortest schedules against the other extreme of strict robustness. Given some train classification instance, let  $h$  denote the length of

an optimal non-robust schedule and  $\bar{h}$  the length of an optimal strictly robust schedule. The values  $\underline{h}$  and  $\bar{h}$  present the lower and upper bounds for the length resulting from any combination of  $j$ ,  $k$ , and  $p$ . Yet, as explained in Sect. II (Related Work),  $\bar{h}$  may be exceeded by the geometric method, i.e. an optimal strictly robust schedule disregarding presorted inbound trains, and even longer schedules than this are obtained by triangular sorting.

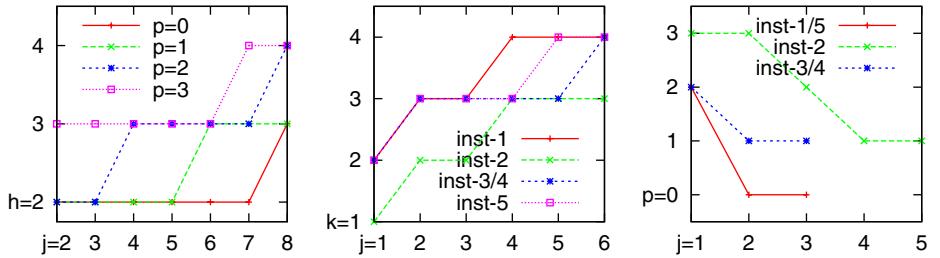
**Table 1.** Optimal length values for  $S_j$  with  $(p, k)$ -extensions for the five traffic instances: the values for the triangular and geometric method are given in the first and second column, resp.,  $\bar{h}$  and  $\underline{h}$  in the third and fourth column, resp. Omitted entries represent no meaningful choice of  $p$ .

	$k$	$t$	$g$	$0\ 0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$								
	$p$	$r$	$e$	$0\ 0$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$	$0$	$1$	$2$	$3$	$4$							
	$j$	$i$	$o$	$0\ 1$	$0$	$1$	$2$	$1$	$2$	$1$	$2$	$1$	$2$	$3$	$4$	$1$	$2$	$3$	$4$	$7$	$8$	$5$	$6$	$7$	$8$	$3$	$4$	$5$	$6$	$7$	$8$					
inst-1	11	6	5	2	2	4	3	4	3	5	4	5	4	5	2	3	2	3	3	4	2	3	4	5	4	5	2	3	3	3	4	4	2	2	2	
inst-2	8	5	5	3	3	4	3	4	3	5	5	3	3	3	3	3	3	3	3	4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
inst-3	8	5	5	2	2	4	2	4	3	4	3	5	5	2	3	2	2	2	3	2	3	3	4	4	2	3	2	2	2	3	3	3	3	3	3	3
inst-4	10	6	5	2	2	4	2	4	3	4	3	5	5	2	3	2	2	2	3	2	3	3	4	5	2	2	2	2	3	3	3	3	3	3	3	3
inst-5	9	6	5	2	2	4	3	4	3	5	5	2	3	2	3	3	3	2	3	3	4	3	3	4	4	2	2	2	2	3	3	3	3	3	3	3

Table II summarizes the computed length of an optimal recovery robust schedule according to the different parameters  $p$ ,  $k$  and  $j$ . As lower and upper bounds for those length inst-2 requires  $\underline{h}=3$ , while all other instances yield  $\underline{h}=2$  and  $\bar{h}=5$  for all instances. The geometric method requires  $h=6$  for three of the instances, and the triangular method even between eight (int-2 and inst-3) and eleven steps (inst-1), which shows that ignoring presorted input wastes a lot of potential for improvement.

If only small amounts of recovery action ( $k=1$ ) are allowed, for  $j=1$  the schedule length does not exceed  $\underline{h}$  for inst-1 and inst-5 with  $p=0$ , for inst-3 and inst-4 with  $p \leq 1$ , and for inst-2 even for  $p \leq 3$ , so yet for lowest degrees of recovery we obtain some robustness without increasing the length beyond that of an optimal non-robust schedule. Raising the degree of disturbance to  $j \geq 2$ , we still obtain a length  $h=4 < \bar{h}$  if the value of  $p$  is increased to  $p=1$  for inst-1, to  $p=2$  for inst-2, inst-4, and inst-5, and even to  $p=3$  for inst-3. These values are significantly smaller than those for the strictly robust methods of geometric or even triangular sorting.

The degree of robustness grows rapidly with increasing degrees of recovery, and for  $k=4$  with  $p \leq \bar{h}$ —except for inst-4 with  $p=2$ —we can allow any number of delayed trains and still achieve the length  $\underline{h}$  of an optimal robust schedule. Between these extremes, Tab. II shows how far the value of  $p$  can be raised for  $k=2$  and arbitrarily high amounts of delay  $j \geq 4$ : most instances allow  $p=1$  to obtain  $h=3$ , and  $h=4$  can be achieved even for  $p=4$  for three out of five instances. For  $k=3$ , Fig. 2 (left) summarizes the values of inst-1: a schedule of length 3 with a recovery action starting after the third sorting step suffices to cope with a delay of up to six trains and  $p=1$  allows  $h \leq 3$  even for any disturbance value  $j$ . Similarly, for a fixed value of  $p=2$ , Fig. 2 (center) shows the rapid growth of robustness: except for inst-2,  $k$  must be raised rather



**Fig. 2.** left: optimal schedule lengths  $h$  of inst-1 for  $k = 3$ ; center: highest possible values of  $k$  to achieve a length of  $\underline{h}$  for  $p = 2$ ; right: smallest possible values of  $p$  to achieve a length of  $\underline{h}$  for  $k = 2$ ;

quickly between  $j = 1$  and  $j = 3$  to achieve a length of  $\underline{h}$ , whereas the required value of  $k$  does not exceed four for higher disturbances  $j \geq 6$ . Conversely, Fig. 2 (right) fixes  $k = 2$  and shows the maximum value of  $p$  that allows a length of  $\underline{h}$ :  $j = 1$  still allows  $p = \underline{h}$  for all instances, but, except for inst-2, this length  $\underline{h}$  cannot be achieved for any choice of  $p$  for high amounts of delay  $j \geq 4$ . Hence, higher values of  $k$  contribute much more to the potential of recovery than low values of  $p$ . Summarizing, through adjusting the recovery parameters  $k$  and  $p$ , our algorithm presents a tool to flexibly trade off between fast classification and robust schedules and, even for high degrees of robustness, we achieve much shorter schedules than the triangular method currently applied in practice.

## 5 Conclusion

We have developed a practically applicable algorithm for deriving robust train classification schedules of minimum length. In contrast to [2], we regard multiple inbound and outbound trains, which allows integrating the most relevant disturbance in form of delayed trains. We have introduced the natural recovery action of  $(p, k)$ -extensions, for which we proved that the problem is **NP**-complete for every constant  $k \geq 1$ . Nevertheless, for the simple yet quite general set of scenarios  $S_j$ , we have shown our generic algorithm of Sect. 3 can be implemented in polynomial time by solving the subproblem of calculating a maximum recoverable set of breaks efficiently. The experimental study of Sect. 4 indicates that the resulting algorithm improves on the current classification practice as it yields shorter schedules and still allows high degrees of robustness. Its flexibility further allows balancing between strictly robust and optimal non-robust schedules and raises potential for increased traffic throughput in classification yards.

**Future Work.** Further practical restrictions, such as a limited number of classification tracks (see [15]), are desirable to be considered in the context of robustness. In a practical setting where the actual sorting is started (through the first pull-out) before all inbound trains have arrived, the online version of the problem becomes relevant. Moreover, the number of cars rolled in presents a

secondary objective, which can be additionally minimized for a minimum length. Finally, making the order of inbound trains part of the optimization yields different robust optimization problems.

## References

1. Cacchiani, V., Caprara, A., Galli, L., Kroon, L., Márton, G.: Recoverable robustness for railway rolling stock planning. In: ATMOS 2008, IBFI, Schloss Dagstuhl (2008)
2. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A.: Robust algorithms and price of robustness in shunting problems. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) ATMOS 2007. IBFI, pp. 175–190. Schloss Dagstuhl (2007)
3. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A., Schachtebeck, M., Schöbel, A.: Recoverable robustness in shunting and timetabling. In: Ahuja, R., Möhring, R., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 28–60. Springer, Heidelberg (2009)
4. Cicerone, S., Di Stefano, G., Schachtebeck, M., Schöbel, A.: Dynamic algorithms for recoverable robustness problems. In: ATMOS 2008. IBFI, Schloss Dagstuhl (2008)
5. Daganzo, C.F., Dowling, R.G., Hall, R.W.: Railroad classification yard throughput: The case of multistage triangular sorting. *Transp. Res.* 17A(2), 95–106 (1983)
6. Dahlhaus, E., Manne, F., Miller, M., Ryan, J.: Algorithms for combinatorial problems related to train marshalling. In: AWOCA 2000, pp. 7–16 (2000)
7. Flandorffer, H.: Vereinfachte Güterzugbildung. *ETR RT* 13, 114–118 (1953)
8. Gatto, M., Maue, J., Mihalak, M., Widmayer, P.: Shunting for dummies: An introductory algorithmic survey. In: Ahuja, R., Möhring, R., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 310–337. Springer, Heidelberg (2009)
9. Hansmann, R.S., Zimmermann, U.T.: Optimal sorting of rolling stock at hump yards. In: Mathematics - Key Technology for the Future: Joint Projects Between Universities and Industry, pp. 189–203. Springer, Heidelberg (2007)
10. Hauser, A., Maue, J.: Experimental evaluation of approximation and heuristic algorithms for sorting railway cars. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 154–165. Springer, Heidelberg (2010)
11. Jacob, R., Márton, P., Maue, J., Nunkesser, M.: Multistage methods for freight train classification. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) ATMOS 2007, pp. 158–174. IBFI, Schloss Dagstuhl (2007)
12. Jacob, R., Márton, P., Maue, J., Nunkesser, M.: Multistage methods for freight train classification. *Networks* (2010) (to appear, 2010)
13. Krell, K.: Grundgedanken des Simultanverfahrens. *ETR RT* 22, 15–23 (1962)
14. Liebchen, C., Lübbecke, M.E., Möhring, R.H., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. In: Ahuja, R., Möhring, R., Zaroliagis, C. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 1–27. Springer, Heidelberg (2009)
15. Márton, P., Maue, J., Nunkesser, M.: An improved classification procedure for the hump yard Lausanne Triage. In: Clausen, J., Di Stefano, G. (eds.) ATMOS 2009. IBFI, Schloss Dagstuhl (2009)
16. Pentinga, K.: Teaching simultaneous marshalling. *Railway Gaz.*, 590–593 (1959)
17. Siddiquee, M.W.: Investigation of sorting and train formation schemes for a railroad hump yard. In: Proc. of the 5th Int. Symposium on the Theory of Traffic Flow and Transportation, pp. 377–387 (1972)

# Cloning Voronoi Diagrams via Retroactive Data Structures

Matthew T. Dickerson<sup>1</sup>, David Eppstein<sup>2</sup>, and Michael T. Goodrich<sup>2</sup>

<sup>1</sup> Dept. of Math and Computer Sci., Middlebury College, Middlebury, Vermont, USA

<sup>2</sup> Computer Science Department, University of California, Irvine, USA

**Abstract.** We address the problem of replicating a Voronoi diagram  $V(S)$  of a planar point set  $S$  by making proximity queries:

1. the exact location of the nearest site(s) in  $S$
2. the distance to and label(s) of the nearest site(s) in  $S$
3. a unique label for every nearest site in  $S$ .

In addition to showing the limits of nearest-neighbor database security, our methods also provide one of the first natural algorithmic applications of retroactive data structures.

## 1 Introduction

In the *algorithmic data-cloning framework* [13], a *data querier*, Bob, is allowed certain types of queries to a data set  $S$  that belongs to a *data owner*, Alice. Once Alice has determined the kinds of queries that she will allow, she must correctly answer every valid query from Bob. The information security question, then, is to determine how many queries and how much processing time is needed for Bob to clone the entire data set. We define a *full cloning* of  $S$  to mean that Bob can answer any validly-formed query as accurately as Alice could. In an  $\epsilon$ -approximate cloning of  $S$ , Bob can answer any validly-formed query to within an accuracy of  $\epsilon > 0$ .

In this paper, we are interested in data sets consisting of a set  $S$  of  $n$  points in the plane, where  $n$  and the contents of  $S$  are initially unknown. We study the risks to  $S$  when Alice supports planar nearest-neighbor queries on  $S$ . We assume that all the sites in  $S$  are inside a known bounding box,  $B$ , which, without loss of generality, can be assumed to be a square with sides normalized to have length 1. Since planar nearest-neighbor queries define a Voronoi diagram in the plane (e.g., see [7]), we can view Bob's goal in this instance of the algorithmic data-cloning framework as that of trying to determine the Voronoi diagram of  $S$  inside the bounding box  $B$ . We consider three types of responses (in decreasing order of information content):

1. the exact location of the nearest site(s) to  $p$  in  $S$
2. the distance and label(s) of the nearest site(s) to  $p$  in  $S$
3. a unique label identifying each nearest site to  $p$  in  $S$ .

With all three cases, we want to know how difficult it is to compute the Voronoi diagram, or an approximation of it, from a set of queries.

**Related work.** Motivated by the problem of having a robot discover the shape of an object by touching it [6], there is considerable amount of related work in the computational geometry literature on discovering polygonal and polyhedral shapes from probing (e.g., see [12, 4, 9, 10, 14, 19, 18]). We refer the interested reader to the survey and book chapter by Skiena [17, 20], and simply mention that, with the notable exception of work by Dobkin *et al.* [9], this prior work is primarily directed at discovering obstacles in a two-dimensional environment using various kinds of contact probes. Translated into this context, their method results in a scheme that would use  $7n - 5$  queries to clone a Voronoi diagram, with a time overhead that is  $\Theta(n^2)$ .

In the framework of *retroactive data structures* [3, 8, 12], each update operation  $o$  to a data structure  $D$ , such as an insertion or deletion of an element, comes with a unique numerical value,  $t_o$ , specifying a *time value* at which the operation  $o$  is assumed to take place. The order in which operations are presented to the data structure is not assumed to be the same as the order of these time values. Just like update operations, query operations also come with time values; a query with time value  $t$  should return a correct response with respect to a data structure on which all operations with  $t_o < t$  have been performed. Thus, an update operation,  $o$ , having a time value,  $t_o$ , will affect any subsequent queries having time values greater than or equal to  $t_o$ . In a *partially retroactive* data structure the time for a query must be at least as large as the maximum  $t_o$  seen so far, whereas in a *fully retroactive* data structures there is no restriction on the time values for queries. Demaine *et al.* [8] show how a general comparison-based ordered dictionary (with successor and predecessor queries) of  $n$  elements (which may not belong to a total order, but which can always be compared when they are in  $D$  for the same time value) can be made fully retroactive in  $O(n \log n)$  space and  $O(\log^2 n)$  query time and amortized  $O(\log^2 n)$  update time in the pointer machine model. Blelloch [3] and Giora and Kaplan [12] improve these bounds, for numerical (totally ordered) items, showing how to achieve a fully retroactive ordered dictionary in  $O(n)$  space and  $O(\log n)$  query and update times in the RAM model. These latter results do not apply to the general comparison-based partially-ordered setting, however.

**Our Results.** Given a set  $S$  of  $n$  points in the plane, with an API that supports nearest-neighbor queries, we show how queries of Type 1 and Type 2 allow an exact cloning of the Voronoi diagram,  $V(S)$ , of  $S$  with  $O(n)$  queries and  $O(n \log^2 n)$  processing time. Our algorithms are based on non-trivial modifications of the sweep-line algorithm of Fortune [11] (see also [7]) so that it can construct a Voronoi diagram correctly in  $O(n \log^2 n)$  time while tolerating unbounded amounts of backtracking. We efficiently accommodate this unpredictable backtracking through the use of a fully retroactive data structure for general comparison-based dictionaries. In particular, our method is based on our showing that the dynamic point location method of Cheng and Janardan [5] can be adapted into a method for achieving a general comparison-based fully retroactive ordered dictionary with  $O(n)$  space,  $O(\log n)$  amortized update times, and  $O(\log^2 n)$  query times. We also provide lower bounds that show that, even with

an adaptation of the Dobkin *et al.* [9] approach optimized for nearest-neighbor searches, there is a sequence of query responses that requires  $\Omega(n^2)$  overhead for their approach applied to these types of *exact* queries. Nevertheless, we show that it is possible to clone  $V(S)$  using only  $3n$  queries. We prove that queries of Type 3 can never exactly clone  $V(S)$ , however, nor even determine with certainty the value of  $n = |S|$ . Nevertheless, we show that with  $n \log(\frac{1}{\epsilon})$  queries we can construct an  $\epsilon$ -approximate cloning of  $V(S)$  that will support approximate nearest neighbor queries guaranteeing a response that is a site within (additive)  $\epsilon > 0$  distance of the exact nearest neighbor of the query point.

## 2 A Fully-Retroactive Ordered Dictionary

In this section, we develop a fully retroactive ordered dictionary data structure using  $O(n)$  space,  $O(\log n)$  amortized update time, and  $O(\log^2 n)$  query time, based on a dynamic point location method of Cheng and Janardan [5]. The main idea is to construct an interval tree,  $B$ , over the intervals between the insertion and deletion times of each item in the dictionary, and to maintain  $B$  as a BB[ $\alpha$ ]-tree [16].

Each item  $x$  is stored at the unique node  $v$  in  $B$  such that  $x$ 's insertion time is associated with  $v$ 's left subtree and  $x$ 's deletion time is associated with  $v$ 's right subtree. We store  $x$  in two priority search trees [15],  $L(v)$  and  $R(v)$ , associated with node  $v$ . These two priority search trees are both ordered by the dictionary ordering of the items stored in them; all such items are active at the time value that separates  $v$ 's left and right subtrees, so they are all comparable to each other. The priority search trees differ, however, in how they prioritize their items. As with priority search trees more generally, each node in  $L(v)$  and  $R(v)$  stores two items, one that is used as a search key and another that has the minimum or maximum priority within its subtree. In  $L(v)$ , the insertion time of an item is used as a priority, and a node in  $L(v)$  stores the item that has the minimum insertion time among all items within the subtree of descendants of that node. In  $R(v)$ , the deletion time of an item is used as a priority, and a node in  $R(v)$  stores the item that has the maximum deletion time within its subtree.

An insertion of an item  $x$  in  $D$  is done by finding the appropriate node  $v$  of the interval tree and inserting  $x$  into  $L(v)$  and  $R(v)$ , and a deletion is likewise done by deletions in  $L(v)$  and  $R(v)$ . Updates that cause a major imbalance in the interval tree structure are processed by rebalancing, which implies, by the properties of BB[ $\alpha$ ]-trees [16], that updates run in  $O(\log n)$  amortized time.

Queries are done by searching the interval tree for the nodes with the property that the retroactive time specified as part of the query could be contained within one of the time intervals associated with that node. For each matching interval tree node  $v$ , we perform a search in either  $L(v)$  or  $R(v)$  depending on the relation between the query time and the time that separates the left and right children of  $v$ . The search method of Cheng and Janardan [5] allows us to find the successor of the query value, among the nodes stored in  $L(v)$  or  $R(v)$  with time intervals that contain the query time, in time  $O(\log n)$ . The result of the overall query is

then formulated by comparing the results found at each interval tree node and choosing the one that is closest to the query value. Thus, the query takes  $O(\log n)$  time to identify the interval tree nodes associated with the query time,  $O(\log^2 n)$  to query each of logarithmically many priority search trees, and  $O(\log n)$  time to combine the results, for a total of  $O(\log^2 n)$  time.

**Theorem 1.** *One can maintain a fully-retroactive general comparison-based dictionary on  $n$  elements, using  $O(n)$  space, so that updates run in  $O(\log n)$  amortized time and predecessor and successor queries run in  $O(\log^2 n)$  time.*

### 3 Exact Query Probes

We begin our study of Voronoi diagram cloning with the strongest sort of queries—Type 1. Given a query point  $p$ , a Type-1 query returns the site  $q$  in  $S$  nearest to  $p$ , that is, it returns the geometric location of  $q$ ,  $p$ 's nearest-neighbor in  $S$ . In the event that  $p$  has more than one nearest neighbors in  $S$ , all nearest neighbors are returned. We show that only  $O(n)$  queries and  $O(n \log^2 n)$  processing time is needed to completely clone  $V(S)$ —which, as implied, also means we explicitly have determined both  $S$  and  $n$ .

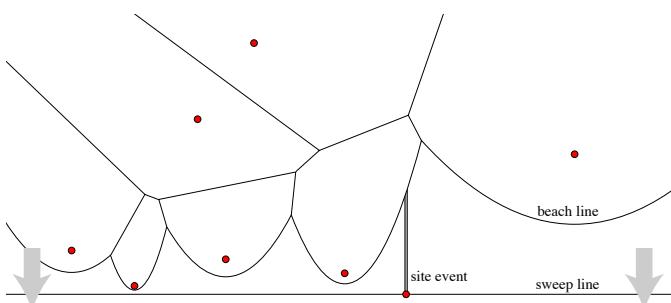
**Overview of Our Algorithm.** Our algorithm is adapted from the plane sweep Voronoi diagram algorithm of Fortune [11], with a significant modification to allow for unbounded and unpredictable amounts of backtracking. The fundamental difference is that the Fortune algorithm begins with the set of sites,  $S$ , completely known; in our case, the only thing we know at the start is a bounding box containing  $S$ . Using the formulation of de Berg *et al.* [7], Fortune's algorithm uses an event queue to controls a sweep line that moves in order of decreasing  $y$  coordinates, with a so-called “beach line”—an  $x$ -monotone curve made up of parabolic segments following above the sweep line. The plane above the beach line is partitioned into cells according to the final Voronoi diagram of  $S$ . There are two types of events, caused when the sweep line crosses point sites in  $S$  and Voronoi vertices in  $V(S)$ ; the latter points are determined as the algorithm progresses. In our version, we need to find both the sites and the Voronoi vertices as the plane sweep advances. And because not all sites are known in advance, we will need to verify *tentative* Voronoi vertex events as we sweep across them, at times backtracking our sweep line when our queries reveal new sites that invalidate tentative Voronoi vertices and introduce new events that are actually above our sweep line. We will show that each query discovers a feature in the Voronoi diagram, that the number of times we backtrack is bounded by the number of these features, and these facts imply that the number of queries and updates we perform in a retroactive dictionary used to implement our sweep-line algorithm is  $O(n)$ . In fact, we will prove that the number of probes is at most  $4n$ .

We begin with an overview of our algorithm. The algorithm begins by finding all the Voronoi regions and edges that intersect the top edge of the bounding box,  $B$ . If there are  $k$  such regions (and thus  $k - 1$  edges), this can be accomplished in  $O(n)$  time with  $2k - 1$  queries. This step initializes our event queue with  $k$  of the point sites in  $S$ .

The algorithm then proceeds much as the Fortune algorithm, but with the following two important changes. Whenever we reach a point site event for some site  $q \in S$  (i.e., when  $q$  is removed from the event queue), we do a nearest-neighbor query on the point of the beach line directly above  $q$ —that is, the point with the same  $x$ -coordinate as  $q$  and a  $y$  coordinate on the beach line that exists for the time value when the sweep line hits  $q$ . The position of this query point can be determined by using a retroactive dictionary queried with respect to the components of the beach line for the time value (in the plane sweep) associated with the point  $q$ . (See Fig. 1) Querying this point will either confirm a Voronoi edge known to be part of the final Voronoi diagram (in which case we proceed with the sweep) or it will discover a new site  $r$  in  $S$  (in which case the sweep line restores point  $q$  to the event queue and backtracks to  $r$ ).

The second type of event is a tentative Voronoi vertex event. We do a nearest-neighbor query at the point believed to be a Voronoi vertex, which either confirms the vertex and all of its adjacent Voronoi edges above it, or it discovers a new site in  $S$ . This new site must be above the sweep line: at the time that Fortune's algorithm processes a Voronoi vertex event its sweep line must be as far from the Voronoi vertex as the three sites generating the vertex, so undiscovered sites below the sweep line cannot be nearest neighbors to the tentative vertex. If the algorithm discovers a new site above the sweep line, then again we backtrack and process that new site. In either case the Voronoi vertex is removed from the event queue—either added to the Voronoi diagram being constructed as a validated vertex, or ignored as a false vertex.

**Correctness and Complexity.** Both the correctness and the analysis of this algorithm make use of the following important observation. Though the algorithm backtracks at certain “false” events—or tentative events that are proven false—it never completely removes any Voronoi components that have been confirmed by probes. Voronoi edges can only be added in two ways: the addition of a new site that creates one new edge, or the addition of a Voronoi vertex that terminates two edges and creates one new edge. In both cases, the edge is verified as an actual edge using a query before it is added to the Voronoi diagram being constructed, thus the diagram never contains edges that could later be falsified. (See Fig. 2)



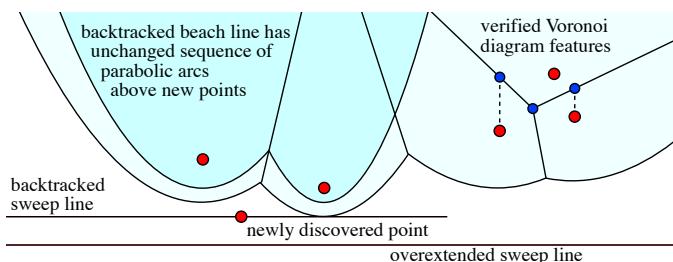
**Fig. 1.** Illustrating the sweep-line algorithm for constructing a Voronoi diagram

The insertion of a new site begins a new edge directly above it, where the parabola of the site being added to the tree  $T$ —a degenerate line-segment parabola at the instant it is added—intersects the existing parabola above it, thus replacing one leaf in the tree with three. But before this site is inserted with its edge, the edge is tested with a query into the existing Voronoi diagram. The other time an edge may be added is at a Voronoi vertex where two existing edges meet and a third new one is created. But all tentative Voronoi vertices are also verified by queries before they become circle events.

There are a few key observations that will lead to the analysis of the algorithm's run time and total number of queries. First, the sweep line will only backtrack when a new site in  $S$  is discovered, and so there are at most  $n$  backtracks. Second, every time we have a tentative Voronoi vertex that turns out to be unverified—that is, an event that turns out not to be part of the final diagram—we have also discovered a new site in  $S$ , and thus we have at most  $O(n)$  phony events that are processed. It follows that the run time of the algorithm is asymptotically equivalent to the original Fortune algorithm, modulo the time needed for our retroactive data structure queries. Furthermore, after our initialization stage, every nearest-neighbor query either finds a site, verifies a site by looking at the Voronoi edge above it, or verifies a Voronoi vertex. The initialization, as noted, requires  $2k - 1$  queries if there are  $k$  sites initially discovered. Queries discover  $n - k$  more sites, verify  $n$  sites, and find at most  $2n - 2 - k$  Voronoi vertices, for a total of  $4n - 3$  which is less than  $4n$  queries.

The algorithm requires the same run time as the original Fortune plane sweep except for the processing of the tentative Voronoi vertices that prove to be phony, and all the backtracking (which is implemented using our retroactive dictionary). As noted, there are  $O(n)$  of these backtracking steps, since these can only occur once for each previously undetected site in  $S$ . So the overall number of updates and queries in our sweep-line-with-backtracking algorithm is  $O(n)$ ; hence, the running time of our algorithm is  $O(n \log^2 n)$ . Thus, we have the following.

**Theorem 2.** *Given a set  $S$  of  $n$  points in  $\mathbf{R}^2$ , we can construct a copy of  $V(S)$  using at most  $4n$  Type-1 queries and  $O(n \log^2 n)$  time.*



**Fig. 2.** Backtracking the sweep-line

**An Alternate Algorithm Using More Time and Fewer Queries.** This second algorithm follows an incremental construction paradigm, based on the general approach of Dobkin *et al.* [9] for discovering a 3-dimensional convex polyhedron using finger probes. The alternative algorithm begins by querying each of the four corners of the bounding box. There are three cases to consider: these probes may discover one, two, or more than two sites in  $S$ . If we discover more than two sites, then we construct the Voronoi diagram of all 3 or 4 of the sites discovered by these four queries, but we mark each Voronoi vertex as tentative and put it into a queue. The algorithm then proceeds as follows until the queue is empty. Remove a tentative Voronoi vertex from the queue, and query it. If the query reveals that it is a Voronoi vertex—that is, it has the three expected nearest neighbors—then we confirm the vertex and continue. If it is not a Voronoi vertex, then it must be closer to some previously undiscovered site in  $S$ , that will be returned by the query. We add that site to our list of known sites and update the Voronoi diagram in worst case  $O(n)$  time using incremental insertion. When the queue is empty, we have a complete Voronoi diagram. Every probe except possibly one of the four corner probes discovered either a new site in  $S$  or confirmed a Voronoi vertex, and so the total number of queries is at most  $n + (2n - 5) + 1 < 3n$ . If the four corner queries discovered only two sites, then we compute the Voronoi edge that would be shared by these two sites if they were the only two sites in  $S$ , and we query both intersections of this edge with the bounding box. If we confirm both edges, then  $n = 2$  and we are done. We have used  $3n = 6$  queries. If at least one of these two additional queries discovers another site, then we have at least three known sites and we proceed as with the previous case. Every query except at most three of the initial queries either confirmed a Voronoi vertex or discovered a new point site, and so the total number of queries is at most  $n + (2n - 5) + 3 < 3n$ . All four corners will belong to the same Voronoi region if and only if there is only one site in the bounding box, in which case 4 queries was sufficient.

**Theorem 3.** *Given a set  $S$  of  $n$  points in  $\mathbf{R}^2$ , we can construct a copy of  $V(S)$  using at most  $3n + 1$  Type-1 queries and  $\Theta(n^2)$  time.*

*Proof.* We have already established the quadratic upper bound. For the sake of a lower bound, imagine that we have a set  $S'$  of  $n/2$  points on the bottom boundary of  $B$ , all within distance  $\delta$  of the point  $(0, 0)$ , for a small parameter  $\delta$  with  $0 < \delta \leq 1/2^n$ . These points, by themselves, construct a Voronoi diagram with parallel edges. Suppose further that there is a single point,  $p_0 = (\delta, 1 - \delta)$ , near the top boundary of  $B$ . The Voronoi region for  $p_0$  intersects the Voronoi region of every point in  $S'$ . The above algorithm therefore, after discovering the boundary points in  $S'$ , would next query a vertex on the Voronoi diagram  $V(S')$  of  $B$ , which will discover  $p_0$ . Next it will probe at vertex that is equi-distant to  $p_0$  and a point in  $S'$ . Suppose that this probe discovers a point  $p_1 = (\delta, 1/2)$ . This point is closer to every point in  $S'$  than  $p_0$ ; hence, updating the Voronoi diagram to go from  $V(S' \cup \{p_0\})$  to  $V(S' \cup \{p_0, p_1\})$  takes  $\Omega(n)$  time. Now, suppose querying a tentative vertex of  $V(S' \cup \{p_0, p_1\})$ , which will be equi-distant from  $p_1$  and a point in  $S'$ , discovers a point  $p_2 = (\delta, 1/2^2)$ . Again, updating the

Voronoi diagram takes  $\Omega(n)$  time. Suppose, therefore, that we continue in this way, with each newly-discovered point  $p_i = (\delta, 1/2^i)$  requiring that we spend  $\Omega(n)$  time to update the current Voronoi diagram. After discovering  $p_{n/2-1}$ , the  $n$ -th point in the set  $S = S' \cup \{p_0, p_1, \dots, p_{n/2-1}\}$ , we will have spent  $\Omega(n^2)$  time in total to discover the Voronoi diagram  $V(S)$ .  $\square$

## 4 Distance Query Probes

We next consider our Voronoi diagram cloning algorithm for the case when we use only distance query probes—probes that return the distance to and label of the nearest site(s). We begin by describing how we can find those sites whose Voronoi regions (and thus also edges) intersect the top boundary of the bounding box where the sweep-line begins. We will speak of a *probe circle* as the set of possible locations of a site returned by a probe  $p$ : it is the circle of radius  $d$  centered at  $p$ , where  $d$  is the distance returned to the nearest site.

**Initializing the Sweep Line.** We begin the initialization process by probing at the two top corners of the bounding box. If both probes return the same site  $p$ , then by convexity of Voronoi regions the entire top edge of the box belongs to the Voronoi region  $V(p)$ . Furthermore, both probes also return a distance  $d$  to the site  $p$ , and so  $p$  must fall on an intersection of two circles of radius  $d$  centered at the two corner probe locations. Since one of these intersections is above the bounding box, the remaining intersection gives the exact location of  $p$ .

Assume that the two corner probes  $p_l$  and  $p_r$  on the left and right respectively return different sites  $q_l$  and  $q_R$  respectively. We know the distance  $p_l$  to  $q_l$  and  $p_R$  to  $q_R$ , but we don't know the exact locations of the two sites, nor do we know if there are any other sites with regions intersecting the bounding box. The segment between these two probes is therefore not fully classified. We will describe a recursive procedure for classification.

Let  $L$  be an unclassified segment, with the probes  $p_l$  and  $p_r$  on the left and right sides of the segments respectively returning different sites  $q_l$  and  $q_r$ . First, we probe the midpoint  $p_m$  of  $L$ . The probe returns either one of the two known sites, or a new site  $q_m$ . If it returns a new site, then we divide  $L$  into two segments that are both unclassified, but which have classified endpoints  $q_l$ ,  $q_m$ , and  $q_r$ , and we recursively classify them.

Suppose query  $p_m$  returns one of the already known sites. If this probe returns  $q_r$ , then we can immediately compute the exact location of  $q_r$  from the two probes  $p_r$  and  $p_m$ , since only one of the intersections of the probe circles is inside the bounding box. We also have classified the segment  $L_r$  between probes  $p_r$  and  $p_m$  as being fully inside the Voronoi region of  $p_r$ . The other half of the segment,  $L_l$ , however, is not classified; we only know the Voronoi regions of its endpoints are the regions of  $q_r$  and  $q_l$ . Here it would be tempting to again probe the midpoint of the segment  $L_l$ ; however that could lead to an unbounded number of probes as we repeatedly divide the segment in half because the midpoint of the remaining unclassified segment  $L_l$  could still belong to  $q_r$  and so we would gain no new information about  $q_l$ . What we do instead is use the known location of  $q_r$ , which

must be outside the probe circle at  $p_l$ , to find a probe location  $p_{l2}$  close enough to  $p_l$  that it is guaranteed not to give us  $q_r$ . This is possible since the perpendicular bisector between  $q_r$  (which is a known point) and any point on the probe circle from  $p_l$ —which is the set of candidate locations for  $q_l$ —must fall between  $p_m$  and  $p_l$  on a finite segment computable in  $O(1)$  time, and thus anything between that range and  $p_l$  is closer to  $q_l$  than to  $q_r$ .

So this new probe  $p_{l2}$  returns either  $q_l$  or a new site. If it returns a new site, then we divide the unclassified segment into two unclassified segments and recursively classify them. If this new probe gives us  $q_l$  then we now know the exact location of  $q_l$  from two probes. From the exact locations of  $q_r$  and  $q_l$ , we can compute and probe where their Voronoi edge ought to cross the bounding box, either confirming that Voronoi edge—which means that the entire edge is now classified—or we discover a new site. If the probe gives us a new site, then again we divide the unclassified segment into two unclassified segments and recursively classify them.

**Lemma 1.** *The initialization stage for the sweep line requires  $O(k)$  time and  $3k - 1$  probes where  $k$  is the number of sites whose Voronoi regions intersect the top of the bounding box.*

*Proof.* Note that every probe either identifies a previously undiscovered site ( $k$  probes), provides a second probe with more information on an already discovered site enabling the exact location of this site to be computed ( $k$  probes), or confirms a Voronoi edge ( $k - 1$  probes for  $k$  regions). So the total number of probes in this section is  $3k - 1$  where  $k$  is the number of sites whose Voronoi regions intersect the top of the bounding box. Each probe is processed in  $O(1)$  time.  $\square$

**Processing the Sweep Line.** The previous subsection explains how to initialize the sweep line. The algorithm making use of distance-only queries now proceeds as with the exact query probe version of the previous section, except that a slightly different approach requiring more probes will be needed to process tentative site events.

As with the algorithm of the previous section, there are two types of tentative events: a tentative Voronoi vertex for three known sites, and a tentative Voronoi edge that falls directly above a known site and is determined by one other known site. Both of these events need to be verified by probe—that is, we need to determine if these events are actually real, or whether there is some other site closer to the events. In both cases, we use a probe  $p$  where the tentative Voronoi feature *should* be. If that problem returns the correct three or two site labels (at the correct distance), then the verification is complete, and we proceed as with the algorithm of the previous section.

However, these probes may discover a new site  $q$ ; in this case, they give only the distance to that site and not its actual location. We need two more probes that return the same site in order to discover its exact location—but these probes may instead return yet other new sites. We now describe how to choose the locations of these probes so that no work is wasted, and each probe either verifies

a Voronoi vertex, verifies a Voronoi edge above a known site, or is one of three probes that exactly locates a site.

Let  $p_1$  be a probe during the sweep line, that attempts to verify a Voronoi vertex or edge, and instead discovers a new site  $q_1$  that was not previously known. Let  $d = d(p_1, q_1)$  be the distance returned from probe  $p_1$  to its site  $q_1$ , and let  $e$  be the distance from  $p_1$  to the nearest previously known sites—that is, the two or three sites whose tentative Voronoi vertex or edge it was seeking to verify. Since probe  $p_1$  returned  $q_1$ , we know that  $d < e$ . Let  $p_2$  be any probe location such that  $d(p_1, p_2) < \frac{e-d}{2}$ . By the triangle inequality, we know  $d(p_2, q_1) < d + \frac{e-d}{2} = \frac{e+d}{2}$ , while  $d(p_2, r) > e - \frac{e-d}{2} = \frac{e+d}{2}$ , where  $r$  is any of the two or three previously known closest sites to  $p_1$ . It follows immediately that probe  $p_2$  cannot return any previously known site except  $q_1$  which was first discovered by probe  $p_1$ . We can choose any probe location meeting this restriction,  $d(p_1, p_2) < \frac{e-d}{2}$ , which is computable in  $O(1)$  time.

So there are two possibilities with probe  $p_2$ : either it returns site  $q_1$  again, or it returns a new site  $q_2$ . If  $p_2$  returns  $q_1$ , we now have two probes returning that site, and distances to that site, so its location is one of at most two intersections between the two probe circles. We can now probe either one of those two intersections, and from the result we determine the exact location of  $q_1$  because the probe either returns  $q_1$  at distance 0, or it returns some other site, or it returns  $q_1$  at a distance  $> 0$ .

If  $p_2$  returns a new site  $q_2$ , then we now have two sites that have been discovered, but whose exact locations are not known. We can discover the exact location using the recursive method of the previous subsection, treating the segment  $p_1p_2$  as an unclassified segment, probing its midpoint, and continuing. However, once we have received a site as the result of two probes, we still require a third probe to exactly locate it since both intersections of the first two probe circles might be inside the bounding box.

**Lemma 2.** *Processing the remaining events (after the initialization) for the sweep line requires at most  $6n - 3k - 5$  probes where  $k$  is the number of sites whose Voronoi regions intersect the top of the bounding box.*

*Proof.* There are  $n - k$  sites to be discovered,  $n$  sites that need to have an edge verified above them, and at most  $2n - 5$  Voronoi vertices in the Voronoi diagram of  $n$  sites. Every probe accomplishes one of five things: it verifies a Voronoi vertex ( $2n - 5$  probes), verifies a Voronoi edge directly above a site ( $n$  probes), or is one of exactly three probes used to discover and then exactly locate a new site ( $3(n - k)$  probes.) The total number of probes required is therefore at most  $6n - 3k - 5$ .  $\square$

**Theorem 4.** *Given a set  $S$  of  $n$  points in  $\mathbf{R}^2$ , we can construct a copy of  $V(S)$  using at most  $6n - 6$  Type-2 queries and  $O(n \log^2 n)$  time.*

## 5 Label-Only Query Probes

Using queries of the third type, it is impossible to *exactly* clone  $V(S)$  or even to determine with certainty the value of  $n = |S|$ . However even with this minimal query

information, we construct an approximate Voronoi diagram  $V(S')$  which, without *explicitly* storing the locations of the sites in  $S$ , will still support later arbitrary approximate proximity queries to  $V(S)$ . We now show that an approximate Voronoi diagram can be constructed to answer nearest-neighbor queries, with a probing process that uses  $O(N \log(1/\epsilon))$  queries and  $O(N(\log N + \log(1/\epsilon)))$  time, where  $N \leq n$  is the number of discovered sites in  $S$ . (Any two sites separated by at least  $\epsilon$  will be distinguished and discovered.) The main idea of the algorithm is to build an approximation to the Voronoi cell of each known site, using  $O(\log(1/\epsilon))$  queries per feature of the cell. This sequence of queries either finds a sufficiently accurate approximation for the location of that feature or discovers the existence of another site label. We begin by querying each corner of our bounding box to find the label of the site in whose region that corner belongs. For any side of the box whose corners are in different Voronoi regions, we do a binary search to find, within a distance of  $\epsilon^2$ , the edge of the Voronoi region for each different site. This may discover new Voronoi regions. For each new region discovered, we also do a binary search to discover its edges to within  $\epsilon^2$ . Each binary search requires  $O(\log \frac{1}{\epsilon})$  queries and time. The result is an ordered list of Voronoi edges crossing each side of the bounding box.

A second similar search a distance of  $2\epsilon$  from each side of the bounding box will find the same Voronoi edges—or will discover a new Voronoi region, indicating that the Voronoi edge has ended. For those Voronoi edges that have not ended within  $2\epsilon$ , we compute an approximation of the line containing the Voronoi edge—that is, the perpendicular bisector of the two sites whose labels we know. An argument using similar triangles shows that our approximation of this edge is accurate enough that we can determine to within a distance of  $< \epsilon$  where this edge crosses the far boundary. We do a doubling search of queries out along each discovered Voronoi edge, and then a binary search back once we have moved past the end of the edge, to find where it ends. Thus, three binary searches of  $O(\log(\frac{1}{\epsilon}))$  queries and time each suffice to discover complete approximations of each Voronoi edge intersecting the bounding box, including an approximate location of the Voronoi vertex terminating these edges. In the worst case, our approximation is within  $\epsilon$ . A constant number of queries in the vicinity of each Voronoi vertex will discover the other edge or edges coming out of the vertex. We repeat this process for each new Voronoi edge as it is discovered, until every Voronoi edge has both ends terminated at Voronoi vertices, at which time the approximate Voronoi diagram is complete.

**Theorem 5.** *Given a set  $S$  of  $n$  points in  $\mathbf{R}^2$ , we can construct a planar subdivision,  $V'$ , using  $O(N \log \frac{1}{\epsilon})$  Type-3 queries and  $O(N(\log N + \log \frac{1}{\epsilon}))$  time, where  $N < n$  is the number of discovered sites in  $S$ , such that any two sites separated by at least  $\epsilon$  will be distinguished and discovered and each point on the 1-dimensional skeleton of  $V$  is within distance  $\epsilon$  of a point on the 1-dimensional skeleton of the Voronoi diagram,  $V(S)$ , of  $S$ .*

## References

1. Alevizos, P.D., Boissonnat, J.-D., Yvinec, M.: Non-convex contour reconstruction. *J. Symbolic Comput.* 10, 225–252 (1990)
2. Aoki, Y., Imai, H., Imai, K., Rappaport, D.: Probing a set of hyperplanes by lines and related problems. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1993. LNCS, vol. 709, pp. 72–82. Springer, Heidelberg (1993)
3. Blelloch, G.E.: Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In: SODA 2008: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 894–903. Society for Industrial and Applied Mathematics, Philadelphia (2008)
4. Boissonnat, J.-D., Yvinec, M.: Probing a scene of non-convex polyhedra. *Algorithmica* 8, 321–342 (1992)
5. Cheng, S.W., Janardan, R.: New results on dynamic planar point location. *SIAM J. Comput.* 21, 972–999 (1992)
6. Cole, R., Yap, C.K.: Shape from probing. *J. Algorithms* 8(1), 19–38 (1987)
7. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Berlin (1997)
8. Demaine, E.D., Iacono, J., Langerman, S.: Retroactive data structures. *ACM Trans. Algorithms* 3(2), 13 (2007)
9. Dobkin, D.P., Edelsbrunner, H., Yap, C.K.: Probing convex polytopes. In: Proc. 18th Annu. ACM Sympos. Theory Comput., pp. 424–432 (1986)
10. Edelsbrunner, H., Skiena, S.S.: Probing convex polygons with  $x$ -rays. *SIAM J. Comput.* 17, 870–882 (1988)
11. Fortune, S.J.: A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2, 153–174 (1987)
12. Giora, Y., Kaplan, H.: Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Trans. Algorithms* 5(3), 1–51 (2009)
13. Goodrich, M.T.: The mastermind attack on genomic data. In: IEEE Symposium on Security and Privacy. IEEE Press, Los Alamitos (2009) (to appear)
14. Joseph, E., Skiena, S.S.: Model-based probing strategies for convex polygons. *Comput. Geom. Theory Appl.* 2, 209–221 (1992)
15. McCreight, E.M.: Priority search trees. *SIAM J. Comput.* 14(2), 257–276 (1985)
16. Mehlhorn, K.: Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. EATCS Monographs on Theoretical Computer Science, vol. 3. Springer, Heidelberg (1984)
17. Skiena, S.S.: Problems in geometric probing. *Algorithmica* 4, 599–605 (1989)
18. Skiena, S.S.: Probing convex polygons with half-planes. *J. Algorithms* 12, 359–374 (1991)
19. Skiena, S.S.: Interactive reconstruction via geometric probing. *Proc. IEEE* 80(9), 1364–1383 (1992)
20. Skiena, S.S.: Geometric reconstruction problems. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 26, pp. 481–490. CRC Press LLC, Boca Raton (1997)

# A Unified Approach to Approximate Proximity Searching

Sunil Arya<sup>1,\*</sup>, Guilherme D. da Fonseca<sup>2,\*\*</sup>, and David M. Mount<sup>3,\*\*\*</sup>

<sup>1</sup> Department of Computer Science,  
The Hong Kong University of Science and Technology, Hong Kong  
[arya@cs.ust.hk](mailto:arya@cs.ust.hk)

<sup>2</sup> Universidade Federal do Estado do Rio de Janeiro (Unirio), Brazil  
[fonseca@uniriotec.br](mailto:fonseca@uniriotec.br)

<sup>3</sup> Department of Computer Science and Institute for Advanced Computer Studies,  
University of Maryland, College Park  
[mount@cs.umd.edu](mailto:mount@cs.umd.edu)

**Abstract.** The inability to answer proximity queries efficiently for spaces of dimension  $d > 2$  has led to the study of approximation to proximity problems. Several techniques have been proposed to address different approximate proximity problems. In this paper, we present a new and unified approach to proximity searching, which provides efficient solutions for several problems: spherical range queries, idempotent spherical range queries, spherical emptiness queries, and nearest neighbor queries. In contrast to previous data structures, our approach is simple and easy to analyze, providing a clear picture of how to exploit the particular characteristics of each of these problems. As applications of our approach, we provide simple and practical data structures that match the best previous results up to logarithmic factors, as well as advanced data structures that improve over the best previous results for all aforementioned proximity problems.

## 1 Introduction

The term *proximity* refers informally to the quality of being close to some point or object. Proximity data structures arise from numerous applications in science and engineering because it is a fundamental fact that nearby objects tend to exert a greater influence and have greater relevance than more distant objects. The inability to answer proximity queries efficiently for spaces of dimension  $d > 2$  has motivated study of approximate solutions to proximity problems. In recent years, a number of different techniques have been proposed for solving these problems. In order to obtain the best performance, the technical elements

---

\* Research supported by the Research Grants Council of Hong Kong, China under project number 610106.

\*\* Research supported by CNPq grant PDJ-151194/2007-6, FAPERJ grant E-26/110.091/2010 and CNPq/FAPERJ grant E-26/110.552/2010.

\*\*\* Research supported by NSF grant CCR-0635099 and ONR grant N00014-08-1-1015.

of these solutions vary considerably, depending on the type of problem being solved and the properties of the underlying entities. Each variant involves its own particular construction and analysis. Consequently, it is difficult to obtain a clear understanding of the basic mechanisms underlying these approaches. In this paper, we present efficient solutions for several approximate proximity problems, all within a simple and unified framework.

Abstractly, a *proximity search problem* involves preprocessing a multidimensional point set into a data structure in order to efficiently answer queries based on distances. Consider a set  $P$  of  $n$  points in  $d$ -dimensional Euclidean space, for a constant  $d \geq 2$ . In problems involving aggregation, it is useful to associate each point  $p \in P$  with a weight,  $w(p)$ , which is assumed to be drawn from some commutative semigroup  $(\mathbf{S}, +)$ . For example, range queries involve computing the semigroup sum of points lying within some region. In such cases, properties of the semigroup may be exploited in order to obtain the most efficient solution. An important semigroup property is *idempotence*, which means that  $x + x = x$ , for all  $x \in \mathbf{S}$ . (As an example, consider the integers under maximum or minimum.)

In this paper we consider the following fundamental proximity problems. In each case, we are given a query point  $q$  and possibly a radius  $r$ . We present each problem in its exact form, but each has a natural approximation version, given an approximation parameter  $0 < \varepsilon < 1$ . Points within distance  $r(1 - \varepsilon)$  of  $q$  must be considered and only points within distance  $r(1 + \varepsilon)$  may be considered.

- *Spherical range queries*: Given a query point  $q$  and a radius  $r$ , determine the number of points of  $P$  (or more generally, the semigroup sum of the weights of the points) lying within distance  $r$  from  $q$ . If the semigroup is idempotent, this is called an *idempotent spherical range query*.
- *Spherical emptiness queries*: This can be viewed as a special case of an idempotent range query in which the goal is to determine whether any point of  $P$  lies within distance  $r$  of  $q$ .
- *Nearest neighbor queries*: Given a query point  $q$  determine the closest point of  $P$  to  $q$  and its associated distance  $r$ .

Spherical emptiness queries and nearest neighbor queries are strongly related. Given the nearest neighbor  $p$  of a query point  $q$ , we can determine whether the ball of radius  $r$  centered at  $q$  is empty by comparing  $r$  and  $\|pq\|$ . In the approximate version, we can answer nearest neighbor queries by computing a constant-factor approximation of the nearest neighbor and then performing a binary search with  $O(\log \frac{1}{\varepsilon})$  spherical emptiness queries [19]. In this paper, we will present methods for answering approximate spherical emptiness queries, and results for approximate nearest neighbor queries will follow as corollaries.

*Prior results.* Approximate nearest neighbor queries in spaces of fixed dimension have been widely studied. Data structures with  $O(n)$  storage space and query times no better than  $O(\log n + 1/\varepsilon^{d-1})$  have been proposed by several authors [8, 9, 11, 15]. In subsequent papers, it was shown that query times could be reduced, at the expense of greater space [10, 19, 13, 23]. Although different tradeoffs were achieved, in all cases the products of the  $\varepsilon$  terms in the storage and query times

are roughly  $O(1/\varepsilon^{d-1})$ . These space-time tradeoffs were improved in [3, 2, 6]. It was shown that  $O(\log n + 1/\varepsilon^{\frac{d-1}{2}})$  query time could be achieved with essentially  $O(n)$  storage, and generally it is possible to achieve space-time tradeoffs where the product of the  $\varepsilon$  terms in the storage and the *square* of the query time is roughly  $O(1/\varepsilon^{d-1})$ .

Early results on approximate range searching for general semigroups provided query times of  $O(\log n + 1/\varepsilon^{d-1})$  with  $O(n)$  space [7]. Space-time tradeoffs for idempotent spherical range searching were presented in [5]. It was shown there that, given a tradeoff parameter  $\gamma \in [1, 1/\varepsilon]$ , queries can be answered in time  $O(\log n + (\log \frac{1}{\varepsilon})/(\varepsilon\gamma)^{\frac{d-1}{2}})$  with  $O(n\gamma^d/\varepsilon)$  storage. The tradeoff was later extended to handle spherical range queries for arbitrary semigroups in  $O(\log(n\gamma) + 1/(\varepsilon\gamma)^{d-1})$  time with  $O(n\gamma^d \log \frac{1}{\varepsilon})$  storage [4]. An approach for arbitrary semigroups that is more similar in spirit to ours is presented in [1], but it is limited to query times in  $\Omega(\log n + 1/\varepsilon^{\frac{d-1}{2}})$ , and it does not benefit from the assumptions of idempotence or emptiness.

The best space-time tradeoffs for answering approximate nearest neighbor queries are based on *Approximate Voronoi Diagrams* (or *AVDs*) [19, 6]. While answering an approximate nearest neighbor query with an AVD consists of a simple point location in a quadtree, building and analyzing the AVDs require relatively sophisticated machinery, including WSPDs, sampling, BBD-trees, separation properties, bisector sensitivity, and spatial amortization. To extend AVDs to handle spherical range queries, several new tools were introduced [5, 4]. The cells of the AVD are divided into three different types, with different query-answering mechanisms and analyses for each.

*Our results.* We present a new and unified approach to proximity searching. We place all the aforementioned proximity queries within a unified framework, providing a clear picture of how to exploit the peculiarities of each problem. We do so without using most of the AVD machinery, thus obtaining data structures that are easy both to implement and analyze. As applications of our approach, we provide simple and practical data structures that match the best previous tradeoffs up to logarithmic factors, as well as advanced data structures that improve over the best previous results for all aforementioned proximity problems.

Our approach is based on a well-known data structure, the compressed quadtree (described in Section 2.1). To perform a search involving a query ball  $b$ , the search algorithm begins by computing a constant number of cells in the compressed quadtree that cover  $b$ . Each cell locally answers the query for the portion of the ball that lies within the cell. To achieve low storage, we divide the cells into two types:

- (i) Cells enclosing a large number of points store a data structure whose storage is not dependent on that number. We call this data structure an *insensitive module*. The insensitive module is generally a table where a query is answered by performing a single lookup.
- (ii) Cells with a small number of points store a data structure whose storage benefits from the low number of points. We call this data structure an *adaptive module*. The adaptive module can either be as simple as a list of points

where queries are answered by brute force or as complex as the most efficient data structures known for exact range searching.

Since our framework is modular, we can plug the appropriate building blocks to obtain different data structures. By plugging simple and practical data structures, we obtain bounds that match the best known bounds up to logarithmic factors. Alternatively, we can plug advanced exact data structures in order to obtain small improvements to the most efficient data structures known for all aforementioned proximity problems. For example, with  $\tilde{O}(n)$  storage, we can perform approximate nearest neighbor queries in  $\tilde{O}(1/\varepsilon^{\frac{d-1}{2}})$  time using only simple data structures, matching the best bound previously known [3], or  $\tilde{O}(1/\varepsilon^{\frac{d-3}{2} + \frac{2}{d+1}})$  time by using exact data structures for halfspace emptiness queries from [22]. (Throughout,  $\tilde{O}(x)$  stand for  $O(x \text{ polylog}(n, 1/\varepsilon))$ .) As another example, we can answer idempotent spherical range queries in polylogarithmic time with  $\tilde{O}(n/\varepsilon^{d+1})$  storage using only simple data structures, thus matching the best bound previously known [5], or with  $\tilde{O}(n/\varepsilon^d)$  storage using exact data structures for halfspace range searching from [21].

Next, we highlight the most efficient tradeoffs obtained using our approach. The tradeoffs are described as a function of the tradeoff parameter  $\gamma \in [1, 1/\varepsilon]$ . Although the improvements are not dramatic, they are significant because they show that our method, while being both simpler and more unified, also offers new insights into the computational complexities of these problems.

- For general spherical range searching with query time  $\tilde{O}(1/(\varepsilon\gamma)^{d-1})$ , we improve the best previous storage [4, 1] from  $\tilde{O}(n\gamma^d)$  to  $\tilde{O}(n\gamma^{d-1}(1 + \varepsilon\gamma^2))$ . This improves the storage by a factor of  $\tilde{O}(\gamma/(1 + \varepsilon\gamma^2))$  for the same query time.
- For idempotent spherical range searching with query time  $\tilde{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$ , we improve the best previous storage [5] from  $\tilde{O}(n\gamma^d/\varepsilon)$  to  $\tilde{O}(n\gamma^{d-\frac{1}{2}}/\sqrt{\varepsilon})$ . This improves the storage by a factor of  $\tilde{O}(\sqrt{\gamma/\varepsilon})$  for the same query time.
- For nearest neighbor searching, the best previous result [3, 2, 6] has query time  $\tilde{O}(1/(\varepsilon\gamma)^{\frac{d-1}{2}})$  with storage  $\tilde{O}(n\gamma^{d-1})$ . We improve this result to query time  $\tilde{O}(1/(\varepsilon\gamma)^{\frac{d-3}{2} + \frac{1}{d}})$  with storage  $\tilde{O}(n\gamma^{d-2})$ , for even  $d \geq 4$ . For odd  $d \geq 3$ , we obtain query time  $\tilde{O}(1/(\varepsilon\gamma)^{\frac{d-3}{2} + \frac{2}{d+1}})$  with storage  $\tilde{O}(n\gamma^{d-2 + \frac{2}{d+1}})$ .

The insensitive modules are of special interest because they work in the absolute error model [14], providing more efficient tradeoffs for approximate spherical range searching in this model. The insensitive general module can be used to reduce the  $\varepsilon$ -dependency in algorithms for approximating the smallest  $k$ -enclosing disk [20] and the unit disk enclosing the most points [17].

## 2 Framework

In this section, we present our new approach as a general framework and show how to analyze its complexity. First, we review preliminary results on compressed

quadtrees. Second, as concrete illustration, we present a simple data structure for approximate spherical emptiness. Finally, we generalize this data structure to an abstract framework for approximate proximity searching.

## 2.1 Quadtrees

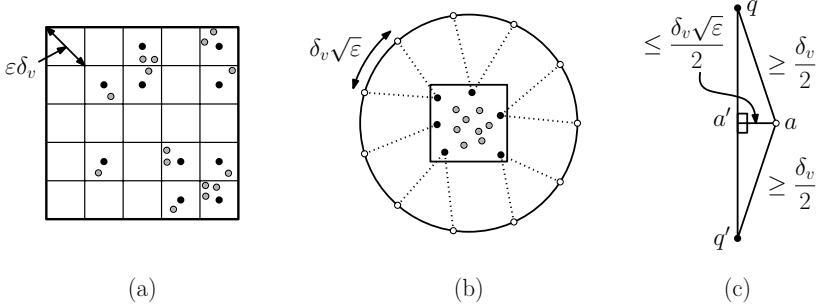
A *quadtree* is a hierarchical decomposition of the data points induced by a hierarchical partition of the space into  $d$ -dimensional hypercubes. The root of the quadtree corresponds to the whole set of data points. An internal node has  $2^d$  children corresponding to the sets of points in the disjoint subdivisions of the parent hypercube. A leaf is a node which contains a single data point. A *quadtree box* is defined recursively as the original bounding hypercube or the hypercubes obtained by evenly dividing a quadtree box.

A *compressed quadtree*, is obtained by replacing all maximal chains of nodes that have a single non-empty child by a single node associated with the coordinates of the smallest quadtree box containing the data points. The size of a compressed quadtree is  $O(n)$  and there are many different ways to build a compressed quadtree with  $n$  points in  $O(n \log n)$  time [8, 18, 16]. Even though the height of the tree can be as much as  $\Theta(n)$ , we can efficiently search a compressed quadtree by using an auxiliary structure which can be a simple hierarchy of separators [18], a skip-quadtree [16], or a BBD-tree [8]. An important type of query that these auxiliary structures answer in  $O(\log n)$  time is called a cell query [18]. Let  $T$  be a compressed quadtree for the set of points  $P$ . Given a query quadtree box  $Q$ , a *cell query* consists of finding the unique cell  $Q'$  in  $T$  such that  $P \cap Q = P \cap Q'$ , if it exists. The quadtree box  $Q'$  exists if  $P \cap Q \neq \emptyset$  and  $Q'$  is unique because  $T$  is compressed.

Let  $v$  be a vertex in a compressed quadtree associated with a quadtree box  $\square_v$  of diameter  $\delta_v$ . Consider a grid with cells of diameter  $\varepsilon\delta_v$  subdividing  $\square_v$ . Let  $c_v$  denote the number of non-empty grid cells (that is, those containing a point of  $P$ ). Since there are  $O(n)$  nodes in  $T$  and  $c_v \leq (\frac{1}{\varepsilon})^d$  by a packing argument, it follows that  $\sum_{v \in T} c_v = O(n(\frac{1}{\varepsilon})^d)$ . The following technical lemma, which will be useful in analyzing the storage requirements of our data structures, shows that the sum is significantly smaller.

**Lemma 1.** *For any compressed quadtree  $T$  with  $n$  points,  $\sum_v c_v = O(n \log 1/\varepsilon)$ . Consequently, the number of nodes  $v$  with  $c_v > \alpha$  is  $O(n(\log 1/\varepsilon)/\alpha)$ .*

*Proof.* The proof proceeds by a charging argument, where each of the  $O(n)$  quadtree nodes receives up to  $O(\log \frac{1}{\varepsilon})$  charges from the ancestors of the node. Assume without loss of generality that  $\varepsilon$  is a power of  $1/2$  and consider an internal node  $v$ . Let  $S_1$  be the set of quadtree nodes corresponding, by cell queries, to the non-empty grid cells of diameter  $\varepsilon\delta_v$  subdividing  $\square_v$ . A node in  $S_1$  may have arbitrarily small diameter because of compression, but the parent of a node in  $S_1$  has diameter at least  $\varepsilon\delta_v$ . Let  $S_2$  be the set of parents of the nodes in  $S_1$ . We have  $|S_2| \geq |S_1|/2^d = c_v/2^d$ . Node  $v$  assigns  $c_v/|S_2| = O(1)$  charges to each cell in  $S_2$ , so the sum of charges over all nodes is equal to  $\sum_v c_v$ . Since a node  $v$  only receives charges from ancestors of diameter at most  $\delta_v/\varepsilon$ , each node receives at most  $O(\log \frac{1}{\varepsilon})$  charges.  $\square$



**Fig. 1.** Spherical emptiness cells: (a) for the case  $c_v \leq 1/\varepsilon^{(d-1)/2}$  and (b) for the case  $c_v > 1/\varepsilon^{(d-1)/2}$ . Part (c) illustrates the correctness proof when  $c_v > 1/\varepsilon^{(d-1)/2}$ .

## 2.2 A Simple Data Structure for Approximate Spherical Emptiness

To illustrate our new approach, we demonstrate how to use it to answer approximate spherical emptiness queries. Given a query ball  $b$  of radius  $r$  and  $\varepsilon > 0$ , the algorithm must return “yes” if the expanded ball of radius  $r(1 + \varepsilon)$  is empty and “no” if the contracted ball of radius  $r/(1 + \varepsilon)$  is nonempty. The basis of our data structure is a compressed quadtree, together with any data structure that can answer cell queries in time  $O(\log n)$ . Each leaf node in the compressed quadtree simply stores the single point (if any) contained in it. Each internal node  $v$  will store a set  $S_v$  of  $O(\min(c_v, 1/\varepsilon^{(d-1)/2}))$  points. We shall see that the approximate emptiness of a ball  $b$  of radius  $r \geq 2\delta_v$  with respect to the points of  $P \cap \square_v$  can be reduced to a simple brute-force test of whether  $S_v \cap b = \emptyset$ . We consider two cases, depending on the size of  $c_v$ .

If  $c_v \leq 1/\varepsilon^{(d-1)/2}$ , consider a grid with cells of diameter  $\varepsilon\delta_v$  subdividing  $\square_v$ , and set  $S_v$  to be the set of at most  $c_v$  center points of the nonempty grid cells (see Figure 1(a)). Since the set  $S_v$  is obtained by moving the points in  $\square_v$  by at most  $\varepsilon\delta_v$ , and the query ball has radius  $r \geq 2\delta_v$ , testing the approximate emptiness of  $b$  with respect to  $P \cap \square_v$  reduces to testing whether  $S_v \cap b = \emptyset$ .

On the other hand, if  $c_v > 1/\varepsilon^{(d-1)/2}$ , we use a coresnet construction like the one in [12]. Consider ball of radius  $\delta_v$  centered at  $\square_v$ 's center, and let  $A$  be a  $\beta$ -dense set of points on the boundary of this ball, for  $\beta = \delta_v\sqrt{\varepsilon}/2$ . (That is, given any point  $a'$  on the boundary of the ball, there is a point  $a \in A$  such that  $\|aa'\| \leq \delta_v\sqrt{\varepsilon}/2$ .) By a simple packing argument, we may assume that  $|A| = O(1/\varepsilon^{(d-1)/2})$ . For each  $a \in A$ , we compute its nearest neighbor among  $P \cap \square_v$ . Let  $S_v$  be the resulting set of nearest neighbors (see Figure 1(b)). Next, we show that this set is sufficient to approximately answer the query.

Since the case when the query ball completely contains  $\square_v$  is trivial, we may assume that the center of the query ball (of radius  $r \geq 2\delta_v$ ) is at distance at least  $\delta_v$  from any point in  $\square_v$ . For a given query point  $q$  and its nearest neighbor  $q'$ , there exists  $a \in A$  such that  $\|qa\| + \|aq'\| \leq \|qq'\|(1 + \varepsilon)$ . This follows by a simple application of the Pythagorean Theorem on the pair of right triangles defined by the obtuse triangle  $\triangle qaq'$  (see Figure 1(c)). Letting

$q_a$  denote the nearest neighbor of  $a$  among the points inside  $\square_v$ , it follows that  $\|qq_a\| \leq \|qa\| + \|aq_a\| \leq \|qa\| + \|aq'\| \leq (1 + \varepsilon)\|qq'\|$ .

To answer a query for a ball  $b$  of radius  $r$ , we start by locating a set  $V$  of  $O(1)$  quadtree boxes of diameter at most  $r/2$  that disjointly cover  $b \cap P$ . This task can be performed in  $O(\log n)$  time by determining the intersection of  $b$  with a grid (of side length  $2^{\lfloor \log_2(r/2\sqrt{d}) \rfloor}$ ) and then performing the corresponding cell queries. As mentioned above, we answer the query by a simple brute-force test that, for each  $v \in V$ ,  $S_v \cap b = \emptyset$ . Since we consider only a constant number of cells, each storing  $O(1/\varepsilon^{(d-1)/2})$  points, the total query time is  $O(\log n + 1/\varepsilon^{(d-1)/2})$ .

To analyze the storage, note that, by Lemma 1, the number of nodes with  $c_v > 1/\varepsilon^{(d-1)/2}$  is  $O(n\varepsilon^{(d-1)/2} \log \frac{1}{\varepsilon})$ . Since the storage for each node is  $O(1/\varepsilon^{(d-1)/2})$ , the total storage for the nodes with  $c_v > 1/\varepsilon^{(d-1)/2}$  is  $O(n \log \frac{1}{\varepsilon})$ . The storage for each node with  $c_v \leq 1/\varepsilon^{(d-1)/2}$  is  $O(c_v)$ . By Lemma 2, the sum of  $c_v$  for all nodes in the quadtree is  $O(n \log \frac{1}{\varepsilon})$ , hence the total storage is  $O(n \log \frac{1}{\varepsilon})$ .

**Lemma 2.** *Given a set of  $n$  points in  $d$ -dimensional space, there exists a data structure of space  $O(n \log \frac{1}{\varepsilon})$  that can answer approximate spherical emptiness queries in time  $O(\log n + 1/\varepsilon^{(d-1)/2})$ , and approximate nearest neighbor queries in time  $O(\log n + (\log \frac{1}{\varepsilon})/\varepsilon^{(d-1)/2})$ .*

### 2.3 Abstract Framework

We now introduce our framework in an abstract setting. The basis of the data structure is a compressed quadtree equipped with an auxiliary structure to answer cell queries. Each internal node  $v$  in the compressed quadtree stores a data structure (called a module) that answers approximate spherical queries for a ball of radius  $r \geq 2\delta_v$  with respect to the points that are inside  $\square_v$ . If  $v$  is a leaf node, then we simply store the single point contained in  $\square_v$ .

Define a *module* to be a data structure that takes two inputs: a box  $\square$  and a set of points  $X$  contained in this box. A module answers  $\varepsilon$ -approximate spherical queries with respect to  $X$  for balls of radius  $2\delta$ , where  $\delta$  is the size of  $\square$ . Let  $k = |X|$ , and let  $s(k)$  denote the module's storage bound. We distinguish between two types of modules. A module is *insensitive* if its storage  $s(k)$  depends only on  $\varepsilon$ , regardless of the number of points in  $\square$ . For the remaining modules, called *adaptive*,  $f(k) = s(k)/k$  is assumed to be a nondecreasing function of  $k$ . For a threshold parameter  $\alpha$  to be specified, if  $c_v > \alpha$ , then node  $v$  stores an insensitive module for the box  $\square_v$  and the set of points  $P \cap \square_v$ . Otherwise,  $v$  stores an adaptive module for the box  $\square_v$  and the set of center points of the grid cells defining  $c_v$ .

To answer a query for a ball of radius  $r$ , we start by identifying a set  $V$  of  $O(1)$  quadtree boxes of diameter at most  $r/2$  that disjointly cover the points in the query ball, in  $O(\log n)$  time. Then, we answer the query inside each of these cells and combine the results. For the sake of approximate nearest neighbor queries, it is useful to observe that, if we are performing multiple queries with consecutive spheres whose radii vary within a constant factor of each other, this  $O(\log n)$  term is incurred only for the first query.

Let  $t$  denote the query time for the module with the largest query time used. Since we are performing a constant number of queries among the quadtree cells after  $O(\log n)$  time to perform cell queries, our data structure has query time  $O(\log n + t)$ . We generally choose the modules in a manner to make the query time equal to  $t$  for all modules, unless this is not an option in the tradeoff.

Let  $S$  denote the storage for each node that uses the insensitive module. Note that this is the same for all nodes, regardless of the number of points in the node. Since the insensitive module is used only for nodes  $v$  with  $c_v > \alpha$ , by Lemma 11, the total storage for the insensitive-module nodes is  $O\left(\frac{nS}{\alpha} \log \frac{1}{\varepsilon}\right)$ .

Let  $s(c_v)$  denote the storage for the adaptive module with query time  $t$  storing the  $c_v$  points in  $\square_v$ . Recall that  $c_v \leq \alpha$  for all nodes  $v$  where the adaptive module is used. Since  $f(c_v) = s(c_v)/c_v$  is monotonically increasing in  $c_v$  for an adaptive module, we have  $s(c_v)/c_v \leq s(\alpha)/\alpha$ . Thus, the total storage for all the adaptive module nodes is  $\sum_{v \in T, c_v \leq \alpha} s(c_v) = \sum_{v \in T} \frac{s(c_v)}{c_v} \cdot c_v \leq \frac{s(\alpha)}{\alpha} \sum_{v \in T} c_v$ , which by Lemma 11 is  $O\left(\frac{ns(\alpha)}{\alpha} \log \frac{1}{\varepsilon}\right)$ . In summary, we have:

**Theorem 1.** *For any threshold parameter  $\alpha$ , given an insensitive module with query time  $t$  and storage  $S$ , and an adaptive module with query time  $t$  and storage  $s(\alpha)$  for  $\alpha$  points, we can build a spherical range searching data structure (for a semigroup compatible with the modules) with  $O(\log n + t)$  query time and storage*

$$O\left(\frac{n(S + s(\alpha))}{\alpha} \log \frac{1}{\varepsilon}\right).$$

### 3 Modules

In this section, we consider the design of efficient modules to be used in our framework. Recall that a module is a data structure for the following simple range searching problem: Preprocess a set of  $n$  points, inside a box of diameter  $\delta$ , in order to efficiently answer approximate spherical queries where the radius of the query ball is at least  $2\delta$ . We remark that, when addressing the design of modules,  $n$  refers to the number of points stored just in the module.

We consider three cases, general spherical range queries, idempotent queries and emptiness queries. We present each of these cases in the next three sections.

#### 3.1 General Spherical Range Queries

In the most general version, we cannot assume any properties about the commutative semigroup. The modules designed for the general version can also be used for the idempotent or emptiness versions. The simplest (and surprisingly useful) adaptive module has both  $O(n)$  storage and query time. We call this a *brute force module*, and it consists of a list of the  $n$  points where queries are answered in  $O(n)$  time by inspecting each point individually.

A much more sophisticated adaptive module consists of reducing spherical range searching to halfspace range searching by lifting the points onto a  $(d+1)$ -dimensional paraboloid and then using Matoušek's exact range searching data

structure [21]. Storage is  $m \in [n, n^{d+1}/\log^{d+1} n]$ , preprocessing is  $O(n^{1+\beta} + m \log^\beta n)$  for arbitrarily small  $\beta$ , and query time is  $O(n/m^{1/(d+1)})$ . We call this an *exact general module*. Next, we describe an insensitive module.

Let  $x_1, \dots, x_d$  denote the orthogonal axes. We call the  $x_d$  axis *vertical*, the hyperplane determined by the remaining axes *horizontal*, and use standard terms such as *top* with respect to these directions. Without loss of generality, we assume that the box of diameter  $\delta$  intersects only a portion of the query ball boundary that has normal vectors within an angle at most  $\pi/4$  of the vertical axis. Separate data structures can be defined for a constant number of rotated sets of points.

Consider a  $d - 1$  dimensional grid subdividing the horizontal hyperplane into cells of diameter  $\rho \leq \delta$ , for a parameter  $\rho$  to be defined later. Each of these cells induces a prism inside the bounding hypercube in the vertical direction, which we call a *column*. The column has height  $O(\delta)$  and size  $O(\rho)$  in the horizontal directions. Next, we define a data structure to answer approximate spherical range queries, in time  $O(1)$ , corresponding to the portion of the query ball that is contained in a column. The number of columns in the box, which we will denote by  $C$ , is  $O((\delta/\rho)^{d-1})$ .

For simplicity, we scale down the space by  $\rho$ , making the horizontal diameter of the column equal to 1 and the height  $\delta' = \Theta(\delta/\rho)$ . We describe a data structure with absolute approximation error  $\phi$ , irrespective of the radius of the query ball (as in the absolute error model [14]).

Let  $B$  be a ball of radius  $r \geq 2$  centered on the vertical axis and tangent to the horizontal hyperplane  $x_d = 0$ . We define  $f(r) = r - \sqrt{r^2 - 1}$  as the length of a vertical segment connecting  $B$  to a point at distance 1 from the origin in the horizontal hyperplane  $x_d = 0$ . Consider two balls of radius  $r$  and  $r'$  that are tangent to each other at a point within the column. If  $|f(r') - f(r)| \leq \phi$ , then it is easy to see that these balls approximate each other inside the column, in the sense that any point on intersection of the column with the boundary of one ball is within (vertical) distance of  $\phi$  of a point on the boundary of the other ball. Let  $R$  be the set of radii  $r \leq 1/\phi$  such that  $f(r)$  is a multiple of  $\phi$ .

Next, we analyze  $|R|$ . It is easy to show that  $f(r) \leq 1/r$  (by setting  $r = \sec x$  with  $0 \leq x < \pi/2$ , and applying standard trigonometric identities). Since  $r \geq 2\delta'$ , we have  $f(r) \leq 1/\delta'$ , and therefore  $|R| = O(1 + 1/(\delta'\phi))$ .

Let  $G$  be a set of balls of radius  $r'$ , for each  $r' \in R$ , centers with vertical coordinates in increments of  $\phi$ , and centers with horizontal coordinates defined by the vertices of a horizontal grid with cells of diameter  $\phi r'$ . Consider that  $G$  only has balls such that the normal vectors inside the column are within an angle at most  $\pi/4$  of the vertical axis, since other balls can be handled using a suitable rotation. We always have a ball  $g \in G$  whose boundary is within vertical distance  $\phi$  of a tangent ball to the query ball inside the column. The set of balls  $G$  define a data structure with  $O(1)$  query time for query balls of radius  $r \geq 2\delta'$  inside a column of horizontal diameter 1. For each radius  $r'$ , the set  $G$  has  $H = O(1/\phi^{d-1})$  horizontal coordinates and, since we only store balls that have normal vectors within an angle at most  $\pi/4$  of the vertical axis,  $G$  has  $V = O(\delta'/\phi)$  vertical coordinates,  $|G| = |R|HV = O\left((1 + \frac{1}{\delta'\phi})\frac{\delta'}{\phi^d}\right)$ .

To change the scale back to columns of diameter  $\rho$  with error  $\varepsilon\delta$ , we set  $\delta' = \delta/\rho$  and  $\phi = \varepsilon\delta/\rho$ , from which we have  $|G| = O((1 + \frac{\rho^2}{\varepsilon\delta^2}) \frac{\rho^{d-1}}{\varepsilon^d \delta^{d-1}})$ .

To obtain the data structure for the whole bounding box, we build a set of balls for each column, cropping the balls to inside the corresponding columns. Let  $\gamma \in [1, 1/\varepsilon]$  be a parameter to control the space-time tradeoff. Setting  $\rho = \delta\varepsilon\gamma$ , and multiplying by the  $C = O((\delta/\rho)^{d-1})$  columns we have storage  $S = C|G| = O((1 + \varepsilon\gamma^2) \frac{1}{\varepsilon^d})$ .

Next, we describe an adaptive module based on the previous idea. Instead of having  $V = O(1/\varepsilon)$  balls in  $C = O(1/(\varepsilon\gamma)^{d-1})$  columns, we only consider balls that have a point on the boundary of the ball. Therefore, if the box has  $n$  points, the total storage is  $n|R|H = O(n(1 + \varepsilon\gamma^2)\gamma^{d-1})$ . The query time increases by an  $O(\log \frac{1}{\varepsilon})$  factor because we need to perform a binary search for each column.

**Lemma 3.** *There are adaptive modules for the general version with (i) query time and storage  $t = s(n) = O(n)$  (brute force module), (ii) query time  $t = O(n/m^{1/(d+1)})$  and storage  $m \in [n, n^{d+1}/\log^{d+1} n]$  (exact general module), and (iii) query time  $t = O((\log \frac{1}{\varepsilon})/(\varepsilon\gamma)^{d-1})$  and storage  $S = O(n(1 + \varepsilon\gamma^2)\gamma^{d-1})$ , for  $\gamma \in [1, 1/\varepsilon]$  (approximate general module). There is an insensitive module for the general version with query time  $t = O(1/(\varepsilon\gamma)^{d-1})$  and storage  $S = O((1 + \varepsilon\gamma^2)/\varepsilon^d)$ , for  $\gamma \in [1, 1/\varepsilon]$  (insensitive general module).*

*Applications.* The modules from Lemma 3 can be used together with Theorem 1 to obtain the following data structures for spherical range searching. If we use the insensitive general module and the brute force module, setting  $\alpha = 1/(\varepsilon\gamma)^{d-1}$ , then we obtain query time  $O(\log n + \alpha)$  with storage  $O(n\gamma^{d-1}(1 + \varepsilon\gamma^2)(\log \frac{1}{\varepsilon})/\varepsilon)$ . If we use only the approximate general module (by setting  $\alpha = n$ ), then we have query time  $O(\log n + (\log \frac{1}{\varepsilon})/(\varepsilon\gamma)^{d-1})$  with storage  $O(n\gamma^{d-1}(1 + \varepsilon\gamma^2)\log \frac{1}{\varepsilon})$ .

### 3.2 Idempotent Spherical Range Queries

It is not known how to exploit idempotence in exact range searching. Thus, we introduce no adaptive module for this case. The insensitive module for the idempotent case is strongly based on the insensitive general module, albeit much more efficient. In the idempotent version, the generators can overlap, therefore we do not need to crop the balls inside each column as in the insensitive module. A careful look at the insensitive module shows that the same balls are used by essentially all columns. Therefore, the storage for a single column is equal to the total storage. In the idempotent and emptiness versions, we set  $\rho = \delta\sqrt{\varepsilon\gamma}$  to obtain query time  $O(1/(\varepsilon\gamma)^{(d-1)/2})$ . We have storage  $S = |G| = O((\gamma/\varepsilon)^{(d+1)/2})$ .

**Lemma 4.** *There is an insensitive module for the idempotent version with query time  $t = O(1/(\varepsilon\gamma)^{(d-1)/2})$  and storage  $S = O((\gamma/\varepsilon)^{(d+1)/2})$ , for  $\gamma \in [1, 1/\varepsilon]$  (insensitive idempotent module).*

*Applications.* The insensitive idempotent module from Lemma 4 can be used together with Theorem 1 to obtain the following data structures for the idempotent

version. If we use brute force as the adaptive module and set  $\alpha = 1/(\varepsilon\gamma)^{(d-1)/2}$ , then we obtain query time  $O(\log n + \alpha)$  with storage  $O(n\gamma^d(\log \frac{1}{\varepsilon})/\varepsilon)$ . If we use the exact general module and set  $\alpha = 1/\varepsilon^{\frac{d}{2}}\gamma^{\frac{d-2}{2}}$ , then we obtain query time  $O(\log n + 1/(\varepsilon\gamma)^{\frac{d-1}{2}})$  with storage  $O(n\gamma^{d-\frac{1}{2}}(\log \frac{1}{\varepsilon})/\sqrt{\varepsilon})$ .

### 3.3 Spherical Emptiness Queries

Halfspace emptiness is a well studied problem. Exact data structures for halfspace emptiness are much more efficient than for general semigroups. By lifting the points onto a  $(d+1)$ -dimensional paraboloid and then using Matoušek's halfspace emptiness data structure [22] we obtain an adaptive module with storage  $m \in [n, n^{\lceil d/2 \rceil}]$  and query time  $\tilde{O}(n/m^{1/\lceil d/2 \rceil})$ . We call this an *exact emptiness module*.

To obtain an insensitive module for the emptiness version, we simply modify the insensitive idempotent module in order to store only the vertical coordinate of the center of the bottommost non-empty ball for each horizontal coordinate of the center. Therefore, storage is reduced by a factor of  $O(1/\varepsilon)$ . Storage becomes  $O(\gamma^{(d+1)/2}/\varepsilon^{(d-1)/2})$  with the same  $O(1/(\varepsilon\gamma)^{(d-1)/2})$  query time. Note that this insensitive module generalizes the one used in Section 2.2.

We can improve upon this by adapting constructs from [6] (such as the Concentric Ball Lemma) and using exact data structure for spherical emptiness. As a result, it is possible to obtain an insensitive module whose storage is  $O(\gamma^{(d+1)/2}/\varepsilon^{(d-1)/2})$  and whose query time is  $\tilde{O}(1/(\varepsilon\gamma)^{(d-3)/2+1/d})$  for even  $d$  and  $\tilde{O}(1/(\varepsilon\gamma)^{(d-3)/2+2/(d+1)})$  for odd  $d$ . The technical details are omitted.

**Lemma 5.** *There is an adaptive module for the emptiness version with query time  $t = \tilde{O}(n/m^{1/\lceil d/2 \rceil})$  and storage  $m \in [n, n^{\lceil d/2 \rceil}]$  (exact emptiness module).*

*There is an insensitive module for the emptiness version with query time  $t = O(1/(\varepsilon\gamma)^{(d-1)/2})$  and storage  $S = O(\gamma^{(d+1)/2}/\varepsilon^{(d-1)/2})$ , for  $\gamma \in [1, 1/\varepsilon]$  (insensitive emptiness module). There is also an insensitive module for the emptiness version with query time  $t = \tilde{O}(1/(\varepsilon\gamma)^{(d-3)/2+1/d})$  for even  $d \geq 4$  and  $t = \tilde{O}(1/(\varepsilon\gamma)^{(d-3)/2+2/(d+1)})$  for odd  $d \geq 3$  and storage  $S = \tilde{O}(\gamma^{(d+1)/2}/\varepsilon^{(d-1)/2})$  (advanced insensitive emptiness module).*

*Applications.* The modules from Lemma 5 can be used together with Theorem 1 to obtain the following data structures. If we use the brute force module, the insensitive emptiness module, and set  $\alpha = 1/(\varepsilon\gamma)^{(d-1)/2}$ , we have query time  $O(\log n + \alpha)$  with storage  $O(n\gamma^d \log \frac{1}{\varepsilon})$ . If we use the exact emptiness module and the advanced insensitive emptiness module, then we obtain query time  $\tilde{O}(1/(\varepsilon\gamma)^{(d-3)/2+1/d})$  with storage  $\tilde{O}(n\gamma^{d-2})$  for even  $d \geq 4$ . For odd  $d \geq 3$ , we obtain query time  $\tilde{O}(1/(\varepsilon\gamma)^{(d-3)/2+2/(d+1)})$  with storage  $\tilde{O}(n\gamma^{d-2+2/(d+1)})$ .

## References

1. Arya, S., da Fonseca, G.D., Mount, D.M.: Tradeoffs in approximate range searching made simpler. In: Proc. 21st SIBGRAPI, pp. 237–244 (2008)
2. Arya, S., Malamatos, T.: Linear-size approximate Voronoi diagrams. In: Proc. 13th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 147–155 (2002)

3. Arya, S., Malamatos, T., Mount, D.M.: Space-efficient approximate Voronoi diagrams. In: Proc. 34th Ann. ACM Symp. Theory of Comput. (STOC), pp. 721–730 (2002)
4. Arya, S., Malamatos, T., Mount, D.M.: Space-time tradeoffs for approximate spherical range counting. In: Proc. 16th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 535–544 (2005)
5. Arya, S., Malamatos, T., Mount, D.M.: On the importance of idempotence. In: Proc. 38th ACM Symp. on Theory of Comput. (STOC), pp. 564–573 (2006)
6. Arya, S., Malamatos, T., Mount, D.M.: Space-time tradeoffs for approximate nearest neighbor searching. J. ACM 57, 1–54 (2009)
7. Arya, S., Mount, D.M.: Approximate range searching. Comput. Geom. 17, 135–163 (2001)
8. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM 45(6), 891–923 (1998)
9. Bespamyatnikh, S.N.: Dynamic algorithms for approximate neighbor searching. In: Proc. 8th Canad. Conf. Comput. Geom. (CCCG), pp. 252–257 (1996)
10. Chan, T.M.: Approximate nearest neighbor queries revisited. Discrete Comput. Geom. 20, 359–373 (1998)
11. Chan, T.M.: Closest-point problems simplified on the ram. In: Proc. 13th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA), pp. 472–473 (2002)
12. Chan, T.M.: Faster core-set constructions and data-stream algorithms in fixed dimensions. Comput. Geom. 35(1), 20–35 (2006)
13. Clarkson, K.L.: An algorithm for approximate closest-point queries. In: Proc. 10th Annu. ACM Symp. Comput. Geom. (SoCG), pp. 160–164 (1994)
14. da Fonseca, G.D., Mount, D.M.: Approximate range searching: The absolute model. Comput. Geom. 43(4), 434–444 (2010)
15. Duncan, C.A., Goodrich, M.T., Kobourov, S.: Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. J. Algorithms 38, 303–333 (2001)
16. Eppstein, D., Goodrich, M.T., Sun, J.Z.: The skip quadtree: a simple dynamic data structure for multidimensional data. In: Proc. 21st ACM Symp. Comput. Geom. (SoCG), pp. 296–305 (2005)
17. Funke, S., Malamatos, T., Ray, R.: Finding planar regions in a terrain: in practice and with a guarantee. Internat. J. Comput. Geom. Appl. 15(4), 379–401 (2005)
18. Har-Peled, S.: Notes on geometric approximation algorithms,  
<http://valis.cs.uiuc.edu/~sariel/teach/notes/aprx/>
19. Har-Peled, S.: A replacement for Voronoi diagrams of near linear size. In: Proc. 42nd Ann. Symp. Foundations of Computer Science (FOCS), pp. 94–103 (2001)
20. Har-Peled, S., Mazumdar, S.: Fast algorithms for comput. the smallest  $k$ -enclosing circle. Algorithmica 41(3), 147–157 (2005)
21. Matoušek, J.: Range searching with efficient hierarchical cutting. Discrete Comput. Geom. 10, 157–182 (1993)
22. Matoušek, J., Schwarzkopf, O.: On ray shooting in convex polytopes. Discrete Comput. Geom. 10, 215–232 (1993)
23. Sabharwal, Y., Sen, S., Sharma, N.: Nearest neighbors search using point location in balls with applications to approximate Voronoi decompositions. J. Comput. Sys. Sci. 72, 955–977 (2006)

# Spatio-temporal Range Searching over Compressed Kinetic Sensor Data

Sorelle A. Friedler\* and David M. Mount\*\*

Dept. of Computer Science, University of Maryland,  
College Park, MD 20742, USA  
[{sorelle,mount}@cs.umd.edu](mailto:{sorelle,mount}@cs.umd.edu)  
<http://www.cs.umd.edu/~sorelle>,  
<http://www.cs.umd.edu/~mount>

**Abstract.** As sensor networks increase in size and number, efficient techniques are required to process the very large data sets that they generate. Frequently, sensor networks monitor objects in motion within their vicinity; the data associated with the movement of these objects are known as kinetic data. In an earlier paper we introduced an algorithm which, given a set of sensor observations, losslessly compresses this data to a size that is within a constant factor of the asymptotically optimal joint entropy bound. In this paper we present an efficient algorithm for answering spatio-temporal range queries. Our algorithm operates on a compressed representation of the data, without the need to decompress it. We analyze the efficiency of our algorithm in terms of a natural measure of information content, the joint entropy of the sensor outputs. We show that with space roughly equal to entropy, queries can be answered in time that is roughly logarithmic in entropy. In addition, we show experimentally that on real-world data our range searching structures use less space and have faster query times than the naive versions. These results represent the first solutions to range searching problems over compressed kinetic sensor data.

## 1 Introduction

Sensor networks and the data they collect have become increasingly prevalent. They are frequently employed to observe objects in motion and are used to record traffic data [14, 23], observe wildlife migration patterns [20, 25], and observe motion from many other settings [2]. In order to perform accurate statistical analyses of this data over arbitrary periods of time, the data must be faithfully recorded and stored. For example, a large sensor network observing a city's traffic patterns may generate gigabytes of data each day [14]. The vast

---

\* The work of Sorelle Friedler has been supported in part by the AT&T Labs Fellowship Program and the University of Maryland Ann G. Wylie Dissertation Fellowship.

\*\* The work of David Mount has been supported in part by the National Science Foundation under grant CCR-0635099 and the Office of Naval Research under grant N00014-08-1-1015.

quantities of such data necessitate compression of the sensor observations, yet analyses of these observations is desirable. Ideally, such analysis should operate over the compressed data without the need to decompress it. In order to perform statistical analyses of the data, it is often desirable that retrieval queries be supported. In this paper, we present the first data structure and algorithms for answering range searching queries over compressed data streams arising from large sensor networks.

In an earlier paper [10], we presented an algorithm for losslessly compressing kinetic sensor data and a framework for analyzing its performance. (See Section 2 for a brief introduction.) We assume that we are given a set of sensors, which are at fixed locations in a space of constant dimension (our results apply generally to metric spaces of constant doubling dimension [18].) These sensors monitor the movement of a number of kinetic objects. Each sensor monitors an associated region of space, and at regular time steps it records an occupancy count of the number of objects passing through its region. Over time, each sensor produces a string of occupancy counts; the problem considered in [10] is how to compress all these strings.

Previous compression of sensor data in the literature has focused largely on approximation algorithms in the streaming model or lossy compression of the data. We consider lossless compression. This is often more appropriate in scientific contexts, where analysis is performed after the data has been collected and accurate results are required. Lossless compression algorithms have been studied in the single-string setting [16, 22, 27, 28] but remain mostly unstudied in a sensor-based setting [10].

In order to query observed sensor data, which ranges over time and space, we need to consider both temporal and spatial queries. *Temporal range queries* are given a time interval and return an aggregation of the observations over that interval. *Spatial range queries* are given some region of space (e.g., a rectangle, sphere, or halfplane) and return an aggregation of the observations within that region. *Spatio-temporal range queries* generalize these by returning an aggregation restricted by both a temporal and a spatial range. We assume that occupancy counts are taken from a commutative semigroup of fixed size, and the result is a semigroup sum over the range. There are many different data structures for range searching (on uncompressed data), depending on the properties of the underlying space, the nature of the ranges, properties of the semigroup, and whether approximation is allowed [1].

We present data structures for storing compressed sensor data and algorithms for performing spatio-temporal range queries over these data. We analyze the quality of these range searching algorithms in terms of both time and space by considering the information content of the set of sensor outputs. There are two well-known ways in which to define the information content of a string, classical statistical (Shannon) entropy and empirical entropy. Statistical entropy [24] is defined under the assumption that the source  $X$  is drawn from a stationary, ergodic random process. The normalized statistical entropy, denoted  $H(X)$ , provides a lower bound on the number of bits needed to encode a character of  $X$ . In

**Table 1.** Time and space bounds for temporal range searching and  $\varepsilon$ -approximate spatio-temporal range searching for fat convex ranges in  $\mathbb{R}^d$ .  $S$  is the number of sensors in the network,  $T$  is the length of the observation period, and  $\text{Enc}(X)$  and  $\text{Enc}(\mathbf{X})$  denote the sizes of the compressed representations for single sensor stream (for temporal range searching) and sensor system (for spatio-temporal range searching), respectively.

### Bounds for Range Searching.

	Temporal	Spatio-temporal
Preprocessing time	$O(\text{Enc}(X))$	$O(\text{Enc}(\mathbf{X}))$
Query time	$O(\log T)$	$O(((1/\varepsilon^{d-1}) + \log S) \log T)$
Space	$O(\text{Enc}(X))$	$O(\text{Enc}(\mathbf{X}) \log S)$

contrast, the empirical entropy [17, 19], denoted  $H_k(X)$ , while similar in spirit to the statistical entropy, assumes no underlying random process and relies only on the observed string and the context of the most recent  $k$  characters. These definitions and distinctions are discussed in more detail in a companion paper [11].

Previously, random access or pattern matching queries over compressed text (without relying on decompression) have been studied in the context of strings [3, 8, 9, 13] and XML files [7]. For example, Ferragina and Venturini [9] show that it is possible to retrieve a substring (indexed by start and end times) in the compressed text with query time equal to  $O(1 + \frac{\ell}{\log T})$  where  $\ell$  is the length of the substring and  $T$  is the length of the string  $X$ . Their space requirement is  $T \cdot H_k(X) + o(T)$  bits. Their data structure allows substring queries, which are very different from semigroup range searching queries, which we consider here. For surveys of this area see [6, 15, 21].

In this paper we present the first range query results over compressed kinetic sensor data. Specifically, we consider the problems of temporal range searching and spatio-temporal range searching for fat convex ranges (e.g. spheres, rectangles with low aspect ratio, etc. [4]).

As mentioned earlier, we analyze our algorithms in terms of the joint entropy of the sensor outputs. The preprocessing makes only one pass over the compressed data, and thus it can be performed as the data are collected. The query bounds are logarithmic in the input size. The space bounds, given in bits, match the entropy lower bound up to constant factors. Specific bounds are given in Table II.

In addition to theoretical results, we present experimental evaluation of our temporal range searching structure. These results show that, in addition to being theoretically efficient, our data structure offers a roughly 50-fold improvement in space. These improvements increase as the data sets become larger. Both our temporal and spatio-temporal data structures are quite practical, being based on very simple data structures (tries, binary trees, and quadtrees, in particular).

## 2 Framework for Kinetic Sensor Data

In an earlier paper [10] we introduced a framework and a lossless compression scheme for discrete kinetic data observed by a sensor network. This framework

will be used as a basis for the results of this paper. We begin with some basic definitions about the structure of the sensor network and the associated observed data streams. Consider a static sensor network with  $S$  sensors, monitoring the motion of a collection of moving objects. Let  $P$  be a point set indicating the sensor locations. All sensors are assumed to operate over  $T$  synchronized time steps. Each sensor observes the motion of objects in some region surrounding it, and records an *occupancy count* indicating the number of objects passing within its region during the observed time step. No assumptions are made about the nature of the point motion nor the nature of the sensor regions.

Central to our framework is the notion that each sensor's output is statistically dependent on a relatively small number of nearby sensors. For some point  $p \in P$ , let  $NN_m(p) \subseteq P$  be the  $m$  nearest neighbors of  $p$ . Sensors  $i$  and  $j$  with associated sensor positions  $p_i, p_j \in P$  are said to be *mutually  $m$ -close* if  $p_i \in NN_m(p_j)$  and  $p_j \in NN_m(p_i)$ . For a constant  $m$ , a sensor system is said to be  *$m$ -local* if all pairs of sensors that are not mutually  $m$ -close are statistically independent.

In [10] we introduced a compression algorithm, *PartitionCompress*, which operates on an  $m$ -local sensor system. It compresses the sensor outputs to within a constant factor  $c$  (depending on dimension) of the optimal joint entropy bound. Intuitively, the compression algorithm is based on the following idea. If two sensor streams are statistically independent, they may be compressed independently from each other. If not, optimal compression can only be achieved if they are compressed jointly. The algorithm works by compressing the outputs from clusters of nearest neighbor groups together, as if they were a single stream. In order to obtain the desired compression bounds, these clusters must be sufficiently well separated so that any two mutually  $m$ -close sensors are in the same cluster. *PartitionCompress* partitions the points into a constant number  $c$  (independent of  $m$  but depending on dimension) of subsets for which this is true and then compresses clusters together to take advantage of local dependencies. The compression of a single cluster may be performed using any string compression algorithm; to obtain the near optimal bound, this algorithm must compress streams to their optimal entropy bound. It is shown in a companion paper [11] that LZ78, the Lempel-Ziv dictionary compression algorithm [28], is sufficient for our purposes.

For the rest of the paper, we will use  $\text{Enc}_{alg}(\mathbf{X})$  to denote the length of the encoded set of sensor outputs  $\mathbf{X}$ , where  $alg$  specifies the string compressor used by the compression algorithm of [10]. Since LZ78 will suffice, let  $\text{Enc}(\mathbf{X}) = \text{Enc}_{\text{LZ78}}(\mathbf{X})$ . In [11], it is shown that  $\text{Enc}(\mathbf{X})$  is on the order of the optimal space bound when analyzed in terms of either the statistical or empirical entropy.

### 3 Temporal Range Searching

In this section we describe a data structure that answers temporal range searching queries over a single compressed sensor stream. Let  $X$  be a sequence of sensor counts over time period  $[1, T]$ , which will be compressed and preprocessed into a data structure so that given any temporal range  $[t_0, t_1] \in [1, T]$ , the aggregated

count over that time period can be calculated efficiently. We assume that the individual sensor counts are drawn from a semigroup, and the sum is taken over this semigroup. The space used by the data structure (in bits) will be asymptotically equal to that of the compressed string, and the query time will be logarithmic in  $T$ . Here is the main result of this section. Recall that, given string  $X$ ,  $\text{Enc}(X)$  denotes the length of the compressed encoding of  $X$ .

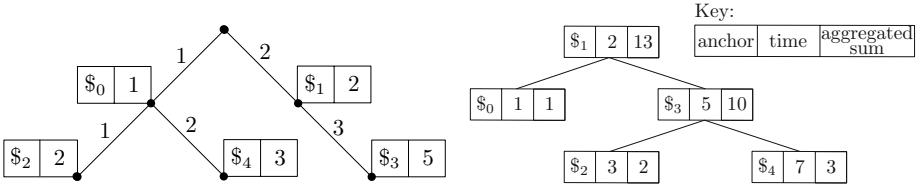
**Theorem 1.** *There exists a temporal range searching data structure, which given string  $X$  over a time period of length  $T$ , can be built in time  $O(\text{Enc}(X))$ , achieves query time  $O(\log T)$ , and uses space  $O(\text{Enc}(X))$  bits.*

The remainder of this section is devoted to proving this theorem. Here we consider the simpler special case where the semigroup is in fact a group, which means that both addition and subtraction of weights are allowed. The semigroup case involves a more sophisticated data structure; that description and the full proofs for this section can be found in [12].

We begin by describing the preprocessing for our data structure in the group context, where subtraction of counts is allowed. First, the given sequence  $X$  is compressed using the LZ78 compression algorithm and the standard accompanying trie (also known as a *dictionary*) containing nodes that represent *words* is created [28]. We begin with a short overview of this algorithm. LZ78 scans over the input, putting characters into a trie so that each edge in the trie represents a single character. As the string is scanned from beginning to end, the prefix is looked up in the trie and the most recent character is added to that path in the trie. The resulting *word* is added to the compressed version of the string by simply storing a pointer to the bottom most node of the path in the dictionary. Let  $d$  be the number of words in the dictionary. Each word in the dictionary (possibly excepting the last) is used in the compressed version of the string exactly once. In addition, each word in the dictionary was generated only after all prefixes had previously been added, so the trie is *prefix-complete* [8]. We will make use of the fact, proved in a companion paper [11], that  $d \log d = \text{Enc}(X)$ .

Let us now discuss our preprocessing of the stream  $X$ . It involves two phases. The first takes place during the single scan through the input. The data are compressed using LZ78 compression, the associated trie is created, and pointers to word endings (called *anchor points*) are stored. Additionally, the aggregated value of each word (e.g. the sum of its component counts, or the *word sum*) is added to the associated node in the dictionary. This value can be found by adding the count at the current node to its parent's stored aggregated value as each letter is added to an existing word in the trie. This phase takes time  $O(T)$  and we will refer to the result of this phase as the *compressed form* of the input.

The second phase, which is the one we will analyze for its additional non-compression related time, consists of creating a binary search tree over the anchor points and initializing auxiliary data structures. Building a binary search tree over the anchor points (stored already sorted by word start time) requires  $O(d)$  time, since there are  $d$  words and each has one associated anchor point. Additionally, we create an aggregation tree over the aggregate word values, so that aggregate values of consecutive words can be easily found when considering



**Fig. 1.** Left: LZ78 trie annotated with associated anchor points and word sums for a single sensor with observation string “12112312”. Considered inline, the string with anchor points as breaks between the words becomes 1 \$0 2 \$1 11 \$2 23 \$3 12 \$4. Right: The corresponding binary search tree based on word start times that also contains aggregated sums for the words contained in each node’s subtree.

substrings. This takes time  $O(d)$  when created as an annotation to the existing binary search tree. Finally, we will later need access to a level ancestor data structure, which can be built in  $O(d)$  time [5].

**Lemma 1.** *Assuming that the input is given in compressed form, temporal range searching takes preprocessing time  $O(d) = O(\text{Enc}(X))$ .*

Next we describe query processing. Each temporal query can be categorized as either *internal* or *overlapping* depending on whether the query interval overlaps one word or multiple words, respectively. Internal queries implicitly divide a word into a prefix, query region, and suffix. Since the trie is prefix-complete, all prefix aggregations are stored in our annotated trie and can be retrieved in  $O(1)$  time using these annotations and the level-ancestor data structure. Entire word aggregate values can be retrieved as a group using the annotated binary search tree created over the aggregate word values. Using these basic retrieval systems, internal queries can be found by subtracting the prefix and suffix values from the word total and overlapping queries can be determined by adding the suffix, complete word sums, and prefix values.

The running time is dominated by the  $O(\log d)$  time needed to lookup which word(s) overlap the given temporal query using a binary search over the sorted anchor points, and the  $O(\log d)$  complete words that might be summed using the aggregation tree for overlapping queries.

**Lemma 2.** *The query time for temporal range searching in the group setting is  $O(\log d) = O(\log T)$ .*

Finally, we consider the total number of bits of space used in this process. The storage of the anchor points requires space  $d$  and the annotated dictionary takes space  $d$ . Under our assumption that the group is of fixed size, the largest sum that can be achieved during this process is  $O(T)$ . These sums annotate dictionary words, so the modified dictionary takes space at most  $O(d \log T)$ , which is  $O(d \log d)$  since  $T = O(d^2)$ . In addition, we make use of an auxiliary data structure to solve the level ancestor problem [5]. This data structure requires

storage only of the tree,  $O(d)$  pointers to nodes in the tree, and a table of  $O(d)$  encoded subtrees that each take  $O(\log d)$  space. Thus, the total size required by this auxiliary data structure is also  $O(d \log d)$ .

**Lemma 3.** *The total space in bits required for our temporal range structure in the group setting is  $O(d \log d) = O(\text{Enc}(X))$ .*

## 4 Spatio-temporal Range Searching

In this section we consider how to extend the results of the previous section on temporal range searching on a single string to range searching for a sensor system, in which queries include both the spatial and temporal components of the data. We assume that we are given an  $m$ -local sensor system with  $S$  sensors. Each sensor is identified with its location  $p_i$  in space and a stream  $X_i$  of occupancy counts over some common time interval  $[1, T]$ . We assume that the sensors reside in real  $d$ -dimensional space,  $\mathbb{R}^d$ , where  $d$  is a constant. Our approach can be generalized to metric spaces with constant doubling dimension. We model each sensor's location as a point, and the answer to a range query consists of the sensors whose associated point lies within the query region. Let  $P$  and  $\mathbf{X}$  denote the sets of sensor locations and observation streams, respectively.

Define a *spatio-temporal range query* to be a pair  $(Q, [t_0, t_1])$  consisting of a geometric query range  $Q$  from some space  $\mathcal{Q}$  of allowable ranges (e.g., rectangles, balls, or halfspaces) and a time interval  $[t_0, t_1] \subseteq [1, T]$ . The problem is to compute the sum of the occupancy counts of the sensors whose locations lie within the range, that is,  $P \cap Q$ , over the given time interval. In general, the occupancy counts are assumed to be drawn from a commutative semigroup, and the sum is taken over this semigroup. The remainder of this section is devoted to proving the following theorem, which shows that approximate spherical spatio-temporal range queries can be answered efficiently. In fact, these techniques hold for all fat convex ranges, but for simplicity of presentation we will limit ourselves to the spherical case here. (Proofs can be found in [12].)

**Theorem 2.** *There exists a data structure for answering  $\varepsilon$ -approximate spatio-temporal spherical range queries for an  $S$ -element  $m$ -local sensor system  $\mathbf{X}$  in  $\mathbb{R}^d$  for all sufficiently long time intervals  $T$  with preprocessing time  $O(\text{Enc}(\mathbf{X}))$ , query time  $O(((1/\varepsilon^{d-1}) + \log S) \log T)$ , and space  $O(\text{Enc}(\mathbf{X}) \log S)$  bits.*

Rather than considering a particular range searching problem, we will show that the above problem can be reduced to a generalization of classical range searching. To motivate this reduction, we recall that the compression algorithm presented in [10] groups sensors into clusters, and the sensor outputs within each cluster are then compressed jointly. In order to answer range queries efficiently, it will be necessary to classify each such cluster as lying entirely inside the range, outside the range, or overlapping the range's boundary. In the last case, we will need to further investigate the cluster's internal structure. Efficiency therefore is dependent on the number of clusters that overlap the range's boundary. We will exploit

spatial properties of the clusters as defined in [10] to achieve this efficiency. To encapsulate this notion abstractly, we introduce the problem of *range searching over clumps*, in which the points are replaced by balls having certain separation properties. Eventually, we will show how to adapt the BBD-tree structure [4] to answer approximate range queries in this context.

Given any metric space of constant dimension, a *set of clumps* is defined to be a finite set  $C$  of balls that satisfies the following *packing property* for some constant  $\gamma$  (depending possibly on dimension): Given any metric ball  $b$  of radius  $r$ , the number of clumps of  $C$  of radius  $r'$  that have a nonempty intersection with  $b$  is at most  $O((1 + (r/r'))^\gamma)$ . Given a range shape  $Q$ , a clump may either lie entirely within  $Q$ , entirely outside  $Q$ , or may intersect the boundary of  $Q$ . In the last case, we say that the clump is *stabbed* by  $Q$ .

The relevance of the notion of clumps to our setting is established in the following lemma. The lemma states that the clusters of sensors within a single partition created by the *PartitionCompress* algorithm of [10], when associated with a bounding ball, form a set of clumps. The *PartitionCompress* algorithm partitions the sensor point set  $P$  into a constant number of *groups*,  $P_1, \dots, P_c$  (where  $c$  depends only on the dimension of the space). Each group  $P_i$  is further partitioned into subsets, called *clusters*, such that if two sensors are in different clusters then their outputs are independent of each other. Given a ball  $b$  and real  $\varphi > 0$ , let  $\varphi b$  denote the ball concentric with  $b$  whose radius is a factor of  $\varphi$  times the radius of  $b$ .

**Lemma 4.** *Given a point set  $P$ , let  $P' \subseteq P$  be any of the groups generated by the PartitionCompress algorithm, and let  $P'_1, \dots, P'_h$  denote the associated set of clusters for this group. Then there exists a set of balls  $C = \{b_1, \dots, b_h\}$  that form a set of clumps such that  $P'_i \subseteq b_i$ .*

The proof of this lemma (given in [12]) relies on the observation that  $\frac{1}{2}b_1, \dots, \frac{1}{2}b_h$  are pairwise disjoint. This is established based on the geometric properties of the repetitive partitioning process of the *PartitionCompress* algorithm.

We define the problem of *range searching among clumps* as follows: Given a space  $\mathcal{Q}$  of allowable ranges and a set  $C$  of clumps, each of which is associated with a numeric weight from some commutative semigroup, preprocess the clumps into a collection of subsets, called *generators*, such that given any query range  $Q \in \mathcal{Q}$ , it is possible to report (1) a subset of these generators that form a disjoint cover of the clumps lying wholly within  $Q$  and (2) the subset of clumps that  $Q$  stabs. The total space requirements of a data structure for the range searching problem over clumps is the sum of space needed to represent the generators and the clumps, together with the space needed for storing the index structure needed to answer queries. The query time includes number of generators and stabbed clumps returned, plus the time to compute them.

Many data structures used in range searching are based on partition trees [1]. In such data structures, space is recursively subdivided into regions and the points are partitioned among these regions, until each region contains a single point. Each node of the tree is associated with a generator corresponding to the elements of the point set that lie in the leaves descended from this node. Our

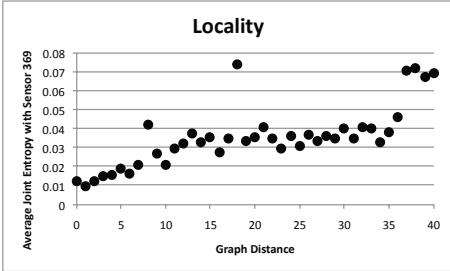
main result shows that, given a partition-tree based solution to the problem of range-searching among clumps, we can use such a structure to answer spatio-temporal range queries. This is done by adding an auxiliary data structure to each of the nodes of the tree to answer the temporal queries.

**Lemma 5.** *Suppose that we have a partition-tree based data structure that, given a set  $C$  of  $n$  clumps, can answer range queries over a query space  $\mathcal{Q}$  with preprocessing time  $pp(n)$ , query time  $qt(n)$ , space  $sp(n)$  bits, and has height  $h(n)$ . Then there exists a data structure that can answer spatio-temporal range queries for an  $m$ -local sensor system  $\mathbf{X}$  of size  $S$  over a range space  $\mathcal{Q}$  and time interval of length  $T$  with preprocessing time  $O(h(S) \cdot pp(S) + \text{Enc}(\mathbf{X}))$ , query time  $O(qt(S) \cdot \log T)$ , and space  $O(sp(S) + h(S) \cdot \text{Enc}(\mathbf{X}))$  bits.*

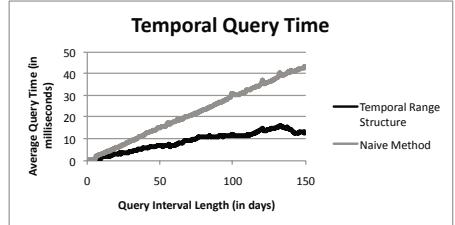
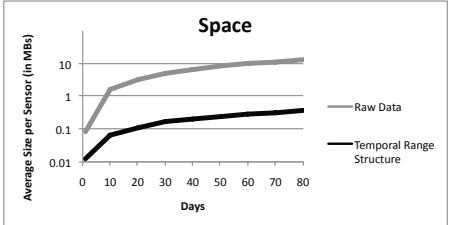
Observe that we can generalize the notion of  $\varepsilon$ -approximate range searching to approximate range searching over clumps. To do so we define two ranges  $Q^-$  and  $Q^+$ , representing the inner and outer approximate ranges. For example, in the case of spherical range searching, given a query ball  $Q$ , we define  $Q^- = Q$  and  $Q^+$  to be the ball concentric with  $Q$  but whose radius is scaled relative to  $Q$ 's radius by a factor of  $(1 + \varepsilon)$ . (See Arya and Mount [4] for further details.) If a generator lies entirely within  $Q^+$  its points may be counted as lying within the approximate range, if it lies entirely outside of  $Q^-$ , its points may be considered to lie outside the approximate range. A clump is classified as being stabbed by  $Q$  if and only if it has a nonempty intersection with both  $Q^-$  and the complement of  $Q^+$ . It is easy to show that such a clump has diameter  $\Omega(\varepsilon \cdot \text{diam}(Q))$  [4]. By the packing property of clumps, the number of such clumps is  $O(1/\varepsilon^\gamma)$ , where the parameter  $\gamma$  depends only on the dimension of the space. We conclude by remarking that it is relatively easy to generalize many standard approximate range searching data structures based on hierarchical partitioning to answer range searching over clumps. We present one example based on the BBD-tree data structure of [4].

**Lemma 6.** *There exists a data structure for answering  $\varepsilon$ -approximate spherical range searching queries over a set  $C$  of  $n$  clumps in  $\mathbb{R}^d$  with preprocessing time  $O(n \log n)$ , query time  $O((1/\varepsilon^{d-1}) + \log n)$ , and space  $O(n \cdot (\text{prec}(C) + \log n))$  bits, where  $\text{prec}(C)$  denotes the maximum number of bits of precision in the geometric coordinates used to define  $C$ .*

By applying Lemma 5 to the above data structure, it follows that we can answer  $\varepsilon$ -approximate spherical range searching queries for a sensor system of size  $S$  over a time period of length  $T$  with preprocessing time  $O((S \log^2 S) + \text{Enc}(\mathbf{X}))$ , query time  $O(((1/\varepsilon^{d-1}) + \log S) \log T)$ , and space  $O((S \cdot (\text{prec}(C) + \log S) + \text{Enc}(\mathbf{X}) \log S)$  bits, where  $\text{prec}(C)$  denotes the maximum number of bits of precision in the geometric coordinates used to define  $C$ . Under the assumption that  $T$  is sufficiently large that the encoding space dominates over time-invariant quantities, this completes the proof of Theorem 2.



**Fig. 2.** Locality shown via a comparison of the graph distance between sensor 369 and other sensors and the average joint entropy between all such pairs of sensors. As the distance between the sensors increases so does the joint entropy, showing that closer sensors are more likely to have related outputs.



**Fig. 3.** *Left:* The space used by the raw data in comparison to that used by the temporal range structure shown for varying numbers of days of data. Note that the size is shown in a logarithmic scale. As the number of days increase, the space saving increases as well. *Right:* Average temporal query times for 100 randomly chosen queries for each interval length. Query times for two methods are shown; one using the temporal range structure and one using a naive method that simply aggregates values one by one.

## 5 Experimental Results

In addition to the theoretical analysis of the range searching results presented here, we evaluated the temporal range searching structure experimentally. Using a data set provided by the Mitsubishi Electronic Research Laboratory (MERL) [26] consisting of activation times for sensors located in the hallways of their building, we analyzed three aspects of our data structure's performance; locality, space, and time. In short, we found that our assumption that sensors closer to each other are more likely to have similar outputs was correct and that our data structure was able to use less space than a naive structure while providing faster query times. In the rest of this section, we describe the data set and our experimental methods in greater detail.

The MERL data set consists of activation times, representing people moving, for 213 sensors. These activation times are given with epoch start and end times. Using these start times, and noting that each activation lasted approximately one second, we translated these activation times into streams of data for each sensor in the form described earlier. Each activation is represented by a count of one and seconds in which no activations were reported are represented by a count of zero. These streams are associated with sensors whose locations are known and relationships are shown in a building map. Using this map, we create a graph in which neighboring sensors are connected by an edge with weight one.

In order to evaluate our assumption that nearby sensors are more likely to have related outputs, we compared the distance between a single sensor (sensor 369, a sensor in the middle of a hallway) to the pairwise joint entropy of that sensor and all other sensors' outputs. Distance was computed as the shortest path distance within the neighborhood graph described earlier, and the joint entropy considered was an empirical generalization of joint entropy[11] with a window size of 10 seconds. The average joint entropy for each distance is shown in Figure 2. The graph shows that those sensors in the neighborhood near sensor 369, those less than distance five away, have outputs with lower joint entropy. After this local neighborhood, the joint entropy raises to a relatively constant threshold for the majority of distances, and finally raises again for far away sensors. The outlying points at distances 8 and 18 represent comparisons with sensors in the unusual areas near the elevators and lunch room, respectively.

We considered the storage space for the sensor data streams for the raw data (consisting of one value per second) versus the temporal range structures (specifically the annotated tries) written to files. The size taken over all sensors by each of these methods as it varies based on the number of days (in increments of 10) of data can be seen in Figure 3. We call the ratio between the space used by the raw data and the space used by the temporal range structure the *improvement ratio*. The improvement ratio increases as the amount of data increases, ranging from a 14-fold improvement for 1 day to a 66-fold improvement for 80 days of data. This increase is likely caused by the observation of repeated patterns; the first observation must be stored in the annotated trie while later observations can simply extend existing patterns, taking less space.

Query time is considered for varying query interval lengths for 150 days of data (at 1 day intervals). We compare our temporal range searching method to the naive method that aggregates by linearly adding each count. Query times do not include the time to read in the file or, in the case of the temporal range structure, the one-time preprocessing cost. A graph showing the interval length versus the query time for each of these methods is given in Figure 3. Each query time depicted on the graph represents the average of 100 randomly chosen queries of the given interval length. As the interval length increases, the temporal range structure's improvement over the naive method increases as well.

## References

- [1] Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In: Chazelle, B., Goodman, J., Pollack, R. (eds.) *Advances in Discrete and Computational Geometry*, pp. 1–56. American Mathematical Society, Providence (1998)
- [2] Akyildiz, I., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: A survey. In: *Computer Networks*, pp. 393–422 (2002)
- [3] Amir, A., Benson, G., Farach-Colton, M.: Let sleeping files lie: Pattern matching in Z-compressed files. *J. Comput. Syst. Sci.* 52(2), 299–307 (1996)
- [4] Arya, S., Mount, D.M.: Approximate range searching. *Computational Geometry: Theory and Applications* 17, 135–152 (2000)
- [5] Bender, M.A., Farach-Colton, M.: The level ancestor problem simplified. *Theoretical Computer Science* 321, 5–12 (2004)

- [6] Ferragina, P., Gonzalez, R., Navarro, G., Venturini, R.: Compressed text indexes: From theory to practice. *Journal of Experimental Algorithmics* 13, 12–31 (2009)
- [7] Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and searching XML data via two zips. In: Proc. of the 15th International Conference on World Wide Web, pp. 751–760 (2006)
- [8] Ferragina, P., Manzini, G.: Indexing compressed text. *Journal of the ACM* 52(4), 552–581 (2005)
- [9] Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science* 372(1), 115–121 (2007)
- [10] Friedler, S.A., Mount, D.M.: Compressing kinetic data from sensor networks. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, pp. 191–202. Springer, Heidelberg (2009)
- [11] Friedler, S.A., Mount, D.M.: Realistic compression of kinetic sensor data. Technical Report CS-TR-4959, University of Maryland, College Park (2010)
- [12] Friedler, S.A., Mount, D.M.: Spatio-temporal range searching over compressed kinetic sensor data. Technical Report CS-TR-4960, U. Maryland (2010)
- [13] González, R., Navarro, G.: Statistical encoding of succinct data structures. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 294–305. Springer, Heidelberg (2006)
- [14] Guitton, A., Trigoni, N., Helmer, S.: Fault-tolerant compression algorithms for sensor networks with unreliable links. Technical Report BBKCS-08-01, Birkbeck, University of London (2008)
- [15] Hon, W.-K., Shah, R., Vitter, J.S.: Compression, indexing, and retrieval for massive string data. In: Amir, A., Parida, L. (eds.) Combinatorial Pattern Matching. LNCS, vol. 6129, pp. 260–274. Springer, Heidelberg (2010)
- [16] Huffman, D.A.: A method for the construction of minimum-redundancy codes. In: Proceedings of the IRE, vol. 40 (September 1952)
- [17] Kosaraju, R.S., Manzini, G.: Compression of low entropy strings with Lempel–Ziv algorithms. *SIAM J. Comput.* 29(3), 893–911 (1999)
- [18] Krauthgamer, R., Lee, J.R.: Navigating nets: Simple algorithms for proximity search. In: Symposium on Discrete Algorithms (2004)
- [19] Manzini, G.: An analysis of the Burrows–Wheeler transform. *J. ACM* 48(3), 407–430 (2001)
- [20] MIT Media Lab. The Owl project, <http://owlproject.media.mit.edu/>
- [21] Navarro, G., Mäkinen, V.: Compressed full-text indexes. *ACM Computing Surveys* 39(1) (2007)
- [22] Rissanen, J.: Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development* 20 (1976)
- [23] Saunier, N., Sayed, T.: Automated analysis of road safety with video data. In: *Transportation Research Record*, pp. 57–64 (2007)
- [24] Shannon, C.E.: A mathematical theory of communication. *The Bell System Technical Journal* 27, 379–423, 623–656 (1948)
- [25] Stutchbury, B.J.M., Tarof, S.A., Done, T., Gow, E., Kramer, P.M., Tautin, J., Fox, J.W., Afanasyev, V.: Tracking long-distance songbird migration by using geolocators. *Science*, 896 (February 2009)
- [26] Wren, C.R., Ivanov, Y.A., Leigh, D., Westbues, J.: The MERL motion detector dataset: 2007 workshop on massive datasets. Technical Report TR2007-069, Mitsubishi Electric Research Laboratories, Cambridge, MA, USA (August 2007)
- [27] Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* IT-23(3) (May 1977)
- [28] Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* 24(5), 530–536 (1978)

# Constructing the Exact Voronoi Diagram of Arbitrary Lines in Three-Dimensional Space<sup>☆</sup> with Fast Point-Location

Michael Hemmer<sup>1</sup>, Ophir Setter<sup>2</sup>, and Dan Halperin<sup>2</sup>

<sup>1</sup> INRIA - Sophia Antipolis, France

[Michael.Hemmer@sophia.inria.fr](mailto:Michael.Hemmer@sophia.inria.fr)

<sup>2</sup> Tel-Aviv University, Israel

[{ophirset,danha}@post.tau.ac.il](mailto:{ophirset,danha}@post.tau.ac.il)

**Abstract.** We introduce a new, efficient, and complete algorithm, and its exact implementation, to compute the Voronoi diagram of lines in space. This is a major milestone towards the robust construction of the Voronoi diagram of polyhedra. As we follow the exact geometric-computation paradigm, it is guaranteed that we always compute the mathematically correct result. The algorithm is complete in the sense that it can handle all configurations, in particular all degenerate ones. The algorithm requires  $O(n^{3+\varepsilon})$  time and space, where  $n$  is the number of lines. The Voronoi diagram is represented by a data structure that permits answering point-location queries in  $O(\log^2 n)$  expected time. The implementation employs the CGAL packages for constructing arrangements and lower envelopes together with advanced algebraic tools.

**Keywords:** Voronoi Diagrams, Point Location, Lower Envelopes, Robust Geometric Computing, Computational Geometry, CGAL.

## 1 Introduction

The Voronoi diagram (*VD*) is among the most fundamental structures in Computational Geometry, and is known to be a useful tool in a variety of domains. For instance, structural biology [19], [34] and robot motion planning [25], [35] apply Voronoi diagrams to encode point sets keeping maximal distance from atoms or obstacles, respectively. A related concept is the medial-axis transform [6], which is considered fundamental in solid modeling and applied to problems such as finite element meshing, shape morphing, and feature recognition. Yet, the adaptation of complex three-dimensional Voronoi diagrams in professional tools has been very slow. Their use is hindered by the difficulty of designing and implementing reliable geometric algorithms for complex structures in three-dimensional space.

Voronoi diagrams have been the subject of a tremendous amount of research. We refer the reader to the survey by Aurenhammer and Klein [2] of work

<sup>☆</sup> This work has been supported in part by the Israel Science Foundation (grant no. 236/06), by the German-Israeli Foundation (grant no. 969/07), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

published up till 2000. Voronoi diagrams in  $\mathbb{R}^2$  are well understood in almost all aspects, that is, in terms of complexity and optimal algorithms as well as in terms of robust and efficient implementations. In  $\mathbb{R}^3$  much less is known, even for simple objects such as lines, segments, or polyhedra. For example, a tight bound on the combinatorial complexity of the *VD* of  $n$  lines or line segments in  $\mathbb{R}^3$  is unknown; it is conjectured that the complexity is near-quadratic; the known lower bound is  $\Omega(n^2)$  [1], but the best known upper bound is<sup>1</sup>  $O(n^{3+\varepsilon})$  [32]. In the case of lines with a fixed number  $c$  of orientations the upper bound was improved to  $O(c^4 n^{2+\varepsilon})$  [26]. A complete analysis of all possible combinatorial cases for three arbitrary lines is presented by Everett *et al.* [16], [17].

Today, there are many published results on robust constructions of different types of Voronoi diagrams in  $\mathbb{R}^2$ . Not only Voronoi diagrams of points are considered, but also Voronoi diagrams of line segments [23], circles [15], ellipses [14], and more [8, §2], [31]. In  $\mathbb{R}^3$ , an exact implementation of the Voronoi diagram of additively-weighted points was analyzed in [2], but we are not aware of any exact, complete, and implemented algorithm that computes Voronoi diagrams of lines, line segments, or polyhedra. Nevertheless, progress has been made toward the exact computation of the arrangement of quadrics [5], [12]. Each Voronoi cell of the diagram of lines in space can be represented as the union of cells of such an arrangement. Other approaches explicitly aim for an exact or robust computation of the Voronoi diagram (or the medial axis) [9], [27]. However, those approaches are not complete. For example, Culver's algorithm [9] does not handle singular trisector-curves.

Finally, Hanniel and Elber [20] provided an algorithm to construct the Voronoi cell of bounded planes, spheres, and cylinders in  $\mathbb{R}^3$ . Similar to ours, the approach utilizes lower envelopes but leaves robustness issues aside, since the use of parametrized intersection curves hinders an efficient implementation using exact arithmetic. Moreover, it does not consider point location.

We present an exact and complete (and thus robust) algorithm for computing the Voronoi diagram of arbitrary lines in three dimensions with respect to the Euclidean metric. The algorithm requires  $O(n^{3+\varepsilon})$  time and space, where  $n$  is the number of input lines. The data structure admits answering of point-location queries in  $O(\log^2 n)$  time. We believe that the nature of the algorithm and the general approach of its implementation constitute a major milestone towards an exact and robust construction of the Voronoi diagram of polyhedra in  $\mathbb{R}^3$ .

We utilize the fact that in Euclidean space the boundary of Voronoi cell (*VC*) can be considered as a lower envelope since the cell essentially has a certain “star shapedness” property: For any point  $p$  inside the Voronoi cell of a specific line site  $\ell$ , the line segment connecting  $p$  to its projection  $p_\ell$  onto  $\ell$ , is fully contained in the cell. This observation enables us to represent the Voronoi cell of  $\ell$  as a minimization diagram, which is (conceptually) embedded on an infinitesimally small cylinder around  $\ell$ . This observation is similar to the well-known connection

---

<sup>1</sup> A bound of the form  $O(f(n) \cdot n^\varepsilon)$  means that the actual upper bound is  $C_\varepsilon f(n) \cdot n^\varepsilon$ , for any  $\varepsilon > 0$ , where  $C_\varepsilon$  is a constant that depends on  $\varepsilon$ , and generally tends to infinity as  $\varepsilon$  goes to 0.

between Voronoi diagrams and lower envelopes [13]. Lower dimensional cells are represented several times, namely as part of the boundary of the *VC* of each line they are associated with. The implementation is developed in and based on CGAL, the Computational Geometry Algorithms Library<sup>2</sup>.

The paper is organized as follows. Section 2 discusses preliminary subjects, such as properties of bisectors and trisectors of lines in space and the lower envelope algorithm. Section 3 describes the details of the construction of a Voronoi cell. Section 4 discusses the point location algorithm and its analysis. Section 5 gives implementation details and presents preliminary experimental results that were obtained with our software.

## 2 Preliminaries

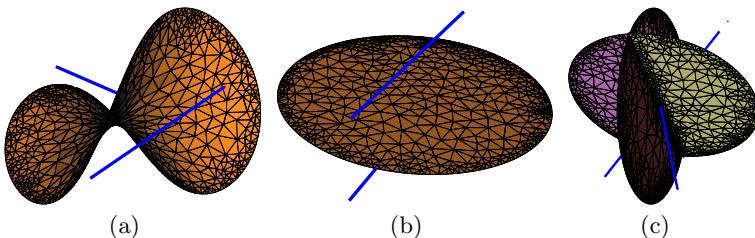
Let  $\mathcal{O} = \{s_1, s_2, \dots, s_n\}$  be a set of objects in  $\mathbb{R}^d$ , also referred to as sites. We follow the Voronoi diagram definition by Everett *et al.* [17]: The *Voronoi diagram*  $VD(\mathcal{O})$  is the subdivision of  $\mathbb{R}^d$  into cells, where each cell  $VC(S)$  is associated with a subset  $S \subseteq \mathcal{O}$ , such that every point in  $VC(S)$  is strictly closer to all sites in  $S$  than to all other sites in  $\mathcal{O}$  and is equidistant from all sites in  $S$ . The formal definition is:

$$VC(S) = \left\{ p \in \mathbb{R}^d \mid \begin{array}{l} \forall s \in S, t \in \mathcal{O} \setminus S : d(p, s) < d(p, t) \\ \forall s, t \in S : d(p, s) = d(p, t) \end{array} \right\}$$

In the context of this paper,  $\mathcal{O}$  denotes a set of arbitrary rational lines in  $\mathbb{R}^3$  and  $d(\cdot, \cdot)$  denotes the Euclidean distance function. The set of points that is of equal distance to two or three sites is called a bisector or trisector, respectively.

### 2.1 Properties of Bisectors and Trisectors

We next state some properties of bisectors and trisectors of the Voronoi diagram of lines in  $\mathbb{R}^3$  that are used throughout this paper. Proposition 1 gives properties of bisectors; see Figure 1 for illustrations.



**Fig. 1.** Bisector of: (a) two generic lines; (b) two parallel lines; (c) two intersecting lines. The diagrams were created with our implementation (see Section 5), and were clipped by a sphere for convenience.

<sup>2</sup> <http://www.cgal.org>

**Proposition 1.** *The bisector of two lines  $\ell_1$  and  $\ell_2$  in three-dimensional space is either (a) a hyperbolic paraboloid (a surface of algebraic degree 2), if  $\ell_1$  and  $\ell_2$  are skew, (b) a plane, if  $\ell_1$  and  $\ell_2$  are parallel, or (c) a pair of orthogonal planes, if  $\ell_1$  and  $\ell_2$  are concurrent. In the latter case, the singular locus of the bisector is a line that perpendicularly intersects  $\ell_1$  and  $\ell_2$  in their intersection.*

The main theorem of Everett et al. [16] provides a good overview of the different cases of the trisector:

**Theorem 1 (Everett et al.).** *The trisector of three lines is either (i) a non-singular quartic, if the three lines are pairwise skew but not all parallel to a common plane nor lie on the surface of a hyperboloid of revolution, (ii) a cubic and a line that do not intersect, if the three lines are pairwise skew and lie on the surface of a hyperboloid of revolution, (iii) a nodal quartic, if the three lines are pairwise skew and all parallel to a common plane, (iv) one parabola or hyperbola, if there is exactly one pair of coplanar lines which are parallel, (v) two parabolas or hyperbolas that intersect, if there is exactly one pair of coplanar lines that intersect, (vi) between 0 and 4 lines, if there are two pairs of coplanar lines, or (vii) one line, in the case of three coplanar concurrent lines, the common singular locus of the bisectors.*

We use a corollary of the above theorem in Section 3, where we describe the construction of a Voronoi cell in the diagram of lines.

## 2.2 Lower Envelope Algorithm

Again, we regard the boundary of each three-dimensional VC as a lower envelope with respect to its line site  $\ell_0$ . This lower envelope is represented as a minimization diagram which is conceptually embedded in the  $uv$ -parameter space of the surface of an infinitesimally small cylinder around  $\ell_0$ .<sup>3</sup> We utilize the divide-and-conquer algorithm for constructing lower envelopes [1] as it is implemented in CGAL [33, §8.5], which we briefly describe next.

Since the algorithm projects bisectors into the parameter space, all bisectors are initially split up into  $uv$ -monotone surfaces. The algorithm then splits the resulting set  $\mathcal{G}$  into two subsets  $\mathcal{G}_1$  and  $\mathcal{G}_2$  of roughly equal size, and recursively computes their minimization diagrams  $\mathcal{M}_1$  and  $\mathcal{M}_2$ . In the conquer step, the two diagrams are merged into one. First, the overlay of  $\mathcal{M}_1$  and  $\mathcal{M}_2$  is computed, where each feature is labeled with up to two sets of labels  $L_1$  and  $L_2$  of candidate surfaces from both diagrams. Thereafter, the arrangement is further refined such that each feature can either be labeled with  $L_1$ ,  $L_2$ , or  $L_1 \cup L_2$ . In particular, each face that is labeled with two bisectors is refined by the corresponding projected trisector curve. Note that this step can also split up edges. After the comparison of bisectors the algorithm removes redundant edges and vertices, which yields the final diagram. The complexity of the above algorithm is  $O(n^{2+\varepsilon})$ , with the condition that the bisector surfaces are “well-behaved”.

Note that the algorithm heavily relies on arrangement operations such as overlay, which are provided by [33, §8.1] and [4]. Though, we treat these as a black

---

<sup>3</sup> See Section 3 for details on the  $uv$ -parameter space setting.

box throughout most of the paper, some details can be found in Section 5. The additional constructions and predicates required by the lower envelope algorithm are: the construction of the projected boundary of  $uv$ -monotone surfaces, the construction of the projected intersection of two  $uv$ -monotone surfaces, and the comparison of two bisectors above a face, an edge, or a vertex.

### 3 Computing a Voronoi Cell

This section discusses the computation of the  $VC$  of one line, referred to as the base line and denoted by  $\ell_0$ .

CGAL’s arrangement package has the infrastructure to compute envelopes over cylinders. However, for the efficiency of the implementation it is important to keep the algebraic degree of the projected curves as low as possible. Therefore, we project the curves on two parallel planes that “sandwich” the base line, while keeping the projection direction normal to the cylinder. This reduces the maximum degree of a projected trisector curve from sixteen down to eight.

#### 3.1 Parametrization and Projection

Let  $F = \{\vec{b}_1, \vec{b}_2, \vec{b}_3\}$  be an orthogonal basis of  $\mathbb{R}^3$  which is chosen such that  $\vec{b}_1$  is the direction of the base line  $\ell_0$ . Moreover, let  $p_0$  be some rational point on  $\ell_0$ . Now, consider the parametrization  $\mathcal{X}(u, v, r) = p_0 + u \cdot \vec{b}_1 + v \cdot \vec{b}_2 + r \cdot \vec{b}_3$ .  $\mathcal{X}(u, v, \pm 1)$  defines two parallel planes ( $uv$ -planes) that sandwich  $\ell_0$ . A point  $\mathcal{X}(u_0, v_0, \pm 1)$  represents a ray that originates from point  $p_0 + u_0 \cdot \vec{b}_1$  on  $\ell_0$  with direction  $\pm(v_0 \cdot \vec{b}_2 + \vec{b}_3)$ . Projecting along these rays onto  $\mathcal{X}(u, v, \pm 1)$ , we denote  $\mathcal{X}(u, v, \pm 1)$  as the positive and the negative projection plane, respectively.

Note that the plane  $H^* = \{x \in \mathbb{R}^3 | (x - p_0)^T \cdot \vec{b}_3 = 0\}$  is not covered by the parametrization. But it is straightforward to glue the two minimization diagrams on the two planes together as long as the chosen frame  $F$  is *generic*, that is, curves are not allowed to touch  $H^*$ , intersect in  $H^*$ , or even be contained in  $H^*$ . However, curves are of course allowed to transversely intersect  $H^*$ , each intersection giving rise to a single vertical asymptote on each projection plane.

In order to avoid these critical cases, we generate the local frame by setting  $\vec{b}_2$  to some random vector that is orthogonal to  $\vec{b}_1$ . Though this frame is generic with high probability, we also check in all relevant predicates that the frame is indeed generic. If necessary, we restart the computation choosing another random frame. We chose the standard strategy that increases the number of random bits used for each iteration. This way we guarantee termination and a small number of additional bits due to the randomization.

We highlight below several major issues in the projection of a trisector. The projection of a bisectors’ boundary and a detailed case analysis is not addressed due to lack of space. We rely merely on the generic frame and on the following corollary that directly follows from Theorem 1.

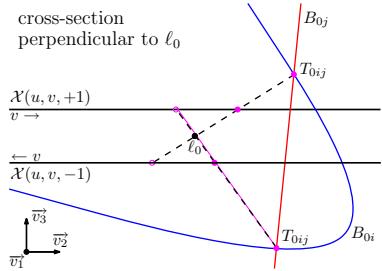
**Corollary 1.** *The set of points where the trisector does not represent a transversal intersection of the bisectors is a 0-dimensional set, namely, the singular*

points of the trisector. The only exception is the case of three coplanar concurrent lines; in this case the trisector is the common singular locus (line) of the three bisectors.

For a trisector  $T_{0ij}$  let  $B_{0i}, B_{0j}, B_{0ij} \in \mathbb{Q}[x_1, x_2, x_3]$  be the three trivariate polynomials of the relevant bisectors. Now let  $B_1$  and  $B_2$  be the two bisectors of minimal degree,  $d_1$  and  $d_2$ , respectively. The projection is carried out by a resultant computation [18]. Since we wish to project towards  $\ell_0$  we first substitute  $\mathcal{X}(u, v, r)$  into  $B_1$  and  $B_2$  and compute the resultant with respect to  $r$ .

$$\text{res}(u, v) := \text{resultant}(B_1(\mathcal{X}(u, v, r)), B_2(\mathcal{X}(u, v, r)), r) \in \mathbb{Q}[u, v].$$

This is at most a bivariate polynomial of degree  $2d_1d_2$ . Thus, in the worst case (the generic case) this is an irreducible polynomial of degree<sup>4</sup> only 8. However, due to its algebraic nature the approach can not immediately distinguish between the parameter spaces of the positive and the negative planes. The Figure to the right illustrates how the resultant projects  $T_{0ij}$  into the positive and negative plane. We first split up the projected curve into  $u$ -monotone arcs using [4]. In particular, curves are split up at vertical asymptotes. In order to decide that an arc  $\alpha$  is on a certain plain we utilize Corollary 1, namely the observation that in all but one exception (which is handled explicitly) two bisectors must intersect transversely along the trisector curve, which implies that  $B_{0i}$  and  $B_{0j}$  must interchange their order while passing the projected trisector.



This is detected by two ray shoots at rational points right above and below  $\alpha$ . Let  $\bar{p}$  and  $p$  be these two points, respectively. To ensure that both points are chosen sufficiently close, we construct a rational vertical line  $L$  that intersects  $\alpha$  in its interior, say at point  $p_\alpha$ . We choose the points on  $L$  such that they isolate the arc from all other intersections of  $L$  with  $\text{res}$ . Now consider the path on  $L$  from  $\bar{p}$  (or  $p$ ) to  $p_\alpha$ .  $\bar{p}$  is sufficiently close to  $\alpha$  since this path does not intersect  $\text{res}$  until it reaches  $\alpha$ . In case  $\alpha$  is vertical, we choose  $L$  to be horizontal.

### 3.2 Lower Envelope Predicates

A core part of the envelope algorithm is the representation of minimization diagrams as labeled arrangements and the overlay of such arrangements. The required constructions and predicates for these operations relate to planar algebraic curves only, which are provided by [4]. However, it remains to ensure that no intersection takes place in  $H^*$ . This boils down to testing that the leading

<sup>4</sup> More precisely, it is a bivariate polynomial of bi-degree at most (4, 4). For a standard rational parametrization of the cylinder, we would obtain a polynomial of bi-degree (8, 8) or 16 in total.

coefficients with respect to  $v$  of two non overlapping (co-prime) curves have no common root. Thus, we provide a slightly modified set of operations that ensure this condition in addition.

The remaining predicates that are required by the envelope algorithm are the comparison of two bisectors above a face, an edge, or a vertex, respectively; see also Section 2.2. For a vertex, which may not have rational coordinates, we first check whether the point is on the projected intersection of the two bisectors and report equality if it is indeed the case. Otherwise it is sufficient to compute a rational point that is close enough to the vertex, and to compare the surfaces along the corresponding ray. In order to compare above an edge we construct a vertex in its interior and compare at that vertex as described above. For a face, it is sufficient to compare (again via ray-shooting) at a rational point in its interior. In each case, the rational point is constructed using strategies similar to the one discussed at the end of Section 3.1.

### 3.3 Complexity

For the time and space complexity analysis we ignore additional costs that may arise due to variable bit-length of various implementations adhering to the exact computation paradigm [36]. We also ignore the additional run-time that can result from a poor choice of a generic frame (Section 3.1), as it is not the general case, and has no impact on performance in expectation.

The bisector surfaces are algebraic and thus comply with the definition of “well-behaved” surfaces required in [32]. Thus, the time-complexity of the lower envelope algorithm is  $O(n^{2+\varepsilon})$  (which is also the best known upper bound). Overall, the run-time complexity of computing the cells for all  $n$  lines is  $O(n^{3+\varepsilon})$ , which also bounds the space complexity.

## 4 Fast Point Location

Given a query point  $q$  we wish to find the closest line to it. Consider the following point-location strategy: We start with a random line site  $\ell$ . First we project  $q$  on  $\ell$  and locate its image in the minimization diagram of  $\ell$ . The image is located on a feature of the minimization diagram which is labeled with a (in general not empty) set of line sites  $\mathcal{S}$ . We then compare the distance  $d(\ell, q)$  to  $d(\ell', q)$  for one line  $\ell' \in \mathcal{S}$ . If  $d(\ell, q)$  is less than or equal to  $d(\ell', q)$  we report  $\ell$  or  $\mathcal{S} \cup \ell$ , respectively. Otherwise we continue in the cell of  $\ell'$ . This walk through the Voronoi diagram terminates since there is only a finite number of cells and the distance of  $q$  to the current line always decreases. We can locate the image of  $q$  inside the minimization diagram in expected  $O(\log n)$  time by using point-location that is based on trapezoidal decomposition [28]. Combining this algorithm with the idea of landmarks [21] may already have good performance in practice. However, the algorithm has a worst-case time complexity  $O(n \log n)$ .

We turn it into an algorithm with a time-complexity  $O(\log^2 n)$  by combining it with a strategy that is similar to skip lists. We build a hierarchy of Voronoi diagrams. The lowest layer contains the *VD* of the full set of lines. The lines for the other layers are from the previous layer, each chosen with probability  $1/k$ , where  $k > 1$  is constant. The highest layer (the root layer) contains only a constant number of lines ( $\leq k$ ). The expected number of layers is  $O(\log n)$ . In order to locate a point  $q$  we first locate it in the root layer using the walk strategy described above. We then proceed to the next layer starting at the line that was found in the preceding layer.

We remark that some special cases are left out in this discussion for brevity (e.g., query points in  $H^*$ ), but they are completely handled in our software. The following theorem summarizes the performance of the point-location structure (see [10] and [24] for a similar analysis in 2D):

**Theorem 2.** *For any query point  $q$  the expected running time of the point-location query in the hierarchical VD structure is  $O(\log^2 n)$ .*

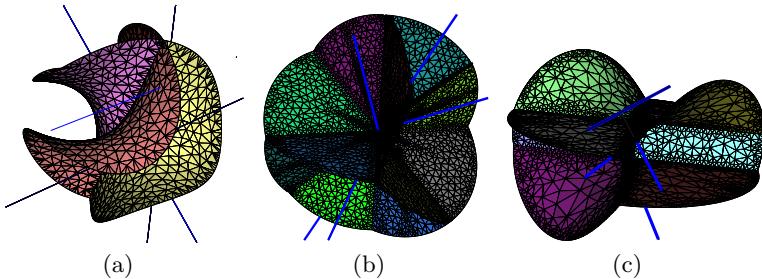
*Proof.* The number of cells visited at the root layer is obviously at most  $k$ . For all other layers, consider the path backward, from its target to the source: For every cell the probability that it is already the source is  $1/k$ . Thus, the expected length of a path is  $\sum_{i=1}^n \frac{i}{k} \left(\frac{k-1}{k}\right)^{i-1} \leq k$ .

That is, the expected running-time is  $k \sum_{i=1}^{\log_k n} T(k^i)$ , where  $T(m)$  is the expected time spent on the point location in the minimization diagram of  $m$  lines. Thus we obtain an expected running-time of  $O(\log^2 n)$  in total.

## 5 Implementation Details

Our implementation is based on CGAL, which follows the generic-programming paradigm [3]. Algorithms are formulated and implemented such that they are abstract from the actual types, constructions, and predicates. Thus, the implementation of every algorithm and data structure in CGAL is parametrized by a so-called traits class [29], in which these functionalities are defined. In particular, users can employ an algorithm with their own types, constructions, and predicates by providing their own traits class. This way it is possible to achieve a great amount of flexibility. At the extreme, it is possible to even partially change the nature of an algorithm, as we do here for the three-dimensional lower envelope class [33, §8.5].

The core of our implementation is the traits class for the lower envelope algorithm, which also needs to be a valid traits class for CGAL’s arrangement package. The required functionalities by the arrangement package are provided by the traits class presented in [4]. The approach reduces all construction and predicates to cylindrical algebraic decompositions of the plane for one or two curves. We essentially wrap this traits class and add the auxiliary functionalities required by the envelope algorithm; see also Section 3. In case we detect that the current frame is not generic an exception is thrown, which is then caught by our primary class that computes a new frame and restarts the computation



**Fig. 2.** Degenerate Voronoi diagrams of lines. The diagrams are clipped by a sphere for convenience. (a) VD of 4 lines, obtained by rotating one line around the  $z$ -axis. All bisectors meet in that axis. (b) VD of 4 lines intersecting in one point. (c) VD of 4 lines, two lines intersect and the others are parallel to each of them, respectively.

of the cell. For each Voronoi cell we keep a separate instance of the traits class, which is used for both planes. This allows caching of relevant results.

Approximation of the three-dimensional coordinates of a vertex, is based on multi-precision floating-point interval arithmetic (MPFI) [8, §8]. Since this is a certified approximation, we obtain a bounding box that contains the vertex. This could be used to easily establish the adjacency among lower dimensional cells. For instance, let  $v$  denote a vertex in a minimization diagram  $\mathcal{M}$ . The label of  $v$  points to all other minimization diagrams that contain a representation of it. Let  $\mathcal{M}'$  be one of these diagrams and  $v'$  be the representation of  $v$  that we wish to find therein. We could use a similar approach to the one used in [12]: By using the labels, we identify all possible candidates in  $\mathcal{M}'$ . Since the bisector surfaces are at most of algebraic degree two, this set contains only up to 8 representations and contains at least  $v'$ . We progressively compute more precise bounding boxes for all candidates until only one (the one of  $v'$ ) overlaps the bounding box of  $v$ .

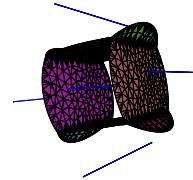
Our implementation can handle arbitrary rational lines, in particular, it can handle all possible degenerate cases. Figure 2 depicts degenerate Voronoi diagrams. Each mesh was generated using CGAL's package for labeled mesh domains [30]. The oracle, which is required by the mesh generation, was written such that it only utilizes (and thereby tests) our point location structure. We used `medit` [22] for the final visualization.

Since we aim to eventually incorporate our code into a CGAL package the software is developed within the revision control system of the project. All experiments within this section were carried out on an internal CGAL release CGAL-3.7-Ic-27, which already comprises all the necessary algebraic tools [4]. However, the trapezoidal map is currently not available for minimization diagrams due to ongoing changes in the arrangement package (it is anticipated soon), which forces us to resort to simpler point location strategies for now.

Finally, we present preliminary results obtained with our software. The point location structure as it is discussed in Section 4 leaves the ratio  $k$  among levels undetermined. In order to show the impact of  $k$  we created random instances of

**Table 1.** Average number of visited cells per query, where  $k$  denotes the ratio of the hierarchy and  $N$  the number of lines. Cases where the hierarchy would only consist of one level are marked with 'n/a', the times would correspond to those in the last column. To the right is depicted a Voronoi diagram of 5 parallel lines.

$N \setminus k$	2	4	8	12	16	20	24	28	$+\infty$
16	6.48	4.30	4.34	n/a	n/a	n/a	n/a	n/a	<b>3.94</b>
36	8.09	6.33	<b>5.33</b>	6.23	5.67	n/a	n/a	n/a	<b>5.62</b>
64	9.77	6.42	<b>5.73</b>	7.34	6.07	6.00	6.12	6.83	6.63
100	9.87	7.22	<b>6.18</b>	7.10	6.45	6.97	6.83	7.13	7.43
144	11.56	8.14	8.47	<b>7.46</b>	7.81	9.64	9.75	8.68	12.72



parallel lines<sup>5</sup> with coefficients in the range  $[0, 2^{10}]$ . For each instance, we created 10 Voronoi diagram hierarchies, which were queried with 1000 random points in  $[0, 2^{10}]^3$  each.

Table I shows the average number of visited cells per query depending on the number of lines and the chosen value for  $k$ . The last column shows the pure walk without a hierarchy, which suggests (for the case of parallel lines) an average query time in  $O(\sqrt{n})$ , as one may also expect due to results in [11]. For larger instances, it seems that choosing  $k$  between 8 and 12 is appropriate.

## 6 Conclusions

We have presented an exact, complete, and thus robust, algorithm that computes the Voronoi diagram of arbitrary rational lines in  $\mathbb{R}^3$ . The algorithm requires  $O(n^{3+\varepsilon})$  time and space, where  $n$  is the number of lines. The introduced data structure admits answering point-location queries in  $O(\log^2 n)$  expected time. The implemented prototype is exact and can handle all degenerate cases.<sup>6</sup>

The algorithm is intentionally designed such that it avoids tedious case distinctions, which makes it implementable, maintainable and, in particular, extensible to other primitives such as points, line segments, and triangles. Thus, we consider our approach as a major milestone towards the exact computation of the Voronoi diagram of polyhedra in three dimensions.

The approach may also be generalized to spheres (see also [20]) which would open the door for innovative solutions to central problems in Structural Biology [25], [35]. Moreover, we expect that it will pave the way to devising a three-dimensional variant of the visibility-Voronoi complex [34], a structure that enables to trade-off clearance and path length in robot motion planning, and has proved to be especially useful in the plane.

**Acknowledgments.** We thank S. Lazard and M. Yvinec for fruitful discussions.

<sup>5</sup> Since the trapezoidal map is not yet available for envelopes, we had to resort to instances that keep the complexity of a cell small.

<sup>6</sup> For the most recent version and supplemental material we refer to: <http://acg.cs.tau.ac.il/projects/internal-projects/3d-lines-vor/project-page>

## References

1. Agarwal, P.K., Schwarzkopf, O., Sharir, M.: The overlay of lower envelopes and its applications. *Disc. Comput. Geom.* 15(1), 1–13 (1996)
2. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J., Urrutia, G. (eds.) *Handb. Comput. Geom.*, ch. 5, pp. 201–290. Elsevier, Amsterdam (2000)
3. Austern, M.H.: *Generic Programming and the STL*. Addison-Wesley, Reading (1999)
4. Berberich, E., Hemmer, M., Kerber, M.: A generic algebraic kernel for non-linear geometric applications. *Research Report 7274*, INRIA (2010)
5. Berberich, E., Hemmer, M., Kettner, L., Schömer, E., Wolpert, N.: An exact, complete and efficient implementation for computing planar maps of quadric intersection curves. In: Mitchell, J., Rote, G., Kettner, L. (eds.) *Proc. 21st Annu. ACM Symp. Comput. Geom.*, pp. 99–106. ACM Press, Pisa (2005)
6. Blum, H.: A transformation for extracting new descriptors of shape. In: Wathen-Dunn, W. (ed.) *Models for the Perception of Speech and Visual Form*. MIT Press, Cambridge (1967)
7. Boissonnat, J.D., Delage, C.: Convex hull and Voronoi diagram of additively weighted points. In: Brodal, G.S., Leonardi, S. (eds.) *ESA 2005*. LNCS, vol. 3669, pp. 367–378. Springer, Heidelberg (2005)
8. Boissonnat, J.D., Teillaud, M. (eds.): *Effective Computational Geometry for Curves and Surfaces*. Mathematics and Visualization. Springer, Heidelberg (2006)
9. Culver, T., Keyser, J., Manocha, D.: Exact computation of the medial axis of a polyhedron. *Computer Aided Geometric Design* 21(1), 65–98 (2004)
10. Devillers, O.: Improved incremental randomized Delaunay triangulation. In: *Proc. 14th Annu. ACM Symp. Comput. Geom.*, pp. 106–115. ACM Press, New York (1998)
11. Devroye, L., Lemaire, C., Moreau, J.M.: Expected time analysis for Delaunay point location. *Computational Geometry* 29(2), 61–89 (2004)
12. Dupont, L., Hemmer, M., Petitjean, S., Schömer, E.: Complete, exact and efficient implementation for computing the adjacency graph of an arrangement of quadrics. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007*. LNCS, vol. 4698, pp. 633–644. Springer, Heidelberg (2007)
13. Edelsbrunner, H., Seidel, R.: Voronoi diagrams and arrangements. *Disc. Comput. Geom.* 1, 25–44 (1986)
14. Emiris, I.Z., Tsigaridas, E.P., Tzoumas, G.M.: The predicates for the Voronoi diagram of ellipses. In: *Proc. 22nd Annu. ACM Symp. Comput. Geom.*, pp. 227–236. ACM Press, New York (2006)
15. Emiris, I.Z., Karavelas, M.I.: The predicates of the Apollonius diagram: Algorithmic analysis and implementation. *Comput. Geom. Theory Appl.* 33(1-2), 18–57 (2006)
16. Everett, H., Gillot, C., Lazard, D., Lazard, S., Pouget, M.: The Voronoi diagram of three arbitrary lines in  $\mathbb{R}^3$ . In: *Abstracts of 25th Eur. Workshop Comput. Geom.* (2009)
17. Everett, H., Lazard, S., Lazard, D., Din, M.S.E.: The Voronoi diagram of three lines. In: *Proc. 23rd Annu. ACM Symp. Comput. Geom.*, pp. 255–264. ACM Press, New York (2007)
18. von zur Gathen, J., Gerhard, J.: *Modern Computer Algebra*. Cambridge University Press, Cambridge (1999)

19. Halperin, D., Kavraki, L.E., Latombe, J.C.: Robotics. In: Goodman, J.E., O'Rourke, J. (eds.) *Handb. Disc. Comput. Geom.*, 2nd edn., ch. 48, pp. 1065–1093. Chapman & Hall/CRC, Boca Raton (2004)
20. Hanniel, I., Elber, G.: Computing the Voronoi cells of planes, spheres and cylinders in  $\mathbb{R}^3$ . *Comput. Aided Geom. Des.* 26(6), 695–710 (2009)
21. Haran, I., Halperin, D.: An experimental study of point location in planar arrangements in CGAL. *ACM Journal of Experimental Algorithms* 13 (2008)
22. Frey, P.J.: MEDIT : An interactive Mesh visualization Software. Technical Report RT-0253, INRIA (December 2001)
23. Karavelas, M.I.: A robust and effient implementation for the segment Voronoi diagram. In: Int. Symp. on Voronoi Diagrams in Sci. and Engineering, pp. 51–62 (2004)
24. Karavelas, M.I., Yvinec, M.: Dynamic additively weighted Voronoi diagrams in 2D. In: Proc. 10th Annu. Eur. Symp. Alg., pp. 586–598. Springer, London (2002)
25. Kim, D.S., Seo, J., Kim, D., Cho, Y., Ryu, J.: The beta-shape and beta-complex for analysis of molecular structures. In: Gavrilova, M.L. (ed.) *Generalized Voronoi Diagram: A Geometry-Based Approach to Computational Intelligence*. Studies in Computational Intelligence, vol. 158, pp. 47–66. Springer, Heidelberg (2008)
26. Koltun, V., Sharir, M.: 3-dimensional Euclidean Voronoi diagrams of lines with a fixed number of orientations. *SIAM J. on Computing* 32(3), 616–642 (2003)
27. Milenkovic, V.: Robust construction of the Voronoi diagram of a polyhedron. In: Proc. 5th Canad. Conf. Comput. Geom., pp. 473–478 (1993)
28. Mulmuley, K.: A fast planar partition algorithm, I. In: Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci., pp. 580–589 (1988)
29. Myers, N.: Traits: A new and useful template technique. *C++ Gems* 17 (1995)
30. Rineau, L., Yvinec, M.: 3D surface mesh generation. In: CGAL Editorial Board *CGAL User and Reference Manual* (ed.), 3.5 edn. (2009)
31. Setter, O., Sharir, M., Halperin, D.: Constructing two-dimensional Voronoi diagrams via divide-and-conquer of envelopes in space. *Transactions on Computational Sciences* (to appear, 2010)
32. Sharir, M.: Almost tight upper bounds for lower envelopes in higher dimensions. *Disc. Comput. Geom.* 12(1), 327–345 (1994)
33. The CGAL Project: CGAL User and Reference Manual. CGAL Editorial Board, 3.6 edn. (2010), <http://www.cgal.org/>
34. Wein, R., van den Berg, J.P., Halperin, D.: The visibility-Voronoi complex and its applications. *Computational Geometry: Theory and Applications* 36(1), 66–87 (2007); special Issue on the 21st European Workshop on Computational Geometry - EWCG 2005
35. Yaffe, E., Halperin, D.: Approximating the pathway axis and the persistence diagram of a collection of balls in 3-space. In: Proc. 24th Annu. ACM Symp. Comput. Geom., pp. 260–269. ACM Press, New York (2008)
36. Yap, C.K., Dubé, T.: The exact computation paradigm. In: Du, D.Z., Hwang, F.K. (eds.) *Computing in Euclidean Geometry*, 2nd edn. LNCS, vol. 1, pp. 452–492. World Scientific, Singapore (1995)

# Local Graph Exploration and Fast Property Testing<sup>\*</sup>

Artur Czumaj

Centre for Discrete Mathematics and its Applications (DIMAP) and

Department of Computer Science, University of Warwick, UK

[A.Czumaj@warwick.ac.uk](mailto:A.Czumaj@warwick.ac.uk)

**Abstract.** We will present some recent results about testing graph properties in sparse graphs and will discuss graph exploration techniques which allow very efficient algorithms for testing graph properties.

## 1 Introduction

*Property testing* is a relaxation of classical decision problems, in which one wants to distinguish objects (for example, graphs, functions, or point sets) that have a given predetermined property (for example, being bipartite, monotone, or in convex position) from those that are *far* from this property. Property testing is motivated by the need to understand how to obtain information from massive structured or semi-structured data sets using small random samples. The notion of property testing was first explicitly formulated by Rubinfeld and Sudan [25] and it arises naturally in the context of program verification [425], learning theory, etc. Goldreich et al. [14] initiated the study of property testing for combinatorial objects, and in the recent years we have seen a number of property testing algorithms to test functions, probability distributions, graph and hypergraph properties, properties of languages, etc. (for references, see the surveys [8][13][23][24]).

One of the main directions in property testing is that of testing graph properties, as introduced by Goldreich, Goldwasser, and Ron [14][15]. The goal is for a given graph  $G = (V, E)$  and a given property  $P$  (e.g., being bipartite), to decide if  $G$  satisfies property  $P$  or  $G$  is  $\varepsilon$ -far<sup>1</sup> from property  $P$ . Here, informally, we say  $G$  is  $\varepsilon$ -far from property  $P$  if it differs in an  $\varepsilon$ -fraction of its description from any graph having the property  $P$ . The tester is usually randomized and we allow it to err with probability at most  $\frac{1}{3}$ ; if it always accepts any  $G$  that satisfies  $P$  then the tester has *one-sided error*; otherwise, it has *two-sided error*.

Property testing can be seen as approximation algorithms for decision problems. Since we only want to “approximately decide” problems, it is often possible to obtain algorithms that are much more efficient than their exact counterparts.

---

\* Research supported by EPSRC award EP/G064679/1 and by the Centre for Discrete Mathematics and its Applications (DIMAP), EPSRC award EP/D063191/1.

<sup>1</sup> Normally one considers  $\varepsilon$  as being a small constant, independent of the input.

As it has been demonstrated in numerous papers in the last decade, many property testers have indeed running times that are sublinear in the input size; for some problems it is even possible to obtain property testers whose complexity is *independent of the input size*. This, in turn, resulted in the recent development of sublinear-time approximation algorithms for many classical combinatorial problems, including dense max-cut, clustering problems, and estimating the cost of the minimum spanning tree and maximum matching (see, e.g., [5, 6, 9, 10, 12, 14, 22] and [8]).

### 1.1 Testing Properties of Dense Graphs

Since the notion of being  $\varepsilon$ -far depends on the input representation, we consider here two most popular models for analyzing graph properties. We begin with the *adjacency matrix model* [14], where the input graph  $G = (V, E)$  on  $n$  vertices is represented by its adjacency matrix of size  $n^2$ ; then  $G$  is  $\varepsilon$ -far from property  $P$  if one has to modify at least  $\varepsilon n^2$  entries in the adjacency matrix to obtain a graph satisfying property  $P$ . (Since we consider here graphs obtained by changing  $\mathcal{O}(\varepsilon n^2)$  edges of the original graph  $G$ , it is easy to see that this definition is suitable mostly for dense graphs.) For such input representation, the complexity of a testing algorithm is measured by the number of queries to the adjacency matrix of  $G$ .

This model has been extensively studied in the last decade and it is now quite well understood [3]. After a series of papers, we know that in the adjacency matrix model, testability of a property in constant time (independent on the input size, but possibly depending on  $\varepsilon$ ) is closely related to Szemerédi partitions of the graph. In fact, it has been shown that a graph property is testable in time independent of the size of  $G$  if and only if it can be reduced to testing finitely many Szemerédi partitions [1]. Moreover, Alon and Shapira [2] show that any natural graph property is testable in time independent of  $n$  with one-sided error if and only if it is either hereditary or it is close (in some well-defined sense) to a hereditary property.

### 1.2 Testing Properties of Sparse Graphs and Graph Exploration

While property testing in the adjacency matrix model for dense graphs is now relatively well understood, much less is known about testing properties of sparse (or arbitrary) graphs in the *adjacency list model*.

Properties of sparse graphs have been traditionally studied in the model of *bounded-degree* graphs introduced by Goldreich and Ron [15]. In this model, the input graph  $G = (V, E)$  is represented by its adjacency list (or, equivalently, by its incidence list) and the vertex degrees are bounded by a constant  $d$  independent of the number of vertices of  $G$ . A testing algorithm has a constant-time access to any entry in the adjacency list by making a query to the  $i^{\text{th}}$  neighbor of a given vertex, and the number of accesses to the adjacency list is the (query) *complexity of the tester*. A property testing algorithm is an algorithm that for a given graph  $G$  determines if it satisfies a predetermined property  $P$  or it is  $\varepsilon$ -far

from property  $P$ ; a graph  $G$  is  $\varepsilon$ -far from property  $P$  if one has to modify more than  $\varepsilon dn$  edges in  $G$  to obtain a graph having property  $P$ .

We will now briefly discuss central results for testing properties of sparse graphs and basic techniques used to efficiently test properties of sparse graphs.

Goldreich and Ron [15][16][17] were the first to study properties of sparse graphs. They show that although some basic properties (eg., being connected,  $k$ -connected, or Eulerian) can be tested in time independent of  $n$ , a number of fundamental properties require a superconstant testing time. For example, it has been shown that testing if a graph  $G$  is bipartite requires  $\Omega(\sqrt{n})$  time [15]. (One needs  $\Omega(\sqrt{n})$  time to distinguish between random graphs from the following two classes: (i) sum of a Hamiltonian cycle  $H$  and a perfect matching, and (ii) sum of a Hamiltonian cycle  $H$  and a perfect matching  $M$  such that each edge from  $M$  creates an even-length cycle when added to  $H$ . Since the latter class consists only of bipartite graphs and a random graph from the former class is with high probability  $\varepsilon$ -far from bipartite, the lower bound follows.) Similar bounds are known for testing if a graph is a good expander, is  $k$ -colorable, etc.

Comparing to the results for the adjacency matrix model, the adjacency list model has much more algorithmic flavor. In particular, to test if a given graph has a predetermined property one typically requires to do much more than just a simple sampling of vertices; many testing algorithms require some graph exploration algorithms to collect information about local and global properties of graphs. Indeed, the main three techniques are random sampling, local search (exploring the neighborhood of a vertex), and random walks.

The aforementioned lower bound for testing bipartiteness was complemented by an upper bound of  $\tilde{O}(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$  time for testing bipartiteness in [16]. The algorithm by Goldreich and Ron is very representative for the area:

### Testing Bipartiteness:

- Pick a random sample  $S$  of  $\mathcal{O}(1/\varepsilon)$  vertices
- For each  $v \in S$ :
  - perform  $\text{poly}(\log n/\varepsilon)\sqrt{n}$  random walks from  $v$ , each of length  $\text{poly}(\log n/\varepsilon)$
  - If the graph induced by all edges visited is not bipartite **then reject**
- If the algorithm did not reject yet **then accept**

The analysis of the algorithm is very elaborate and it establishes an interesting connection with the analysis of the convergence times of Markov chains. Similar techniques of exploring the input graph by many independent random walks have been used in several recent works, for example, in  $\tilde{O}(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$ -time algorithms for testing if a graph is a good expander [11][19][21] (see also [17]).

A new approach has been proposed recently by Czumaj, Sohler, and Shapira [7], who consider testing graph properties not for all graphs, but rather for some specific classes of graphs. For example, they show that if the underlying graph is

planar, then any hereditary graph property (e.g., bipartiteness,  $k$ -colorability, or perfectness) is testable in time independent of the input size. This approach can be generalized to any class of graphs that can be partitioned into constant size components by removing  $\varepsilon n$  edges of the graph, for any  $\varepsilon > 0$ ; we call such graphs *hyperfinite*. Benjamini et al. [3] extended this result and show that every minor-closed graph property is testable in time independent of the input size in general bounded-degree graphs (with two-sided error); this is shown by first proving that hyperfiniteness is testable for general bounded-degree graphs, and then by observing that every minor-closed graph property is hyperfinite. In particular, testing if a graph is planar can be done in time independent on  $n$  with two-sided error. This result (the complexity) has been improved by Hassidim et al. [18]; in particular, testing if a bounded degree graph is planar can be done with  $2^{\text{poly}(1/\varepsilon)}$  queries. Hassidim et al. [18] show also how to apply similar techniques to not only test graph properties, but also to approximate the distance to almost any hereditary property in any bounded degree hereditary families of graphs.

**Future directions and challenges.** Despite seeing a lot of progress in the last several years, we still have only a partial picture of the complexity of testing graph properties in the model of sparse graphs. For example, we know that in the two-sided error model, testing planarity can be done in time  $2^{\text{poly}(1/\varepsilon)}$ ; can we do it in time polynomial in  $1/\varepsilon$ ? Or, what is the complexity of testing planarity in the one-sided error model, where it is conjectured that the complexity is  $\Theta(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$ . Similar question can be asked for testing other minor-closed properties. In fact, a more ambitious challenge would be to provide a characterization of properties of bounded degree graphs represented by adjacency lists that can be tested in constant time, or in  $\Theta(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$  time. In this flavor, it has been shown recently that testing (with one-sided error) the property of having an  $H$ -minor can be done in time independent of  $n$  if and only if  $H$  is cycle-free.

The results for sparse graphs mentioned above are dealing solely with bounded-degree graphs. While the model of bounded-degree graphs is elegant and very natural, it is also desirable to consider the model of arbitrary graphs represented by adjacency list (see, e.g., [20]). How quickly can we test basic graph properties in this model? For example, it is easy to see that testing planarity require  $\Omega(\sqrt{n})$ ; can we design an algorithm that could match this bound?

## References

1. Alon, N., Fischer, E., Newman, I., Shapira, A.: A combinatorial characterization of the testable graph properties: it's all about regularity. *SIAM Journal on Computing* 39(1), 143–167 (2009)
2. Alon, N., Shapira, A.: A characterization of the (natural) graph properties testable with one-sided error. *SIAM Journal on Computing* 37(6), 1703–1727 (2008)
3. Benjamini, I., Schramm, O., Shapira, A.: Every minor-closed property of sparse graphs is testable. In: *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 393–402 (2008)
4. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences* 47(3), 549–595 (1993)

5. Chazelle, B., Rubinfeld, R., Trevisan, L.: Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing* 34(6), 1370–1379 (2005)
6. 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 Journal on Computing* 35(1), 91–109 (2005)
7. Czumaj, A., Shapira, A., Sohler, C.: Testing hereditary properties of nonexpanding bounded-degree graphs. *SIAM Journal on Computing* 38, 2499–2510 (2009)
8. Czumaj, A., Sohler, C.: Sublinear-time algorithms. *Bulletin of the EATCS* 89, 23–47 (2006)
9. Czumaj, A., Sohler, C.: Sublinear-time approximation algorithms for clustering via random sampling. *Random Structures and Algorithms* 30(1-2), 226–256 (2007)
10. Czumaj, A., Sohler, C.: Estimating the weight of metric minimum spanning trees in sublinear-time. *SIAM Journal on Computing* 39(3), 904–922 (2009)
11. Czumaj, A., Sohler, C.: Testing expansion in bounded-degree graphs. In: *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 570–578 (2007)
12. Frieze, A., Kannan, R.: The regularity lemma and approximation schemes for dense problems. In: *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 12–20 (1996)
13. Goldreich, O.: Introduction to testing graph properties. *Electronic Colloquium on Computational Complexity (ECCC)*, Report No. 082 (2010)
14. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connection to learning and approximation. *Journal of the ACM* 45(4), 653–750 (1998)
15. Goldreich, O., Ron, D.: Property testing in bounded degree graphs. *Algorithmica* 32(2), 302–343 (2002)
16. Goldreich, O., Ron, D.: A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica* 19(3), 335–373 (1999)
17. Goldreich, O., Ron, D.: On testing expansion in bounded-degree graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, Report No. 7 (2000)
18. Hassidim, A., Kelner, J.A., Nguyen, H.N., Onak, K.: Local graph partitions for approximation and testing. In: *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 22–31 (2009)
19. Kale, S., Seshadhri, C.: An expansion tester for bounded degree graphs. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I*. LNCS, vol. 5125, pp. 527–538. Springer, Heidelberg (2008)
20. Kaufman, T., Krivelevich, M., Ron, D.: Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing* 33(6), 1441–1483 (2004)
21. Nachmias, A., Shapira, A.: Testing the expansion of a graph. *Information and Computation* 208(4), 309–314 (2010)
22. Nguyen, H.N., Onak, K.: Constant-time approximation algorithms via local improvements. In: *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 327–336 (2008)
23. Ron, D.: Property testing. In: *Handobook of Randomized Algorithms*, vol. II, pp. 597–649 (2001)
24. Rubinfeld, R.: Sublinear time algorithms. In: *Proceedings of the International Congress of Mathematicians, ICM* (2006)
25. Rubinfeld, R., Sudan, M.: Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing* 25(2), 252–271 (1996)

# A Fully Compressed Algorithm for Computing the Edit Distance of Run-Length Encoded Strings<sup>\*</sup>

Kuan-Yu Chen<sup>1</sup> and Kun-Mao Chao<sup>1,2,3</sup>

<sup>1</sup> Department of Computer Science and Information Engineering

<sup>2</sup> Graduate Institute of Biomedical Electronics and Bioinformatics

<sup>3</sup> Graduate Institute of Networking and Multimedia

National Taiwan University, Taipei, Taiwan 106

**Abstract.** In this paper, a commonly used data compression scheme, called run-length encoding, is employed to speed up the computation of edit distance between two strings. Our algorithm is the first to achieve “fully compressed,” meaning that it runs in time polynomial in the number of runs of both strings. Specifically, given two strings, compressed into  $m$  and  $n$  runs,  $m \leq n$ , we present an  $O(mn^2)$ -time algorithm for computing the edit distance of the two strings. Our approach also gives the first fully compressed algorithm for approximate matching of a pattern of  $m$  runs in a text of  $n$  runs in  $O(mn^2)$  time.

## 1 Introduction

The edit distance (a.k.a Levenshtein distance) is a common similarity measure between two strings. It is defined as the minimum steps required to transform one string into the other via operations of insertions, deletions, or substitutions. The problem of computing the edit distance between two strings has been explored for decades. The classic dynamic programming solution takes  $O(N^2)$  time, where  $N$  denotes the length of both strings. All known techniques for breaking the  $O(N^2)$  time bound essentially follow the paradigm of *acceleration via compression* (see [8] for more details.). The first breakthrough to  $O(N^2 / \log N)$ -time was made by Masek and Paterson [12], who used the Four-Russians technique to speed up the edit-distance computation. The Four-Russians technique can be seen as a naïve compression utilizing the fact that sufficiently short substrings over a constant alphabet must appear many times. After that, Crochemore *et al.* [6] exploited LZ-factorization of strings and gave an  $O(hN^2 / \log N)$ -time algorithm, where  $h \leq 1$  is the entropy of the text. Their solution is general enough for the sequence alignment problem with unrestricted scoring matrices.

In this paper, we accelerate the edit-distance computation by exploiting symbol repetitions in strings. The underlying compression scheme is called *run-length*

---

\* Partially supported by NSC grants 97-2221-E-002-097-MY3 and 98-2221-E-002-081-MY3 from the National Science Council, Taiwan.

*encoding* (RLE), which is widely applied in many areas, e.g. FAX transmission, image compression, and optical character recognition. Previous results under this paradigm include the computation of indel-distance (dual of longest common subsequence) of two run-length encoded strings which requires  $O(mn \log mn)$  time [2][4], where  $m$  and  $n$  denote the number of runs of input strings of lengths  $M$  and  $N$ . Several papers showed how to compute the edit distance of two run-length encoded strings in  $O(Mn + mN)$  time [3][6][13]. Recently, Huang *et al.* [10] and Liu *et al.* [11] improved the bound to  $O(\min\{Mn, mN\})$  time. (It should be noted that the works of [3][11] focused on the Levenshtein distance in which each edit operation has a unit cost, while the results of [6][10][13] apply to the general edit distance problem with weighted costs.) To date, all the time bounds for the edit distance problem still depend on the uncompressed string lengths. Therefore, an intriguing question is (as asked in [10][13]) whether one can design an algorithm that is “fully compressed,” i.e. its time complexity depends solely on the number of runs of the RLE strings. For Levenshtein distance, this paper answers the question affirmatively by giving an  $O(mn^2)$ -time solution.

On the other hand, a closely related problem, called *compressed pattern matching* is, given a compressed text  $T$  and an uncompressed pattern  $P$ , to find all occurrences of  $P$  in  $T$  without decompressing  $T$ . If pattern  $P$  is also compressed, the problem is referred to as *fully compressed pattern matching*. Many studies have been made around this subject under different compression schemes, e.g. RLE compression, LZ-family compression, and straight-line programs. For RLE inputs, the exact string matching problem can be easily solved in  $O(m+n)$  time, where  $m$  and  $n$  denote the number of runs of the pattern and the text. The  $k$ -mismatch with wildcards problem can be solved in  $O(mn \log m)$  time [5]. The *approximate matching problem* seeks for occurrences of an RLE string within another RLE string, allowing up to  $k$  edit operations. For this problem, Mäkinen *et al.* [13] proposed an  $O(mNM)$ -time algorithm, where  $M$  denotes the uncompressed pattern length. Huang *et al.* later improved the bound to  $O(nM)$  time [10]. The approach presented in this paper gives the first fully compressed algorithm running in  $O(mn^2)$  time.

## 2 Preliminaries

### 2.1 Edit Graph

Given two strings  $A[1 \dots M]$  and  $B[1 \dots N]$ , the edit distance problem can be solved as follows. Initially,  $ED(i, 0) = i$  and  $ED(0, j) = j$  for  $0 \leq i \leq M$  and  $1 \leq j \leq N$ . For  $1 \leq i \leq M$  and  $1 \leq j \leq N$ ,  $ED(i, j) = \min\{ED(i - 1, j) + 1, ED(i, j - 1) + 1, ED(i - 1, j - 1) + \delta(A[i], B[j])\}$ , where  $\delta(A[i], B[j]) = 0$  if  $A[i]$  matches  $B[j]$ , and  $\delta(A[i], B[j]) = 1$  otherwise. The edit distance between  $A$  and  $B$  is the value of  $ED(M, N)$ . This dynamic programming solution can be represented in terms of a weighted, acyclic grid graph  $G$  of  $(M + 1) \times (N + 1)$  vertices, called *edit graph*. Each vertex  $(i, j)$  stores the value of  $ED(i, j)$ , and any shortest path in  $G$  from vertex  $(0, 0)$  to vertex  $(M, N)$  specifies an optimal *edit trace* (a sequence of edit operations).

## 2.2 Propagation over Run-Sized Blocks

Run-length encoding (abbreviated as RLE) is a well-known coding scheme that performs lossless data compression. RLE compression simply groups consecutive, identical symbols of a string into a run, usually denoted by  $\sigma^i$ , where  $\sigma$  is an alphabet symbol and  $i$  is its repetition times. For example, string  $bcccdaaaaaa$  can be compressed into RLE format as  $b^2c^3d^2a^5$ . Based on the RLE factorization of input strings  $A$  and  $B$ , the edit graph  $G$  can be partitioned into  $mn$  blocks, where  $m$  and  $n$  denote the number of runs of  $A$  and  $B$ , respectively. We refer to a subgraph of  $G$  as a *match block* if it corresponds to a run of  $A$  and a run of  $B$  that encode the same symbol, and as a *mismatch block* otherwise. Note that two adjacent blocks in  $G$  share the vertices in their adjoining borders. As observed in [3613], instead of computing all the vertex values of  $G$ , it suffices to compute only the vertex values in the block borders. That is, given a block  $H$  of  $G$ , we refer to the left and top borders of  $H$  as its *input border*, denoted by  $I$ , and refer to the bottom and right borders of  $H$  as its *output border*, denoted by  $O$ . Our algorithm follows the framework of [3613], traversing the blocks of  $G$  in a left-to-right, top-to-bottom order, and propagates the accumulated values from  $I$  to  $O$  without filling in the vertices lying in-between.

## 2.3 Problem Reduction

In this subsection, we briefly review the observations made in previous work [4313]. Given two vertices  $(i', j')$  and  $(i, j)$  of  $G$ , where  $i' \leq i$  and  $j' \leq j$ , we let  $dist(i', j', i, j)$  denote the total weight of the shortest path from  $(i', j')$  to  $(i, j)$ . Moreover, we say that vertex  $(i, j)$  of  $G$  is in *diagonal*  $j - i$  of  $G$ . Let vertex  $(i_0, j_0)$  denote the upper-left corner of block  $H$ . For each vertex  $(i, j)$  in  $O$ , we have the following recurrence relation:

$$ED(i, j) = \min \left\{ \begin{array}{l} \min_{i_0 \leq i' \leq i} (ED(i', j_0) + dist(i', j_0, i, j)) \\ \min_{j_0 \leq j' \leq j} (ED(i_0, j') + dist(i_0, j', i, j)) \end{array} \right\} \quad (1)$$

**Lemma 1** ([4]). *If  $H$  is a match block, for each vertex  $(i, j)$  in  $O$ , we have that  $ED(i, j) = ED(i_d, j_d)$ , where  $(i_d, j_d)$  is the intersection of diagonal  $j - i$  with  $I$ .*

By Lemma 1, the propagation over a match block is easy. To obtain the vertex values in the output border, we simply copy their diagonal values in the input border. Therefore, the difficulty lies in the propagation over a mismatch block, which can be reduced to the so-called *sliding-window minima problem* as follows.

**Lemma 2** ([3,13]). *If  $H$  is a mismatch block, for vertex  $(i, j)$  in  $O$ , we have that  $ED(i, j) = \min(\min_{i_d \leq i' \leq i} (ED(i', j_0) + j - j_0), \min_{j_d \leq j' \leq j} (ED(i_0, j') + i - i_0))$ , where  $(i_d, j_d)$  is the intersection of diagonal  $j - i$  with  $I$ .*

Let  $LEFT[1 \dots h]$  and  $TOP[1 \dots w]$  denote the vertex values of the left border and the top border of  $I$ . The entries of  $LEFT$  and  $TOP$  are numbered in a bottom-to-top and left-to-right direction, respectively. Let  $OUT[1 \dots w + h - 1]$

denote the vertex values of  $O$ . The entries of  $OUT$  are numbered in a counter-clockwise direction, starting from the lower-left corner. For simplicity, we only discuss the case that block  $H$  is flat, i.e.  $h \leq w$ . The other case where  $h > w$  can be argued symmetrically.

**Definition 1.** For  $i \in [1, w+h-1]$ , we let  $OUT_{left}[i]$  (resp.,  $OUT_{top}[i]$ ) denote the weight of the shortest path passing through the left border (resp., top border) of  $I$  and ending at the vertex corresponding to  $OUT[i]$ .

By Equation (1), we can write:

$$OUT[i] = \min\{OUT_{left}[i], OUT_{top}[i]\} \text{ for } i \in [1, w+h-1]. \quad (2)$$

*Problem 1.* Given a numerical array  $S[1 \dots \ell]$  and a positive integer  $h$ , denoting the window size, we define the sliding-window minima array of  $S$  as  $S^{(h)}[i] = \min\{S[j] \mid i-h+1 \leq j \leq i \text{ and } 1 \leq j \leq \ell\}$  for  $i \in [1, \ell+h-1]$ . The sliding-window minima problem (abbreviated as the SWM problem) is, given array  $S$ , to compute array  $S^{(h)}$ .

Given input array  $S$ , the SWM problem can be solved in  $O(|S|)$  time using *deques with heap orders* described in [7]. By Lemma 2, we can derive the following equations, which are expressed in terms of the sliding-window minima arrays of  $LEFT$  and  $TOP$ .

$$OUT_{left}[i] = \begin{cases} LEFT^{(h)}[i] & +i-1, \text{ for } i \in [1, h]; \\ LEFT^{(h)}[h] & +i-1, \text{ for } i \in [h, w]; \\ LEFT^{(h)}[i-w+h] + w-1, & \text{for } i \in [w, w+h-1]; \end{cases} \quad (3)$$

$$OUT_{top}[i] = \begin{cases} TOP^{(h)}[i] + h-1, & \text{for } i \in [1, w]; \\ TOP^{(h)}[i] + w+h-1-i, & \text{for } i \in [w, w+h-1]; \end{cases} \quad (4)$$

According to Equations (2)–(4), once arrays  $LEFT^{(h)}$  and  $TOP^{(h)}$  are obtained, the values of array  $OUT$  are easily computed. Therefore, the propagation over a mismatch block is reduced to two instances of the SWM problem.

### 3 A Geometric View

In this section, we show that the propagation described in the previous section can be further accelerated by computing only a subset of the border values.

#### 3.1 A Succinct Representation of Border Values

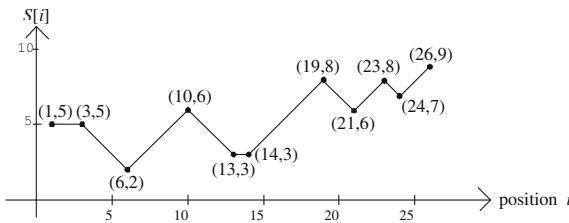
Instead of storing the  $I/O$  values explicitly in arrays, we represent them with a series of two-dimensional points, which we call the *turning points* of arrays.

**Definition 2.** Given a numerical array  $S[1 \dots \ell]$ , we define  $\Delta S(i) = S[i+1] - S[i]$  for  $i \in [1, \ell-1]$ . For simplicity's sake, we let  $\Delta S(0) = \Delta S(\ell) = \infty$ . We call  $(i, S[i])$  a *turning point* of  $S$  if  $\Delta S(i-1) \neq \Delta S(i)$  for  $i \in [1, \ell]$ . The geometric encoding of  $S$ , denoted by  $GE(S)$ , is the list of turning points of  $S$  from left to right.

By definition  $(1, S[1])$  and  $(\ell, S[\ell])$  are the first and the last turning points of  $S[1 \dots \ell]$ . If we plot points  $(i, S[i])$  for all  $i \in [1, \ell]$  in the plane and connect two neighboring points with a straight line, we obtain the *trajectory* (piecewise line segments) of  $S$ , and  $GE(S)$  is the list of turning points of the trajectory from left to right. It is easily seen that  $|GE(S)| \leq |S|$ , where  $|GE(S)|$  denotes the size of list  $GE(S)$  and  $|S|$  denotes the length of numerical array  $S$ .

**Definition 3.** We call  $(i, S[i])$  a *valley point* of  $S$  if  $\Delta S(i-1) < 0$  and  $\Delta S(i) > 0$  for  $i \in [2, \ell - 1]$ . Similarly, we let  $VP(S)$  denote the list of valley points of  $S$  and  $|VP(S)|$  denote its size.

Figure 1 gives an example of turning points and valley points. As will be discussed in Section 5, the number of turning points and valley points in the block borders is crucial to the time analysis of our algorithm.



**Fig. 1.** The geometric encoding of array  $S[1 \dots 26] = 55543234565433456787678789$ . Array  $S$  comprises 26 integers, whereas its geometric encoding  $GE(S)$  is composed of 11 turning points. Among them there are three valley points,  $(6, 2)$ ,  $(21, 6)$ , and  $(24, 7)$ .

### 3.2 Propagation of Turning Points

We use geometric encoding to encode the border values. Since  $ED(i, 0) = i$  for  $1 \leq i \leq M$  and  $ED(0, j) = j$  for  $1 \leq j \leq N$ , we have that each of the leftmost and topmost block borders of  $G$  contains exactly two turning points. Again, by Lemma 1 the propagation over a match block is easily handled by copying the turning points from the input border to the output border. Thus, the difficulty lies in the propagation over a mismatch block. Based on the discussion of Section 2.3, it can be handled by procedure PROPAGATE of Figure 2.

---

#### Procedure PROPAGATE

- Step 1: Use  $GE(LEFT)$  and  $GE(TOP)$  to compute  $GE(LEFT^{(h)})$  and  $GE(TOP^{(h)})$ ;
  - Step 2: Use  $GE(LEFT^{(h)})$  and  $GE(TOP^{(h)})$  to compute  $GE(OUT_{left})$  and  $GE(OUT_{top})$ ;
  - Step 3: Use  $GE(OUT_{left})$  and  $GE(OUT_{top})$  to compute  $GE(OUT)$ ;
- 

**Fig. 2.** Procedure of propagating turning points over a mismatch block

In Step 1 of PROPAGATE, we need to solve the SWM problem with its input and output arrays represented as lists of turning points. We call the problem the *continuous sliding-window minima problem* (abbreviated as CSWM).

*Problem 2.* Let  $S$  be a numerical array and  $h$  be a positive integer, denoting the window size. The CSWM problem is, given  $GE(S)$ , to compute  $GE(S^{(h)})$ .

The CSWM problem can be described graphically as follows. We are given a trajectory  $\mathcal{T}$  in the plane, represented by its turning points, and a window  $\mathcal{W}$ , having a fixed width and an unlimited height. Window  $\mathcal{W}$  is slid, from left to right, across  $\mathcal{T}$  in a *continuous* manner, and at any time, the lowest point of the portion of  $\mathcal{T}$  covered by  $\mathcal{W}$  is plotted in the plane. The CSWM problem seeks for a list of turning points describing the resulting trajectory drawn as above. We will show in Section 4 that the CSWM problem can be solved in  $O(|GE(S)|)$  time, a gain of efficiency over  $O(|S|)$  time.

Given a point  $p$ , we let  $x(p)$  and  $y(p)$  denote its  $x$ -coordinate and its  $y$ -coordinate, i.e.  $p = (x(p), y(p))$ . According to Equations (3) and (4), Step 2 of PROPAGATE can be handled as follows. We split list  $GE(LEFT^{(h)})$  into two lists  $\mathcal{L}_1 = GE(LEFT^{(h)}[1 \dots h])$  and  $\mathcal{L}_2 = GE(LEFT^{(h)}[h \dots 2h - 1])$ . We retrieve each point  $p \in \mathcal{L}_1$  in order and output point  $(x(p), y(p) + x(p) - 1)$ , and then retrieve point  $q \in \mathcal{L}_2$  in order and output  $(x(q) + w - h, y(q) + w - 1)$ . This produces all the turning points of list  $GE(OUT_{left})$ . Similarly, we can easily compute list  $GE(OUT_{top})$  from list  $GE(TOP^{(h)})$ .

According to Equation (2), Step 3 of PROPAGATE can be handled by traversing lists  $GE(OUT_{left})$  and  $GE(OUT_{top})$  simultaneously and output the lower part of the two trajectories.

To conclude, both Step 2 and Step 3 of PROPAGATE can be done in linear time. Hence, if the CSWM problem is also solvable in linear time, the propagation over a mismatch block can be done in time linear in the number of turning points of  $LEFT$  and  $TOP$ , which implies the following crucial theorem.

**Theorem 1.** *The edit distance problem can be solved in  $O(\mathcal{R})$  time, where  $\mathcal{R}$  denotes the total number of turning points in all block borders.*

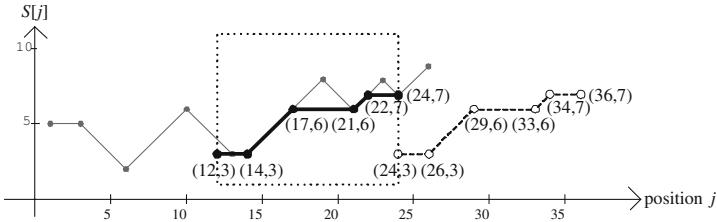
Note that the time of Theorem 1 is never worse than that of [36]. We will show in Section 5 that  $\mathcal{R}$  is bounded by  $O(mn^2)$ , leading to the *fully compressed* feature of our algorithm.

## 4 A Linear-Time Algorithm for the Continuous Sliding-Window Minima Problem

In this section, we present a linear-time algorithm for the CSWM problem. We are given  $GE(S)$ , the geometric encoding of array  $S$ , and a positive integer  $h$ , the window size. Our algorithm simulates the window sliding from left to right across the trajectory of  $S$ , and performs actions whenever the right boundary of the window meets a turning point. We begin by describing the information maintained by our algorithm. Given point  $p$  and a positive number  $d$ , we define  $p \oplus d = (x(p) + d, y(p))$ . Similarly, given a list of points  $\mathcal{L}$ , we define  $\mathcal{L} \oplus d$  to be the list of points obtained by moving the points in  $\mathcal{L}$  distance  $d$  to the right.

**Definition 4.** *For each prefix array  $S[1 \dots j]$  of  $S[1 \dots \ell]$ ,  $j \leq \ell$ , we define its suffix-minimum array as  $SM_j[i] = \min\{S[i], S[i + 1], \dots, S[j]\}$  for  $i \leq j$ .*

Note that the values of  $SM_j$  are increasing, i.e.  $SM_j[i] \leq SM_j[i + 1]$  for  $i \in [1, j - 1]$ . Let  $GE(S) = \langle s_1, s_2, \dots, s_k \rangle$ , where  $s_i$  is the  $i$ -th turning point of  $S$ . At iteration  $i$ , our algorithm needs to consult the values of  $SM_{x(s_i)}[x(s_i) - h + 1 \dots x(s_i)]$ . In implementation, the algorithm maintains list  $\mathcal{L} = GE(SM_{x(s_i)}[x(s_i) - h + 1 \dots x(s_i)]) \oplus (h - 1)$ . See Figure 3 for an example. The window is of size 13 and its right boundary is currently at position 24. The bold lines depict the trajectory of  $SM_{24}[12 \dots 24]$ , and the dashed lines depict the trajectory of list  $\mathcal{L}$ . As the window slides from position 24 to the right, the trajectory of  $\mathcal{L}$  specifies the minimal values contributed by the values of  $S$  before position 24.



**Fig. 3.** The data structure, list  $\mathcal{L}$ , maintained by the algorithm

Our algorithm, named SLIDE (see Figure 4), is devised to follow the trajectory of  $S^{(h)}$  from left to right and output the turning points of  $S^{(h)}$  on the fly. Specifically, at iteration  $i$  the algorithm follows the sub-trajectory of  $S^{(h)}[x(s_i) \dots x(s_{i+1})]$  and outputs the turning points of  $S^{(h)}[x(s_i) \dots x(s_{i+1})]$ .

**Lemma 3.** *Given a numerical array  $S$  and a positive number  $h$ , algorithm SLIDE correctly outputs a list of points containing  $GE(S^{(h)})$  in  $O(|GE(S)|)$  time.*

*Proof.* We show that at iteration  $i$ , the algorithm correctly computes the turning points of  $S^{(h)}[x(s_i) \dots x(s_{i+1})]$  and updates list  $\mathcal{L}$  to  $GE(SM_{i+1}[x(s_{i+1}) - h + 1 \dots x(s_{i+1})]) \oplus (h - 1)$  for the use of iteration  $i + 1$ . Suppose by induction that list  $\mathcal{L} = GE(SM_{x(s_i)}[x(s_i) - h + 1 \dots x(s_i)]) \oplus (h - 1)$  is computed at iteration  $i - 1$ . Observe that for  $j \in [x(s_i), x(s_i) + h - 1]$ ,  $S^{(h)}[j] = \min\{S[j - h + 1], \dots, S[j]\} = \min(SM_{x(s_i)}[j - h + 1], \min(S[x(s_i)], \dots, S[j]))$ . Hence, the trajectory of  $S^{(h)}[x(s_i) \dots x(s_{i+1})]$  is the lower part of the trajectories of  $\mathcal{L}$  and  $\overline{s_i s_{i+1}}$ . The algorithm proceeds in the following cases (see Figure 5 for an illustration):

**Case 1:** The slope of  $\overline{s_i s_{i+1}}$  is nonnegative, which implies that  $\overline{s_i s_{i+1}}$  never goes below the trajectory of  $\mathcal{L}$ . The algorithm thus outputs the turning points belonging to  $\mathcal{L}$  (see line 6). Observe that  $SM_{x(s_{i+1})}[j] = SM_{x(s_i)}[j]$  for  $j \in [x(s_i) - h + 1, x(s_i)]$ , and  $SM_{x(s_{i+1})}[j] = S[j]$  for  $j \in (x(s_i), x(s_{i+1}))$ . Hence, to update  $\mathcal{L}$  the algorithm appends point  $s_{i+1} \oplus (h - 1)$  to its end and then cuts off its front piece before position  $x(s_{i+1})$  (see lines 7–10).

**Algorithm SLIDE**


---

**Input:**  $GE(S) = \langle s_1, s_2, \dots, s_k \rangle$ , where  $x(s_1) < \dots < x(s_k)$ , and a positive number  $h$ .  
**Output:** A list of points containing  $GE(S^{(h)})$ .

```

1   Output point  $s_1$ ;
2   Initialize list  $\mathcal{L} \leftarrow \langle s_1, s_1 \oplus (h - 1) \rangle$ ;
3   for  $i \leftarrow 1$  to  $k - 1$  do
4     Retrieve the next point  $s_{i+1}$  from  $GE(S)$ ;
5     Case 1:  $y(s_i) \leq y(s_{i+1})$ 
6       Output all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in (x(s_i), x(s_{i+1}))$ ;
7       Insert point  $s_{i+1} \oplus (h - 1)$  into the end of  $\mathcal{L}$ ;
8       Traverse  $\mathcal{L}$  from left to right to point  $q$  such that  $x(q) = x(s_{i+1})$ ;
9       Delete all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in [x(s_i), x(q)]$ ;
10      Insert  $q$  into the front of  $\mathcal{L}$ ;
11    Case 2:  $y(s_i) > y(s_{i+1})$ 
12      Traverse  $\mathcal{L}$  from left to right and check if it intersects with  $\overline{s_i s_{i+1}}$ ;
13      Case 2a: there is no intersection.
14        Output all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in (x(s_i), x(s_{i+1}))$ ;
15        Traverse  $\mathcal{L}$  from left to right to point  $q$  such that  $x(q) = x(s_{i+1})$ ;
16        Delete all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in [x(s_i), x(q)]$ 
17        Insert  $q$  into the front of  $\mathcal{L}$ ;
18        Traverse  $\mathcal{L}$  from right to left to the leftmost  $q'$  such that  $y(q') = y(s_{i+1})$ ;
19        Delete all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in [x(q'), x(s_i) + h - 1]$ ;
20        Insert  $q'$  and  $s_{i+1} \oplus (h - 1)$  into the end of  $\mathcal{L}$ ;
21      Case 2b: there is an intersection  $r$ .
22        Output all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in (x(s_i), x(r))$ ;
23        Output points  $r$  and  $s_{i+1}$ ;
24        Reset list  $\mathcal{L} \leftarrow \langle s_{i+1}, s_{i+1} \oplus (h - 1) \rangle$ ;
25  end for
26  Output all the points  $p$  in  $\mathcal{L}$ , where  $x(p) \in (x(s_k), x(s_k) + h - 1)$ ;

```

---

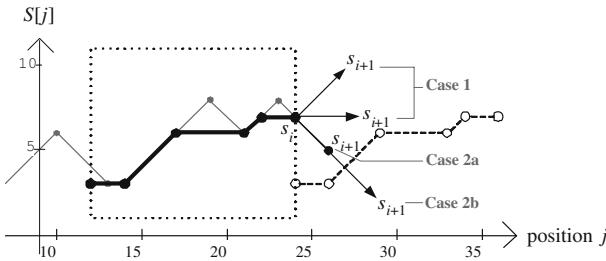
**Fig. 4.** Algorithm for the continuous sliding-window minima problem

**Case 2:** The slope of  $\overline{s_i s_{i+1}}$  is negative.

– **Case 2a:** If  $\overline{s_i s_{i+1}}$  does not intersect with the trajectory of  $\mathcal{L}$ , we have that  $\overline{s_i s_{i+1}}$  never goes below the trajectory of  $\mathcal{L}$ . The algorithm again outputs the turning points belonging to  $\mathcal{L}$ . Observe that  $SM_{x(s_{i+1})}[j] = SM_{x(s_i)}[j]$  for  $j \in [x(s_i) - h + 1, x(q'))$ , and  $SM_{x(s_{i+1})}[j] = S[x(s_{i+1})]$  for  $j \in [x(q'), x(s_{i+1})]$ , where  $q'$  is the leftmost point in  $\mathcal{L}$  such that  $y(q') = y(s_{i+1})$ . Therefore, to update list  $\mathcal{L}$  the algorithm cuts off its front piece before position  $x(s_{i+1})$  and its back piece after position  $x(q')$ , and then appends point  $s_{i+1} \oplus (h - 1)$  to its end (see lines 15–20).

– **Case 2b:** If  $\overline{s_i s_{i+1}}$  intersects with the trajectory of  $\mathcal{L}$  at point  $r$ , we have that the trajectory of  $S^{(h)}[x(s_i) \dots x(s_{i+1})]$  coincides with the trajectory of  $\mathcal{L}$  before position  $x(r)$  and coincides with  $\overline{s_i s_{i+1}}$  after position  $x(r)$ . The algorithm thus outputs the turning points of  $\mathcal{L}$  lying in  $(x(s_i), x(r))$ , and then points  $r$  and  $s_{i+1}$  (see lines 22–23). List  $\mathcal{L}$  is reset to  $\langle s_{i+1}, s_{i+1} \oplus (h - 1) \rangle$  due to the fact that  $SM_{x(s_{i+1})}[j] = S[x(s_{i+1})]$  for  $j \in [x(s_{i+1}) - h + 1, x(s_{i+1})]$  (see line 24).

Observe that the total time spent in the above cases is proportional to the number of elements deleted from list  $\mathcal{L}$ . Since there are  $O(|GE(S)|)$  points inserted into list  $\mathcal{L}$ , the algorithm runs in  $O(|GE(S)|)$  time.  $\square$



**Fig. 5.** The three possible positions of the next turning point  $s_{i+1}$ . The dashed lines depict the trajectory of  $\mathcal{L}$ , which should be compared with line segment  $\overline{s_is_{i+1}}$  in order to determine the trajectory of  $S^{(h)}[x(s_i) \dots x(s_{i+1})]$ .

According to algorithm SLIDE, below we prove two properties of the trajectory of  $S^{(h)}$ , which will be used in the next section.

**Lemma 4.** *Given a numerical array  $S$  and a positive number  $h$ , we have that  $|GE(S^{(h)})| \leq |GE(S)| + |VP(S)|$ .*

*Proof.* We examine the points output by algorithm SLIDE and show that among them there are at most  $|GE(S)| + |VP(S)|$  turning points of  $S^{(h)}$ . Let  $\sigma_1$ ,  $\sigma_2$ , and  $\sigma_3$  denote the numbers of points of Case 1, Case 2a, and Case 2b, encountered by algorithm SLIDE, respectively. Note that  $|GE(S)| = \sigma_1 + \sigma_2 + \sigma_3 + 1$ . Observe that if  $s_i$  is a point of Case 1 or Case 2a, then  $s_i$  is not output but deposited in list  $\mathcal{L}$  as point  $s_i \oplus (h - 1)$ . If  $s_i$  is a point of Case 2b, it is output as well as deposited in  $\mathcal{L}$  as point  $s_i \oplus (h - 1)$ . That is, a point of Case 1 or Case 2a contributes at most one turning point to  $S^{(h)}$ , whereas a point of Case 2b may contribute two turning points. Hence, we have that  $|GE(S^{(h)})| \leq \sigma_1 + \sigma_2 + 2\sigma_3 + 1 = |GE(S)| + \sigma_3$ . We next examine those points of Case 2b in more details. Let  $s_i$  be a point of Case 2b. If the slope of  $\overline{s_is_{i+1}}$  is non-positive, the deposited point  $s_i \oplus (h - 1)$  will not be a turning point of  $S^{(h)}$ . This is because after iteration  $i - 1$ ,  $\mathcal{L}$  is set to  $\langle s_i, s_i \oplus (h - 1) \rangle$ , a horizontal line segment. Thus, if the slope of  $\overline{s_is_{i+1}}$  is non-positive, then either  $\overline{s_is_{i+1}}$  intersects with the trajectory of  $\mathcal{L}$  immediately or both of their slopes are 0. Therefore,  $s_i$  contributes two turning points to  $S^{(h)}$  only when the slope of  $\overline{s_is_{i+1}}$  is positive (in this case,  $s_i$  is a valley point). Hence, we can refine the bound into  $|GE(S^h)| \leq |GE(S)| + |VP(S)|$ .  $\square$

**Lemma 5.** *Given a numerical array  $S$  and a positive number  $h$ , we have that  $|VP(S^{(h)})| = 0$ .*

*Proof.* Because the trajectory of  $\mathcal{L}$  is always rising, algorithm SLIDE follows a negative-slope line only when a point of Case 2b is encountered. Observe that it

cannot next follow a positive-slope line immediately, for  $\mathcal{L}$  is set to a horizontal line segment after Case 2b. Hence, we have that the complete trajectory of  $S^{(h)}$  contains no valley point, i.e.  $|VP(S^{(h)})| = 0$ .  $\square$

## 5 Time Complexity

Since there are  $mn$  propagations in total, if each propagation doubles the number of turning points, the number of turning points will grow in an exponential manner as they cascade down. Below, we show that the number of turning points can only grow by a constant in one propagation. This is obviously correct for propagations over match blocks, since they are simple copies of the turning points from the input border. Thus, we focus on propagations over mismatch blocks. We begin by introducing the *monotonicity property* of the border values. This property was observed by [1] and applied in [16,10,15].

**Lemma 6** ([1,15]). *Given two vertices  $(x_4, y_4)$  and  $(x_3, y_3)$  in  $I$ , and two vertices  $(x_1, y_1)$  and  $(x_2, y_2)$  in  $O$ , such that  $x_4 \leq x_3 \leq x_1 \leq x_2$  and  $y_3 \leq y_4 \leq y_2 \leq y_1$ , we have that if  $ED(x_3, y_3) + dist(x_3, y_3, x_1, y_1) \leq ED(x_4, y_4) + dist(x_4, y_4, x_1, y_1)$ , then  $ED(x_3, y_3) + dist(x_3, y_3, x_2, y_2) \leq ED(x_4, y_4) + dist(x_4, y_4, x_2, y_2)$ .*

**Lemma 7.** *The trajectories of  $OUT_{left}$  and  $OUT_{top}$ , obtained in Step 2 of PROPAGATE, cross at most once. That is, if  $OUT_{top}[i] \leq OUT_{left}[i]$  for some  $i$ , then  $OUT_{top}[j] \leq OUT_{left}[j]$  for all  $j \geq i$ .*

*Proof.* Suppose that  $OUT[i]$  and  $OUT[j]$  correspond to vertex  $(x_1, y_1)$  and vertex  $(x_2, y_2)$  in  $G$ . From  $OUT_{top}[i] \leq OUT_{left}[i]$ , we know that there exists vertex  $(x_3, y_3)$  in the top border such that  $ED(x_3, y_3) + dist(x_3, y_3, x_1, y_1) \leq ED(x_4, y_4) + dist(x_4, y_4, x_1, y_1)$  for all vertices  $(x_4, y_4)$ ,  $y_4 \leq y_2$ , in the left border. By Lemma 6 we have that  $ED(x_3, y_3) + dist(x_3, y_3, x_2, y_2) \leq ED(x_4, y_4) + dist(x_4, y_4, x_2, y_2)$  for all vertices  $(x_4, y_4)$ ,  $y_4 \leq y_2$ , in the left border. Since  $OUT_{left}[j] \leq ED(x_3, y_3) + dist(x_3, y_3, x_2, y_2)$ , the lemma thus follows.  $\square$

**Lemma 8.** *After Step 1 of PROPAGATE, we have that  $|GE(LEFT^{(h)})| \leq |GE(LEFT)| + 1$  and  $|GE(TOP^{(h)})| \leq |GE(TOP)| + 1$ .*

*Proof.* We show by induction that the number of valley points in each block border is at most one, and the lemma thus follows from Lemma 4. Initially, each of the leftmost and topmost block borders contains no valley point. By Lemma 1, the propagation over a match block produces no extra valley point in the output border. As for the propagation over a mismatch block, by Lemma 5 we know that Step 1 leads to no valley point in  $LEFT^{(h)}$  and  $TOP^{(h)}$ . Step 2 clearly produces no extra valley point, and we know by Lemma 7 that Step 3 produces at most one valley point. Hence, we conclude that the propagation over a mismatch block results in at most one valley point in the output border.  $\square$

It is also not hard to see that after Steps 2–3 of PROPAGATE, the number of turning points can only grow by a constant. Now, we are ready to prove the time complexity of our algorithm.

**Theorem 2.** *Given two strings  $A$  and  $B$ , compressed into  $m$  and  $n$  runs,  $m \leq n$ , computing the edit distance between  $A$  and  $B$  can be done in  $O(mn^2)$  time.*

*Proof.* Let  $H_{i,j}$  denote the block corresponding to the  $i$ -th run of  $A$  and the  $j$ -th run of  $B$  for  $i \in [1, m]$  and  $j \in [1, n]$ . Let  $u_{i,j}$  (resp.,  $v_{i,j}$ ) denote the number of turning points in the top border (resp., left border) of  $H_{i,j}$ . Let  $U_i = \sum_{j=1}^n u_{i,j}$  and  $V_j = \sum_{i=1}^m v_{i,j}$ . By Theorem 11, the time of our algorithm is proportional to  $\sum_{i=1}^m U_i + \sum_{j=1}^n V_j$ . Let  $v_{i',n+1}$  (resp.,  $u_{m+1,j'}$ ) denote the number of turning points in the bottom border of  $H_{n,j'}$  (resp., the right border of  $H_{i',n}$ ) for  $i' \in [1, m]$  (resp., for  $j' \in [1, n]$ ). Since the number of turning points can only grow by a constant  $c$  in each propagation, we have that  $u_{i+1,j} + v_{i,j+1} \leq u_{i,j} + v_{i,j} + c$  for all  $i \in [1, m]$ ,  $j \in [1, n]$ . Hence, we have that  $\sum_{j=1}^n (u_{i+1,j} + v_{i,j+1}) \leq \sum_{j=1}^n (u_{i,j} + v_{i,j}) + cn$ , implying  $\sum_{j=1}^n u_{i+1,j} \leq \sum_{j=1}^n u_{i,j} + cn + v_{i,1} - v_{i,n+1} \leq \sum_{j=1}^n u_{i,j} + (cn + 1)$ . That is,  $U_{i+1} \leq U_i + (cn + 1)$ . Since  $U_1 = \sum_{j=1}^n u_{1,j} = 2n$ , it is not hard to derive that  $\sum_{i=1}^m U_i = O(m^2n)$ . Similarly, we can also derive  $\sum_{j=1}^n V_j = O(mn^2)$ . The theorem thus follows.  $\square$

To recover an optimal edit trace, we start from vertex  $(M, N)$  and trace a series of block-crossing paths back to vertex  $(0, 0)$ . This requires additional computation and storage of information during the propagation stage. As long as we associate each point output by algorithm SLIDE with its source entry, we can trace, for each vertex in the output border, back to its optimal source vertex in the input border. This requires  $O(mn^2)$  space in total. Hirschberg's space reduction method [9] can be used to reduce the space to  $O(mn)$  without impairing the  $O(mn^2)$  time bound.

## 6 Concluding Remarks

We present the first fully compressed algorithm for computing the edit distance between two RLE strings, of  $m$  and  $n$  runs, in  $O(mn^2)$  time and  $O(mn)$  space. The approximate matching problem is, given pattern  $P$  and text  $T$ , to identify substrings  $T'$  of  $T$  such that the edit distance between  $P$  and  $T'$  is at most  $k$ . The dynamic programming solution for this problem requires the following setting. The vertex values in the first row of the edit graph are initialized as zeros, and the goal becomes to identify the vertex values in the last row that are less than or equal to  $k$ . Our approach can be easily adapted to this setting within the same time and space bound. Furthermore, our algorithm in fact runs in  $O(\mathcal{R})$  time, where  $\mathcal{R}$  denotes the total number of turning points in the block borders. In the paper, we prove that  $\mathcal{R}$  is bounded by  $O(mn^2)$ . Providing a tighter bound for  $\mathcal{R}$  implies better time complexity of our algorithm.

## References

1. Aggarwal, A., Park, J.K.: Notes on Searching in Multidimensional Monotone Arrays. In: FOCS 1998, pp. 497–512 (1998)
2. Apostolico, A., Landau, G.M., Skiena, S.: Matching for Run-Length Encoded Strings. Journal of Complexity 15(1), 4–16 (1999)

3. Arbell, O., Landau, G.M., Mitchell, J.S.B.: Edit Distance of Run-Length Encoded Strings. *Information Processing Letters* 83(6), 307–314 (2002)
4. Bunke, H., Csirik, J.: An Improved Algorithm for Computing the Edit Distance of Run-Length Coded Strings. *Information Processing Letters* 54(2), 93–96 (1995)
5. Chen, K.-Y., Hsu, P.-H., Chao, K.-M.: Approximate Matching for Run-Length Encoded Strings Is 3SUM-Hard. *Journal of Complexity* (accepted); A preliminary version appeared in CPM 2009 (2009)
6. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A Subquadratic Sequence Alignment Algorithm for Unrestricted Scoring Matrices. *SIAM Journal on Computing* 32(6), 1654–1673 (2003)
7. Gajewska, H., Tarjan, R.E.: Deques with Heap Order. *Information Processing Letters* 22(4), 197–200 (1986)
8. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A Unified Algorithm for Accelerating Edit-Distance Computation via Text-Compression. In: STACS, pp. 529–540 (2009)
9. Hirschberg, D.S.: A Linear Space Algorithm for Computing Maximal Common Subsequences. *Communications of the ACM* 18(6), 341–343 (1975)
10. Huang, G.-S., Liu, J.J., Wang, Y.-L.: Sequence Alignment Algorithms for Run-Length-Encoded Strings. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 319–330. Springer, Heidelberg (2008)
11. Liu, J.J., Huang, G.-S., Wang, Y.-L., Lee, R.C.-T.: Edit Distance for a Run-Length-Encoded String and an Uncompressed String. *Information Processing Letters* 105(1), 12–16 (2007)
12. Masek, W.J., Paterson, M.: A Faster Algorithm Computing String Edit Distances. *Journal of Computer and System Sciences* 20(1), 18–31 (1980)
13. Mäkinen, V., Ukkonen, E., Navarro, G.: Approximate Matching of Run-Length Compressed Strings. *Algorithmica* 35(4), 347–369 (2003)
14. Mitchell, J.S.B.: A Geometric Shortest Path Problem, with Application to Computing a Longest Common Subsequence in Run-Length Encoded Strings. Technical Report, SUNY Stony Brook (1997)
15. Schmidt, J.P.: All Highest Scoring Paths in Weighted Grid Graphs and Their Application to Finding All Approximate Repeats in Strings. *SIAM Journal on Computing* 27(4), 972–992 (1998)

# Fast Prefix Search in Little Space, with Applications

Djamal Belazzougui<sup>1</sup>, Paolo Boldi<sup>2</sup>, Rasmus Pagh<sup>3</sup>, and Sebastiano Vigna<sup>2</sup>

<sup>1</sup> Université Paris Diderot—Paris 7, France

<sup>2</sup> Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy

<sup>3</sup> IT University of Copenhagen, Denmark

**Abstract.** A *prefix search* returns the strings out of a given collection  $S$  that start with a given prefix. Traditionally, prefix search is solved by data structures that are also dictionaries, that is, they actually contain the strings in  $S$ . For very large collections stored in slow-access memory, we propose extremely compact data structures that solve *weak* prefix searches—they return the correct result only if *some* string in  $S$  starts with the given prefix. Our data structures for weak prefix search use  $O(|S| \log \ell)$  bits in the worst case, where  $\ell$  is the average string length, as opposed to  $O(|S|\ell)$  bits for a dictionary. We show a lower bound implying that this space usage is optimal.

## 1 Introduction

In this paper we are interested in the following problem (hereafter referred to as *prefix search*): given a collection of  $n$  strings, find all the strings that start with a given prefix  $p$ . In particular, we will be interested in the space/time tradeoffs needed to do prefix search in a static context (i.e., when the collection does not change over time).

There is a large literature on indexing string collections. We refer to Ferragina et al. [11, 4] for state-of-the-art results, with emphasis on the cache-oblivious model. Roughly speaking, results can be divided into two categories based on the power of queries allowed. As shown by Pătrașcu and Thorup [15], any data structure for bit strings that supports predecessor (or rank) queries must either use super-linear space, or use time  $\Omega(\log |p|)$  for a query on a prefix  $p$ . On the other hand, it is known that prefix queries, and more generally range queries, can be answered in constant time using linear space [1].

Another distinction is between data structures (typically comparison-based) where the query time grows with the number of strings in the collection, versus those (typically some kind of trie) where the query time depends only on the length of the query string [1]. In this paper we fill a gap in the literature by considering data structures for *weak prefix search*, a relaxation of prefix search, with query time depending only on the length of the query string. In a weak prefix search we have the guarantee that the input  $p$  is a prefix of some string in

---

<sup>1</sup> Obviously, one can also combine the two in a single data structure.

the set, and we are only requested to output the ranks (in lexicographic order) of the strings that have  $p$  as prefix.

Our first result is that weak prefix search can be performed by accessing a data structure that uses just  $O(n \log \ell)$  bits, where  $\ell$  is the average string length. This is much less than the space of  $n\ell$  bits used for the strings themselves. We also show that this is the minimum possible space usage for any such data structure, regardless of query time. We investigate different time/space tradeoffs: At one end of this spectrum we have constant-time queries (for prefixes that fit in  $O(1)$  words), and still asymptotically vanishing space usage for the index. At the other end, space is optimal and the query time grows logarithmically with the length of the prefix. Precise statements can be found in the technical overview below.

*Technical overview.* For simplicity we consider strings over a binary alphabet, but our methods generalise to larger alphabets. Our main result is that weak prefix search needs just  $O(|p|/w + \log |p|)$  time and  $O(n \log \ell)$  space, where  $\ell$  is the average length of the strings,  $p$  is the query string, and  $w$  is the machine word size. In the cache-oblivious model [12], we use  $O(p/B + \log |p|)$  I/Os. For strings of fixed length  $w$ , this reduces to query time  $O(\log w)$  and space  $O(n \log w)$ , and we show that the latter is *optimal* regardless of query time. Throughout the paper we strive to state all space results in terms of  $\ell$ , and time results in terms of the length of the actual query string  $p$ , because in a realistic setting (e.g., term dictionaries of a search engine) string lengths might vary wildly, and queries might be issued that are significantly shorter than the average (let alone maximum) string length. Actually, the data structure size depends on the *hollow trie size* of the set  $S$ —a data-aware measure related to the trie size [13] that is much more precise than the bound  $O(n \log \ell)$ .

Building on ideas from [1], we then give an  $O(|p|/w+1)$  solution (i.e., constant time for prefixes of length  $O(w)$ ) that uses space  $O(n\ell^{1/c} \log \ell)$  (for any  $c > 0$ ). This structure shows that weak prefix search is possible in constant time using sublinear space; queries require  $O(|p|/B+1)$  I/Os in the cache-oblivious model.

*Comparison to related results.* If we study the same problem in the I/O model or in the cache-oblivious model, the nearest competitors are the String B-tree [10] and its cache-oblivious version [4], albeit they require access to the set  $S$ . The *static* String B-tree can be modified to use space  $O(n \log n + n \log \ell)$ ; it has very good search performance with  $O(|p|/B + \log_B n)$  I/Os per query (supporting all query types discussed in this paper), and its cache-oblivious version guarantees the same bounds with high probability. However, a search for  $p$  inside the String B-tree may involve  $\Omega(|p| + \log n)$  RAM operations ( $\Omega(|p|/w + \log n)$  for the cache-oblivious version), so it may be too expensive for intensive computations. Our first method, which achieves the optimal space usage of  $O(n \log \ell)$  bits, uses  $O(|p|/w + \log |p|)$  RAM operations and  $O(|p|/B + \log |p|)$  I/Os instead. The number of RAM operations is a strict improvement over String B-trees, while the I/O bound is better for large enough sets. Our second method uses slightly more space ( $O(n\ell^{1/c} \log \ell)$  bits for any  $c > 0$ ) but features  $O(|p|/w + 1)$  RAM operations and  $O(|p|/B + 1)$  I/Os.

In [11], the authors discuss very succinct static data structures for the same purposes (on a generic alphabet), decreasing the space to a lower bound that is, in the binary case, the trie size. The search time is logarithmic in the number of strings. As in the previous case, we improve on RAM operations and on I/Os for large enough sets.

The first cache-oblivious dictionary supporting prefix search was devised by Brodal *et al.* [5] achieving  $O(|p|)$  RAM operations and  $O(|p|/B + \log_B n)$  I/Os. We note that the result in [5] is optimal in a comparison-based model, where we have a lower bound of  $\Omega(\log_B n)$  I/Os per query. By contrast, our result, like those in [4,11], assumes an integer alphabet where there is no such lower bound.

Implicit in the paper of Alstrup *et al.* [1] on range queries is a linear-space structure for constant-time weak prefix search on fixed-length bit strings. Our constant-time data structure, instead, uses sublinear space and allows for variable-length strings.

*Applications.* Data structures that allow weak prefix search can be used to solve the non-weak version of the problem, provided that the original data is stored (typically, in some slow-access memory): a single probe is sufficient to determine if the result set is empty; if not, access to the string set is needed just to retrieve the strings that match the query. By the same means we can answer prefix counting queries. It is also possible to solve range queries with two additional probes to the original data (w.r.t. the output size), improving the results in [1]. We finally show that our results extend to the cache-oblivious model, where we provide an alternative to the results in [5,4,11] that removes the dependence on the data set size for prefix searches and range queries.

*Our contributions.* The main contribution of this paper is the identification of the weak prefix search problem, and the proposal of a solutions based on techniques developed in [2]. Optimality (in space or time) of the solution is also a central result of this research. The second interesting contribution is the description of *range locators* for variable-length strings; they are an essential building block in our weak prefix search algorithms, and can be used whenever it is necessary to recover in little space the range of leaves under a node of a trie.

## 2 Notation and Tools

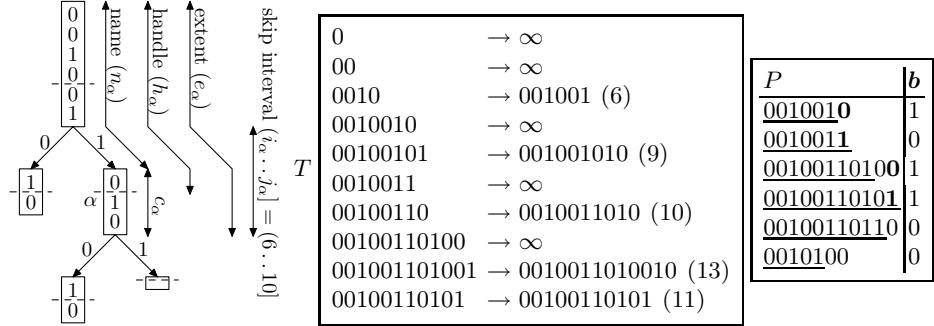
In the following sections, we will use the toy set of strings shown in Figure 1 to display examples of our constructions. We use von Neumann's definition and notation for natural numbers:  $n = \{0, 1, \dots, n - 1\}$ , so  $2 = \{0, 1\}$  and  $2^*$  is the set of all binary strings.

**Weak prefix search.** Given a prefix-free set of strings  $S \subseteq 2^*$ , the *weak prefix search* problem requires, given a prefix  $p$  of some string in  $S$ , to return the range of strings of  $S$  having  $p$  as prefix; this set is returned as the interval of integers that are the ranks (in lexicographic order) of the strings in  $S$  having  $p$  as prefix.

**Model and assumptions.** The model of computation considered in most of the paper is a unit-cost word RAM with word size  $w$ . We assume that  $|S| = O(2^{cw})$

for some constant  $c$ , so that constant-time static data structures depending on  $|S|$  can be used. We extend several results also to the cache-oblivious model [12].

**Compacted tries.** Consider the compacted trie built for a prefix-free set of strings  $S \subseteq 2^*$ . For a given node  $\alpha$  of the trie, we define (see Figure 1):



**Fig. 1.** The trie built on the sample set  $\{001001010, 0010011010010, 00100110101\}$ , and the associated map and range locator.  $T$  maps handles to extents; the corresponding hollow z-fast prefix trie just returns the *lengths* (shown in parentheses) of the extents. In the range locator table, we boldface the zeroes and ones appended to extents, and we underline the actual keys (as trailing zeroes are removed). The last two keys are  $00100110101^+$  and  $0010011^+$ , respectively.

- $e_\alpha$ , the *extent of node  $\alpha$* , is the longest common prefix of the strings represented by the leaves that are descendants of  $\alpha$  (this was called the “string represented by  $\alpha$ ” in [2]);
- $c_\alpha$ , the *compacted path of node  $\alpha$* , is the string stored at  $\alpha$ ;
- $n_\alpha$ , the *name of node  $\alpha$* , is the string  $e_\alpha$  deprived of its suffix  $c_\alpha$  (this was called the “path leading to  $\alpha$ ” in [2]);
- given a string  $x$ , we let  $\text{exit}(x)$  be the exit node of  $x$ , that is, the only node  $\alpha$  such that  $n_\alpha$  is a prefix of  $x$  and  $e_\alpha$  is not a proper prefix of  $x$ ;
- the *skip interval* ( $i_\alpha \dots j_\alpha$ ) associated to  $\alpha$  is  $(0 \dots |c_\alpha|)$  for the root, and  $(|n_\alpha| - 1 \dots |e_\alpha|)$  for all other nodes.

**Data-aware measures.** Consider the compacted trie on a set  $S \subseteq 2^*$ . We define the *trie measure* of  $S$  [13] as

$$\text{T}(S) = \sum_{\alpha} (j_\alpha - i_\alpha) = \sum_{\alpha} (|c_\alpha| + 1) - 1 = 2n - 2 + \sum_{\alpha} |c_\alpha| = O(n\ell),$$

where the summation ranges over all nodes of the trie. For the purpose of this paper, we will also use the *hollow<sup>2</sup> trie measure*

$$\text{HT}(S) = \sum_{\alpha \text{ internal}} (\text{bitlength}(|c_\alpha|) + 1) - 1.$$

<sup>2</sup> A compacted trie is made *hollow* by replacing the compacted path at each node by its length *and then discarding all its leaves*. A recursive definition of hollow trie appears in [3].

Since  $\text{bitlength}(x) = \lceil \log(x + 1) \rceil$ , we have  $\text{HT}(S) = O(n \log \ell)$ .

**Storing functions.** The problem of storing statically an  $r$ -bit function  $f : A \rightarrow 2^r$  from a given set of keys  $A$  has recently received renewed attention [7]. For the purposes of this paper, we simply recall that these methods allow us to store an  $r$ -bit function on  $n$  keys using  $rn + cn + o(n)$  bits for some constant  $c \geq 0$ , with  $O(|x|/w)$  access time for a query string  $x$ . Practical implementations are described in [3]. In some cases, we will store a *compressed* function using a minimal perfect function ( $O(n)$  bits) followed by a compressed data representation (e.g., an Elias–Fano compressed list [3]). In that case, storing natural numbers  $x_0, x_1, \dots, x_{n-1}$  requires space  $\sum_i \lfloor \log(x_i + 1) \rfloor + n \log(\sum_i \lfloor \log(x_i + 1) \rfloor / n) + O(n)$ .

**Relative dictionaries.** A *relative dictionary* stores a set  $E$  relatively to some set  $S \supseteq E$ . That is, the relative dictionary answers questions about membership to  $E$ , but its answers are required to be correct only if the query string is in  $S$ . It is possible to store such a dictionary in  $|E| \log(|S|/|E|)$  bits of space with  $O(|x|/w)$  access time [2].

**Rank and select.** We will use two basic blocks of several succinct data structures—rank and select. Given a bit array (or bit string)  $\mathbf{b} \in 2^n$ , whose positions are numbered starting from 0,  $\text{rank}_{\mathbf{b}}(p)$  is the number of ones up to position  $p$ , exclusive ( $0 \leq p \leq n$ ), whereas  $\text{select}_{\mathbf{b}}(r)$  is the position of the  $r$ -th one in  $\mathbf{b}$ , with bits numbered starting from 0 ( $0 \leq r < \text{rank}_{\mathbf{b}}(n)$ ). It is well known that these operations can be performed in constant time on a string of  $n$  bits using additional  $o(n)$  bits, see [14, 16].

### 3 From Prefixes to Exit Nodes

We break the weak prefix search problem into two subproblems. Our first goal is to go from a given prefix of some string in  $S$  to its exit node.

*Hollow z-fast prefix tries.* We start by describing an improvement of the *z-fast trie*, a data structure first defined in [2]. The main idea behind a z-fast trie is that, instead of representing explicitly a binary tree structure containing compacted paths of the trie, we will store a function that maps a certain prefix of each extent to the extent itself. This mapping (which can be stored in linear space) will be sufficient to navigate the trie and obtain, given a string  $x$ , the *name of the exit node of  $x$*  and the exit behaviour (left, right, or possibly equality for leaves). The interesting point about the z-fast trie is that it provides such a name in time  $O(|x|/w + \log |x|)$ , and that it leads easily to a probabilistically relaxed version, or even to hollow variants.

To make the paper self-contained, we recall the main definitions from [2]. The *2-fattest* number in a nonempty interval of positive integers is the number in the interval whose binary representation has the largest number of trailing zeros. Consider the compacted trie on  $S$ , one of its nodes  $\alpha$ , and the 2-fattest number  $f$  in its skip interval  $(i_\alpha \dots j_\alpha]$ ; if the interval is empty, which can happen only at the root, we set  $f = 0$ . The *handle*  $h_\alpha$  of  $\alpha$  is  $e_\alpha[0 \dots f]$ , where  $e_\alpha[0 \dots f]$  denotes the first  $f$  bits of  $e_\alpha$ . A (*deterministic*) *z-fast trie* is a dictionary  $T$  mapping each

**Algorithm 1**

**Input:** a prefix  $p$  of some string in  $S$ .  
**Output:** the name of  $\text{exit}(p)$ .

```

 $a, b \leftarrow 0, |p|$ 
while  $b - a > 1$  do
   $f \leftarrow$  the 2-fattest number in  $(a \dots b)$ 
   $g \leftarrow T(p[0 \dots f])$ 
  if  $g \geq |p|$  then
     $b \leftarrow f$ 
  else
     $a \leftarrow g$ 
  end if
end while
if  $a = 0$  then
  return  $\varepsilon$ 
else
  return  $p[0 \dots a + 1]$ 
end if

```

**Algorithm 2**

**Input:** the name  $x$  of a node.  
**Output:** the interval  $[i \dots j)$  of strings prefixed by  $x$ .

```

if  $x = \varepsilon$  then
   $i \leftarrow 0, j \leftarrow n$ 
else
   $i \leftarrow \text{rank}_b h(x^\leftarrow)$ 
  if  $x = 111 \dots 11$  then
     $j \leftarrow n$ 
  else
     $j \leftarrow \text{rank}_b h((x^+)^{\leftarrow})$ 
  end if
end if
return  $[i \dots j)$ 

```

**Fig. 2.** Algorithms for weak prefix search and range location

handle  $h_\alpha$  to the corresponding extent  $e_\alpha$ . In Figure 1, the part of the mapping  $T$  with non- $\infty$  output is the z-fast trie built on the trie of Figure 1.

We now introduce a more powerful structure, the (*deterministic*) *z-fast prefix trie*. Consider again a node  $\alpha$  of the compacted trie on  $S$  with notation as above. The *pseudohandles* of  $\alpha$  are the strings  $e_\alpha[0 \dots f')$ , where  $f'$  ranges among the 2-fattest numbers of the intervals  $(i_\alpha \dots t)$ , with  $i_\alpha < t < f$ . Essentially, pseudohandles play the same rôle as handles for every *prefix* of the handle that extends the node name. We note immediately that there are at most  $\log(f - i_\alpha) \leq \log |e_\alpha|$  pseudohandles associated with  $\alpha$ , so the overall number of handles and pseudohandles is bounded by  $\text{HT}(S) + \sum_{x \in S} \log |x| = O(n \log \ell)$ . It is now easy to define a z-fast prefix trie: the dictionary providing the map from handles to extents is enlarged to pseudohandles, which are mapped to the special value  $\infty$ .

We are actually interested in a *hollow* version of a z-fast prefix trie—more precisely, a version implemented by a function  $T$  that maps handles of internal nodes to the length of their extents, and handles of leaves and pseudohandles to  $\infty$ . The function (see again Figure 1) can be stored in a very small amount of space; nonetheless, we will still be able to compute the name of the exit node of any string that is a prefix of some string in  $S$  using Algorithm 1:

**Theorem 1.** Let  $p$  be a nonempty string that is a prefix of some string in  $S$  and  $X = \{p_0 = \varepsilon, p_1, \dots, p_t\}$ , where  $p_1, p_2, \dots, p_t$  are the extents of the nodes of the trie that are proper prefixes of  $p$ , ordered by increasing length. Let  $(a \dots b)$  be the interval maintained by Algorithm 1. Before and after each iteration the following invariant is satisfied:  $a = |p_j|$  for some  $j$ , and  $a \leq |p_t| < b$ .

*Proof.* We note that the invariant is trivially true at the start, as the initial interval is  $(0 \dots |p|)$ . We now prove by induction that in the rest of execution the

invariant is true. At each step we pick the 2-fattest number  $f \in (|p_i| \dots b)$ , and change interval. We have two cases (we follow the notation of Algorithm 1):

- If  $f > |p_t|$ , since  $f < |p|$  then it is either the length of the handle of the exit node of  $p$ , or the length of a pseudohandle associated with the exit node of  $p$ , so we set  $b = f$  and the invariant is preserved.
- Otherwise,  $f$  is 2-fattest in  $(|p_i| \dots |p_t|)$ , so  $p[0 \dots f]$  must be the handle of an ancestor of  $\text{exit}(p)$  (as  $f$  is 2-fattest in every subinterval of  $(|p_i| \dots |p_t|)$  that contains it) which implies  $g = |p_k|$  for some  $k$ . Thus, by setting  $a = g$  the invariant is preserved.  $\square$

By the previous theorem, Algorithm 1 is correct and completes in at most  $\log |p|$  iterations. We note that finding the 2-fattest number in an interval requires the computation of the most significant bit<sup>3</sup>, but alternatively we can check that  $(1 \ll i) \& a \neq (1 \ll i) \& b$  for decreasing  $i$ : if the test is satisfied, the number is  $b$  &  $-1 \ll i$ , otherwise we decrement  $i$ .

*Space and time.* The space needed for a hollow z-fast prefix trie depends on the component chosen for its implementation. The most trivial bound uses a function mapping handles and pseudohandles to one bit that makes it possible to recognise handles of internal nodes ( $O(n \log \ell)$  bits), and a function mapping handles to extent lengths ( $O(n \log L)$  bits, where  $L$  is the maximum string length).

These results, however, can be significantly improved. First of all, we can store handles of internal nodes in a relative dictionary. The dictionary will store  $n - 1$  strings out of  $O(n \log \ell)$  strings, using  $O(n \log((n \log \ell)/n)) = O(n \log \log \ell)$  bits. Then, the mapping from handles to extent lengths  $h_\alpha \mapsto |e_\alpha|$  can actually be recast into a mapping  $h_\alpha \mapsto |e_\alpha| - |h_\alpha|$ . But since  $|e_\alpha| - |h_\alpha| \leq |c_\alpha|$ , by storing these data by means of a compressed function we will use space

$$\begin{aligned} & \sum_{\alpha} \lfloor \log(|e_\alpha| - |h_\alpha| + 1) \rfloor + O(n \log \log \ell) + O(n) \\ & \leq \sum_{\alpha} \lfloor \log(|c_\alpha| + 1) \rfloor + O(n \log \log \ell) \leq \text{HT}(S) + O(n \log \log \ell), \end{aligned}$$

where  $\alpha$  ranges over internal nodes.

Algorithm 1 cannot iterate more than  $\log |p|$  times; at each step, we query constant-time data structures using a prefix of  $p$ : using incremental hashing [6, Section 5], we can preprocess  $p$  in time  $O(|p|/w)$  (and in  $|p|/B$  I/Os) so that hashing prefixes of  $p$  requires constant time afterwards. We conclude that Algorithm 1 requires time  $O(|p|/w + \log |p|)$ .

*Faster, faster, faster...* We now describe a data structure mapping prefixes to exit nodes inspired by the techniques used in [1] that needs  $O(n\ell^{1/2} \log \ell)$  bits of space and answers in time  $O(|p|/w)$ , thus providing a different space/time tradeoff. The basic idea is as follows: let  $s = \lceil \ell^{1/2} \rceil$  and, for each node  $\alpha$  of

<sup>3</sup> More precisely, the 2-fattest number in  $(a..b]$  is  $-1 \ll \text{msb}(a \oplus b) \& b$ .

the compacted trie on the set  $S$ , consider the set of prefixes of  $e_\alpha$  with length  $t \in (i_\alpha \dots j_\alpha]$  such that either  $t$  is a multiple of  $s$  or is smaller than the first such multiple. More precisely, we consider prefixes whose length is either of the form  $ks$ , where  $ks \in (i_\alpha \dots j_\alpha]$ , or in  $(i_\alpha \dots \min\{\bar{k}s, j_\alpha\}]$ , where  $\bar{k}$  is the minimum  $k$  such that  $ks > i_\alpha$ .

We store a function  $F$  mapping each prefix  $p$  defined above to the length of the name of the corresponding node  $\alpha$  (actually, we can map  $p$  to  $|p| - |n_\alpha|$ ). Additionally, we store a mapping  $G$  from each node name to the length of its extent (again, we can just map  $n_\alpha \mapsto |c_\alpha|$ ).

To retrieve the exit node of a string  $p$  that is a prefix of some string in  $S$ , we consider the string  $q = p[0 \dots |p| - |p| \bmod s]$  (i.e., the longest prefix of  $p$  whose length is a multiple of  $s$ ). Then, we check whether  $G(p[0 \dots F(q)]) \geq |p|$  (i.e., whether  $p$  is a prefix of the extent of the exit node of  $q$ ). If this is the case, then clearly  $p$  has the same exit node as  $q$  (i.e.,  $p[0 \dots F(q)]$ ). Otherwise, the map  $F$  provides directly the length of the name of the exit node of  $p$ , which is thus  $p[0 \dots F(p)]$ . All operations are completed in time  $O(|p|/w)$ . The proof that this structure uses space  $O(n\ell^{1/2} \log \ell)$  is deferred to the full paper.

## 4 Range Location

Our next problem is determining the range (of lexicographical ranks) of the leaves that appear under a certain node of a trie. Actually, this problem is pretty common in static data structures, and usually it is solved by associating with each node a pair of integers of  $\log n \leq w$  bits. However, this means that the structure has, in the worst case, a linear ( $O(nw)$ ) dependency on the data.

To work around this issue, we propose to use a *range locator*—an abstraction of a component used in [2]. Here we redefine range locators from scratch, and improve their space usage so that it is dependent on the average string length, rather than on the maximum string length. A range locator takes as input *the name of a node*, and returns the range of ranks of the leaves that appear under that node. For instance, in our toy example the answer to 0010011 would be  $[1 \dots 3]$ . To build a range locator, we need to introduce *monotone minimal perfect hashing*.

Given a set of  $n$  strings  $T$ , a *monotone minimal perfect hash function* [2] is a bijection  $T \rightarrow n$  that preserves lexicographical ordering. This means that each string of  $T$  is mapped to its rank in  $T$  (but strings not in  $T$  give random results). We use the following results from [3]:<sup>4</sup>

**Theorem 2.** *Let  $T$  be a set of  $n$  strings of average length  $\ell$  and maximum length  $L$ , and  $x \in 2^*$  be a string. Then, there are monotone minimal perfect hashing functions on  $T$  that:*

1. *use space  $O(n \log \ell)$  and answer in time  $O(|x|/w)$ ;*
2. *use space  $O(n \log \log L)$  and answer in time  $O(|x|/w + \log |x|)$ .*

---

<sup>4</sup> Actually, results in [3] are stated for prefix-free sets, but it is trivial to make a set of strings prefix-free at the cost of doubling the average length.

We show how a reduction can relieve us from the dependency on  $L$ ; this is essential to our goals, as we want to depend just on the average length:

**Theorem 3.** *There is a monotone minimal perfect hashing function on  $T$  using space  $O(n \log \log \ell)$  that answers in time  $O(|x|/w + \log |x|)$  on a query string  $x \in 2^*$ .*

*Proof.* We divide  $T$  into the set of strings  $T^-$  shorter than  $\ell \log n$ , and the remaining “long” strings  $T^+$ . Setting up a  $n$ -bit vector  $\mathbf{b}$  that records the elements of  $T^-$  with select-one and select-zero structures ( $n + o(n)$  bits), we can reduce the problem to hashing monotonically  $T^-$  and  $T^+$ . We note, however, that using Theorem 2 the set  $T^-$  can be hashed in space  $O(|T^-| \log \log(\ell \log n)) = O(|T^-| \log \log \ell)$ , as  $2\ell \geq \log n$ , and  $T^+$  can be hashed explicitly using a  $(\log n)$ -bit function; since  $|T^+| \leq n/\log n$  necessarily, the function requires  $O(n)$  bits. Overall, we obtain the required bounds.  $\square$

We now describe in detail our range locator, using the notation of Section 2. Given a string  $x$ , let  $x^\leftarrow$  be  $x$  with all its trailing zeroes removed. We build a set of strings  $P$  as follows: for each extent  $e$  of an internal node, we add to  $P$  the strings  $e^\leftarrow$ ,  $e1$ , and, if  $e \neq 111\cdots 11$ , we also add to  $P$  the string  $(e1^+)^\leftarrow$ , where  $e1^+$  denotes the successor of length  $|e1|$  of  $e1$  in lexicographical order (numerically, it is  $e1 + 1$ ). We build a monotone minimal perfect hashing function  $h$  on  $P$ , noting the following easily proven fact:

**Proposition 1.** *The average length of the strings in  $P$  is at most  $3\ell$ .*

The second component of the range locator is a bit vector  $\mathbf{b}$  of length  $|P|$ , in which bits corresponding to the names of leaves are set to one. The vector is endowed with a ranking structure  $\text{rank}_\mathbf{b}$  (see Figure 1).

It is now immediate that given a node name  $x$ , by hashing  $x^\leftarrow$  and ranking the bit position thus obtained in  $\mathbf{b}$ , we obtain the left extreme of the range of leaves under  $x$ . Moreover, performing the same operations on  $(x^+)^\leftarrow$ , we obtain the right extreme. All these strings are in  $P$  by construction, except for the case of a node name of the form  $111\cdots 11$ ; however, in that case the right extreme is just the number of leaves (see Algorithm 2 for the details).

A range locator uses at most  $3n + o(n)$  bits for  $\mathbf{b}$  and its selection structures. Thus, space usage is dominated by the monotone hashing component. Using the structures described above, we obtain:

**Theorem 4.** *There are structures implementing range location in time  $O(|x|/w)$  using  $O(n \log \ell)$  bits of space, and in  $O(|x|/w + \log |x|)$  time using  $O(n \log \log \ell)$  bits of space.*

We remark that other combinations of monotone minimal perfect hashing and succinct data structures can lead to similar results. Among several such asymptotically equivalent solutions, we believe ours is the most practical.

## 5 Putting It All Together

In this section we gather the main results about prefix search:

**Theorem 5.** *There are structures implementing weak prefix search in space  $\text{HT}(S) + O(n \log \log \ell)$  with query time  $O(|p|/w + \log |p|)$ , and in space  $O(n\ell^{1/2} \log \ell)$  with query time  $O(|p|/w)$ .*

*Proof.* The first structure uses a hollow z-fast prefix trie followed by the range locator of Theorem 3; the first component provides the name  $n_\alpha$  of exit node of  $|p|$ ; given  $n_\alpha$ , the range locator returns the correct range. For the second structure, we use the structure defined in Section 3 followed by the first range locator of Theorem 2.  $\square$

Actually, the second structure described in Theorem 5 can be made to occupy space  $O(n\ell^{1/c} \log \ell)$  for any constant  $c > 0$  (the proof will be given in the full version):

**Theorem 6.** *For any constant  $c > 0$ , there is a structure implementing weak prefix search in space  $O(n\ell^{1/c} \log \ell)$  with query time  $O(|p|/w)$ .*

We note that all our time bounds can be translated into I/O bounds in the *cache-oblivious model* if we replace the  $O(|p|/w)$  terms by  $O(|p|/B)$  (where  $B$  is the I/O block size in bits). The  $O(|p|/w)$  term appears in two places: first, in the phase of precalculation of a hash-vector of  $\lceil |p|/w \rceil$  hash words on the prefix  $p$  which is later used to compute all the hash functions on prefixes of  $p$ ; second, in the range location phase, where we need to compute  $x^\leftarrow$  and  $(x^+)^{\leftarrow}$ , where  $x$  is a prefix of  $p$  and subsequently compute the hash vectors on  $x^\leftarrow$  and  $(x^+)^{\leftarrow}$ . Observe that the above operations can be carried on using arithmetic operations only, without any additional I/O (we can use 2-wise independent hashing involving only multiplications and additions for computing the hash vectors and only basic arithmetic operations for computing  $x^\leftarrow$  and  $(x^+)^{\leftarrow}$ ) except for the writing of the result of the computation which occupies  $O(|p|/w)$  words of space and thus take  $O(|p|/B)$  I/Os. Thus in both cases we need only  $O(|p|/B)$  I/Os corresponding to the time needed to read the pattern and to write the result.

## 6 A Space Lower Bound

In this section we show that the space usage achieved by the weak prefix search data structure described in Theorem 5 is optimal up to a constant factor. In fact, we show a matching lower bound for the easier problem of prefix counting (i.e., counting how many strings start with a given prefix), and consider the more general case where the answer is only required to be correct up to an additive constant less than  $k$ . We note that any data structure supporting prefix counting can be used to achieve approximate prefix counting, by building the data structure for the set that contains every  $k$ -th element in sorted order.

**Theorem 7.** Consider a data structure (possibly randomised) indexing a set  $S$  of  $n$  strings with average length  $\ell > \log(n) + 1$ , supporting  $k$ -approximate prefix count queries: Given a prefix of some key in  $S$ , the structure returns the number of elements in  $S$  that have this prefix with an additive error of less than  $k$ , where  $k < n/2$ . The data structure may return any number when given a string that is not a prefix of a key in  $S$ . Then the expected space usage on a worst-case set  $S$  is  $\Omega((n/k) \log(\ell - \log n))$  bits. In particular, if no error is allowed and  $\ell > (1 + \varepsilon) \log n$ , for constant  $\varepsilon > 0$ , the expected space usage is  $\Omega(n \log \ell)$  bits.

*Proof.* Let  $u = 2^\ell$  be the number of possible keys of length  $\ell$ . We show that there exists a probability distribution on key sets  $S$  such that the expected space usage is  $\Omega((n/k) \log \log(u/n))$  bits. By the “easy directions of Yao’s lemma,” this implies that the expected space usage of any (possibly randomised) data structure on a worst case input is at least  $\Omega((n/k) \log \log(u/n))$  bits. The bound for  $\ell > (1 + \varepsilon) \log n$  and  $k = 1$  follows immediately.

Assume without loss of generality that  $n/(k+1)$  and  $k$  are powers of 2. All strings in  $S$  will be of the form  $abc \in 2^*$ , where  $|a| = \log_2(n/(k+1))$ ,  $|b| = \ell - \log_2(n/(k+1)) - \log_2 k$ , and  $|c| = \log_2 k$ . Let  $t = \ell - \log_2(n/(k+1)) - \log_2 k$  denote the length of  $b$ . For every value of  $a$  the set will contain exactly  $k+1$  elements: One where  $b$  and  $c$  are strings of 0s, and for  $b$  chosen uniformly at random among strings of Hamming weight 1 we have  $k$  strings for  $c \in 2^{\log_2 k}$ . Notice that the entropy of the set  $S$  is  $n/(k+1) \log_2 t$ , as we choose  $n/(k+1)$  values of  $b$  independently from a set of  $t$  strings. To finish the argument we will need to show that any two such sets require different data structures, which means that the entropy of the bit string representing the data structure for  $S$  must also be at least  $n/(k+1) \log_2 t$ , and in particular this is a lower bound on the expected length of the bit string.

Consider two different sets  $S'$  and  $S''$ . There exists a value of  $a$ , and distinct values  $b', b''$  of Hamming weight 1 such that  $S'$  contains all  $k$   $\ell$ -bits strings prefixed by  $ab'$ , and  $S''$  contains all  $k$   $\ell$ -bits strings prefixed by  $ab''$ . Assume without loss of generality that  $b'$  is lexicographically before  $b''$ . Now consider the query for a string of the form  $a0^\ell$ , which is a prefix of  $ab'$  but not  $ab''$  – such a string exists since  $b'$  and  $b''$  have Hamming weight 1. The number of keys with this prefix is  $k+1$  and 1, respectively, for  $S'$  and  $S''$ , so the answers to the queries must be different (both in the multiplicative and additive case). Hence, different data structures are needed for  $S'$  and  $S''$ .  $\square$

Note that the trivial information-theoretical lower bound does not apply, as it is impossible to reconstruct  $S$  from the data structure.

It is interesting to note the connections with the lower and upper bounds presented in [11]. This paper shows a lower bound on the number of bits necessary to represent a set of strings  $S$  that, in the binary case, reduces to  $T(S) + \log \ell$ , and provide a matching data structure. Theorem 5 provides a *hollow* data structure that is sized following the naturally associated measure:  $HT(S) + O(n \log \log \ell)$ . Thus, Theorem 5 and 7 can be seen as the hollow version of the results presented in [11], albeit our lower bound is a match only asymptotically. Improving Theorem 7 to  $HT(S) + o(HT(S))$  is an interesting open problem.

## References

1. Alstrup, S., Brodal, G.S., Rauhe, T.: Optimal static range reporting in one dimension. In: STOC 2001, pp. 476–482 (2001)
2. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Monotone minimal perfect hashing: Searching a sorted table with  $O(1)$  accesses. In: SODA 2009, pp. 785–794. ACM Press, New York (2009)
3. Belazzougui, D., Boldi, P., Pagh, R., Vigna, S.: Theory and practise of monotone minimal perfect hashing. In: ALENEX 2009. SIAM, Philadelphia (2009)
4. Bender, M.A., Farach-Colton, M., Kuszmaul, B.C.: Cache-oblivious string B-trees. In: PODS 2006, pp. 233–242. ACM, New York (2006)
5. Brodal, G.S., Fagerberg, R.: Cache-oblivious string dictionaries. In: SODA 2006, pp. 581–590 (2006)
6. Dietzfelbinger, M., Gil, J., Matias, Y., Pippenger, N.: Polynomial hash functions are reliable (extended abstract). In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 235–246. Springer, Heidelberg (1992)
7. Dietzfelbinger, M., Pagh, R.: Succinct data structures for retrieval and approximate membership (extended abstract). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórssen, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 385–396. Springer, Heidelberg (2008)
8. Elias, P.: Efficient storage and retrieval by content and address of static files. J. Assoc. Comput. Mach. 21(2), 246–260 (1974)
9. Elias, P.: Universal codeword sets and representations of the integers. IEEE Trans. on Info. Theory 21, 194–203 (1975)
10. Ferragina, P., Grossi, R.: The string B-tree: a new data structure for string search in external memory and its applications. Journal of the ACM 46(2), 236–280 (1999)
11. Ferragina, P., Grossi, R., Gupta, A., Shah, R., Vitter, J.S.: On searching compressed string collections cache-obliviously. In: PODS 2008, pp. 181–190 (2008)
12. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: FOCS 1999, pp. 285–297. IEEE Comput. Soc. Press, Los Alamitos (1999)
13. Gupta, A., Hon, W.-K., Shah, R., Vitter, J.S.: Compressed data structures: Dictionaries and data-aware measures. Theor. Comput. Sci. 387(3), 313–331 (2007)
14. Jacobson, G.: Space-efficient static trees and graphs. In: FOCS 1989, pp. 549–554 (1989)
15. Pătrașcu, M., Thorup, M.: Randomization does not help searching predecessors. In: SODA 2007, pp. 555–564 (2007)
16. Raman, R., Raman, V., Rao, S.S.: Succinct indexable dictionaries with applications to encoding  $k$ -ary trees and multisets. In: SODA 2002, pp. 233–242. ACM Press, New York (2002)

# On the Huffman and Alphabetic Tree Problem with General Cost Functions

Hiroshi Fujiwara<sup>1,\*</sup> and Tobias Jacobs<sup>2,\*\*</sup>

<sup>1</sup> Department of Information and Computer Sciences, Toyohashi University of Technology, 1-1 Tenpaku-cho, Toyohashi 441-8580, Japan  
[h-fujiwara@cs.tut.ac.jp](mailto:h-fujiwara@cs.tut.ac.jp)

<sup>2</sup> National Institute of Informatics, 2-1-2 Hitotsubashi,  
Chiyoda-ku, Tokyo 101-8430, Japan  
[jacobs@nii.ac.jp](mailto:jacobs@nii.ac.jp)

**Abstract.** We study a wide generalization of two classical problems, the Huffman Tree and Alphabetic Tree Problem. We assume that the cost caused by the  $i$ th leaf is  $f_i(d_i)$ , where  $d_i$  is its depth in the tree under consideration, and  $f_i : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$  is an arbitrary function. All solution methods known for the classical cases fail to compute the optimum here.

For the generalized Alphabetic Tree Problem, we give a dynamic programming algorithm solving it in time  $O(n^4)$ , using space  $O(n^3)$ . Furthermore, we show that the runtime can be reduced to  $O(n^3)$  if the cost functions are nondecreasing and convex. The improved algorithm can also be used in the setting where the cost functions are nondecreasing and the objective function is the maximum leaf cost.

We also prove that the Huffman Tree Problem in its full generality is inapproximable unless P=NP, no matter if the objective function is the sum of leaf costs or their maximum. For the latter problem, we show that the case where the cost functions are nondecreasing admits a polynomial time algorithm.

## 1 Introduction

Computing minimum cost binary trees is a classical combinatorial problem having applications in various areas of informatics. Given a set  $\{\ell_1, \dots, \ell_n\}$  of leaves having weights  $\{w_1, \dots, w_n\}$ , the famous Huffman Tree Problem describes the task to compute a binary tree  $T$  with leaf set  $\{\ell_1, \dots, \ell_n\}$ , such that the weighted total distance between the tree root and a leaf is minimized. The Alphabetic Tree Problem differs from the Huffman Problem by the additional constraint that the left-to-right order of the leaves in the solution tree must be exactly  $\ell_1, \dots, \ell_n$ . The objective function of both versions is given as

$$\sum_{i=1}^n w_i \cdot \text{dist}(\text{root}(T), \ell_i) .$$

---

\* This work was supported by KAKENHI (19700015).

\*\* This work was supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).

In the above model it is assumed that the access cost of a leaf is proportional to its depth in the tree. In this work we investigate a generalized problem: we assume that the cost of leaf  $\ell_i$  having distance  $d_i$  from the root is determined by  $f_i(d_i)$ , where  $f_i : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$  is an arbitrary function. Instances of our generalized problem are determined by the  $n$  cost functions  $f_1, \dots, f_n$ , one for each leaf. The corresponding optimization problems can be formulated as follows.

**General Cost Huffman Tree Problem, GHT.** Given  $n$  arbitrary functions  $f_1, \dots, f_n : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ , the objective of GHT is to determine a binary tree  $T$  having  $n$  leaves and a bijection  $g : \{1, \dots, n\} \rightarrow \text{leaves}(T)$  such that  $\sum_{i=1}^n f_i(\text{depth}(g(i), T))$  is minimized, where  $\text{depth}(g(i), T)$  is the distance between the root of  $T$  and the leaf  $g(i)$ .

**General Cost Alphabetic Tree Problem, GAT.** Given  $n$  arbitrary functions  $f_1, \dots, f_n : \mathbb{N}_0 \rightarrow \mathbb{R}_0^+$ , the objective of GAT is to determine a binary tree  $T$  whose leaves in left-to-right order are  $\ell_1, \dots, \ell_n$ , such that  $\sum_{i=1}^n f_i(\text{depth}(\ell_i, T))$  is minimized.

We also investigate problems **max-GHT** and **max-GAT**, where the objective function is  $\max_{i=1}^n f_i(\text{depth}(g(i), T))$  and  $\max_{i=1}^n f_i(\text{depth}(\ell_i, T))$ , respectively.

**Related Work.** The Huffman Tree Problem is named after D. A. Huffman [1], who gave an algorithm solving it in time  $O(n \log n)$ , which is the fastest possible. For the Alphabetic Tree Problem, the first polynomial time algorithm was a dynamic programming approach having a runtime of  $O(n^3)$ , proposed by Gilbert and Moore [2]. Knuth [3] identified a property of optimal alphabetic trees that admits a speedup of the DP algorithm. The resulting runtime is  $O(n^2)$ . The  $O(n \log n)$  time method discovered by Hu and Tucker [4] uses a bottom-up approach which resembles the Huffman-Algorithm.

Up to today it is unknown whether the Alphabetic Tree Problem can be solved in time  $o(n \log n)$ . However, for certain classes of special weight assignments, linear time algorithms were given [7, 8], and for certain classes of algorithms  $\Theta(n \log n)$  was shown to be a lower bound for the runtime [7]. Flajolet and Prodinger [6] gave an asymptotic estimate of the number of feasible solutions to the Alphabetic Tree Problem.

Efforts to solve the Alphabetic Tree Problem with non-linear costs were made by Hu et al. in [9]. The authors identified a class of cost functions where the Hu-Tucker Algorithm is applicable, including power summations of the type  $\text{cost}(T) = w_i t^{d_i}$  for any  $t \geq 1$ . Baer [15] recently showed that for  $t < 1$  neither the Hu-Tucker algorithm nor the approach of Knuth leads to optimal solutions, and he proposes to use the  $O(n^3)$  algorithm by Gilbert and Moore instead.

Another direction of generalization is to impose additional constraints on the structure of the output tree. The Alphabetic Tree Problem can be interpreted as the task to determine an optimal binary search strategy for a totally ordered set. A considerable number of papers about search in partially ordered sets have been published, see e.g. [10, 12, 16]. In the even more general *Binary Identification Problem*, the input is a number of subsets  $S_1, \dots, S_m$  of leaves, and the goal is to compute a minimum cost search strategy using queries of the type “is leaf  $\ell_i$  in set  $S_j$  or not?”. Recent results from this area can be found in e.g. [11, 13, 14].

**Our Results.** We contribute a number of insights concerning the General Cost Alphabetic Tree (GAT) and Huffman Tree Problem (GHT), including the versions where the maximum leaf cost has to be minimized instead of the sum.

In Section 2 we show that an extension of the Gilbert-Moore Algorithm solves GAT in time  $O(n^4)$  and space  $O(n^3)$ , regardless of the cost functions. We then define two properties of cost functions, *subtree optimality* and *structural continuity*. Both properties admit a speedup of the algorithm by the factor of  $n$ . The speedups are independent from each other, so problem instances whose cost functions satisfy both properties admit a  $O(n^2)$  time optimal algorithm, which happens to be exactly the method proposed by Knuth [3].

We prove that if the cost functions are nondecreasing and convex, then the property of structural continuity is satisfied. Therefore, this case of the Alphabetic Tree Problem can be solved in time  $O(n^3)$ . Furthermore, we show that for max-GAT instances to satisfy structural continuity it even suffices that the cost functions are nondecreasing. These results can be found in Section 3.

In Section 4 we address the Huffman Tree Problem. We show for both GHT and max-GHT that it is NP-hard to decide whether an instance admits a cost zero solution. This means that those problems are much harder than their Alphabetic Tree counterparts: they are inapproximable unless P=NP. As a positive result, we show that max-GHT admits a polynomial time algorithm if the cost functions are nondecreasing. The computational tractability of standard GHT with nondecreasing cost functions remains an open problem.

## 2 Dynamic Programming Algorithm for GAT

This section begins with a recapitulation of the Gilbert-Moore algorithm for the classical Alphabetic Tree Problem. We believe that the property of *subtree optimality* can be well understood in the context of that algorithm, because this property is essentially required for its correctness. Subsequently, we give an extended algorithm which solves GAT without requiring subtree optimality. The runtime and space requirements of that algorithm are however by a factor of  $n$  higher. We then introduce the property of *structural continuity* and show how it helps to make the algorithm more time-efficient again.

**The Gilbert-Moore Algorithm and Subtree Optimality.** The algorithm by Gilbert and Moore employs a dynamic programming approach to solve the classical problem with  $f_i(x) = w_i x$  for  $i = 1, \dots, n$ . Subproblems are determined by two integer parameters  $(l, r)$  with  $1 \leq l \leq r \leq n$ . A subproblem  $(l, r)$  asks about an optimal alphabetic tree for  $f_l, \dots, f_r$ . The value  $\tilde{c}(l, r)$  of an optimal solution to that subproblem is calculated recursively as  $\min_{l \leq i < r} (\tilde{c}(l, i) + \tilde{c}(i + 1, r))$ , and the optimal search tree is obtained by making the optimal solution tree to  $(l, j)$  and  $(j + 1, r)$  the left and right subtree of the root, respectively, where  $j$  is the value of  $i$  for which the minimum is reached in the above formula. In the basic case of  $l = r$ , the optimal tree only consists of one leaf.

There are  $O(n^2)$  different subproblems, and each of them requires computation time  $O(n)$ , so the overall runtime is  $O(n^3)$ , while the space requirements

are  $O(n^2)$ . The algorithm successively merges optimal alphabetic trees for subsequences of cost functions into optimal trees for larger subsequences. The reason why this works out is because an optimal tree for  $f_1, \dots, f_n$  is always the combination of optimal trees for  $f_1, \dots, f_i$  and  $f_{i+1}, \dots, f_n$ , for some  $i \in \{1, \dots, n\}$ . This property is called *subtree optimality*. Note that by induction it follows that for any internal node  $v$  in an optimal alphabetic tree, the subtree under  $v$  is an optimal alphabetic tree for the sequence of leaves that are descendants of  $v$ .

**Algorithm for GAT.** A simple counterexample (see full paper) shows that GAT is not subtree-optimal in general, i.e. the left and right subtree under the root of an optimal alphabetic tree are not necessarily optimal alphabetic trees.

For some problem instance  $(f_1, \dots, f_n)$ , assume that  $i$  is such that the leaves  $\ell_1, \dots, \ell_i$  and the leaves  $\ell_{i+1}, \dots, \ell_n$  are in the left and right subtree  $T_1$  and  $T_2$  under the root of an optimal alphabetic tree  $T$ , respectively. We have that

$$\begin{aligned} \text{cost}(T) &= \sum_{1 \leq j \leq i} f_j(\text{depth}(\ell_j, T)) + \sum_{i < j \leq n} f_j(\text{depth}(\ell_j, T)) \\ &= \sum_{1 \leq j \leq i} f_j(\text{depth}(\ell_j, T_1) + 1) + \sum_{i < j \leq n} f_j(\text{depth}(\ell_j, T_2) + 1). \end{aligned}$$

The optimality of  $T$  implies that the structure of say  $T_1$  must be such that the term  $\sum_{1 \leq j \leq i} f_i(\text{depth}(\ell_i, T_1) + 1)$  is minimized. This differs from the optimization term for problem instance  $(f_1, \dots, f_i)$  only by the offset of 1 that is added to each depth value. By the same argument, the two subtrees under the root of  $T_1$  are optimal alphabetic trees with respect to the cost function where an offset of 2 is added to the depth values before applying the  $f_j$ s, and so on.

This offset is added as an additional parameter to the description of subproblems, so subproblem  $(l, r, k)$  is to determine an alphabetic tree  $T'$  having leaves  $\ell_l, \dots, \ell_r$  so as to minimize  $\sum_{i=l}^r f_i(\text{depth}(\ell_i, T') + k)$ . It can also be interpreted as the task to compute a tree  $T'$  which minimizes the sum of access costs under the assumption that the root of  $T'$  is appended to a path of length  $k$ .

The cost  $\tilde{c}$  of an optimal solution to a subproblem is calculated as

$$\tilde{c}(l, r, k) = \begin{cases} f_r(k) & \text{if } l = r \\ \min_{i=l}^{r-1} \{\tilde{c}(l, i, k+1) + \tilde{c}(i+1, r, k+1)\} & \text{otherwise.} \end{cases} \quad (1)$$

The original problem instance  $I$  is given as subproblem  $(1, n, 0)$ . Each time a new subproblem with  $k$  incremented by one is generated, the difference between  $l$  and  $r$  decreases by at least one, which implies that  $k$  never grows larger than  $n$ . Consequently, we have no more than  $O(n^3)$  different subproblems. Each subproblem requires an effort of  $O(n)$ , so the runtime of this algorithm is  $O(n^4)$ .

**Structural Continuity.** The property of *structural continuity* was proven by Knuth [3] to hold for the classical Alphabetic Tree Problem. It roughly states that the root of an optimal alphabetic tree can only move left when the interval under consideration is extended to the left. For making this more precise, recall that the root of an alphabetic tree divides the sequence of leaves into a left and a right

subsequence,  $\ell_1, \dots, \ell_i$  and  $\ell_{i+1}, \dots, \ell_n$ . We say that  $i$  is the position of the root. Now assume that  $i$  is the position of the root of an optimal tree for subsequence  $\ell_1, \dots, \ell_r$ . Then the property of structural continuity guarantees that there is an optimal alphabetic tree for subsequence  $\ell_{l-1}, \dots, \ell_r$  where the root is at a position smaller than or equal to  $i$ . Symmetrically, for  $\ell_l, \dots, \ell_{r+1}$ , there is an optimal solution where the index of the root's position is not smaller than  $i$ .

Structural continuity yields an improved algorithm for the GAT problem. This algorithm computes optimal solutions to the subproblems in the following order: For  $a = 0, \dots, n - 1$ , for  $k = 0, \dots, n$ , compute the solutions to all subproblems  $(l, r, k)$  with  $r - l = a$ . It is not hard to see that this order of computation guarantees that each solution to a subproblem is computed before it is needed during the computation of another subproblem's optimal solution.

We reason about the behavior of the algorithm during some fixed assignment of  $a$  and  $k$ . Denote the choice of  $i$  in Equation 1 as  $i[l, r, k]$ . For  $j = 1, \dots, n - a$ , the optimal solution to  $(j, j+a, k)$  is computed. Structural continuity implies that  $i[j, j+a-1, k] \leq i[j, j+a, k] \leq i[j+1, j+a, k]$ . The values of  $i[j, j+a-1, k]$  and  $i[j+1, j+a, k]$  have already been determined due to the order of computation. There are only  $i[j+1, j+a, k] - i[j, j+a-1, k] + 1$  possible values for  $i[j, j+a, k]$  to be considered by the algorithm. Summing them up for  $i[j, j+a, k]$ ,  $j = 1, \dots, n - a$ , results in a telescope sum which evaluates to  $O(n)$ . There are  $O(n^2)$  different configurations of  $a$  and  $k$ , so the improved runtime is  $O(n^3)$ .

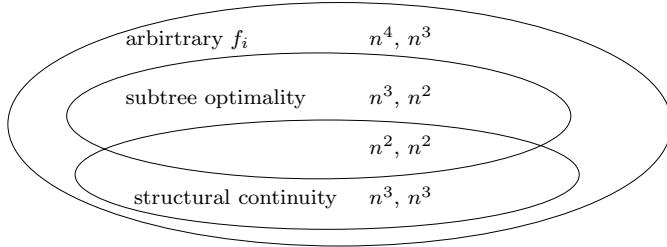
If both subset optimality and structural continuity holds, the Gilbert-Moore algorithm can modified in a similar manner to obtain runtime  $O(n^2)$ , details can be found in [3]. The resource requirements of the dynamic programming approach are visualized in Figure 1.

### 3 Cost Functions Satisfying Structural Continuity

In this section we prove that structural continuity is satisfied by GAT instances whose cost functions are nondecreasing and convex. We also show that nondecreasing cost functions lead to structural continuity in the case of max-GAT.

**Theorem 1.** *Let  $(f_1, \dots, f_n)$  be an instance of GAT where the cost functions are nondecreasing and convex, i.e.  $0 \leq f_i(x) - f_i(x-1) \leq f_i(x+1) - f_i(x)$  for each  $i = 1, \dots, n$  and  $x \geq 1$ . Then the instance satisfies structural continuity.*

For proving the theorem, we use an alternative characterization of the GAT problem. Imagine the infinite binary tree  $\mathcal{T}$ , where every node is an internal one. Given the functions  $f_1, \dots, f_n$ , we seek to find a minimum cost injective assignment  $g$  from the index set  $\{1, \dots, n\}$  to nodes of  $\mathcal{T}$ , under the restriction that any downward path in  $\mathcal{T}$  encounters at most one of these  $n$  indices. Furthermore, the assignment has to be alphabetical, i.e. for any two indices  $i < j$ ,  $g(i)$  and  $g(j)$  must have a common ancestor  $w$  such that  $g(i)$  is in the left and  $g(j)$  is in the right subtree under  $w$ . The objective function to be minimized is  $\sum_{i=1}^n f_i(\text{depth}(g(i), \mathcal{T}))$ .



**Fig. 1.** Runtime and space requirements of the DP approach for GAT

As the  $f_i$ s are nondecreasing, it is not hard to verify that there always exists an optimal solution where any infinite downward path starting at the root of  $\mathcal{T}$  contains a node some index is assigned to. Assignments satisfying that property are called *regular*. When  $g$  is regular, then, by turning the  $n$  nodes in  $g(\{1, \dots, n\})$  into leaves, one obtains a finite tree that has  $n$  nodes and is a solution to the GAT instance. Conversely, any solution for the GAT problem can be interpreted as a regular assignment  $g$  having the same cost.

The subproblems considered by the dynamic programming approach also have a natural interpretation in terms of  $\mathcal{T}$ . Subproblem  $(l, r, k)$  describes the task to find a minimum cost regular assignment of the index set  $\{l, \dots, r\}$  to  $\mathcal{T}$  under the same restrictions as above, but with each  $f_i$  replaced with  $f_i^{+k}$ , where  $f_i^{+k}(d) := f_i(d + k)$ .

Theorem I claims that the root of the optimal tree never has to move right when the interval under consideration is extended to the left. In that proposition, the sequence of functions is interpreted to be fixed, while the tree structure is flexible and adapts to the subproblem under consideration. However, from the point of view just introduced, the tree  $\mathcal{T}$  is fixed and the assignment  $g$  of the index set to nodes in  $\mathcal{T}$  is the flexible part. In this context, Theorem I claims that indices never move left when the interval is extended to the left.

To make our statements more formal, we need to introduce some more notation. Let  $i$  be an index that is assigned to a node of  $\mathcal{T}$  by  $g$ . Assume that  $g$  is modified. We say that  $i$  moves upwards/downwards, if its new position is an ancestor/descendant of its old position. We further say that  $i$  moves left/right, when its new position is left/right of its old position, where a node  $u$  is defined to be left (right) of node  $v$ , when there exists a node  $w$  in  $\mathcal{T}$  such that  $u$  is in the left (right) and  $v$  is in the right (left) subtree under  $w$ . Any movement of  $i$  is either an upward, downward, left, or right movement. The proposition we are going to show is actually more general than Theorem I. The complete proof of the following lemma will appear in the full version of this paper.

**Lemma 1.** *Let  $g$  be the assignment corresponding to an optimal solution to  $(l, r, k)$ . Then there is a regular optimal solution  $g'$  to  $(l - 1, r, k)$  such that for any index  $i = l, \dots, r$ , it either holds that  $g'(i) = g(i)$ ,  $g'(i)$  is right of  $g(i)$ , or  $g'(i)$  is a descendant of  $g(i)$ .*

*Proof (sketched).* Assume for contradiction that there is some index  $i$  moving upwards or left, no matter which optimal regular solution  $g'$  is chosen. Let  $i$  the minimum index with that behavior in some optimal solution  $g'$ .

If  $i = l$ , a contradiction is caused by the fact that  $i$  is assigned to the leftmost path of  $\mathcal{T}$  by  $g$ , but  $g'$  must assign index  $i - 1$  to a node on this path.

If  $i > l$ , first consider the case that index  $i$  moves upwards. We split the transformation from  $g$  into  $g'$  into the transformation from  $g|\{l, \dots, i-1\}$  into  $g'|\{l-1, \dots, i-1\}$ , and the transformation from  $g|\{i, \dots, r\}$  into  $g'|\{i, \dots, r\}$ . The further transformation only changes the positions of indices  $< i$ , and the latter only affects indices  $\geq i$ . As index  $i - 1$  moves neither left nor upwards, these two transformations take place in non-overlapping parts of  $\mathcal{T}$  and can therefore be performed independently of each others. Only the transformation from  $g|\{i, \dots, r\}$  into  $g'|\{i, \dots, r\}$  causes indices to move left or downwards, and from the optimality of  $g$  and  $g'$  we conclude that the solution obtained from  $g$  by only performing the first transformation is as good as  $g'$ .

For analyzing the case where  $i$  moves left, observe that  $i - 1$  can only move downwards in order to make room for  $i$ . We define an alternative solution  $h'$  for  $(l-1, r, k)$ . Solution  $h'$  is constructed starting from  $g'$ , by changing the positions of the indices  $i, \dots, r$  to the positions they are assigned to by  $g$ . After that, we can move index  $i - 1$  a certain number of steps upwards. We then show that  $h'$  is optimal for  $(l-1, r, k)$ , although no index moves down or left during the transformation from  $g$  into  $h'$ .  $\square$

*Proof (of Theorem 1).* Let  $i$  be the position of the root in an alphabetic tree  $T$  for subproblem  $(l, r, k)$ . The corresponding function  $g$  assigns indices  $l, \dots, i$  to the left subtree under the root of  $T$ , and indices  $i+1, \dots, r$  are assigned to the right subtree. Conversely, the position of the root of  $T$  is exactly the largest index being assigned to the left subtree under the root of  $T$ .

So when during the transformation from  $g$  to  $g'$  no index moves left, no index larger than  $i$  can move into the left subtree under the root of  $T$ , and this means that the root of the alphabetic tree  $T'$  corresponding to  $g'$  cannot have a greater position than the root of  $T$ . Using this argumentation, Theorem 1 follows immediately from Lemma 1.  $\square$

**Theorem 2.** *Let  $(f_1, \dots, f_n)$  be an instance of max-GAT where the cost functions are nondecreasing, i.e.  $f_i(x) \geq f_i(x-1)$  for each  $i = 1, \dots, n$  and  $x > 1$ . Then the instance satisfies structural continuity.*

This theorem is much simpler to prove than Theorem 1. The argumentation is based on a simple observation about the cost  $\tilde{c}(l, r, k)$  of the optimal solution to subproblem  $(l, r, k)$ . The correctness of the following lemma is established by the fact that any solution to  $(l-1, r, k)$  can easily be transformed into a cheaper solution to  $(l, r, k)$  when the cost functions are nondecreasing.

**Lemma 2.** *For any instance of max-GAT with nondecreasing cost functions and any subproblem  $(l, r, k)$  of it, it holds that  $\tilde{c}(l, r, k) \leq \tilde{c}(l-1, r, k)$  and  $\tilde{c}(l, r, k) \leq \tilde{c}(l, r+1, k)$ .*

*Proof (of Theorem 2).* Let  $i$  be the position of the root in an optimal solution  $T$  to subproblem  $(l, r, k)$ . For some  $j > i$ , let  $T'$  be an optimal solution to subproblem  $(l - 1, r, k)$  with the root's position at  $j$ . We show that the tree  $T''$ , which is defined as the best solution to  $(l - 1, r, k)$  having the root at position  $i$ , is not more expensive than  $T'$ . The cost of  $T''$  is given as  $\text{cost}(T'') = \max\{\tilde{c}(l - 1, i, k + 1), \tilde{c}(i + 1, r, k + 1)\}$ .

Let us first assume that the maximum in the formula for  $\text{cost}(T'')$  is defined by the first term, i.e.  $\tilde{c}(l - 1, i, k + 1) \geq \tilde{c}(i + 1, r, k + 1)$ . Multiple application of Lemma 2 gives

$$\tilde{c}(l - 1, j, k + 1) \geq \tilde{c}(l - 1, i, k + 1) \geq \tilde{c}(i + 1, r, k + 1) \geq \tilde{c}(j + 1, r, k + 1) ,$$

and this implies that  $\text{cost}(T') = \max\{\tilde{c}(l - 1, j, k + 1), \tilde{c}(j + 1, r, k + 1)\}$  is not smaller than  $\text{cost}(T'')$ .

Now we assume that the second term establishes the maximum, i.e.  $\tilde{c}(l - 1, i, k + 1) \leq \tilde{c}(i + 1, r, k + 1)$ . By Lemma 2,  $\tilde{c}(l, i, k + 1) \leq \tilde{c}(i + 1, r, k + 1)$ , which means that  $\tilde{c}(i + 1, r, k + 1)$  is the maximum in the cost term for  $T$  as well. In other words,  $\text{cost}(T) = \text{cost}(T'')$ . Lemma 2 states that no solution to  $(l - 1, r, k + 1)$ , including  $T'$ , can be cheaper than  $T$ . Therefore,  $T''$  is optimal.  $\square$

## 4 Huffman Tree Problem

In this section we address the General Cost Huffman Tree Problem (GHT). In the classical linear cost model, this problem can be reduced to the Alphabetic Tree Problem by sorting the nodes by weight. A simple counterexample (to appear in the full paper) demonstrates that this reduction does not work correctly in the case of general costs, even if the cost functions are monotonic and convex. We show that GHT in its full generality is inapproximable unless P=NP, which holds for both GHT and max-GHT. Subsequently, we prove that the latter problem admits a polynomial time algorithm if the cost functions are nonincreasing.

**Complexity of GHT.** The computational complexity of sum-GHT and max-GHT will both be settled by one reduction from the 3-Set Cover Problem, which is well-known to be NP-hard [5].

**Exact Cover by 3-Sets, X3C.** Given some set  $C$  with  $|C| = 3k$ ,  $k \in \mathbb{N}$ , and a collection  $D$  of 3 element subsets of  $C$ , the problem X3C is to decide whether there is a sub-collection  $D' \subseteq D$ , such that each element of  $C$  occurs in exactly one member of  $D'$ .

Let  $(C, D)$  be an instance of X3C with  $|D| = m$  and  $|C| = n$ . In the following, we show how to construct an equivalent instance  $I$  of GHT. We consider the solution tree for instance  $I$  to be partitioned into  $m$  layers, each consisting of three levels. In order to simplify notation, let  $l_q^i = 3(i - 1) + q$  denote the  $q$ th level of the  $i$ th layer for  $i = 1, \dots, m$  and  $q = 1, 2, 3$ . The tree root level 0 is not considered to be in one of the layers.

In instance  $I$  there are three different types of cost functions. First, there are functions  $f_2^i$  and  $f_3^i$  for  $i = 1, \dots, m$ . For  $i < m$ ,  $f_q^i$  is defined as  $f_q^i(l_q^i) = 0$  and  $f_q^i(x) = 1$  for any  $x \neq l_q^i$ . The  $m$ th pair is defined as  $f_2^m(x) = f_3^m(x) = 0$  for  $x = l_2^m$ , and  $f_2^m(x) = f_3^m(x) = 1$  otherwise. Note that there are no functions  $f_1^i$ .

Second, we introduce  $m - k$  functions  $g_1, \dots, g_{m-k}$ . Those functions are all identical, for  $t = 1, \dots, m - k$  they are defined as  $g_t(l_1^i) = 0$  for  $i = 1, \dots, m$ , and  $g_t(x) = 1$  for all other values of  $x$ .

Finally,  $I$  contains  $n$  different functions  $h_1, \dots, h_n$ , one for each element of  $C = \{c_1, \dots, c_n\}$ . Let  $D = \{D_1, \dots, D_m\}$ . For each  $i = 1, \dots, m$ , select one element  $d_i \in D_i$  arbitrarily. Now, for  $j = 1, \dots, n$ , define

$$h_j(x) = \begin{cases} 0 & \text{if } x = l_2^i \text{ for some } i \text{ with } c_j = d_i \\ 0 & \text{if } x = l_3^i \text{ for some } i \text{ with } c_j \in D_i \setminus \{d_i\} \\ 1 & \text{otherwise.} \end{cases}$$

**Lemma 3.** *There is a solution to instance  $I$  having cost 0 if and only if instance  $(C, D)$  admits an exact cover.*

*Proof.* “ $\Rightarrow$ ” Assume that there is a solution  $T$  to instance  $I$  having cost 0. Then the leaf associated with function  $f_q^i$  must be on level  $l_q^i$  for  $q = 1, 2$  and  $i = 1, \dots, m - 1$ , and the leaves associated with  $f_2^m$  and  $f_3^m$  must be on level  $l_2^m$ . Because of the leaves on level  $l_2^m$ , there must be a downward path  $v_1^1, v_2^1, v_3^1, v_2^2, \dots, v_2^m$  starting in the root  $v_1^1$  of  $T$ , such that  $v_q^i$  is on level  $l_q^i - 1$ , and  $v_2^m$  is the parent of the leaves associated with  $f_2^m$  and  $f_3^m$ .

We can assume that in  $T$ , for  $1 \leq i \leq m - 1$ , the internal nodes  $v_2^i$  and  $v_3^i$  are the parents of the leaves associated with  $f_2^i$  and  $f_3^i$ , respectively. If this property does not hold, then the tree  $T$  can be modified in order to satisfy it: simply interchange children between  $v_t^i$  and the parent of the leaf associated with  $f_t^i$  appropriately. This does not change the level of any leaf, so the cost of  $T$  remains zero.

Now consider the subtrees  $T_1, \dots, T_m$ , where  $T_i$  is defined as the subtree under  $v_1^i$  which does not contain  $v_2^i$ . The set of leaves associated with the  $g$ -type and  $h$ -type functions is exactly the set of leaves being in those subtrees. For  $0 \leq i \leq m$ , the unique  $h_j$  with  $c_j = d_i$  is the only function besides  $f_2^i$  which evaluates to 0 for input  $l_2^i$ . Furthermore, only the  $g$ -type functions evaluate to 0 for input  $l_1^i$ . Therefore, if some  $T_i$  does not contain any  $g$ -type leaf, then it has at least three leaves which are associated with  $h$ -type functions.

As there are only  $m - k$  functions of type  $g$ ,  $k$  of the  $m$  subtrees  $T_i$  must contain at least three  $h$ -type leaves. As the total number of  $h$ -type leaves is  $n = 3k$ , each of those subtrees must contain exactly three of them.

Let  $T_i$  be such a subtree. Function  $h_j$  with  $c_j = d_i$  is the only function besides  $f_2^i$  which evaluates to 0 for input  $l_2^i$ , and  $h_{j'}, h_{j''}$  with  $\{c_{j'}, c_{j''}\} = D_i \setminus \{d_i\}$  are the only two functions besides  $f_3^i$  evaluating to 0 for input  $l_3^i$ .  $T_i$  must contain exactly those three functions, because otherwise it would have more than three leaves.

As the number of subtrees  $T_i$  of this kind is  $m$ , and each  $h$ -type function can only occur in one of them, the corresponding selection of  $D_i$ s must be an exact cover of  $U$ .

“ $\Leftarrow$ ” If  $D' \subset D$  is an exact cover, then one can construct a zero cost solution tree  $T$  from it which has the structure just described.  $\square$

We have given a reduction showing that it is NP-hard to decide whether a GHT instance admits a zero cost solution. This establishes the following theorem.

**Theorem 3.** *GHT and max-GHT are inapproximable unless P=NP.*

**Max-GHT with Monotonic Costs.** We give a polynomial time algorithm for the version of GHT where the cost functions are monotonic and the objective is to minimize the maximum cost caused by a leaf.

Let  $(f_1, \dots, f_n)$  be an instance of max-GHT. The cost of any solution is fully characterized by the *level assignment* function  $d : \{1, \dots, n\} \rightarrow \{0, \dots, n\}$ , which assigns the tree level of the corresponding leaves to indices of the cost functions (note that no leaf can have a depth greater than  $n$ ). Given  $d$ , the cost of the corresponding tree can be calculated as  $\text{cost}(d) = \max_i f_i(d(i))$ .

Our approach is to compute an optimal level assignment function and then derive an optimal tree from it. Given  $d$ , let  $\bar{d} : j \mapsto |\{d(i) = j\}|$  be the function which assigns to each tree level the number of leaves assigned to it by  $d$ .

**Lemma 4.** *A function  $d : \{1, \dots, n\} \rightarrow \{0, \dots, n\}$  is the level assignment function of a binary tree having  $n$  leaves, if and only if*

$$\sum_{k=0}^n \bar{d}(k) 2^{n+1-k} = 2^{n+1}. \quad (2)$$

Furthermore, for any level assignment function satisfying that equation, a corresponding binary tree can be computed in polynomial time.

*Proof.* “ $\Rightarrow$ ”: Let  $d$  be the level function of a binary tree  $T$  having leaves  $\ell_1, \dots, \ell_n$ . For  $i = 1, \dots, n$ , replace  $\ell_i$  with a full binary tree having depth

$$(n+1) - \text{depth}(\ell_i, T) = (n+1) - d(i).$$

By this transformation, we obtain the full binary tree  $T_{n+1}$  having depth  $n+1$ . For  $i = 1, \dots, n$ , the tree replacing  $\ell_i$  has  $2^{(n+1)-d(i)}$  leaves. The claim follows from the fact that  $T_{n+1}$  has  $2^{n+1}$  leaves.

“ $\Leftarrow$ ”: From the level function  $d$  we construct a tree  $T$  level by level, keeping track of the total number  $v(j)$  of nodes (internal nodes and leaves) on each level  $j$ . Our construction will maintain the invariant

$$\sum_{k=0}^{j-1} \bar{d}(k) 2^{n+1-k} + v(j) 2^{n+1-j} = 2^{n+1}.$$

If  $n = 1$ , then it must hold that  $f(1) = 0$ , so we can simply place the only leaf at the root of  $T$ . Otherwise, we place an internal node at the root of  $T$  at level 0. In both cases, the above invariant is established with respect to  $j = 0$ .

For  $1 \leq j \leq n$ , assume that we have already created level  $0, \dots, j-1$  of  $T$  and have established

$$\sum_{k=0}^{j-2} \bar{d}(k)2^{n+1-k} + v(j-1)2^{n+1-(j-1)} = 2^{n+1}.$$

We have  $v(j-1) - d(j-1)$  internal nodes at level  $j-1$ , so the total number of nodes on level  $j$  is  $v(j) = 2v(j-1) - 2d(j-1)$ . This implies  $v(j)2^{n+1-j} = v(j-1)2^{n+1-(j-1)} - d(j-1)2^{n+1-(j-1)}$ , so

$$\sum_{k=0}^{j-1} \bar{d}(k)2^{n+1-k} + v(j)2^{n+1-j} = \sum_{k=0}^{j-2} \bar{d}(k)2^{n+1-k} + v(j-1)2^{n+1-(j-1)} = 2^{n+1},$$

which establishes the invariant with respect to  $j$ . For showing that level  $j$  of the tree can be constructed, we need to prove that  $\bar{d}(j) \leq v(j)$ . This can be shown using the invariant:  $2^{n+1-j}v(j) =$

$$2^{n+1} - \sum_{k=0}^{j-1} \bar{d}(k)2^{n+1-k} \geq 2^{n+1} - \left( \sum_{k=0}^{j-1} \bar{d}(k)2^{n+1-k} + \sum_{k=j+1}^n \bar{d}(k)2^{n+1-k} \right)$$

$= 2^{n+1-j}\bar{d}(j)$ , where both equalities are implied by Equation 2.

When  $j$  is the largest index with  $\bar{d}(j) > 0$ , the above formula gives  $v(j) = d(j)$ , so any node on the deepest level of  $T$  is a leaf. Thus,  $T$  is a feasible binary tree with  $n$  leaves. The construction of  $T$  clearly takes only polynomial time.  $\square$

Lemma 4 reduces max-GHT to the search for a minimum cost assignment  $d$  satisfying Equation 2. We further simplify this task by relaxing the Equation to an Inequation. The proof of the following lemma is deferred to the full paper.

**Lemma 5.** *Let  $d : \{1, \dots, n\} \rightarrow \{0, \dots, n\}$  be a function satisfying*

$$\sum_{k=0}^n \bar{d}(k)2^{n+1-k} \leq 2^{n+1}. \quad (3)$$

*Then there is a feasible level assignment function  $d'$  with  $d'(x) \leq d(x)$  for  $1 \leq x \leq n$ . Furthermore,  $d'$  can be determined from  $d$  in polynomial time.*

**Theorem 4.** *The version of max-GHT where all cost functions are monotonic can be solved in polynomial time.*

*Proof.* We have reduced max-GHT to the problem of determining a minimum cost assignment  $d$  satisfying Inequation 3. This problem can be solved using binary search to determine the minimum cost of any feasible  $d$ . The search is performed over the set of all  $n^2$  function values of  $f_1, \dots, f_n$ , because the cost of any solution for our problem instance is one of those function values.

In each iteration of binary search, guess a value  $c$  for  $\text{cost}(d)$ . Recall that we have assumed the cost functions to be monotonic. So, for  $i = 1, \dots, n$ , there is

some  $x_i$  such that  $f_i(x) \leq c$  for  $x \leq x_i$  and  $f_i(x) > c$  otherwise. Assign  $d(i) = x_i$  for  $i = 1, \dots, n$  and check Inequation 3. If it is satisfied, then there is a solution tree to our problem instance having cost equal or less than  $c$ , which is computable in polynomial time, due to Lemma 5 and 4. Conversely, if Inequation 3 is not satisfied, then there is no solution tree to the instance having cost  $c$  or less. This is because the level function  $d'$  of such a tree could be obtained from  $d$  by decreasing the function values of a certain set of indices, and changing  $d$  in that manner can only further increase the left side of Inequation 3.  $\square$

## References

1. Huffman, D.A.: A Method for the Construction of Minimum-Redundancy Codes. In: Proceedings of the I.R.E., pp. 1098–1101 (1952)
2. Gilbert, E.N., Moore, E.F.: Variable Length Binary Encodings. Bell System Tech. J. 38, 933–968 (1959)
3. Knuth, D.E.: Optimum Binary Search Trees. Acta Informatica 1, 14–25 (1971)
4. Hu, T.C., Tucker, A.C.: Optimal Computer Search Trees and Variable-Length Alphabetical Codes. SIAM Journal on Applied Mathematics 21(4), 514–532 (1971)
5. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
6. Flajolet, P., Prodinger, H.: Level Number Sequences for Trees. Discrete Mathematics 65(2), 149–156 (1987)
7. Klawe, M.M., Mumey, B.: Upper and Lower Bounds on Constructing Alphabetical Binary Trees. In: Proc. 4th ACM-SIAM Symposium on Discrete Algorithms, pp. 185–193 (1993)
8. Hu, T.C.: A New Proof of the T-C Algorithm. SIAM J. Appl. Math 25, 83–94 (1973)
9. Hu, T.C., Kleitman, D.J., Tamaki, J.K.: Binary Trees Optimum Under Various Criteria. SIAM Journal on Applied Mathematics 37(2), 246–256 (1979)
10. Carmo, R., Donaldelli, J., Kohayakawa, Y., Laber, E.: Searching in Random Partially Ordered Sets. Theor. Comp. Science 321, 41–57 (2004)
11. Chakaravarthy, V., Pandit, V., Roy, S., Awasthi, P., Mohania, M.: Decision Trees for Entity Identification: Approximation Algorithms and Hardness Results. In: Proceedings of PODS (2007)
12. Mozes, S., Onak, K., Weizmann, O.: Finding an Optimal Tree Searching Strategy in Linear Time. In: Proceedings of SODA (2008)
13. Adler, M., Heeringa, B.: Approximating Optimal Binary Decision Trees. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 1–9. Springer, Heidelberg (2008)
14. Chakaravarthy, V., Pandit, V., Roy, S., Sabharwal, Y.: Approximating Decision Trees with Multiway Branches. In: Proceedings of ICALP (2009)
15. Baer, M.B.: Alphabetic Coding with Exponential Costs. Information Processing Letters 110(4), 139–142 (2010)
16. Cicalese, F., Jacobs, T., Laber, E., Molinaro, M.: On The Complexity of Searching in Trees: Average-case Minimization. In: Proceedings of ICALP 2010 (to appear, 2010)

# Medium-Space Algorithms for Inverse BWT<sup>\*</sup>

Juha Kärkkäinen<sup>1</sup> and Simon J. Puglisi<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Helsinki, Finland

[juha.karkkainen@cs.helsinki.fi](mailto:juha.karkkainen@cs.helsinki.fi)

<sup>2</sup> School of Computer Science and Information Technology,

Royal Melbourne Institute of Technology, Australia

[simon.puglisi@rmit.edu.au](mailto:simon.puglisi@rmit.edu.au)

**Abstract.** The Burrows–Wheeler transform is a powerful tool for data compression and has been the focus of intense research in the last decade. Little attention, however, has been paid to the inverse transform, even though it is a bottleneck in decompression. We introduce three new inversion algorithms with improved performance in a wide range of the space-time spectrum, as confirmed by both theoretical analysis and experimental comparison.

## 1 Introduction

The Burrows–Wheeler transform (BWT) [20] is an invertible transformation of a text that has a central role in some of the best data compression methods. The transform itself performs no compression — the result is just a permutation of the text — but the transformed text is easier to compress using simple and fast methods [15]. Much effort has gone into developing efficient algorithms for the forward transform, largely owing to its close relation to constructing the suffix array [17] and compressed text indexes [16]. The less studied problem of inverting the transform is the subject of this paper.

The inverse transform is a bottleneck in decompression and thus needs to be fast, particularly in applications requiring frequent decompression such as on-the-fly disk compression. The space requirement is also an issue: a typical, fast implementation requires 5 times the space of the text. As already proposed in the original paper [2], the text can be broken into smaller blocks, each of which is compressed separately. However, a large block size is preferable because it allows better compression (see e.g. [4]). Furthermore, the block size is determined during the forward transform, possibly on a machine with more memory or using a space efficient algorithm (e.g. [11]), and the inverse transform is impossible unless a sufficiently space-efficient algorithm is available.

The key operation in the inversion is a rank query:

$$\text{rank}(j) \equiv |\{i \mid i < j \text{ and } L[i] = L[j]\}|,$$

---

\* This work is supported by the Academy of Finland grant 118653 (ALGODAN) and by the Australian Research Council.

where  $L$  is the transformed text. A single scan of  $L$  is sufficient to answer all distinct rank queries in the sequential order, but during the inversion they are needed in a different order. By tabulating the answers, we obtain a simple, linear time inversion algorithm, already described in [2]. There are a few variations but all of them need at least  $n \log n$  bits of space for the tabulated answers or something equivalent. We call these *large-space algorithms*.

Space-efficient data structures for rank queries are widely used with compressed text indexes [16], but the query needed there is of a more general form:

$$\text{rank}(c, j) \equiv |\{i \mid i < j \text{ and } L[i] = c\}|.$$

We call this the *general rank query* as opposed to the *special rank query* needed in inverse BWT algorithms. Obviously, any data structure for general rank queries can be used for special rank queries, so using the techniques from compressed text indexes, we obtain space-efficient algorithms for inverse BWT. Many of these algorithms need at most  $n \log \sigma + o(n \log \sigma)$  bits of space, where  $\sigma$  is the size of the alphabet. This is only slightly more than needed for the text itself — and sometimes even less by way of compression — but this comes at a significant cost in query time, especially in practice. We call these *small-space algorithms*.

The focus of this paper is on *medium-space algorithms* that are between large-space and small-space algorithms with respect to both time and space complexity. The key characteristic of medium-space algorithms is the tabulation of *partial* answers to all special rank queries.

*Related work.* Seward [19] describes several large-space BWT inversion algorithms, among them the original algorithm from [2], and compares them experimentally. One of the algorithms, `mergedTL`, is the fastest known algorithm in practice. Seward also has two algorithms in the small-space category, but they are not competitive either in theory or in practice.

The only previous medium-space algorithm we are aware of is by Lauther and Lukovszki [13]. They also propose two small-space algorithms and provide experimental results for two of their algorithms. They identify the central role of the special rank query but not its relation to the general rank query.

Ferragina, Gagie and Manzini [3] have recently described an external memory algorithm for the inversion. It is, however, rather complicated and unlikely to be competitive except when external memory algorithms are the only option.

There is a large body of research on space-efficient data structures for (general) rank queries in the area of compressed text indexes and succinct data structures. The theoretically best results on BWT inversion achievable using those techniques are reported at the bottom of Table II.

*Our contribution.* We introduce three new medium-space algorithms for BWT inversion, offering improved space–time tradeoffs, including the most space-efficient linear-time algorithm for large alphabets. The time and space complexities are shown in Table II. The table also shows our improved analysis of the only previous medium-space algorithm in [13]. Experimental results show that the favorable tradeoff properties extend to practice too.

**Table 1.** Time and space complexities of BWT inversion algorithms. The sections correspond to large-, medium- and small-space algorithms. The space complexities exclude the space for input, output and  $\mathcal{O}(\sigma \log n)$ -bit data structures.

space (bits/symbol)	time per symbol	comment
$\lceil \log n \rceil + \lceil \log \sigma \rceil$	$\mathcal{O}(1)$	mergeTL in [19]
$\lceil \log n \rceil$	$\mathcal{O}(\log \sigma)$	indexF in [19]
$\log \lceil \log n \rceil + \log \sigma + \lceil \log \sigma \rceil$	$\mathcal{O}(1)$	[13]
$1 + \log b + \lceil \log \sigma \rceil$	$\mathcal{O}(\log(n/b))$ $-H_0 + b\sigma/n$	this paper $\lceil \log n \rceil \leq b \leq n/\sigma$
$2 + \log (\lceil \log n \rceil + 3\lceil \log \sigma \rceil)$ $+ \log \sigma + \frac{2 \log \log n}{\log n}$	$\mathcal{O}(1)$	this paper
$1 + \log b + \log \sigma$	$\mathcal{O}(\log(n/b) - \log \sigma)$	this paper $2(1 + \lceil \log n \rceil) \leq b \leq n/\sigma$
$\lceil \log \sigma \rceil + (\sigma \log n)/b$	$\mathcal{O}(b)$	[13]
$\lceil \log \sigma \rceil + \frac{\sigma}{b_1} \left( \log b_1 + \frac{\log n}{b_2} \right)$	$\mathcal{O}(b_1 + b_2)$	[13]
$H_k + \mathcal{O}\left(\frac{\log \sigma \log \log n}{\log n} + \frac{\sigma^{k+1} \log n}{n}\right)$	$\mathcal{O}(1 + \frac{\log \sigma}{\log \log n})$	[5,14]
$\log \sigma + \mathcal{O}(\frac{\log \sigma}{\log \log \sigma})$	$\mathcal{O}(\log \log \sigma)$	[6]

Perhaps of independent interest is the identification of the special rank query as an operation of interest, separate from the general rank query. The separate nature is illustrated by the fact that extending the techniques used by the large- and medium-space algorithms to general rank queries would blow up the space by factor  $\sigma$ , which is usually too much. We note that, besides inverse BWT, the locate and display procedures over BWT-based compressed indexes (see [16]) perform repeated special rank queries.

## 2 Preliminaries

Let  $S = S[0..n] = S[0]S[1]\dots S[n]$  be a string of  $n + 1$  symbols or characters. The first  $n$  characters of  $S$  are drawn from an ordered alphabet  $\Sigma$ , and the final character  $S[n]$  is a special “end of string” symbol,  $\$$ , distinct from and lexicographically smaller than all the other symbols. We assume that the symbols in  $\Sigma$  are encoded with the integers  $\{0, 1, \dots, \sigma - 1\}$  in an order preserving way.

For any  $i \in 0..n$ , the string  $S[i..n]S[0..i - 1]$  is a *rotation* of  $S$ . Let  $\mathcal{M}$  be the  $(n + 1) \times (n + 1)$  matrix whose rows are all the rotations of  $S$  in lexicographic order. Let  $F$  be the first and  $L$  the last column of  $\mathcal{M}$ , both taken to be strings of length  $n + 1$ . The string  $L$  is the **Burrows–Wheeler transform** of  $S$ . An example is given in Fig. 1. Note that  $F$  and  $L$  are permutations of  $S$ .

For a string  $X$ , integers  $j, r \in \{0, \dots, |X| - 1\}$  and a symbol  $c$ , define the following functions:

$$\begin{aligned} \text{access}_X(j) &\equiv X[j] \\ \text{rank}_X(j) &\equiv |\{i \mid i < j \text{ and } X[i] = X[j]\}| \end{aligned}$$

$F$	$L$	
\$ B A N A N	A	
A \$ B A N A N	N	
A N A \$ B A N		
A N A N A \$ B		
B A N A N A \$		
N A \$ B A N A		
N A N A \$ B A		

$F$	$L$	rank $_L$
\$	A	0
A	N	0
A	N	1
A	B	0
B	\$	0
N	A	1
N	A	2

**Fig. 1.** BWT matrix  $\mathcal{M}$  and inverse BWT permutation for text  $S = \text{BANANA\$}$ **Inverse BWT in forward order**

```

1: construct  $F$  by sorting  $L$ 
2:  $j \leftarrow \text{select}_L(\$, 0)$ 
3: for  $i \leftarrow 0$  to  $n$  do
4:    $S[i] \leftarrow c \leftarrow \text{access}_F(j)$ 
5:    $r \leftarrow \text{rank}_F(j)$ 
6:    $j \leftarrow \text{select}_L(c, r)$ 

```

**Inverse BWT in reverse order**

```

1: construct  $F$  by sorting  $L$ 
2:  $j \leftarrow \text{select}_L(\$, 0)$ 
3: for  $i \leftarrow n$  downto 0 do
4:    $S[i] \leftarrow c \leftarrow \text{access}_L(j)$ 
5:    $r \leftarrow \text{rank}_L(j)$ 
6:    $j \leftarrow \text{select}_F(c, r)$ 

```

**Fig. 2.** Two abstract algorithms for the inverse Burrows–Wheeler transform

$$\text{select}_X(c, r) \equiv \begin{cases} j & \text{if } X[j] = c \text{ and } \text{rank}_X(j) = r \\ \text{undefined} & \text{if there is no such } j \end{cases}$$

The notation  $\text{access}_X(j)$  is used instead of  $X[j]$  when  $X$  might be stored in a form that does not support trivial character access.

Two abstract inversion algorithms are given in Fig. 2. The first (left-hand side) algorithm constructs  $S$  from the beginning to the end and the second in the reverse order. Both algorithms follow the same unicyclic permutation but in different directions. An example of the permutation is shown in Fig. 1. To obtain concrete algorithms, we need to define the implementation of the operations  $\text{access}$ ,  $\text{rank}$ , and  $\text{select}$ .

### 3 Basic Large-Space Algorithms

Of the two abstract algorithms in Fig. 2, we will focus on the reverse order algorithm, as its operations are easier to implement and faster in practice. All the algorithms mentioned in this paper are based on it.

Another feature shared by all the algorithms is the implementation of  $\text{select}_F$  based on the special nature of  $F$ . The string  $F$  contains the characters of  $S$  in sorted order and all copies of the same symbol are grouped together. For any symbol  $c$ , let  $C[c]$  be the position of the first occurrence of  $c$  in  $F$ . We can implement  $\text{select}_F$  as

$$\text{select}_F(c, r) = C[c] + r .$$

The array  $C$  can be easily computed by scanning  $L$ .

The difference between various algorithms is the implementation of  $\text{access}_L$  and  $\text{rank}_L$ . We will describe next the algorithm from the seminal paper by Burrows and Wheeler [2]. They store  $L$  explicitly, making  $\text{access}_L$  trivial. The values  $\text{rank}_L(j)$  are stored in a table  $R[0..n]$ , which can be computed by scanning  $L$  while keeping account of the number of occurrences of each symbol.

The algorithm runs in linear time and needs  $n(\lceil \log n \rceil + \lceil \log \sigma \rceil) + (\sigma + \mathcal{O}(1))\lceil \log n \rceil$  bits of space. It would be very fast in practice, but for cache misses. In the main loop, the sequence of accesses to  $L$  and  $R$  is essentially random with a high likelihood of a cache miss for each access. Seward [19] describes an optimized version that replaces the arrays  $L$  and  $R$  with a single array  $LR$  that stores both values. This can reduce the number of cache misses to a half, leading to a significant improvement in speed. This algorithm, which we call Algorithm LR (`mergeTL` in [19]), is the fastest known algorithm for BWT inversion. It is also the starting point for our medium-space algorithms.

## 4 Basic Medium-Space Algorithms

In this section we describe two simple medium-space algorithms. One of them is by Lauther and Lukovszki [13] and one is new.

Both algorithms modify Algorithm LR by storing only partial information about ranks in  $R$  (i.e., in the  $R$ -fields of the array  $LR$ ). Every position  $j \in \{0, \dots, n\}$  is associated with a nearby *reference point*  $\text{ref}(j) \in \{0, \dots, n\}$ , and

$$R[j] = \text{rank}_L(j) - \text{rank}_L(L[j], \text{ref}(j)) .$$

Now we can compute a rank query as  $\text{rank}_L(j) = \text{rank}_L(L[j], \text{ref}(j)) + R[j]$ . The difference between the two algorithms is the choice of reference points and the computation of  $\text{rank}_L(L[j], \text{ref}(j))$ .

### 4.1 Algorithm LR-B

The first algorithm is by Lauther and Lukovszki [13]. We provide an improved analysis.

Divide  $R$  into  $\lceil (n+1)/b \rceil$  blocks of size  $b$ . Every position in a block is associated with the same reference point, which is the center of the block. In other words, the reference points are the positions  $b/2, b+b/2, 2b+b/2, \dots$ . As a small twist to the basic scheme, if  $j$  is in the first half of a block, i.e., if  $j < \text{ref}(j)$ , we set

$$R[j] = \text{rank}_L(L[j], \text{ref}(j)) - \text{rank}_L(j) - 1 .$$

Otherwise, i.e., if  $j \geq \text{ref}(j)$ , we use the basic scheme and set

$$R[j] = \text{rank}_L(j) - \text{rank}_L(L[j], \text{ref}(j)) .$$

Now all the values in  $R$  are in the range  $[0, b/2 - 1]$  and can be stored using  $\lceil \log b \rceil - 1$  bits. The ranks at the reference points are stored in a two-dimensional array  $R_{\text{ref}}$ , i.e., for all  $c \in \Sigma$  and  $j \in \{0, \dots, \lceil (n+1)/b \rceil - 1\}$ ,

$$R_{\text{ref}}[c, j] = \text{rank}_L(c, b/2 + jb) .$$

We need at most  $\sigma(n/b + 1)\lceil \log n \rceil$  bits for the array  $R_{\text{ref}}$ .

The following theorem summarizes the properties of the algorithm. All proofs are omitted here due to lack of space and are provided in the full paper.

**Theorem 1.** *Setting  $b = 2^k$  for  $k = \lfloor \log(\sigma \lceil \log n \rceil) \rfloor$ , Algorithm LR-B computes the inverse Burrows-Wheeler transform in  $\mathcal{O}(n)$  time using at most*

$$n(\log \lceil \log n \rceil + \log \sigma + \lceil \log \sigma \rceil) + \mathcal{O}(\sigma \lceil \log n \rceil)$$

*bits of space.*

## 4.2 Algorithm LR-I

In our new algorithm, reference points are separate for each symbol of the alphabet. For a symbol  $c$ , the reference points are at every  $b$ th occurrence of  $c$ , i.e., at positions  $\text{select}_L(c, 0), \text{select}_L(c, b), \text{select}_L(c, 2b), \dots$ . A position  $j$  is assigned to the closest preceding reference point for the symbol  $L[j]$ , i.e.,

$$\text{ref}(j) = \text{select}_L(L[j], b \lfloor \text{rank}_L(j)/b \rfloor).$$

The array  $R$  is as in the basic scheme, i.e.,  $R[j] = \text{rank}_L(j) - \text{rank}_L(L[j], \text{ref}(j))$ , and we need  $\lceil \log b \rceil$  bits for each entry. The reference points for a symbol  $c$  are stored in an array  $I_c$ , i.e.,  $I_c[i] = \text{select}_L(c, ib)$ . The arrays  $I_c$ ,  $c \in \Sigma$ , can be seen as sparse inverted lists for the symbols. The total space for them is  $n \lceil \log n \rceil / b + \mathcal{O}(\sigma \log n)$  bits. To compute  $\text{rank}_L(j)$ , we binary search  $I_{L[j]}$  to find  $i$  such that  $I_{L[j]}[i] \leq j < I_{L[j]}[i+1]$ , and then  $\text{rank}_L(j) = ib + R[j]$ .

Unlike LR-B, Algorithm LR-I offers a space-time tradeoff as shown by the following result.

**Theorem 2.** *Let  $b = 2^k$  for an integral  $k$ . If  $k = \lfloor \log \lceil \log n \rceil \rfloor$ , the space requirement of Algorithm LR-I is at most*

$$n(1 + \log \lceil \log n \rceil + \lceil \log \sigma \rceil) + \mathcal{O}(\sigma \log n).$$

*For  $\lfloor \log \lceil \log n \rceil \rfloor < k \leq \log(n/\sigma)$ , the space requirement is at most*

$$n(1 + k + \lceil \log \sigma \rceil) + \mathcal{O}(\sigma \log n).$$

*The time complexity is  $\mathcal{O}(n(\log(n/b) - H_0) + b\sigma)$ , where  $H_0 \leq \log \sigma$  is the zeroth order empirical entropy of  $S$  (see Section 5).*

## 5 Variable-Length Encoding

In this section, we show how to improve the algorithms of the previous section using variable-length encoding.

For a string  $X$ , let  $\Sigma_X$  be the set of symbols occurring in  $X$ , let  $|X_c|$  be the number of occurrences of a symbol  $c$  in  $X$ , and let  $f_X(c) = |X_c|/|X|$  be the frequency of  $c$ . The *zeroth order empirical entropy* of  $X$  is

$$H_0(X) = \sum_{c \in \Sigma_X} f_X(c) \log(1/f_X(c)) .$$

A *canonical prefix code* [18] for  $X$  is characterized by a non-decreasing sequence  $\ell = (\ell_1, \dots, \ell_{|\Sigma_X|})$  of positive, integral code lengths satisfying Kraft's inequality:  $\sum_{i=1}^{|\Sigma_X|} 2^{-\ell_i} \leq 1$ . The code lengths are assigned to symbols in decreasing order of symbol frequency; let  $\ell(c)$  denote the code length of a symbol  $c$ . There exists an assignment of binary code words  $\text{code}(c)$  of  $\ell(c)$  bits to each symbol  $c$  so that, for every  $c, c' \in \Sigma_X$  with  $f_X(c) < f_X(c')$ ,

- $\text{code}(c)$  is not a prefix of  $\text{code}(c')$  (the code is *prefix-free*), and
- $\text{code}(c)$  is lexicographically smaller than  $\text{code}(c')$  (the code is *canonical*).

Let  $\ell(X)$  be the encoded length of  $X$  for a code  $\ell$ :

$$\ell(X) = \sum_{i=0}^{m-1} \ell(X[i]) = m \sum_{c \in \Sigma_X} f_X(c) \ell(c) .$$

For any prefix code,  $\ell(X) \geq mH_0(X)$ . The equality is achieved with the *fractional* lengths  $\ell(c) = \log(1/f_X(c))$ . The Huffman code [9] is known to be the optimal code with integral lengths. However, for our purposes, we need a code where the code length of every symbol is close to the fractional optimum, which the Huffman code does not guarantee [12]. Furthermore, with one of our algorithms (VRL-I, Section 5.2), we have a strict upper limit  $h$  on the code lengths. We will be using the *length-limited rounded code*  $\hat{\ell}_X^h$  with

$$\hat{\ell}_X^h(c) = \lceil h - \log(f_X(c)(2^h - \sigma_X) + 1) \rceil ,$$

for any integer  $h \geq \lceil \log \sigma_X \rceil$ . When there is no upper limit, the code is  $\hat{\ell}_X^h = \hat{\ell}_X^\infty$  with  $\hat{\ell}_X(c) = \lceil \log(1/f_X(c)) \rceil$ . The properties of the code are established in the following lemma.

**Lemma 1.** *The code lengths  $\hat{\ell}_X^h$  define a valid prefix code for  $X$  with  $\hat{\ell}(c) \leq h$  for all  $c \in \Sigma$ . Furthermore, for all  $c \in \Sigma$ ,*

$$\hat{\ell}_X^h(c) < \log(1/f_X(c)) + \log(2^h/(2^h - \sigma_X)) + 1 ,$$

and if  $\log(1/f_X(c)) \geq h$ , then  $\hat{\ell}_X^h(c) = h$ .

## 5.1 Algorithm VLR-B

As with LR-B, we divide the array  $LR$  into blocks of size  $b$ . The reference point for all positions in a block is now the beginning of the block (instead of the center).

Let  $B$  be a block. We encode the  $L$ -fields in  $B$  with the unlimited rounded code  $\hat{\ell}_B$ , and the  $R$ -fields using  $\lceil \log |B_c| \rceil$  bits for a symbol  $c$ . The combined length of the two fields for a symbol  $c$  is

$$\lceil \log(1/f_B(c)) \rceil + \lceil \log |B_c| \rceil = \lceil \log(b/|B_c|) \rceil + \lceil \log |B_c| \rceil \leq \lceil \log b \rceil + 1.$$

Thus we need  $n(\lceil \log b \rceil + 1)$  bits for the whole  $LR$  array.

For each block  $B$ , we have a table  $V$  with an entry for each symbol  $c$  in  $\Sigma_B$  containing three fields

- The  $e$ -field has  $\lceil \log b \rceil + 1$  bits with  $\text{code}(c)$  in the beginning and the rest of the field filled with zeros.
- The  $s$ -field contains the original code for  $c$  using  $\lceil \log \sigma \rceil$  bits.
- The  $r$ -field is the rank of the symbol  $c$  at the reference point, i.e., at the beginning of the block in  $\lceil \log n \rceil$  bits.

The table  $V$  is ordered by the  $e$ -field. Given  $LR[j]$ , we find the entry in  $V[i]$  such that  $V[i].e \leq LR[j] < V[i+1].e$ . We obtain the symbol  $L[j]$  from  $V[i].s$  and its rank at the reference point from  $V[i].r$ . The rank relative to the reference point is  $LR[j] - V[i].e$ . Thus  $\text{rank}_L(j) = V[i].r + (LR[j] - V[i].e)$ .

To speed up the search in  $V$ , there is another table  $U[0..2^q - 1]$ ,  $0 \leq q \leq \lceil \log b \rceil + 1$ . The entries of  $U$  represent the bitstrings of length  $q$ . The entry for a bitstring  $Q$  contains a pointer to the first position in  $V$  with a code beginning with  $Q$ . If a code is shorter than  $q$ , say  $\hat{\ell}_B(c) < q$ , all bitstrings beginning with  $\text{code}(c)$  point to  $V[c]$ . We need  $\lceil \log \sigma \rceil$  bits for each pointer.

Using  $U$  we can short-cut to a good starting point for the search in  $V$ . The search itself can be done linearly, and we still obtain a linear-time algorithm as shown by the following theorem.

**Theorem 3.** *Setting  $q = \lceil \log \sigma \rceil$  and  $b = 2^k$  for  $k = \lceil \log(\sigma(\lceil \log n \rceil + 3\lceil \log \sigma \rceil)) \rceil$ , Algorithm VLR-B computes the inverse Burrows-Wheeler transform in  $\mathcal{O}(n)$  time using at most*

$$n \left( 2 + \log(\sigma) + \log(\lceil \log n \rceil + 3\lceil \log \sigma \rceil) + \frac{2 \log \log n}{\log n} \right) + \mathcal{O}(\sigma \log n)$$

bits of space.

## 5.2 Algorithm VLR-I

Our final algorithm is a modification of Algorithm LR-I to use variable-length fields in the  $LR$  array. Each entry in the  $LR$  array is  $h > \log \sigma$  bits. The  $L$ -fields use the length-limited rounded code  $\hat{\ell}_L^h$ , leaving  $h - \hat{\ell}_L^h(c)$  bits for the  $R$  field. Thus the reference points are placed at every  $b(c)$ th occurrence for  $b(c) = 2^{h - \hat{\ell}_L^h(c)}$ . The decoding of the  $LR$  entries is done as in Algorithm VLR-B. Otherwise the algorithm works as LR-I. The properties are summarized in the following theorem.

**Theorem 4.** Let  $h_{\min} = \lfloor 1 + \log \lceil 1 + \log n \rceil + \log \sigma \rfloor$ . When  $h = h_{\min}$ , the space requirement of Algorithm VLR-I is at most

$$n(2 + \log \lceil 1 + \log n \rceil + \log \sigma) + \mathcal{O}(\sigma \log n)$$

bits. For  $h > h_{\min}$ , the space requirement is at most

$$n(h+1) + \mathcal{O}(\sigma \log n)$$

bits. The time complexity is  $\mathcal{O}(n(\max(1, \log n - h)))$ .

## 6 Experimental Results

For testing we used the files listed in Table 2<sup>1</sup>. All tests were conducted on a 3.0 GHz Intel Xeon CPU with 4Gb main memory and 1024K L2 Cache. The machine had no other significant CPU tasks running. The operating system was Fedora Linux running kernel 2.6.9. The compiler was g++ (gcc version 4.1.1) executed with the -O3 option. The times given are the minima of three runs and were recorded with the standard C `getrusage` function. The memory requirements are sums of the sizes of all data structures as reported by the `sizeof` function.

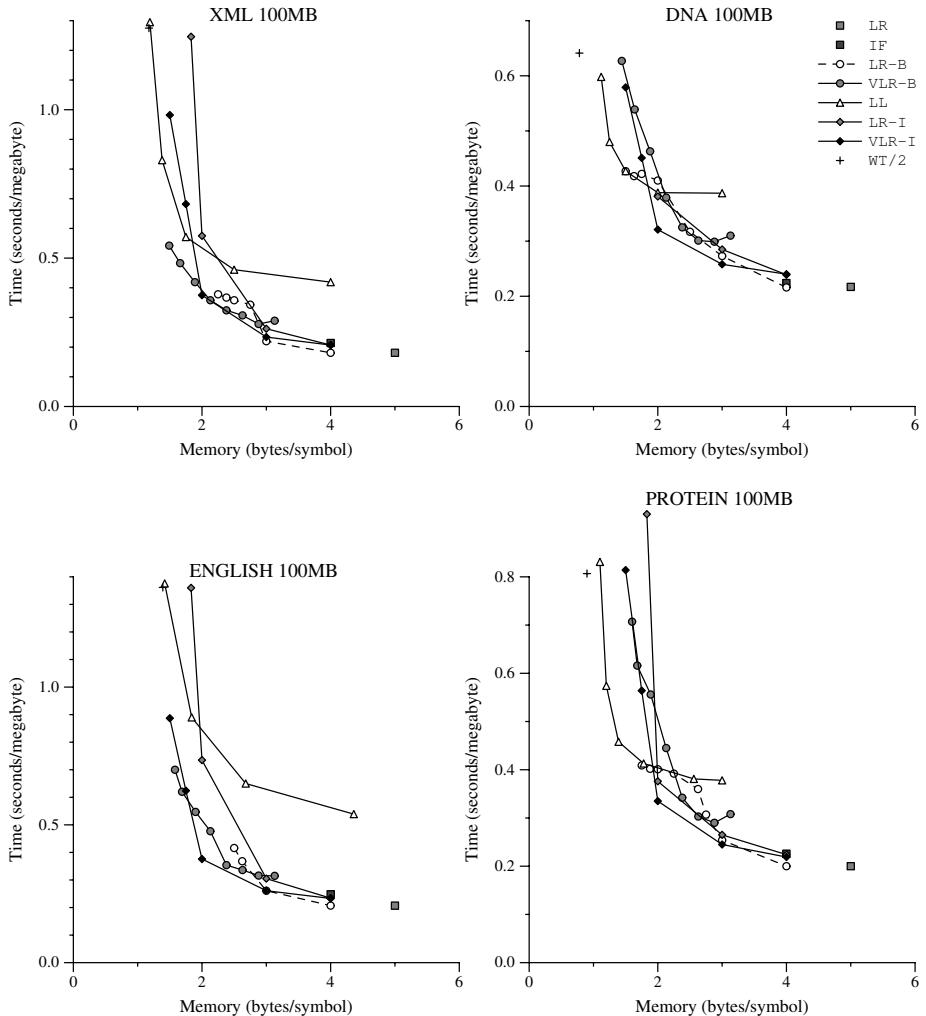
**Table 2.** Data sets used for empirical tests. For each type of data (DNA, XML, ENGLISH, PROTEIN) a 100Mb file was used.

Data set name	$\sigma$	$H_0$	mean LCP
XML	97	5.23	44
DNA	16	1.98	31
ENGLISH	239	4.53	2,221
PROTEIN	27	4.20	166

**Table 3.** Algorithms and their parameter settings. Underlined parameter values indicate that the implementation is optimized for the byte or word alignment provided by those parameters values.

Alg.	Description
LR	Algorithm LR (Sect. 3) = <code>mergedTL</code> in [19] with 32-bit integers
IF	<code>indexF</code> in [19] with 32-bit integers
LR-B	$k \in 5 + \lfloor \log \sigma \rfloor, \dots, \underline{17}, \underline{25}$
VLR-B	$k \in 10, \dots, 24$
LR-I	$k + \lfloor \log \sigma \rfloor \in 14, \underline{16}, \underline{24}, \underline{32}$
VLR-I	$h \in 12, 14, \underline{16}, \underline{24}, \underline{32}$
LL	The simple small-space algorithm in [13] Blocksizes are powers of two in $[\max(32, \sigma) \dots \min(2048, 40\sigma)]$
WT	A simple algorithm using wavelet tree for rank queries (see text)

<sup>1</sup> Available from <http://pizzachili.dcc.uchile.cl/>



**Fig. 3.** Time-memory tradeoff for various inversion algorithms. For clarity, the time shown for WT is half of the actual time, which would be far outside the graph.

The focus of the experiments is on the four algorithms described in Sections 4 and 5, but for comparison we also implemented two large-space algorithms by Seward [19] and two simple small-space algorithms, one by Lauther and Lukovszki [13] and one based on the wavelet tree [7], which is a commonly used rank data structure with compressed text indexes [16]. We optimized the wavelet tree implementation for special rank queries and used the method of Vigna [21] (the fastest we know) for bitvector rank queries. For canonical prefix coding, we use the techniques of Turpin and Moffat [20]<sup>2</sup> instead of the technique

<sup>2</sup> Originally downloaded from [http://www.cs.mu.oz.au/~alistair/mr\\_coder/](http://www.cs.mu.oz.au/~alistair/mr_coder/)

of Sect. 5.1. In all medium- and small-space algorithms, we use  $\sigma = |\Sigma_S|$  (see Table 2), which affects arrays of size  $\sigma$ , the height  $\lceil \log \sigma \rceil$  of the wavelet tree, and the size  $\lceil \log \sigma \rceil$  of the  $L$ -field in the  $LR$  array for Algorithms LR-B and LR-I. The algorithms and their parameter settings are summarized in Table 3.

The time and space requirements during BWT inversion are shown in Fig. 3. The times do not include reading the input or writing the output. The input and output are held in memory during the computation but are excluded from space requirements when the algorithm accesses them only sequentially.

All the medium-space algorithms display a fairly smooth space-time tradeoff curve, even the constant-time algorithms with no theoretical tradeoff. This is explained by cache effects. As the  $LR$  array (which always dominates the space) gets bigger, the other data structures get smaller and start to fit in the cache.

At the fast end of the space-time tradeoff, LR-B matches the speed of the fastest known algorithm, LR, in less memory. Note that this parameter setting ( $k = 25$ ) was not implemented or even suggested by Lauther and Lukovszki [13]. The middle area is dominated by the algorithms VLR-B and VLR-I using variable-length encoding. They reduce the space by a factor of 2–3 compared with LR without slowing down by more than a factor of two. The results for the small end are mixed, and anyway should be considered incomplete, since there are many possibilities for improving the small-space algorithms.

## 7 Concluding Remarks

We have introduced three new algorithms for the BWT inversion and demonstrated, theoretically and experimentally, that they improve the state of the art, particularly in the middle area of the space-time tradeoff spectrum. We are continuing our research by focusing on the extremes of the spectrum.

At the small end, the two-level version of the small-space algorithm by Lauther and Lukovszki [13], and advanced techniques from compressed text indexes such as Huffman-shaped wavelet trees [8] and implicit compression boosting [14] appear promising approaches.

At the large end, we are experimenting with an algorithm that reduces cache misses by taking advantage of repetitions in the text. Another interesting avenue for future work is exploiting properties of modern processors such as parallelism and out-of-order execution [10].

## References

1. Adjeroh, D., Bell, T., Mukherjee, A.: The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching. Springer, Heidelberg (2008)
2. Burrows, M., Wheeler, D.J.: A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California (1994)
3. Ferragina, P., Gagie, T., Manzini, G.: Lightweight data indexing and compression in external memory. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 697–710. Springer, Heidelberg (2010)

4. Ferragina, P., Manzini, G.: On compressing the textual web. In: Proc. 3rd ACM International Conference on Web Search and Data Mining, pp. 391–400. ACM, New York (2010)
5. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms* 3, Article 20 (2007)
6. Golynski, A., Munro, J.I., Rao, S.S.: Rank/select operations on large alphabets: a tool for text indexing. In: Proc. 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 368–373. ACM, New York (2006)
7. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proc. 14th ACM-SIAM Symposium on Discrete Algorithms, pp. 841–850. SIAM, Philadelphia (2003)
8. Grossi, R., Gupta, A., Vitter, J.S.: When indexing equals compression: experiments with compressing suffix arrays and applications. In: Proc. 15th ACM-SIAM Symposium on Discrete Algorithms, pp. 636–645. SIAM, Philadelphia (2004)
9. Huffman, D.A.: A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.* 40, 1098–1101 (1952)
10. Kärkkäinen, J., Rantala, T.: Engineering radix sort for strings. In: Amir, A., Turpin, A., Moffat, A. (eds.) SPIRE 2008. LNCS, vol. 5280, pp. 3–14. Springer, Heidelberg (2008)
11. Kärkkäinen, J.: Fast BWT in small space by blockwise suffix sorting. *Theoretical Computer Science* 387, 249–257 (2007)
12. Katona, G.O.H., Nemetz, T.O.H.: Huffman codes and self-information. *IEEE Transactions on Information Theory* IT-22, 337–340 (1976)
13. Lauther, U., Lukovszki, T.: Space efficient algorithms for the Burrows-Wheeler backtransformation. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 293–304. Springer, Heidelberg (2005)
14. Mäkinen, V., Navarro, G.: Implicit compression boosting with applications to self-indexing. In: Ziviani, N., Baeza-Yates, R. (eds.) SPIRE 2007. LNCS, vol. 4726, pp. 229–241. Springer, Heidelberg (2007)
15. Manzini, G.: An analysis of the Burrows-Wheeler transform. *Journal of the ACM* 48, 407–430 (2001)
16. Navarro, G., Mäkinen, V.: Compressed full-text indexes. *ACM Computing Surveys* 39, Article 2 (2007)
17. Puglisi, S.J., Smyth, W.F., Turpin, A.: A taxonomy of suffix array construction algorithms. *ACM Computing Surveys* 39, 1–31 (2007)
18. Schwartz, E.S., Kallick, B.: Generating a canonical prefix encoding. *Communications of the ACM* 7, 166–169 (1964)
19. Seward, J.: Space-time tradeoffs in the inverse B-W transform. In: Storer, J., Cohn, M. (eds.) Proc. IEEE Data Compression Conference, pp. 439–448. IEEE Computer Society, Los Alamitos (2001)
20. Turpin, A., Moffat, A.: Housekeeping for prefix coding. *IEEE Transactions on Communications* 48, 622–628 (2000)
21. Vigna, S.: Broadword implementation of rank/select queries. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 154–168. Springer, Heidelberg (2008)

# Median Trajectories<sup>\*</sup>

Kevin Buchin<sup>1</sup>, Maike Buchin<sup>2</sup>, Marc van Kreveld<sup>2</sup>,  
Maarten Löffler<sup>3</sup>, Rodrigo I. Silveira<sup>4</sup>, Carola Wenk<sup>5</sup>, and Lionov Wiratma<sup>2</sup>

<sup>1</sup> Dept. of Mathematics and Computer Science, TU Eindhoven  
[k.a.buchin@tue.nl](mailto:k.a.buchin@tue.nl)

<sup>2</sup> Dept. of Information and Computing Sciences, Utrecht University  
[{maike,marc}@cs.uu.nl](mailto:{maike,marc}@cs.uu.nl), [lionov@gmail.com](mailto:lionov@gmail.com)

<sup>3</sup> Dept. of Computer Science, University of California, Irvine  
[mloffler@uci.edu](mailto:mloffler@uci.edu)

<sup>4</sup> Dept. de Matemàtica Aplicada II, Universitat Politècnica de Catalunya  
[rodrigo.silveira@upc.edu](mailto:rodrigo.silveira@upc.edu)

<sup>5</sup> Dept. of Computer Science, University of Texas at San Antonio  
[carola@cs.utsa.edu](mailto:carola@cs.utsa.edu)

**Abstract.** We investigate the concept of a *median* among a set of trajectories. We establish criteria that a “median trajectory” should meet, and present two different methods to construct a median for a set of input trajectories. The first method is very simple, while the second method is more complicated and uses homotopy with respect to sufficiently large faces in the arrangement formed by the trajectories. We give algorithms for both methods, analyze the worst-case running time, and show that under certain assumptions both methods can be implemented efficiently. We empirically compare the output of both methods on randomly generated trajectories, and analyze whether the two methods yield medians that are according to our intuition. Our results suggest that the second method, using homotopy, performs considerably better.

## 1 Introduction

A relatively new type of geometric data that is being collected and analyzed more and more often is the *trajectory*: a path through space and time that a certain object traverses. This is due to technological advances like GPS, RFID tags, and mobile phones, and has caused an increase in demand for analysis possibilities. New analysis methods for trajectory data have been developed in the last few years, but a number of basic concepts are still lacking a satisfactory study. One of these concepts is the *median trajectory* for a given collection of

---

\* This research has been supported by the Netherlands Organisation for Scientific Research (NWO) under BRICKS/FOCUS grant number 642.065.503, under the project GOGO, and under project no. 639.022.707. M. B. is supported by the German Research Foundation (DFG) under grant number BU 2419/1-1. M. L. is further supported by the U.S. Office of Naval Research under grant N00014-08-1-1015. R. I. S. is also supported by the Netherlands Organisation for Scientific Research (NWO). C. W. is supported by the National Science Foundation grant NSF CCF-0643597.

trajectories. Intuitively, a median trajectory is a trajectory that uses pieces of the trajectories of the collection and is somehow in the middle. However, it is not clear how this concept should be defined. In this paper we establish criteria that we believe a median trajectory should meet, and we develop two median definitions that meet these criteria. Furthermore, we give algorithms to compute the median trajectory according to these definitions, and analyze experimentally whether our definitions give useful output.

**Trajectories.** Trajectories are a type of geographic data that have a temporal and a spatial component. Trajectories describe the locations over time of an entity that can move. The entity can be a person, animal, vehicle, hurricane (eye of), shopping basket (with an RFID tag), or any other moving object. We assume that the movement is continuous, but is measured at a discrete set of times.

Formally, a trajectory is the time-stamped path taken by a moving object, and is typically represented by a sequence of tuples of points and time stamps, that are points in space-time, where space is two- or three-dimensional. A collection of  $m$  trajectories  $\tau_1, \dots, \tau_m$  therefore gives rise to an input size of  $\Theta(nm)$ . In some applications, the time stamps of the  $m$  trajectories are exactly the same, while in other applications they are different. In general, trajectories can be collected with different or irregular sampling rates, at different times, and data can be missing. In between time stamps, we have no knowledge of the movement of the entity. The standard assumption is that the moving object moves with constant velocity from a time-stamped point to the next time-stamped point. Therefore, the path of a trajectory is a polygonal curve that can self-intersect, and can have repeated vertices if the entity does not move. Often, the number of vertices of a trajectory is much larger than the number of trajectories, that is,  $n \gg m$ .

**Trajectory analysis.** Analysis methods for trajectories have been developed in GIScience and in data mining. Sets of trajectories can be analyzed in a variety of ways. They can be clustered into a collection of subsets that have a high within-subset similarity and a low across-subset similarity (e.g. [11][18] and many more). They can be classified if a clustering is given [19]. Movement patterns on them can be computed [5][12][17]. Movement patterns that have been defined and for which algorithms have been suggested are flocking, convoys, herds, leadership, commuting, encounter, and various others. These intuitively represent similar movement in a group (same location), similar movement over a time span (same heading), or movement to the same position (same destination).

Several analysis tasks require a definition of (and algorithms for) similarity of trajectories. For instance, a simple similarity measure for trajectories is the average distance at corresponding times. With a similarity measure, or its inverse, a distance measure, clustering methods can easily be given. Single linkage and complete linkage clustering need a similarity measure only, and that defines the clustering. On the other hand,  $k$ -means and  $k$ -medoids clustering requires a definition of the mean and the median, respectively, regardless of whether the data are numbers or trajectories.

**Mean and median trajectories.** The intuition behind a mean trajectory is that it averages locations, one of each trajectory, like a center of gravity. In contrast, the intuition behind a median trajectory is that it always is central with respect to the number of given trajectories. Imagine a collection of GPS tracks from hikes by different people on different days. The hikers may have followed the same route globally, but there may have been options like going left around a lake or right, or taking a detour to a viewpoint. From their tracks, we want to extract a good global route. Notice that if in such a data set, seven hikers went left around a lake and three went right, then a mean trajectory would actually go through the lake, whereas a median trajectory would go with the group of seven. Similarly, with a side path to a viewpoint and back, if the majority goes to the viewpoint, then a median trajectory should do so as well. A mean trajectory might go partially to the viewpoint, which does not make sense in this context.

**Overview of results.** In Section 2 we discuss the idea of median trajectories. No definition has been suggested yet, so we investigate properties that a suitable median should have. We first propose a simple definition (*simple median*) that directly follows the definition of a *level* in an *arrangement of lines*. We also propose a more refined definition (*homotopic median*) that uses geometric and topological concepts, and may be better suited to most applications that involve trajectories. Then we discuss the maximum combinatorial complexity of a median according to these definitions.

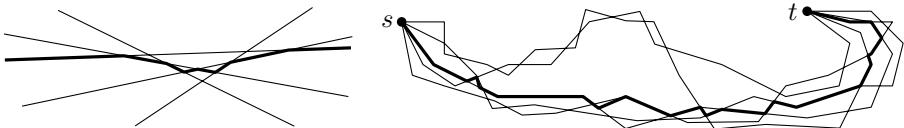
In Section 3 we present algorithms that compute median trajectories according to the two definitions. We can compute the simple median in  $O((nm)^2)$  time, and the homotopic median in  $O((nm)^{2+\epsilon})$  time for any  $\epsilon > 0$ , in the worst case. Here,  $m$  is the number of given trajectories and  $n$  is the maximum complexity of any trajectory. We improve our algorithms for practical situations. We can compute the simple median in  $O((nm + k)\alpha(nm) \log(nm))$  time, where  $\alpha$  is the inverse Ackermann function and  $k$  is the number of vertices of the median, i.e., the output complexity. Under certain assumptions related to the sampling of the trajectory, we can compute the homotopic median in  $O(nm \log^2(nm) + (nm + k)\alpha(nm) \log(nm))$  time. We note that  $k = O((nm)^2)$  in the worst case, as we show in Section 2.2. One would expect that typically,  $k = \Omega(n)$  and  $k = O(nm)$ .

In Section 4 we give results of tests that we obtain from an implementation. They mainly serve to analyze the quality of the median trajectories. In Section 5 we discuss our results and suggest directions of further research.

## 2 On the Definition of a Median Trajectory

The *mean* and the *median* are notions that define a “middle” of a set of data of a certain type. For sets of numbers, their definitions are clear, but for points in the plane, for instance, there are already many different possibilities. Existing notions of middle include the center of gravity, the center of the smallest enclosing disk (also known as *Gaussian center* or *Steiner center* [9][10]), the point that

minimizes the sum of distances to the other points (the *Weber point* [8], or the *center point* [4]). The one-dimensional equivalents of these notions correspond to the mean or the median. Observe that for some point sets, such as a set of points in convex position, no point in the set would intuitively be the mean or median. For a set of  $m$  real-valued, continuous functions the median of the functions corresponds (assuming  $m$  is odd) to the  $\lceil m/2 \rceil$ -*level* in the arrangement of the graphs of the functions. Figure 1 (left) shows an example for lines.



**Fig. 1.** Left, the median level in an arrangement of lines. Right, a median trajectory from  $s$  to  $t$  that always switches trajectory at every intersection.

Let us consider median trajectories. Trajectories include a temporal component as well as a spatial component, but it is not clear whether a median can take the temporal component into account in a useful way. We discuss some examples. Suppose the trajectories came from a group of animals that were traveling in a herd. Then we can use the temporal component because we know that the animals were together at any point in time. Next, suppose that the animals were traveling solitary, according to a similar route. Then they traveled on different days or months, and we cannot use the temporal component. Even if the animals had the same starting location of the route, we cannot simply align the starting times of the travel, because one animal may have been held up due to a predator, which upsets the time correspondence that we assumed at the start. The same is true for trajectories of cars with the same origin and destination: an initial time correspondence may easily be upset due to traffic lights or traffic conditions.

Thus, in many situations we want a median trajectory that does not take the temporal component into account. Similar motivation was given for trajectory similarity measures: many of these are partly shape-based, like dynamic time warping or largest common subsequence, or fully shape-based, like Hausdorff distance or Fréchet distance. Hence, we will concentrate on medians of trajectories based on the path of the trajectory. The median that we will define and compute will therefore be the path of a median trajectory. With slight abuse of terminology, we will just write “median trajectory” for brevity. We note that with a temporal component, some research on modeling motion and kinetic data structures is related to the median (or mean) trajectory (e.g. [123]).

## 2.1 Requirements for a Median Trajectory

Let a set  $T = \{\tau_1, \dots, \tau_m\}$  of  $m$  trajectories be given, each containing  $n$  vertices. We assume for convenience that all trajectories start at the same point  $s$  and end at the same point  $t$ ; this is a strong and unrealistic assumption but we

are interested in a clean definition of the median where behavior at the ends is not considered important. We also assume that no trajectory passes through  $s$  or  $t$  a second time and that  $s$  and  $t$  are incident to the unbounded face of the arrangement of curves corresponding to the trajectories. Note that we also assume a direction on the trajectories, namely from  $s$  to  $t$ , and for convenience we assume that  $m$  is odd. Finally, we assume that the curves do not touch or coincide with each other or themselves at any point, unless they cross, and no three trajectories pass through a common point. Several of these assumptions can be removed, but they make the description easier.

We list several required properties of the median trajectory:

1. The median trajectory is a polygonal curve from  $s$  to  $t$ .
2. Any point on the median trajectory lies on some trajectory of the input.
3. For any point  $p$  on the median trajectory, the minimum number of distinct trajectories that  $p$  must cross to reach the unbounded face (including the one(s) on which  $p$  lies) is  $(m + 1)/2$ .

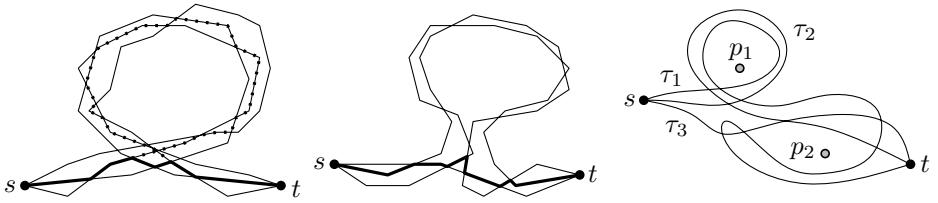
Besides these requirements, a number of desirable properties of the median trajectory can be given: Its length, total angular change, and number of vertices should be about the same as in the input trajectories. Finally, the median trajectory should be robust with respect to outliers: if ten trajectories follow the same route but one or two are completely different, the presence of these two outliers should not influence the median trajectory much. In particular, in the presence of outlier trajectories, the third property should be restated with  $m$  the number of non-outlying trajectories instead of the total number of trajectories.

Let  $\mathcal{A}$  be the arrangement formed by the (paths of the) trajectories in  $T$ . It is composed of  $O(nm)$  line segments and therefore it may have complexity up to  $\Theta((nm)^2)$ . The median trajectory is a path that follows edges of this arrangement. In the immediate neighborhood of  $s$ , it is clear how the median trajectory leaves  $s$ . Since we assume that  $s$  is on the outer face, we can order the  $m$  edges adjacent to  $s$  with the first and last edge adjacent to the outer face. Then the edge the median starts on is simply the  $(m/2)$ nd edge in the order.

**Simple definition.** Inspired by the median level in an arrangement of lines, we can give a simple definition of the median: It is the trajectory obtained after leaving  $s$  in the only possible way, and then switching the trajectory at every intersection point, following the next trajectory in the forward direction (see Figure 11 (right)). Note that if the trajectories are  $x$ -monotone, then this definition gives the same result as the  $\lceil m/2 \rceil$ -level or median function given before. We refer to a median by this definition as a *simple median*. The proofs of the following and other lemmas are omitted due to space constraints.

**Lemma 1.** *The simple median satisfies the three required properties for a median.*

Although this definition often gives desired results, it can behave badly when trajectories are self-intersecting, see Figure 12 (left). All three trajectories make the loop, but the median does not. Similarly, when the trajectories are not self-intersecting, the majority route can also be missed. In Figure 12 (middle), two



**Fig. 2.** Left, a loop in all trajectories makes the outcome (bold) of always switching trajectory undesirable. Middle, similar undesirable outcome can occur when no trajectory has self-intersections. Right, homotopy with respect to two poles.

trajectories go up and make a detour, while one trajectory stays low. The simple median in this situation also stays low.

**Homotopy definition.** To deal with this situation better, we identify parts of the plane with respect to which the median should behave the same as most of the input trajectories. In both examples of Figure 2, we have a region in the plane that is a bounded face of the arrangement  $\mathcal{A}$  that is relatively large. We propose placing *poles* in such large faces and require that the median goes in the same way around the poles as the input trajectories, using the concept of *homotopy*.

We make this more precise. Let  $T = \{\tau_1, \dots, \tau_m\}$  be the input trajectories and let  $P = \{p_1, \dots, p_h\}$  be a set of  $h$  poles which are assumed to not lie on any trajectory. Since the trajectories all go from  $s$  to  $t$ , we can use deformability of the trajectories into each other in the punctured plane [13]. Two trajectories  $\tau_i$  and  $\tau_j$  are *homotopic* if one can be deformed continuously into the other without passing over any pole, and while keeping  $s$  and  $t$  fixed. In Figure 2 (right),  $\tau_1$  and  $\tau_2$  are homotopic to each other, while  $\tau_3$  is not homotopic to  $\tau_1$  or  $\tau_2$ .

We first discuss how we find the median when all trajectories in  $T$  are homotopic with respect to  $P$ . We now use a variation of the trajectory switching approach: follow the median over the correct edge at  $s$ . Assume we have followed the median and we are on an edge of trajectory  $\tau_i$  and we encounter an intersection  $v$  with a trajectory  $\tau_j$ . If the median so far, concatenated with trajectory  $\tau_j$  from  $v$  until  $t$ , has the same homotopy type as the input trajectories, then we switch to  $\tau_j$ , otherwise we ignore the intersection and stay on  $\tau_i$ . This approach maintains the invariant that if we would simply stay on the current trajectory until  $t$ , the homotopy type of the median is correct. A median by this definition is referred to as a *homotopic median*. In Figure 2 (left) the dotted loop would be included in the homotopic median.

**Lemma 2.** *The homotopic median satisfies the required properties for a median.*

The remaining question is how to find a set of poles  $P$  such that all trajectories in  $T$  are homotopic with respect to it. A number of different strategies for this are conceivable. We choose to use a simple approach that places a pole in a face of  $\mathcal{A}$  whenever it is larger than  $r$ , i.e., a disk of size  $r$  fits in the face, for some value of  $r$  to be determined later, motivated by the fact that large faces are more

likely to be important. This choice gives no guarantee that the trajectories will be homotopic though. To solve this we could either increase  $r$  or allow some of the trajectories to have a different homotopy, and instead compute the median for a subset  $T' \subset T$ , i.e., effectively treating the remaining trajectories as outliers.

## 2.2 The Complexity of Median Trajectories

Since the arrangement  $\mathcal{A}$  formed by the  $m$  trajectories has complexity  $O((nm)^2)$ , the median trajectory cannot have more edges than that. In the full version of the paper we show that  $m$  non-self-intersecting trajectories with  $n$  edges each can indeed give rise to a median of complexity  $\Omega((nm)^2)$ , using both definitions. We also note that using the median level lower bound for arrangements of lines, we immediately obtain an  $\Omega(nm \log m)$  lower bound on the complexity of the median of  $x$ -monotone trajectories.

## 3 Algorithms to Compute a Median Trajectory

In this section we show that both methods can be implemented efficiently. The simple median can be computed in  $O((nm)^2)$  time in the worst case, while the homotopic median can be computed in  $O((nm)^2 \log(nm))$  time in the worst case. In practice, running times will be faster because they depend on complexities of intermediate results that should be much less than the worst-case situations. In particular, let  $A$  be the complexity of the arrangement formed by the  $nm$  edges of the trajectories. Although  $A = O((nm)^2)$  and this is tight in the worst case, for trajectories we typically expect it to be much smaller. Therefore, making the time bound depend on  $A$  instead of  $(nm)^2$  is desirable. Similarly, let  $h$  be the number of poles. Then  $h = O(A) = O((nm)^2)$ , but only large enough faces give poles so  $h$  is typically much smaller than  $A$ . Finally, the complexity of the median itself, the output size  $k$ , is  $O(A) = O((nm)^2)$ , but typically we expect it to be smaller. Notice that  $h$  or  $k$  can be large when the other is small.

### 3.1 Computing the Simple Median

A simple algorithm to compute the median with the simple definition is via the construction of the arrangement  $\mathcal{A}$ . The arrangement can be constructed in  $O(nm \log(nm) + A)$  time, where  $A$  is the complexity of the arrangement [13]. Then we simply follow the median trajectory through this arrangement, taking  $O(1)$  time at every intersection point or trajectory vertex. Hence, this algorithm takes  $O(nm \log(nm) + A) = O((nm)^2)$  time.

For an output-sensitive algorithm, we use Har-Peled's randomized algorithm for an on-line walk in a planar arrangement [14]. This algorithm has an expected runtime of  $O((nm+I)\alpha(nm+I) \log(nm))$  in an arrangement of  $nm$  line segments, where  $I$  is the number of intersections between the walk and the arrangement, and  $\alpha$  denotes the inverse Ackermann function. In our case,  $I = k$  is the complexity of the median, because the median switches trajectories at every intersection. Furthermore,  $I = O((nm)^2)$  and then  $\alpha(nm + I) = O(\alpha(nm))$ .

**Theorem 1.** *The simple median of  $m$  trajectories with  $n$  edges can be computed in  $O((nm)^2)$  time or in  $O((nm + k)\alpha(nm) \log(nm))$  expected time, where  $k$  is the size of the output.*

### 3.2 Computing the Homotopic Median

We give an algorithm that computes the homotopic median for a given value of  $r$ . If for the resulting poles not all trajectories are homotopic, we compute the median trajectory of the largest homotopy class with respect to these poles. The algorithm consists of three steps.

1. Compute poles, one for each face in which a disk of radius  $r$  fits.
2. Compute the homotopy type of each trajectory, and determine the type that occurs most often. Remove all trajectories that do not have this type.
3. Follow the median from  $s$ : at every intersection, determine whether the continuation on the new trajectory yields the correct homotopy type (when the new trajectory is followed to  $t$ ).

Step 1 can be performed in  $O(nm \log(nm) + A)$  time by constructing the arrangement [13] and then computing the medial axis in each face [7]; the medial axis gives the largest disk that fits inside. Step 2 can be performed using an algorithm of Cabello et al. [6]: Deciding whether two paths are homotopic takes  $O(n\sqrt{h} \log h)$  time, assuming the two paths have  $n$  edges and there are  $h$  poles. The algorithm makes use of a spanning tree on the poles so that any line intersects at most  $O(\sqrt{h})$  edges of this spanning tree, which can be constructed in  $O(h^{1+\epsilon})$  time for any  $\epsilon > 0$ . We can adapt this algorithm to determine the largest subset of homotopic trajectories. Step 3 can be performed by explicitly computing the arrangement of the trajectories. We trace the median, switching between two intersecting trajectories only if this intersection also occurs in the universal cover. The running time of this step is  $O(nm \log(nm) + A)$  time to compute the arrangement, and  $O(mn\sqrt{h} + A)$  time to trace the median.

**Theorem 2.** *The homotopic median of  $m$  trajectories with  $n$  edges can be computed in  $O((nm)^{2+\epsilon})$  time or in  $O((nm\sqrt{h} + k)\alpha(nm) \log(nm) + h^{1+\epsilon} + A)$  expected time for any  $\epsilon > 0$ , where  $h$  is the number of poles,  $A$  is the arrangement size and  $k$  is the output size.*

**Sampling assumption.** It seems reasonable to assume that the size of faces that are relevant and get a pole, represented by  $r$ , is not much less than the length  $s$  of the longest edge in any trajectory. Suppose for instance that  $r \leq s/6$ . Then the trajectories are sampled so sparsely that it can happen that two trajectories have exactly the same edge of length  $2r$ , whereas one traversed in that time unit a distance  $6r$  using three sides of a square of side length  $2r$  instead of one side; note that such a square contains a disk of radius  $r$  and therefore would contain a pole. However, we could not know that the trajectories were very different, although the choice of  $r$  suggests that this is relevant. This makes the assumption  $r = \Omega(s)$  reasonable; we refer to it as the *sampling assumption*. Under this assumption, we can improve the running times of our algorithms.

For Step 1, we determine all faces that get a pole without constructing the arrangement. Let  $D$  be a disk of radius  $r$  centered at the origin. Take the Minkowski sum of every edge of every trajectory and  $D$ , and compute the union of these  $O(nm)$  “race tracks”. The complement of the union contains parts of all faces that are large enough, although the same face of  $\mathcal{A}$  may appear as several faces in the complement of the union. Notice that homotopic equivalence is not influenced if any face of  $\mathcal{A}$  has more than one pole. The sampling assumption implies that all race tracks are *fat objects* of similar size, and hence the union complexity is bounded by  $O(nm)$  [20]. We use the algorithm of Kedem et al. [16] to construct it in  $O(nm \log^2(nm))$  time. This gives us a set of  $h = O(nm)$  poles.

For efficiency reasons in Steps 2 and 3, we must avoid having two poles closer than  $2r$ . Two such poles would lie in the same face of  $\mathcal{A}$ , so we can remove either one. We determine a subset of the poles such that every big face of  $\mathcal{A}$  has at least one pole in the subset, and any two poles in the subset are at least  $2r$  apart. We do this by computing the Delaunay triangulation of the poles. We then remove all edges that have length more than  $2r$ , and choose one pole per connected component in our subset. Let  $P$  be this subset of poles. The idea is to construct a spanning tree on  $P$  with stabbing number  $O(1)$  for line segments of length at most  $s$ .

We do this as follows: take a set of vertical lines that are exactly  $r$  apart. Within each vertical slab, connect the poles by  $y$ -coordinate. Across vertical slabs, consider every two consecutive non-empty slabs; there may be empty slabs in between. We connect the rightmost point of the left slab with the leftmost point of the right slab. Any segment of length at most  $s$  crosses  $O(1)$  slabs, and due to vertical spacing within a slab, it intersects  $O(1)$  spanning tree edges per slab. Hence, the spanning tree on  $P$  has  $O(1)$  stabbing number for line segments of length at most  $s$ , in particular, for the edges of the trajectories.

Now we again use the same algorithm as above, but replace the spanning tree with stabbing number  $O(\sqrt{h})$  by this spanning tree with  $O(1)$  stabbing number. The resulting expected running time for Step 3 is  $O((nm + k)\alpha(nm) \log(nm))$ .

**Theorem 3.** *The homotopic median of  $m$  trajectories with  $n$  edges can be computed in  $O(nm \log^2(nm) + (nm + k)\alpha(nm) \log(nm))$  expected time, where  $k$  is the size of the output, if the sampling assumption is satisfied.*

## 4 Experimental Results for Median Trajectories

In this section we present experimental results that aim at analyzing the quality of the medians generated by our two definitions. The experiments compare the definitions quantitatively, with respect to the desirable properties mentioned in Section 2—number of vertices, total length, and total turning angle—and also qualitatively, by analyzing visually in which cases one or the other method produces counterintuitive results.

**Experimental set-up.** We implemented a random trajectory generator that generates sets of “similar trajectories”. For each of these sets of trajectories, the medians for both definitions were computed and analyzed.

The random trajectory generator starts by generating a given number of waypoints uniformly distributed at random inside a rectangle, with the restriction that two consecutive waypoints are further than some minimum distance apart, and for three consecutive waypoints, the angle between them is not too small (to avoid U-turns). Given these waypoints, a given number of trajectories is generated that all follow the sequence of waypoints. For each trajectory, edges are generated that have a variation in length and in heading, but always somewhat towards the next waypoint. A waypoint is considered reached if a trajectory has a vertex within a certain distance from it, and then the next edge proceeds towards the next waypoint. The longest-to-shortest edge length ratio is 2, and the heading is up to  $45^\circ$  off from being directed to the next waypoint. With some small probability, a trajectory may temporarily not head to the next waypoint but somewhere else, resulting in outliers. Also with a small probability, a generated trajectory may skip waypoints. All trajectories start at the first waypoint and end at the last one.

**Data set.** We generated many sets of 8 waypoints and then 9 trajectories for each of them. A set was accepted if at least 6 out of 9 trajectories were homotopic for a fixed value of  $r$ . For each accepted set, the medians according to both definitions were computed, giving the length, angular change, and number of vertices for both. We distinguished four types of waypoint sets, depending on two properties of the polygonal line implied by the sequence of 8 waypoints: we compare no self-intersections (1) to self-intersections (2), and low angular change (a) to high angular change (b) (angles below or above  $3.8\pi$ ). We repeatedly generated sets until we had 100 sets in each of the four classes. Note that the classes refer to the properties of the waypoints, not the trajectories themselves (trajectories may self-intersect even if the waypoint sequence does not).

**Table 1.** Left: average length and standard deviation according to both definitions ( $\mu_S$  and  $\sigma_S$  for the simple median;  $\mu_H$  and  $\sigma_H$  for the homotopic median), where the average length of the input trajectories is normalized to 1. Middle: same for angular change. Right: same for number of vertices, again after normalization.

	length				angular change				no. of vertices			
	$\mu_S$	$\sigma_S$	$\mu_H$	$\sigma_H$	$\mu_S$	$\sigma_S$	$\mu_H$	$\sigma_H$	$\mu_S$	$\sigma_S$	$\mu_H$	$\sigma_H$
1a	0.960	0.171	0.986	0.060	5.387	1.025	4.680	0.747	3.446	0.690	3.188	0.476
1b	0.947	0.186	0.992	0.060	5.757	1.148	4.977	0.900	3.571	0.776	3.254	0.554
2a	0.513	0.263	0.963	0.115	3.807	2.112	4.656	0.970	2.071	1.062	3.433	0.753
2b	0.520	0.268	0.956	0.087	4.120	2.402	4.698	1.009	2.143	1.160	3.426	0.720

**Results.** We summarize the results in Table 1. Observe that with self-intersections the simple median method gives medians that are on average significantly shorter than the input trajectories. The other tables also indicate that often the simple method does not deal well with self-intersecting trajectories. Regarding the angular change and the number of vertices, we see that the median has much higher values than the average input. The fact that the number

of vertices of the median is higher is not surprising, and this nearly implies that the angular change is higher as well. This higher angular change is undesirable, because it can make the median trajectory appear very different from the input trajectories. Comparing the definitions, the lower standard deviation suggests that the results for the homotopic median are more consistent.

Visual inspection showed that the simple median occasionally made “errors” (against intuition) even for inputs without self-intersections, and nearly always for inputs with self-intersections. The homotopic median nearly always gave intuitive results, although an occasional “error” could be observed, due to the absence of poles in regions that are split by unrelated pieces of the trajectories. We also tested how the number of vertices of the median is influenced by the number of trajectories. There seems to be a linear dependence, suggesting that the output size  $k = \Theta(mn)$ , but this observation is highly dependent on our random trajectory generator. In summary, except for the high angular change and occasional missing parts, the homotopic median performs well even for intersecting trajectories.

## 5 Discussion and Future Research

We discussed the fundamental—but up to now missing—concept of the median of a set of trajectories. We make a first step in this direction by proposing necessary and desirable conditions that a trajectory median should satisfy. Based on them, we presented two definitions of the path of a median trajectory of a set of trajectories, together with efficient methods to compute them. We also proved properties of the resulting medians and analyzed them experimentally.

Given the importance of the concept of a median trajectory and its novelty, we believe this paper opens up many venues of further research. We made several restrictions in this paper that may be unrealistic. We assumed the start and end points of all trajectories to coincide and to lie in the unbounded face. We also assumed that most trajectories are similar enough; the homotopy method can deal with some trajectories that are outliers, but it cannot deal properly with the situation where *parts* of many trajectories are outliers. This can result in a situation where the largest homotopy class has only one trajectory, whereas an intuitively correct median may still exist. We also assumed the parameter  $r$  to be given; it would be desirable to choose it automatically in an efficient manner.

We have not addressed the question how to assign time stamps to the median or how to use the time stamps of the input to guide the computation. And of course it may be possible to define a median that has good properties in a completely different way. Finally, it would be interesting to test the definitions of medians for various types of real-world data, instead of generated data.

## References

1. Agarwal, P.K., de Berg, M., Gao, J., Guibas, L.J.: Staying in the middle: Exact and approximate medians in R1 and R2 for moving points. In: Proc. CCCG, pp. 43–46 (2005)

2. Agarwal, P.K., Gao, J., Guibas, L.J.: Kinetic medians and kd-trees. In: Möhring, R.H., Raman, R. (eds.) *ESA 2002*. LNCS, vol. 2461, pp. 5–16. Springer, Heidelberg (2002)
3. Agarwal, P.K., Guibas, L.J., Hershberger, J., Veach, E.: Maintaining the extent of a moving point set. *Discrete Comput. Geom.* 26, 353–374 (2001)
4. Amenta, N., Bern, M.W., Eppstein, D., Teng, S.-H.: Regression depth and center points. *Discrete Comput. Geom.* 23, 305–323 (2000)
5. Buchin, K., Buchin, M., Gudmundsson, J., Löffler, M., Luo, J.: Detecting commuting patterns by clustering subtrajectories. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) *ISAAC 2008*. LNCS, vol. 5369, pp. 644–655. Springer, Heidelberg (2008)
6. Cabello, S., Liu, Y., Mantler, A., Snoeyink, J.: Testing homotopy for paths in the plane. *Discrete Comput. Geom.* 31, 61–81 (2004)
7. Chin, F.Y.L., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.* 21, 405–420 (1999)
8. Durocher, S., Kirkpatrick, D.: The projection median of a set of points. *Comput. Geom.* 42, 364–375 (2009)
9. Durocher, S., Kirkpatrick, D.G.: The Steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. *Int. J. Comput. Geom. Appl.* 16, 345–372 (2006)
10. Durocher, S., Kirkpatrick, D.G.: Bounded-velocity approximation of mobile Euclidean 2-centres. *Int. J. Comput. Geom. Appl.* 18, 161–183 (2008)
11. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: Proc. 5th KDD, pp. 63–72 (1999)
12. Gudmundsson, J., van Kreveld, M., Speckmann, B.: Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica* 11, 195–215 (2007)
13. Halperin, D.: Arrangements. In: Goodmann, J.E., O'Rourke, J. (eds.) *Handbook of Discrete and Comput. Geom*, pp. 529–562. Chapman & Hall/CRC, Boca Raton (2004)
14. Har-Peled, S.: Taking a walk in a planar arrangement. *SIAM J. Comput.* 30(4), 1341–1367 (2000)
15. Hershberger, J., Snoeyink, J.: Computing minimum length paths of a given homotopy class. *Comput. Geom.* 4, 63–97 (1994)
16. Kedem, K., Livne, R., Pach, J., Sharir, M.: On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.* 1, 59–70 (1986)
17. Laube, P., Purves, R.S.: An approach to evaluating motion pattern detection techniques in spatio-temporal data. *Comp., Env. and Urb. Syst.* 30, 347–374 (2006)
18. Lee, J., Han, J., Whang, K.-Y.: Trajectory clustering: a partition-and-group framework. In: Proc. ACM SIGMOD Int. Conf. Man. of Data, pp. 593–604 (2007)
19. Lee, J.-G., Han, J., Li, X., Gonzalez, H.: TraClass: Trajectory classification using hierarchical region-based and trajectory-based clustering. In: *PVLDB* 2008, pp. 1081–1094 (2008)
20. van der Stappen, A.F., Halperin, D., Overmars, M.H.: The complexity of the free space for a robot moving amidst fat obstacles. *Comput. Geom.* 3, 353–373 (1993)

# Optimal Cover of Points by Disks in a Simple Polygon

Haim Kaplan\*, Matthew J. Katz\*\*, Gila Morgenstern\*\*\*, and Micha Sharir†

**Abstract.** Let  $P$  be a simple polygon, and let  $Q$  be a set of points in  $P$ . We present an almost-linear time algorithm for computing a minimum cover of  $Q$  by disks that are contained in  $P$ . We generalize the algorithm above, so that it can compute a minimum cover of  $Q$  by homothets of any fixed compact convex set  $\mathcal{O}$  of constant description complexity that are contained in  $P$ . This improves previous results of Katz and Morgenstern [20]. We also consider the disk-cover problem when  $Q$  is contained in a (not too wide) annulus, and present a nearly linear algorithm for this case too.

## 1 Introduction

Let  $P$  be a simple  $n$ -gon in the plane, and let  $Q$  be a set of  $m$  points in  $P$ . A *disk cover of  $Q$  with respect to  $P$*  is a set  $\mathcal{D}$  of disks (of variable radii), such that the union of the disks of  $\mathcal{D}$  covers (i.e., contains)  $Q$  and is contained in  $P$ . In other words, each disk  $D \in \mathcal{D}$  is contained in  $P$ , and each point  $q \in Q$ , lies in at least one disk  $D \in \mathcal{D}$ . A *minimum disk cover of  $Q$  with respect to  $P$*  is a disk cover of  $Q$  with respect to  $P$  of minimum cardinality. The problem of computing a minimum disk cover of  $Q$  with respect to  $P$  was introduced and studied by Katz and Morgenstern [20]. They also considered the case where the

---

\* School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: [haimk@post.tau.ac.il](mailto:haimk@post.tau.ac.il). Work by Haim Kaplan was partially supported by the U.S.-Israeli Binational Science Foundation, project number 2006-204, and by grant 975/06 from the Israel Science Fund.

\*\* Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel. E-mail: [matya@cs.bgu.ac.il](mailto:matya@cs.bgu.ac.il). Work by Matthew Katz was partially supported by the MAGNET program of the Israel Ministry of Industry, Trade & Labor (COR-NET consortium), and by the Lynn and William Frankel Center for Computer Sciences.

\*\*\* Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel. E-mail: [gila@cs.bgu.ac.il](mailto:gila@cs.bgu.ac.il). Work by Gila Morgenstern was partially supported by the Lynn and William Frankel Center for Computer Sciences.

† School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA. E-mail: [michas@post.tau.ac.il](mailto:michas@post.tau.ac.il). Work by Micha Sharir was partially supported by NSF Grant CCF-08-30272, by grant 2006-194 from the U.S.-Israeli Binational Science Foundation, by grant 338/09 from the Israel Science Fund, Israeli Academy of Sciences, and by the Hermann Minkowski-MINERVA Center for Geometry at Tel Aviv University.

covering objects are homothets (contained in  $P$ ) of any fixed compact convex set  $\mathcal{O}$  of constant description complexity. In both cases, exact polynomial-time solutions were presented. In this paper we present alternative and significantly faster solutions for both disks and homothets, and also consider the case where the points are in an annulus. All our solutions run in close to linear time.

*Background.* Geometric covering problems have been studied extensively. These problems are instances induced by geometric settings of the well-known set cover problem. Most of these instances are known to be NP-hard. Let us briefly review several geometric covering problems that are related to the problems studied in this paper.

In the unit disk cover problem, the goal is to cover a given set of points with the smallest possible number of unit disks. A polynomial-time approximation scheme (PTAS) for this problem was given by Hochbaum and Maass [17]. In the discrete version of this problem, the covering unit disks must be selected from a given set of unit disks. Until recently only constant-factor approximation algorithms were known for the discrete version; see [2, 5, 6, 30]. This was recently improved by Mustafa and Ray [29], who presented a PTAS for the discrete version (as well as for several other problems), which is based on local search.

Hurtado et al. [19] studied the related problem of computing a minimum enclosing disk of a given set of  $m$  points, whose center must lie in a given convex  $n$ -gon; they presented an  $O(m+n)$ -time algorithm for this problem. The 2-center problem with obstacles was studied by Halperin et al. [16]. In this problem, the goal is to find two congruent disks of smallest radius whose union covers a given set of  $m$  points and whose centers lie outside a given set of disjoint simple polygons with a total of  $n$  edges. They presented a randomized  $O(n \log^2(mn) + mn \log^2 m \log(mn))$  expected time algorithm for this problem. The analogous 1-center problem was studied by Halperin and Linhart [15], who presented an  $O((m+n) \log(mn))$ -time algorithm for this problem.

The solutions of Katz and Morgenstern [20] for the problems mentioned above are based on the “perfect graph approach”, previously used in the solution of several art-gallery problems, under restricted models of visibility; see, e.g., [21, 22, 27, 28, 32]. In this approach, one first defines a graph  $G$  corresponding to the input scene. Next, the following two theorems are proven: (i) There is a one-to-one correspondence between a minimum cover of the desired kind (e.g., disk cover) and a minimum clique cover of  $G$ , and (ii)  $G$  is perfect. Note that the second claim is crucial, since, in general, minimum clique cover is NP-complete, but is polynomial for perfect graphs [13, 14] and in particular for chordal graphs<sup>1</sup> [11]. The algorithms of Katz and Morgenstern consist of three stages. First, construct  $G$ , next, find a minimum clique cover of  $G$ , and finally, construct the cover of  $Q$  corresponding to the minimum clique cover of  $G$ . The bottlenecks of their algorithms are the first and last stages, which, in the case of covering by disks within a simple polygon, required  $O(nm^2)$  time.

---

<sup>1</sup> A graph is *chordal* if every cycle of at least four edges has a *chord*, i.e., an edge connecting two non-consecutive vertices of the cycle; see [13].

In this paper we take a different approach, avoiding the explicit construction of the graph  $G$ , and computing the cover itself by following the algorithm of Gavril [11] for finding a minimum clique cover in a chordal graph, and by exploiting the special geometric structure of  $G$ . This leads to improved solutions, which, when carefully implemented, run in nearly linear time.

The rest of this paper is organized as follows: In Section 2, we describe an algorithm for computing a minimum cover of  $Q$  by disks contained in  $P$  in  $O((n + m(\log n + \log^2 m)))$  time and  $O(n + m)$  space. In Section 3, we extend this result to the case of  $\mathcal{O}$ -cover, that is, computing a minimum cover of  $Q$  by homothets of an object  $\mathcal{O}$  which are contained in  $P$ , where  $\mathcal{O}$  is as above. We show that such a cover can be computed within the same bounds. Finally, in Section 4 we consider the case where the point set  $Q$  is contained in a “not too wide” annulus  $R$ , and give an  $O(m \log m)$ -time algorithm for computing a minimum-disk cover of  $Q$  by disks contained in  $R$ .

## 2 Minimum Disk Cover in a Simple Polygon

Let  $P$  be a simple polygon with  $n$  edges and let  $Q$  be a set of  $m$  points inside  $P$ . We present a nearly linear time algorithm for finding a minimum cover of  $Q$  by disks contained in  $P$ .

Consider the Voronoi diagram of the relatively open edges and reflex vertices of  $P$ , confined to within  $P$ . We refer to these relatively open edges and reflex vertices collectively as *boundary features* (or simply *features*) of  $P$ , and let  $P^*$  denote the set of these features. The *medial axis*  $M$  is the network of vertices, straight edges, and parabolic arcs of this Voronoi diagram which are strictly inside  $P$ , including also the non-reflex vertices of  $P$ . More precisely, a vertex of  $M$  which is not a vertex of  $P$  is a point at equal and smallest distance from three features of  $P$  (including the case where two of these features are an edge  $e$  and a reflex endpoint of  $e$ ). An edge of  $M$  is the locus of all points at equal (and smallest) distance from two features of  $P$  (this time excluding the case of an edge and one of its endpoints). It is well known (and easy to show) that the network  $M$  is in fact a tree; that is, it is a connected network without cycles. See Fig. 1(a)<sup>2</sup>.

As we will argue shortly, when constructing a disk cover of  $Q$  inside  $P$ , it suffices to consider disks whose centers lie on  $M$ , and in fact only maximal such disks, whose boundary touches  $\partial P$  (necessarily in at least two points).

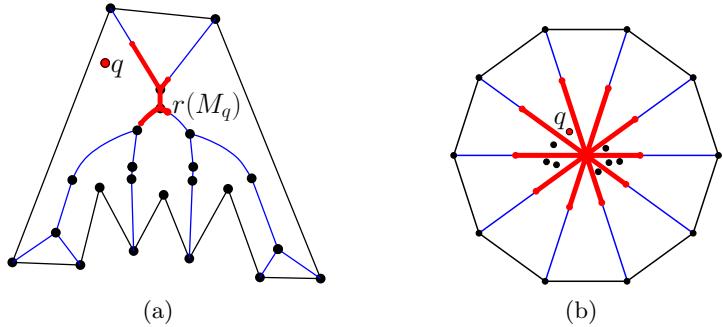
The proof of Lemma 1 below, a major geometric component of our analysis, can be found in the full version of this paper<sup>3</sup>.

**Lemma 1.** *For each point  $q \in Q$ , the portion  $M_q$  of the medial axis consisting of centers of maximal disks that contain  $q$  and are contained in  $P$  is connected.*

<sup>2</sup> The tree property may fail if we include in  $M$  edges at equal distance from an edge  $e$  of  $\partial P$  and from a reflex endpoint of  $e$ .

<sup>3</sup> The full version of this paper can be found at:

<http://www.cs.bgu.ac.il/~gilamor/papers/KKMS10.pdf>



**Fig. 1.** (a) The medial axis of a simple polygon and the subtree  $M_q$  of a point  $q \in Q$  (drawn bold, with its root  $r(M_q)$  highlighted). (b) The complexity of each subtree  $M_q$  is  $\Theta(n)$  in this example.

Consider the graph  $G$  whose vertices are the points of  $Q$ , and there is an edge between two points  $p, q \in Q$  if there exists a disk containing both  $p$  and  $q$  and contained in  $P$ . Clearly there is such a disk if and only if there is a disk containing  $p$  and  $q$  whose center is on  $M$  and whose radius is the distance from its center to  $\partial P$ . This follows by noting that every disk  $D$  contained in  $P$  is contained in a maximal disk of the above kind, which is obtained by inflating  $D$  about its center until it touches  $\partial P$ , and then by moving its center away from the contact with  $\partial P$ , maintaining that contact, until a second contact is made.

For each  $q \in Q$ , the portion  $M_q$  of  $M$  considered in Lemma 1 is equal to the intersection of  $M$  with the Voronoi cell of  $q$  in the diagram of  $P^* \cup \{q\}$ . The lemma asserts that  $M_q$  is a connected subset of  $M$ . We refer to  $M_q$  as a *subtree* of  $M$ , but note that the leaves of  $M_q$  (points of  $M_q$  whose removal does not disconnect  $M_q$ ) need not necessarily be vertices of  $M$ , but can lie in the relative interior of edges of  $M$ ; see Fig. 1(a). It follows by definition that, by identifying each point  $q$  of  $Q$  with its subtree  $M_q$ ,  $G$  becomes the intersection graph of these subtrees. Therefore, by the characterization of Buneman and Gavril [4,12],  $G$  is a *chordal* graph.

We note that computing the subtrees  $M_q$  explicitly is too expensive, because their overall complexity can be  $\Theta(mn)$  in the worst case, as depicted in Fig. 1(b).

As shown by Katz and Morgenstern [20], a cover of  $Q$  by disks contained in  $P$  corresponds to a clique cover of  $G$  and vice versa. This also follows from the fact that subtrees of a tree  $T$  satisfy the *2-Helly-property*: if every pair of subtrees in a given collection intersect, then they all have a common intersection [13]. Therefore, a clique of  $G$  can be covered by a single disk which is a maximal disk centered at a common point of all subtrees  $M_q$  corresponding to the points  $q$  of the clique. (The converse direction is trivial.) So our problem is to find a minimum clique cover of  $G$ . As is known [11], the chordality of  $G$  implies that this latter problem is solvable in polynomial time, but, exploiting the special geometric structure of the graph  $G$ , we are able to solve it particularly efficiently, by an algorithm which runs in  $O(n+m(\log n+\log^2 m))$  time (improving upon the cubic algorithm of Katz and Morgenstern in [20] mentioned in the introduction).

We follow the algorithm of Gavril [11] for finding a minimum clique cover in a chordal graph. The algorithm is greedy and is based on the fact (see, e.g., [13]) that a chordal graph contains a *simplicial vertex*  $v$ , that is, a vertex whose neighbors induce a clique. The greedy clique cover algorithm takes as the first clique in the cover such a simplicial vertex  $v$  and its neighbors. It then deletes  $v$  and its neighbors and iterates this step on the subgraph induced by the remaining vertices (which is still chordal). It is not hard to see that the output clique cover is indeed minimum, because the simplicial vertices picked at each iteration form an independent set in the original  $G$ , and clearly the size of any clique cover is at least the size of this (or any) independent set.

We can implement this algorithm using the subtree representation of  $G$ , as follows. Root  $M$  at an arbitrary vertex of  $\partial P$ . Making  $M$  rooted induces a root for each subtree  $M_q$ , for  $q \in Q$ , which we denote by  $r(M_q)$ . Note that  $r(M_q)$  is not necessarily a vertex of  $M$  but can lie in the relative interior of an edge. See Fig. 1(a).

Pick  $M_q$  such that the (appropriately defined) subtree of  $M$  rooted at  $r(M_q)$  contains no other root. The points of  $Q$  corresponding to all subtrees which contain  $r(M_q)$  are the first clique in our cover ( $q$  is a simplicial vertex). The disk corresponding to this clique can be taken to be the maximal disk within  $P$  centered at  $r(M_q)$ . We then delete all subtrees containing  $r(M_q)$  and iterate, until all of  $Q$  is exhausted.

We now present an efficient implementation of this algorithm. It consists of the following steps.

1. For each point  $q \in Q$  compute an “anchor point”  $A(q) \in M_q$ .
2. Starting from each  $A(q)$ , search  $M$  to find the corresponding root  $r(M_q)$ .
3. Maintain the set of disks  $\mathcal{D}$  that have already been placed in the cover in a data structure  $\mathcal{S}$  that can efficiently test whether a query point is in the union of the disks in  $\mathcal{D}$ .
4. Search  $M$  in a bottom-up manner to find the next simplicial vertex  $q$  of  $G$  (whose root  $r(M_q)$  is lowest in  $M$ , ignoring subtrees corresponding to points that are already covered). When the search reaches a root  $r(M_q)$  we check whether  $q$  is in the union of the disks in  $\mathcal{D}$ . If so, we skip  $r(M_q)$  and continue. Otherwise we add to  $\mathcal{D}$  the maximal disk centered at  $r(M_q)$ , update  $\mathcal{S}$  accordingly, and continue. At the end  $\mathcal{D}$  contains the desired minimum disk cover.

We now give the details of the implementation of each of these steps. We first compute  $M$  in  $O(n)$  time, using the algorithm of Chin et al. [10]. We regard the edges and vertices of  $M \cup \partial P$  as the edges and vertices of a planar map  $H$ . We further partition  $H$  into “pseudo-trapezoids” (referred to as “trapezoids”, in short), by connecting each vertex of  $M$  (lying in the interior of  $P$ , including breakpoints along edges which are equidistant from an edge of  $P$  and an endpoint of that edge) to its nearest points on  $\partial P$ .

For each point  $q \in Q$  we compute the trapezoid  $T(q)$  in the decomposition of  $H$  containing  $q$ . To do so we preprocess the decomposition of  $H$  in  $O(n)$  time and construct the point location data structure of Kirkpatrick [24], which supports

logarithmic-time point-location queries. Then we locate the trapezoids  $T(q)$ , for  $q \in Q$ , by  $m$  point location queries to this data structure, in  $O(m \log n)$  time.

For each  $q \in Q$ , let  $e$  be the feature of  $P^*$  on  $\partial T(q)$  (that is,  $T(q)$  is contained in the Voronoi cell of  $e$ ). We compute the closest point  $q'$  to  $q$  on  $e$  and take  $A(q)$  to be the intersection of the line  $q'q$  with  $M$  which is closest to  $q$  (and which also lies on  $\partial T(q)$ ). It is easy to see that a maximal disk centered at  $A(q)$  inside  $P$  contains  $q$ , and therefore  $M_q$  indeed contains  $A(q)$ .

We compute the roots  $r(M_q)$ , for  $q \in Q$ , as follows. We fix a root for  $M$ , for example, the rightmost vertex of  $P$ . We then traverse  $M$  in depth-first order from the root, and maintain its vertices on the stack (those vertices whose subtrees are still being traversed) in an array  $L$ , stored in their reverse order on the stack, i.e., in their order along the path of  $M$  from the root to the vertex currently being explored ( $L$  is in fact just a concrete structure for implementing the stack). When the search moves from a vertex  $v$  to a new vertex  $w$ , we insert  $w$  as the rightmost element of  $L$ , and when the search backtracks from a vertex  $v$ , we delete  $v$  from (the end of)  $L$ .

When we traverse the edge  $(v, w)$  during the depth-first search, we find  $r(M_q)$  for every point  $q$  such that  $A(q)$  is on the edge  $(v, w)$ , as follows. First observe that testing whether a point  $z \in M$  belongs to  $M_q$  is easy to do in  $O(1)$  time, provided we know the feature (edge or vertex) of  $M$  containing  $z$ : This is equivalent to testing whether  $|zq|$  is at most the radius of the maximal disk centered at  $z$ , which is readily available since we know the features of  $P^*$  nearest to  $z$ .

So let  $q \in Q$  be a point whose anchor  $A(q)$  lies on  $(v, w)$ . If  $v \notin M_q$  then  $r(M_q)$  is on the edge  $(v, w)$ . Furthermore, it is a point on  $(v, w)$  at equal distances to the two features of  $P$  defining the edge  $(v, w)$  and to  $q$ . We compute it by solving the appropriate system of algebraic equations (as is well known, there can be at most two solutions, for otherwise, since Voronoi regions are star-shaped, we would get an impossible planar embedding of  $K_{3,3}$ ), and by taking the solution which lies on  $(v, w)$  closest to the root of  $M$  (i.e., to  $v$ ).

If  $v \in M_q$ , we use binary search on the array  $L$  to find the farthest ancestor  $u$  of  $v$  which is still in  $M_q$ . The root  $r(M_q)$  is on the edge from  $u$  to its parent, and we find it by solving a system of algebraic equations analogous to the one described above.

Having collected all the roots  $r(M_q)$ , we sort them along the edges of  $M$  containing them, and split the edges at these roots, making the roots additional vertices of  $M$ .

Finally we collect the roots which are centers of the maximal disks in our cover, using the greedy algorithm described above. For that we use a data structure  $\mathcal{S}$  that maintains a set  $\mathcal{D}$  of disks, subject to insertions of disks and queries of the form: Given a point  $q$ , determine whether  $q$  is in the union of the disks currently in  $\mathcal{D}$ . The structure  $\mathcal{S}$  is initially empty, but, as we add disks to the cover  $\mathcal{D}$ , we insert them into  $\mathcal{S}$ .

We traverse  $M$  again bottom-up, stopping at each root  $r(M_q)$ . When we encounter a root  $r(M_q)$ , we use the data structure  $\mathcal{S}$  to determine whether the corresponding point  $q$  lies in the union of the disks currently in  $\mathcal{D}$ . If the answer

is yes, we continue traversing  $M$ , effectively ignoring  $q$ . Otherwise, we add to  $\mathcal{D}$  the maximal disk centered at  $r(M_q)$  and contained in  $P$ , update  $\mathcal{S}$  accordingly, and continue. When the traversal terminates,  $\mathcal{D}$  is the desired minimum disk cover.

We now analyze the complexity of the algorithm. Computing  $M$  and the Voronoi diagram  $H$  induced by  $M \cup \partial P$ , the triangulation of  $H$  into trapezoids, and the point location data structure for this triangulation, takes  $O(n)$  time [10, 24]. For each point  $q \in Q$ , computing  $A(q)$  takes  $O(\log n)$  time, for a total of  $O(m \log n)$  time. The computation of the roots  $r(M_q)$  takes  $O(n + m \log n)$  time: maintaining the array  $L$  takes  $O(n)$  time, and the binary searches for the roots take a total of  $O(m \log n)$  time. Sorting the roots along the edges of  $M$  and subdividing  $M$  at these roots takes  $O(n + m \log m)$  time.

Finally, in the last step we again traverse  $M$  and use the data structure  $\mathcal{S}$  to determine, for each root  $r(M_q)$ , whether its corresponding point  $q$  is in the union of the disks that we have already added to the cover. We construct  $\mathcal{S}$  using a standard reduction [3] that converts a static point location data structure to a dynamic incremental one, as follows. We recall that the combinatorial complexity of the boundary of the union of  $k$  disks is  $O(k)$  [23], and that one can compute this union and preprocess it for logarithmic-time point location queries in time  $O(k \log k)$ . Let  $k$  be the number of disks currently in  $\mathcal{D}$ , and let  $b_{\lfloor \log k \rfloor} \cdots b_1 b_0$  be the binary representation of  $k$ . For each  $i$ ,  $0 \leq i \leq \lfloor \log k \rfloor$ , such that  $b_i = 1$ ,  $\mathcal{D}$  contains a static point location data structure for the union of a subset of  $2^i$  disks. Moreover, each of the  $k$  disks belongs to exactly one of these subsets. Thus, given a query point  $q$ , one can determine in  $O(\log^2 k)$  time whether  $q$  lies in the union of the  $k$  disks, by performing a point location query in at most  $\lfloor \log k \rfloor + 1$  substructures. To insert a new disk into  $\mathcal{S}$ , we construct a new static data structure containing the new disk and all the disks in the subsets corresponding to the maximal block of least significant bits  $b_0, b_1, \dots$ , which are equal to 1 in the binary representation of the current  $k$ . This is analogous to performing an increment of the binary counter  $b_{\lfloor \log k \rfloor} \cdots b_1 b_0$ . Since each disk participates in the construction of at most  $O(\log k)$  static structures, an insertion takes  $O(\log^2 k)$  amortized time. In summary, we obtain:

**Theorem 2.** *Let  $P$  be a simple polygon with  $n$  edges and  $Q$  a set of  $m$  points contained in  $P$ . We can compute a minimum cover of  $Q$  by disks contained in  $P$  in  $O(n + m(\log n + \log^2 m))$  time and  $O(n + m)$  space.*

**Remark.** If the minimum cover consists of  $k \ll m$  disks, the running time improves to  $O(n + m(\log n + \log m + \log^2 k))$ . Hence, if  $k = O(1)$ , say, or if we just want to decide whether  $Q$  can be covered by at most  $k = O(1)$  disks, the superlinearity of the bound is caused only by the steps which compute the anchor points  $A(q)$  and the roots  $r(M_q)$ , for  $q \in Q$ . It is a challenging open problem to come up with an alternative approach which avoids the supelinear cost of these steps.

### 3 Covering by Homothetic Copies of a Convex Set

Let  $P$  and  $Q$  be as in Section 2, and let  $\mathcal{O}$  be some fixed compact convex set with nonempty interior. For a large part of the analysis, this is essentially all we assume about  $\mathcal{O}$ . For the algorithmic part, however, we need to assume that  $\mathcal{O}$  has a sufficiently simple shape so as to facilitate efficient implementation of certain operations on  $\mathcal{O}$  as well as efficient construction of a dynamic point location data structure, similar to the structure  $\mathcal{S}$  used above. For the time being, we only add the assumption that  $P$ ,  $Q$  and  $\mathcal{O}$  are in general position, to avoid possible degeneracies in the constructs that extend those studied in the preceding section. Further assumptions will be elaborated below.

We fix some point  $o$  inside  $\mathcal{O}$  as its “center point”, and assume that  $\mathcal{O}$  is initially specified so that  $o$  lies at the origin. Thinking of each point of  $\mathcal{O}$  as a vector,  $\lambda\mathcal{O}$  then denotes the convex set obtained from  $\mathcal{O}$  by scaling it about  $o$  by  $\lambda$ , for any positive  $\lambda$ . The *convex distance function* induced by  $\mathcal{O}$  is  $d_{\mathcal{O}}(p, q) = \inf\{\lambda \mid q \in p + \lambda\mathcal{O}\}$ . We refer to it as the  $\mathcal{O}$ -distance. Chew and Drysdale [8] were the first to study Voronoi diagrams under convex distance functions; see also Leven and Sharir [25]. Note that  $d_{\mathcal{O}}$  is a metric if and only if  $\mathcal{O}$  is centrally symmetric with respect to its center.

The medial axis  $M_{\mathcal{O}}$  of  $P$  under the  $\mathcal{O}$ -distance is defined analogously to the medial axis of  $P$  under the Euclidean distance, as the locus of all points inside  $P$  whose  $\mathcal{O}$ -distance to the boundary is attained in at least two points. Assuming general position,  $M_{\mathcal{O}}$  is a connected 1-dimensional network, consisting of vertices and edges. Each edge is the locus of all points which are at the same (nearest)  $\mathcal{O}$ -distance from two features of  $P^*$  (where  $P^*$  is defined as in the previous section). A vertex of  $M_{\mathcal{O}}$  which is not a vertex of  $P$  is a point at the same (nearest)  $\mathcal{O}$ -distance from three features of  $P^*$ . The shape of the edges of  $M_{\mathcal{O}}$  depends on the shape of  $\mathcal{O}$ . For example, if  $\mathcal{O}$  is a convex polygon then each edge is a polygonal curve, whose breakpoints correspond to placements of the center  $o$  of  $\mathcal{O}$  at which a vertex of  $\mathcal{O}$  touches a vertex of  $P$ . See [8][25] for more details. Finally, as in the previous section, it follows from basic properties of generalized Voronoi diagrams that  $M_{\mathcal{O}}$  is a tree.

As in the preceding section, for a point  $q \in Q$ , we denote by  $M_q$  the portion of  $M_{\mathcal{O}}$  consisting of centers of maximal homothets of  $\mathcal{O}$  that contain  $q$  (and are contained in  $P$ ). Lemma 3 below is analogous to Lemma 1 asserting that for each  $q \in Q$ ,  $M_q$  is a subtree of  $M_{\mathcal{O}}$ ; its proof can be found in the full version of this paper.<sup>3</sup> The analysis below uses the well known fact that homothetic copies of  $Q$  (in general position) are *pseudo-disks*; see, e.g., [23].

**Lemma 3.**  $M_q$  is connected for each point  $q \in Q$ .

Consider the graph  $G_{\mathcal{O}}$  defined similarly to  $G$  in the previous section. Its vertices are the points of  $Q$ , and it contains an edge between two points  $p, q \in Q$  if there exists a homothet of  $\mathcal{O}$  containing both  $p$  and  $q$  and contained in  $P$ . As in the case of disks, one can show that there is such a homothet if and only if there is a homothet containing  $p$  and  $q$  whose center is on  $M_{\mathcal{O}}$  and whose boundary touches  $\partial P$  (at least twice).

Again, if we identify each point  $q$  of  $Q$  with its subtree  $M_q$ , then  $G_{\mathcal{O}}$  is the intersection graph of these subtrees and thus  $G_{\mathcal{O}}$  is a chordal graph. As in Section 2, since pairwise intersecting subtrees of a tree  $T$  have a point in common, covering  $Q$  by homothets of  $\mathcal{O}$  is equivalent to a clique cover of  $G_{\mathcal{O}}$ , so our problem now is to find a minimum clique cover of  $G_{\mathcal{O}}$ .

We use the same high-level algorithm as in Section 2. Below, we mainly refer to the steps of the algorithm that are affected by the use of  $\mathcal{O}$ -distance instead of Euclidean distance. Recall that so far the analysis did not require any further assumptions concerning the actual shape of  $\mathcal{O}$ . However, to facilitate an efficient implementation of the algorithm, we need to assume that this shape is sufficiently simple, so as to allow various operations on  $\mathcal{O}$  and on a constant number of other features (points and/or line segments) to be performed in constant time. Examples of such operations are computing the  $\mathcal{O}$ -distance between two points, or between a point and a line segment, finding a point at the same  $\mathcal{O}$ -distance from three features of  $P^*$ , etc. The simplest way to enforce these properties is to assume that  $\mathcal{O}$  has *constant description complexity* (see, e.g., [31]).

The combinatorial complexity of the medial axis  $M_{\mathcal{O}}$  is  $O(n)$ , we compute it in time  $O(n)$  using the algorithm of Chin et al. [10] (which also applies to convex distance functions). The planar map  $H$  (induced by  $M_{\mathcal{O}} \cup \partial P$ ) is partitioned, in time  $O(n)$ , into simply-shaped cells, by connecting each vertex of  $M_{\mathcal{O}} \setminus \partial P$  to its nearest point(s) (in the  $\mathcal{O}$ -distance) on  $\partial P$ .

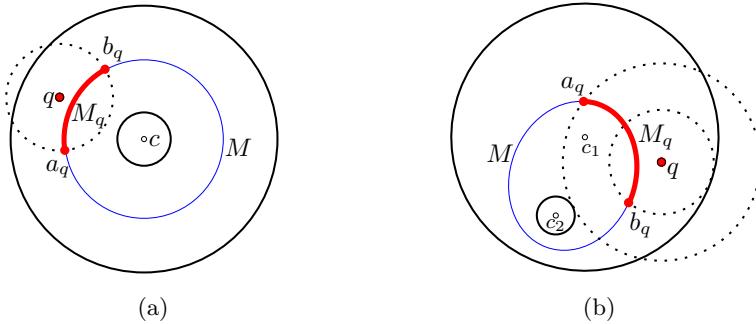
As in the previous section, we perform, in total time  $O(m \log n)$ , point location queries in  $H$  for the points in  $Q$ , as in [24]. Let  $T(q)$  be the trapezoid containing a point  $q \in Q$ , and let  $e$  be the feature of  $P^*$  on  $\partial T(q)$ . We compute the closest point  $q'$  to  $q$  on  $e$  under the  $\mathcal{O}$ -distance, and take  $A(q)$  to be the intersection of the line  $q'q$  with  $M_{\mathcal{O}}$  which is closest to  $q$  (and which also lies on  $\partial T(q)$ ). It is easy to see, arguing as above, that the maximal copy of  $\mathcal{O}$  centered at  $A(q)$  and contained in  $P$  touches  $\partial P$  at  $q'$  (and at another point) and contains  $q$ , and therefore  $M_q$  contains  $A(q)$ . Computing the roots  $r(M_q)$  and sorting them along the edges of  $M_{\mathcal{O}}$ , is done exactly as in the previous section, in  $O(n + m(\log n + \log m))$  time.

Finally, in the last step, we maintain, in a data structure  $\mathcal{S}$ , the union of all the homothets of  $\mathcal{O}$  that have so far been placed in the cover, and perform on it point location queries. Since homothets of  $\mathcal{O}$  are pseudo-disks, we can use essentially the same structure described in the previous section, with the same time and space bounds. We thus obtain (the remark following Theorem 2 applies here as well):

**Theorem 4.** *Let  $\mathcal{O}$  be a fixed compact convex set of constant description complexity, let  $P$  be a simple polygon with  $n$  edges, and let  $Q$  be a set of  $m$  points in  $P$ . We can compute a minimum cover of  $Q$  by homothets of  $\mathcal{O}$  contained in  $P$ , in  $O(n + m(\log n + \log^2 m))$  time, in an appropriate model of computation, using  $O(n + m)$  storage.*

## 4 Covering by Disks in a Sufficiently Narrow Annulus

Assume that the points of  $Q$  lie in an annulus  $R$  rather than in a simple polygon. In this case,  $M$ , the medial axis of  $R$ , is not a tree but a circle. Specifically, let  $c$



**Fig. 2.** (a) The arc  $M_q$  of a point  $q \in Q$ , obtained as the intersection of  $M$  with the (dotted) disk of radius  $\frac{r_2-r_1}{2}$  around  $q$ . (b)  $R$  is a disk centered at  $c_1$  with a hole centered at  $c_2$ ; the medial axis  $M$  is elliptic. The endpoints of  $M_q$  are at equal distance to the two circles bounding  $R$  and its hole and to  $q$ .

be the center of  $R$  and let  $r_1$  and  $r_2$  be inner and outer radii of  $R$ , respectively, then  $M$  is a circle of radius  $r_M = \frac{r_1+r_2}{2}$ , centered at  $c$ .

As in the previous sections, each point  $q \in Q$  is associated with an arc  $M_q$  of the circle  $M$ , which is the portion of  $M$  consisting of centers of maximal disks that cover  $q$  (and are contained in  $R$ ). See Fig. 2(a), which illustrates a proof of the property that  $M_q$  is the intersection of  $M$  with the disk of radius  $\frac{r_2-r_1}{2}$  centered at  $q$ . Again, we consider the intersection graph  $G$  of these arcs. However, unlike the previous cases,  $G$  is not chordal, since  $M$  is not a tree. Instead,  $G$  is a circular-arc graph (see, e.g., [13]). Moreover, we argue that if  $r_2 < \frac{2+\sqrt{3}}{2-\sqrt{3}}r_1 \approx 13.93r_1$  then  $G$  has the 2-Helly property, which means that the circular arcs in a clique of  $G$  have a point in common.

Indeed, since  $M_q$  is the intersection of  $M$  with the disk of radius  $\frac{r_2-r_1}{2}$  centered at  $q$ , it is maximal when  $a_q$ ,  $q$ , and  $b_q$  are collinear, where  $a_q$  and  $b_q$  are the endpoints of  $M_q$ . Let  $\theta = \angle a_q c b_q$ . If  $a_q$ ,  $q$ , and  $b_q$  are indeed collinear then  $\sin \frac{\theta}{2} = \frac{r_2-r_1}{r_2+r_1}$ . So if  $r_2 < \frac{2+\sqrt{3}}{2-\sqrt{3}}r_1$  then  $\sin \frac{\theta}{2} < \frac{\sqrt{3}}{2}$ , and therefore  $\theta < 2\pi/3$ . Consider a clique  $\mathcal{C}$  in  $G$ . Since all arcs are of length smaller than  $1/3$  the length of  $M$ , and they all intersect one specific arc of  $\mathcal{C}$ , then they cannot cover  $M$  completely. Consequently,  $\mathcal{C}$  can be viewed as a set of pairwise intersecting intervals on a line and it follows that all the arcs of  $\mathcal{C}$  must have a common intersection.

We conclude that, again, a cover of  $Q$  by disks contained in  $R$  corresponds to a clique cover of  $G$ , and vice versa. We thus compute a minimum clique cover of  $G$  by applying the  $O(m)$ -time algorithm of Hsu and Tsai [18], after sorting the arcs  $M_q$  by their endpoints in  $O(m \log m)$  time. In summary, we obtain

**Theorem 5.** *Let  $R$  be an annulus such that  $r_2 < \frac{2+\sqrt{3}}{2-\sqrt{3}}r_1$ , where  $r_1, r_2$  are the inner and outer radii of  $R$ , respectively, and let  $Q$  a set of  $m$  points contained in  $R$ . We can compute a minimum cover of  $Q$  by disks contained in  $R$  in  $O(m \log m)$  time and  $O(m)$  space.*

Theorem 5 also applies to the slightly more general case, where  $R$  is a disk with a circular hole, not necessarily concentric; see Fig. 2(b). In this case, the medial axis is an ellipse with foci at the centers of  $R$  and of its hole. For each  $q \in Q$ ,  $M_q$  is still a connected arc of  $M$ . Specifically, the endpoints of  $M_q$  are the points at equal distance to the two circles bounding  $R$  and its hole and to  $q$ , and there can be at most two such points. (Otherwise, arguing as in Section 2, we would get an impossible planar embedding of  $K_{3,3}$ .)

The graph  $G$  is a circular-arc graph, and it possesses the 2-Helly property provided that the hole is not too small. (The exact condition is that there do not exist three points  $q_1, q_2, q_3 \in R$  whose arcs  $M_{q_1}, M_{q_2}, M_{q_3}$  cover  $M$ .) Hence, if this condition holds then a minimum clique cover can be found as above, with the same asymptotic bounds on the running time and storage.

Finally, we do not know how critical is the assumption that  $R$  is not too wide, as in Theorem 5. Does the problem become hard if  $R$  is wider?

**Acknowledgments.** The authors wish to thank Lior Kapelushnik and an anonymous referee for independently suggesting that we use the data structure for union of disks instead of a less efficient range reporting data structure that was used in a previous version of this paper.

## References

1. Agarwal, P.K., Efrat, A., Sharir, M.: Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.* 29, 912–953 (2000)
2. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite vc-dimension. *Discrete Comput. Geom.* 14(4), 469–479 (1995)
3. Bentley, J.L., Saxe, J.B.: Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms* 1(4), 301–358 (1980)
4. Buneman, P.: A characterization of rigid circuit graphs. *Discrete Math.* 9, 205–212 (1974)
5. Calinescu, G., Mandoiu, I.I., Wan, P.-J., Zelikovsky, A.: Selecting forwarding neighbors in wireless ad hoc networks. *MONET* 9(2), 101–111 (2004)
6. Carmi, P., Katz, M.J., Lev-Tov, N.: Covering points by unit disks of fixed location. In: Tokuyama, T. (ed.) *ISAAC 2007*. LNCS, vol. 4835, pp. 644–655. Springer, Heidelberg (2007)
7. Chan, T.M.: A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM* 57(3), Article 16 (2010)
8. Chew, L.P., Drysdale III, R.L.: Voronoi diagrams based on convex distance functions. In: *SCG 1985: Proc. First Annual Sympos. Comput. Geom.*, pp. 235–244 (1985)
9. Chiang, Y., Tamassia, R.: Dynamic algorithms in computational geometry. *Proc. IEEE* 80(9), 1412–1434 (1992)
10. Chin, F.Y.L., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.* 21(3), 405–420 (1999)
11. Gavril, F.: Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.* 1(2), 180–187 (1972)

12. Gavril, F.: The intersection graphs of subtrees of a tree are exactly the chordal graphs. *J. Combinat. Theory Ser. B* 16, 47–56 (1974)
13. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
14. Grötschel, M., Lovász, L., Schrijver, A.: Polynomial algorithms for perfect graphs. In: Berge, C., Chvátal, V. (eds.) Topics on Perfect Graphs. Ann. Discrete Math., vol. 21, pp. 325–356. North-Holland, Amsterdam (1984)
15. Halperin, D., Linhart, C.: The minimum enclosing disk with obstacles (1999) (Manuscript)
16. Halperin, D., Sharir, M., Goldberg, K.Y.: The 2-center problem with obstacles. *J. Algorithms* 42(1), 109–134 (2002)
17. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM* 32(1), 130–136 (1985)
18. Hsu, W.L., Tsai, K.H.: Linear time algorithms on circular-arc graphs. *Inform. Process. Lett.* 40(3), 123–129 (1991)
19. Hurtado, F., Sacristán, V., Toussaint, G.: Some constrained minimax and maximin location problems. *Studies in Locational Analysis* 15, 17–35 (2000)
20. Katz, M.J., Morgenstern, G.: A scheme for computing minimum covers within simple regions. In: Dehne, F., et al. (eds.) 11th Int. Sympos. Algorithms and Data Structures (WADS). LNCS, vol. 5664, pp. 447–458. Springer, Heidelberg (2009)
21. Katz, M.J., Morgenstern, G.: Guarding orthogonal art galleries with sliding cameras. In: 25th European Workshop on Comput. Geom, pp. 159–162 (2009)
22. Katz, M.J., Roisman, G.S.: On guarding the vertices of rectilinear domains. *Comput. Geom.* 39(3), 219–228 (2008)
23. Kedem, K., Livne, R., Pach, J., Sharir, M.: On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.* 1(1), 59–71 (1986)
24. Kirkpatrick, D.: Optimal search in planar subdivisions. *SIAM J. Comput.* 12(1), 28–34 (1983)
25. Leven, D., Sharir, M.: Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams. *Discrete Comput. Geom.* 2, 9–31 (1987)
26. Mehlhorn, K., Näher, S.: Dynamic fractional cascading. *Algorithmica* 5(2), 215–241 (1990)
27. Motwani, R., Raghunathan, A., Saran, H.: Perfect graphs and orthogonally convex covers. *SIAM J. Discrete Math.* 2(3), 371–392 (1989)
28. Motwani, R., Raghunathan, A., Saran, H.: Covering orthogonal polygons with star polygons: The perfect graph approach. *J. Comput. Syst. Sci.* 40(1), 19–48 (1990)
29. Mustafa, N.H., Ray, S.: PTAS for geometric hitting set problems via local search. In: SCG 2009: Proc. 25th Annual Sympos. Comput. Geom, pp. 17–22 (2009)
30. Narayanappa, S., Vojtečhovský, P.: An improved approximation factor for the unit disk covering problem. In: 18th Canadian Conf. on Comput. Geom, pp. 15–18 (2006)
31. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and their Geometric Applications. Cambridge University Press, New York (1995)
32. Worman, C., Keil, J.M.: Polygon decomposition and the orthogonal art gallery problem. *Int. J. Comput. Geometry Appl.* 17(2), 105–138 (2007)

# Stability of $\varepsilon$ -Kernels

Pankaj K. Agarwal, Jeff M. Phillips, and Hai Yu

Duke University, University of Utah, and Google

**Abstract.** Given a set  $P$  of  $n$  points in  $\mathbb{R}^d$ , an  $\varepsilon$ -kernel  $K \subseteq P$  approximates the directional width of  $P$  in every direction within a relative  $(1-\varepsilon)$  factor. In this paper we study the stability of  $\varepsilon$ -kernels under dynamic insertion and deletion of points to  $P$  and by changing the approximation factor  $\varepsilon$ . In the first case, we say an algorithm for dynamically maintaining a  $\varepsilon$ -kernel is stable if at most  $O(1)$  points change in  $K$  as one point is inserted or deleted from  $P$ . We describe an algorithm to maintain an  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$  in  $O(1/\varepsilon^{(d-1)/2} + \log n)$  time per update. Not only does our algorithm maintain a stable  $\varepsilon$ -kernel, its update time is faster than any known algorithm that maintains an  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$ . Next, we show that if there is an  $\varepsilon$ -kernel of  $P$  of size  $\kappa$ , which may be dramatically less than  $O(1/\varepsilon^{(d-1)/2})$ , then there is an  $(\varepsilon/2)$ -kernel of  $P$  of size  $O(\min\{1/\varepsilon^{(d-1)/2}, \kappa^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon)\})$ . Moreover, there exists a point set  $P$  in  $\mathbb{R}^d$  and a parameter  $\varepsilon > 0$  such that if every  $\varepsilon$ -kernel of  $P$  has size at least  $\kappa$ , then any  $(\varepsilon/2)$ -kernel of  $P$  has size  $\Omega(\kappa^{\lfloor d/2 \rfloor})$   $\blacksquare$

## 1 Introduction

With recent advances in sensing technology, massive geospatial data sets are being acquired at an unprecedented rate in many application areas, including GIS, sensor networks, robotics, and spatial databases. Realizing the full potential of these data sets requires developing scalable algorithms for analyzing and querying them. Among many interesting algorithmic developments to meet this challenge, there is an extensive amount of work on computing a “small summary” of large data sets that preserves certain desired properties of the input data and on obtaining a good trade-off between the quality of the summary and its size. A coresnet is one example of such approximate summaries. Specifically, for an input set  $P$  and a function  $f$ , a coresnet  $C \subseteq P$  is a subset of  $P$  (with respect to  $f$ ) with the property that  $f(C)$  approximates  $f(P)$ . If a small-size coresnet  $C$  can be computed quickly (much faster than computing  $f(P)$ ), then one can compute an approximate value of  $f(P)$  by first computing  $C$  and then computing  $f(C)$ . This coresnet-based approach has been successfully used in a wide range of geometric optimization problems over the last decade; see [2].

<sup>1</sup> Research supported by subaward CIF-32 from NSF grant 0937060 to CRA, by NSF under grants CNS-05-40347, CFF-06-35000, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and W911NF-07-1-0376, by an NIH grant 1P50-GM-08183-01, by a DOE grant OEG-P200A070505, and by a grant from the U.S.–Israel Binational Science Foundation.

**$\varepsilon$ -kernels.** Agarwal *et al.* [1] introduced the notion of  $\varepsilon$ -kernels and proved that it is a coresset for many functions. For any direction  $u \in \mathbb{S}^{d-1}$ , let  $P[u] = \arg \max_{p \in P} \langle p, u \rangle$  be the extreme point in  $P$  along  $u$ ;  $\omega(P, u) = \langle P[u] - P[-u], u \rangle$  is called the *directional width* of  $P$  in direction  $u$ . For a given  $\varepsilon > 0$ ,  $K \subset P \subset \mathbb{R}^d$  is called an  $\varepsilon$ -kernel of  $P$  if

$$\langle P[u] - K[u], u \rangle \leq \varepsilon \omega(P, u)$$

for all directions  $u \in \mathbb{S}^{d-1}$ .<sup>2</sup> For simplicity, we assume  $\varepsilon \in (0, 1)$ , because for  $\varepsilon \geq 1$ , one can choose a constant number of points to form an  $\varepsilon$ -kernel. By definition, if  $X$  is an  $\varepsilon$ -kernel of  $P$  and  $K$  is a  $\delta$ -kernel of  $X$ , then  $K$  is a  $(\delta + \varepsilon)$ -kernel of  $P$ .

Agarwal *et al.* [1] showed that there exists an  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$  and it can be computed in time  $O(n+1/\varepsilon^{3d/2})$ , when  $d$  is fixed (assumed throughout the paper). The running time was improved by Chan [6] to  $O(n+1/\varepsilon^{d-3/2})$  (see also [10]). In a number of applications, the input point set is being updated periodically, so algorithms have also been developed to maintain  $\varepsilon$ -kernels dynamically. Agarwal *et al.* [1] had described a data structure to maintain an  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$  in  $(\log(n)/\varepsilon)^{O(d)}$  time per update. The update time was recently improved by Chan [7] to  $O((1/\varepsilon^{(d-1)/2}) \log n + 1/\varepsilon^{d-3/2})$ . His approach can also maintain an  $\varepsilon$ -kernel of size  $O((1/\varepsilon^d) \log n)$  with update time  $O(\log n)$ . If only insertions are allowed (e.g. in a streaming model), the size of the data structure can be improved to  $O(1/\varepsilon^{(d-1)/2})$  [4, 11].

In this paper we study two problems related to the *stability* of  $\varepsilon$ -kernels: how  $\varepsilon$ -kernels change as we update the input set or vary the value of  $\varepsilon$ .

**Dynamic stability.** Since the aforementioned dynamic algorithms for maintaining an  $\varepsilon$ -kernel focus on minimizing the size of the kernel, changing a single point in the input set  $P$  may drastically change the resulting kernel. This is particularly undesirable when the resulting kernel is used to build a dynamic data structure for maintaining another information. For example, kinetic data structures (KDS) based on coresets have been proposed to maintain various extent measures of a set of moving points [2]. If an insertion or deletion of an object changes the entire summary, then one has to reconstruct the entire KDS instead of locally updating it. In fact, many other dynamic data structures for maintaining geometric summaries also suffer from this undesirable property [9].

We call an  $\varepsilon$ -kernel *s-stable* if the insertion or deletion of a point causes the  $\varepsilon$ -kernel to change by at most  $s$  points. For brevity, if  $s = O(1)$ , we call the  $\varepsilon$ -kernel to be *stable*. Chan's dynamic algorithm can be adapted to maintain a stable  $\varepsilon$ -kernel of size  $O((1/\varepsilon^{d-1}) \log n)$ ; see Lemma 1 below. An interesting question is whether there is an efficient algorithm for maintaining a stable  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$ , as points are being inserted or deleted. Maintaining a stable  $\varepsilon$ -kernel dynamically is difficult for two main reasons. First, for an input set  $P$ ,

<sup>2</sup> This is a slightly stronger version of the definition than defined in [1] and an  $\varepsilon$ -kernel  $K$  gives a relative  $(1 + 2\varepsilon)$ -approximation of  $\omega(P, u)$  for all  $u \in \mathbb{S}^{d-1}$  (i.e.  $\omega(K, u) \leq \omega(P, u) \leq (1 + 2\varepsilon)\omega(K, u)$ ).

many algorithms compute  $\varepsilon$ -kernels in two or more steps. They first construct a large  $\varepsilon$ -kernel  $K'$  (e.g. see [17]), and then use a more expensive algorithm to create a small  $\varepsilon$ -kernel of  $K'$ . However, if the first algorithm is unstable, then  $K'$  may change completely each time  $P$  is updated. Second, all of the known  $\varepsilon$ -kernel algorithms rely on first finding a “rough shape” of the input set  $P$  (e.g., finding a small box that contains  $P$ ), estimating its fatness [5]. This rough approximation is used crucially in the computation of the  $\varepsilon$ -kernel. However, this shape is itself very unstable under insertions or deletions to  $P$ . Overcoming these difficulties, we prove the following in Section 2.

**Theorem 1.** *Given a parameter  $0 \leq \varepsilon \leq 1$ , a stable  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$  of a set of  $n$  points in  $\mathbb{R}^d$  can be maintained under insertions and deletions in  $O(1/\varepsilon^{(d-1)/2} + \log n)$  amortized time.*

Note that the update time of maintaining an  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$  is better than that in [7].

**Approximation stability.** If the size of an  $\varepsilon$ -kernel  $K$  is  $O(1/\varepsilon^{(d-1)/2})$ , then decreasing  $\varepsilon$  changes  $K$  quite predictably. However, this is the worst-case bound, and it is possible that the size of  $K$  may be quite small, e.g.,  $O(1)$ , or in general much smaller than the  $1/\varepsilon^{(d-1)/2}$  maximum (efficient algorithms are known for computing  $\varepsilon$ -kernels of near-optimal size [2]). Then how much can the size increase as we reduce the allowable error from  $\varepsilon$  to  $\varepsilon/2$ ? For any  $\varepsilon > 0$ , let  $\kappa(P, \varepsilon)$  denote the minimum size of an  $\varepsilon$ -kernel of  $P$ . Unlike many shape simplification problems, in which the size of simplification can change drastically as we reduce the value of  $\varepsilon$ , we show (Section 3) that this does not happen for  $\varepsilon$ -kernels and that  $\kappa(P, \varepsilon/2)$  can be expressed in terms of  $\kappa(P, \varepsilon)$ .

**Theorem 2.** *For any point set  $P$  and for any  $\varepsilon > 0$ ,*

$$\kappa(P, \varepsilon/2) = O(\min\{\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon), 1/\varepsilon^{(d-1)/2}\}).$$

Moreover, there exist a point set  $P$  and some  $\varepsilon > 0$  such that  $\kappa(P, \varepsilon/2) = \Omega(\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor})$ .

## 2 Dynamic Stability

In this section we describe an algorithm that proves Theorem 1. The algorithm is composed of a sequence of modules, each with certain property. We first state that Chan’s dynamic coresset algorithm [7] can be made stable (see proof in full version [3]):

**Lemma 1.** *For any  $0 < \varepsilon < 1$ , an  $\varepsilon$ -kernel  $K$  of  $P$  of size  $O((1/\varepsilon^{d-1}) \log n)$  can be maintained in  $O(\log n)$  time with  $O(1)$  changes to  $K$  per update.*

We first define the notion of anchor points and fatness of a point set and describe two algorithms for maintaining stable  $\varepsilon$ -kernels with respect to a fixed anchor: one of them maintains a kernel of size  $O(1/\varepsilon^{d-1})$  and the other of size  $O(1/\varepsilon^{(d-1)/2})$ ; the former has smaller update time. Then we describe the algorithm for updating anchor points and maintaining a stable kernel as the anchors

change. Finally, we put these modules together to obtain the final algorithm. We make the following simple observation, which will be crucial for combining different modules.

**Lemma 2 (Composition Lemma).** *If  $K$  is an  $s$ -stable  $\varepsilon$ -kernel of  $P$  and  $K'$  is an  $s'$ -stable  $\varepsilon'$ -kernel of  $K$ , then  $K'$  is an  $(s \cdot s')$ -stable  $(\varepsilon + \varepsilon')$ -kernel of  $P$ .*

**Anchors and fatness of a point set.** We call a point set  $P$   $\beta$ -fat if

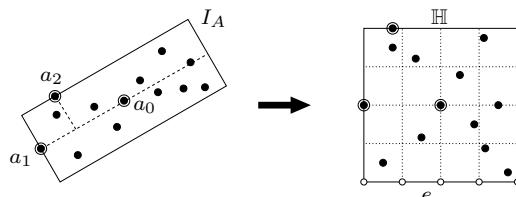
$$\max_{u,v \in \mathbb{S}^{d-1}} \omega(P, u) / \omega(P, v) \leq \beta.$$

If  $\beta$  is a constant, we sometimes just say that  $P$  is *fat*. An arbitrary point set  $P$  can be made fat by applying an affine transform: we first choose a set of  $d+1$  anchor points  $A = \{a_0, a_1, \dots, a_d\}$  using the following procedure of Barequet and Har-Peled [5]. Choose  $a_0$  arbitrarily. Let  $a_1$  be the farthest point from  $a_0$ . Then inductively, let  $a_i$  be the farthest point from the flat span( $a_0, \dots, a_{i-1}$ ). (See Figure 1) The anchor points  $A$  define a bounding box  $I_A$  with center at  $a_0$  and orthogonal directions defined by vectors from the flat span( $a_0, \dots, a_{i-1}$ ) to  $a_i$ . The extents of  $I_A$  in each orthogonal direction is defined by placing each  $a_i$  on a bounding face and extending  $I_A$  the same distance from  $a_0$  in the opposite direction. Next we perform an affine transform  $T_A$  on  $P$  such that the vector from the flat span( $a_0, \dots, a_{i-1}$ ) to  $a_i$  is equal to  $e_i$ , where  $e_0 = (0, \dots, 0)$ ,  $e_1 = (1, 0, \dots, 0), \dots, e_d = (0, \dots, 0, 1)$ . This ensures that  $T_A(P) \subseteq T_A(I_A) = [-1, 1]^d$ . The next lemma shows that  $T_A(P)$  is fat, and follows easily from [8].

**Lemma 3.** *For all  $u \in \mathbb{S}^{d-1}$  and for  $\beta_d \leq 2^d d^{5/2} d!$ ,*

$$\omega(T_A(A), u) \leq \omega(T_A(P), u) \leq \omega(T_A(I_A), u) \leq \beta_d \cdot \omega(T_A(A), u). \quad (1)$$

Agarwal *et al.* [4] show if  $K$  is an  $\varepsilon$ -kernel of  $P$ , then  $T(K)$  is an  $\varepsilon$ -kernel of  $T(P)$  for any affine transform  $T$ , which implies that one can compute an  $\varepsilon$ -kernel of  $T(P)$ . We will need the following generalization of the definition of  $\varepsilon$ -kernel. For two point sets  $P$  and  $Q$ , a subset  $K \subseteq P$  is called an  $\varepsilon$ -kernel of  $P$  with respect to  $Q$  if  $\langle P[u] - K[u], u \rangle \leq \varepsilon \omega(Q, u)$  for all  $u \in \mathbb{S}^{d-1}$ .



**Fig. 1.** Anchor points  $A = \{a_0, a_1, a_2\}$ , rectangle  $I_A$ , and transform  $T_A$  applied to  $P$ ; square  $\mathbb{H}$ , two-dimensional grid  $\mathbb{G}$ , and one-dimensional grid  $\mathbb{G}_e$  on the edge  $e$  of  $\mathbb{H}$

**Stable  $\varepsilon$ -kernels for a fixed anchor.** Let  $A$  be a set of anchor points of  $P$ , as described above. We describe algorithms for maintaining stable  $\varepsilon$ -kernels (with respect to  $A$ ) under the assumption that  $A$  remains a set of anchor points of  $P$ , i.e.,  $A \subseteq P \subset I_A$ , as  $P$  is being updated by inserting and deleting points. In view of the above discussion, without loss of generality, we assume  $I_A = [-1, +1]^d$  and denote it by  $\mathbb{H}$ . As for the static case [16], we first describe a simpler algorithm that maintains a stable  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{d-1})$ , and then a more involved one that maintains a stable  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$ .

Set  $\delta = \varepsilon/\sqrt{d}$  and draw a  $d$ -dimensional grid  $\mathbb{G}$  inside  $\mathbb{H}$  of size  $\delta$ , i.e., the side-length of each grid cell is at most  $\delta$ ;  $\mathbb{G}$  has  $O(1/\delta^d)$  cells. For each grid cell  $\tau$ , let  $P_\tau = P \cap \tau$ . For a point  $x \in \mathbb{H}$  lying in a grid cell  $\tau$ , let  $\hat{x}$  be the vertex of  $\tau$  nearest to the origin; we can view  $x$  being *snapped* to the vertex  $\hat{x}$ . For each facet  $f$  of  $\mathbb{H}$ ,  $\mathbb{G}$  induces a  $(d-1)$ -dimensional grid  $\mathbb{G}_f$  on  $f$ ;  $\mathbb{G}$  contains a *column* of cells for each cell in  $\mathbb{G}_f$ . For each cell  $\Delta \in \mathbb{G}_f$ , we choose (at most) one point of  $P$  as follows: let  $\tau$  be the nonempty grid cell in the column of  $\mathbb{G}$  corresponding to  $\Delta$  that is closest to  $f$ . We choose an arbitrary point from  $P_\tau$ ; if there is no nonempty cell in the column, no point is chosen. Let  $L_f$  be the set of chosen points. Set  $\mathcal{L} = \bigcup_{f \in \mathbb{H}} L_f$ . Agarwal *et al.* [1] proved that  $\mathcal{L}$  is an  $\varepsilon$ -kernel of  $P$ . Insertion or deletion of a point in  $P$  affects at most one point in  $L_f$ , and it can be updated in  $O(\log(1/\varepsilon))$  time. Hence, we obtain the following:

**Lemma 4.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$ , let  $A \subseteq P$  be a set of anchor points of  $P$ , and let  $0 < \varepsilon < 1$  be a parameter.  $P$  can be preprocessed in  $O(n + 1/\varepsilon^{d-1})$  time, so that a  $(2d)$ -stable  $\varepsilon$ -kernel of  $P$  with respect to  $A$  of size  $O(1/\varepsilon^{d-1})$  can be maintained in  $O(\log 1/\varepsilon)$  time per update provided that  $A$  remains an anchor set of  $P$ .*

Agarwal *et al.* [1] and Chan [6] have described algorithms for computing an  $\varepsilon$ -kernel of size  $O(1/\varepsilon^{(d-1)/2})$ . We adapt Chan's algorithm to maintain a stable  $\varepsilon$ -kernel with respect to a fixed anchor  $A$ . We begin by mentioning a result of Chan that lies at the heart of his algorithm.

**Lemma 5 (Chan [6]).** *Let  $E \in \mathbb{N}$ ,  $E^\tau \leq F \leq E$  for some  $0 < \tau < 1$ , and  $P \subseteq [0 : E]^{d-1} \times \mathbb{R}$  a set of at most  $n$  points. For all grid points  $b \in [0 : F]^{d-1} \times \mathbb{R}$ , the nearest neighbors of each  $b$  in  $P$  can be computed in time  $O(n + E^{d-2}F)$ .*

We now set  $\gamma = \sqrt{\varepsilon}/c$  for a constant  $c > 1$  to be used in a much sparser grid than with  $\delta$ . Let  $\mathbb{C} = [-2, +2]^d$  and  $f$  be a facet of  $\mathbb{C}$ . We draw a  $(d-1)$ -dimensional grid on  $f$  of size  $\gamma$ . Assuming  $f$  lies on the plane  $x_d = -2$ , we choose a set  $B_f = \{(i_1\gamma, \dots, i_{d-1}\gamma, -2) \in \mathbb{Z}^d \mid -\lceil 2/\gamma \rceil \leq i_1, \dots, i_{d-1} \leq \lceil 2/\gamma \rceil\}$  of grid points. For a subset  $X \subseteq P$  and a point  $b$ , we define  $\psi(X, b) = \arg \min_{x \in X} \|\hat{x} - b\|$ , i.e., the point in  $X$  such that the snapped point is nearest to  $b$ . For a set  $R$ ,  $\psi(X, R) = \{\psi(X, r) \mid r \in R\}$ . There is a one to one mapping between the faces of  $\mathbb{C}$  and  $\mathbb{H}$ , so we also use  $f$  to denote the corresponding facet of  $\mathbb{H}$ . Let  $L_f$  be the set of points chosen in the previous algorithm corresponding to facet  $f$  of  $\mathbb{H}$  for computing an  $(\varepsilon/2)$ -kernel of  $P$ . Set  $G_f = \psi(L_f, B_f)$ . Chan showed that  $\mathcal{G} = \bigcup_{f \in \mathbb{C}} G_f$  is an  $(\varepsilon/2)$ -kernel of  $\mathcal{L}$  and thus an  $\varepsilon$ -kernel of  $P$ . Scaling  $\mathbb{G}$  and

$B_f$  appropriately and using Lemma 5,  $G_f$  can be computed in  $O(n + 1/\varepsilon^{d-3/2})$  time. Hence,  $\mathcal{G}$  can be computed in  $O(n + 1/\varepsilon^{d-3/2})$  time.

Note that  $\psi(L_f, b)$  can be the same for many points  $b \in B_f$ , so insertion or deletion of a point in  $P$  (and thus in  $L_f$ ) may change  $G_f$  significantly, thereby making  $\mathcal{G}$  unstable. We circumvent this problem by introducing two new ideas. First,  $\psi(L_f, B_f)$  is computed in two stages, and second it is computed in an iterative manner. We describe the construction and the update algorithm for  $f$ ; the same algorithm is repeated for all facets.

We partition  $\mathbb{H}$  into  $O(1/\gamma^{d-1})$  boxes: for  $J = \langle i_1, \dots, i_{d-1} \rangle \in [-1/\gamma, 1/\gamma]^{d-1} \cap \mathbb{Z}^{d-1}$ , we define  $\mathbb{H}_J = [i_1\gamma, (i_1+1)\gamma] \times \dots \times [i_{d-1}\gamma, (i_{d-1}+1)\gamma] \times [-1, +1]$ . We maintain a subset  $X \subseteq L_f$ . Initially, we set  $X = L_f$ . Set  $X_J = X \cap \mathbb{H}_J$ . We define a total order on the points of  $B_f$ . Initially, we sort  $B_f$  in lexicographic order, but the ordering will change as insertions and deletions are performed on  $P$ . Let  $\langle b_1, \dots, b_u \rangle$  be the current ordering of  $B_f$ . We define a map  $\varphi : B_f \rightarrow L_f$  as follows. Suppose  $\varphi(b_1), \dots, \varphi(b_{i-1})$  have been defined. Let  $J_i = \arg \min_J \|\hat{\psi}(X_J, b_i) - b_i\|$ ; here  $\hat{\psi}(\cdot, \cdot)$  denotes the snapped point of  $\psi(\cdot, \cdot)$ . We set  $\varphi(b_i) = \psi(X_{J_i}, b_i)$ . We delete  $\varphi(b_i)$  from  $X$  (and from  $X_{J_i}$ ) and recompute  $\hat{\psi}(X_{J_i}, B_f)$ . Set  $K_f = \{\varphi(b) \mid b \in B_f\}$  and  $K = \bigcup_f K_f$ . Computing  $J_i$  and  $\varphi(b_i)$  takes  $O(1/\varepsilon^{(d-1)/2})$  time, and, by Lemma 5,  $\psi(X_{J_i}, B_f)$  can be computed in  $O(|X_J| + 1/\gamma^{d-2} \cdot 1/\gamma) = O(1/\varepsilon^{(d-1)/2})$  time.

It can be proved that the map  $\varphi$  and the set  $K_f$  satisfy the following properties:

- (P1)  $\varphi(b_i) \neq \varphi(b_j)$  for  $i \neq j$ ,
- (P2)  $\varphi(b_i) = \psi(L_f \setminus \{\varphi(b_j) \mid j < i\}, b_i)$ ,
- (P3)  $K_f \supseteq \psi(L_f, B_f)$ .

Indeed, (P1) and (P2) follow from the construction, and (P3) follows from (P2). (P3) immediately implies that  $K$  is an  $\varepsilon$ -kernel of  $P$ . Next, we describe the procedures for updating  $K_f$  when  $L_f$  changes. These procedures maintain (P1)–(P3), thereby ensuring that the algorithm maintains an  $\varepsilon$ -kernel.

*Inserting a point.* Suppose a point  $p$  is inserted into  $L_f$ . We add  $p$  to  $X$ . Suppose  $p \in \mathbb{H}_J$ . We recompute  $\psi(X_J, B_f)$ . Next, we update  $\varphi(\cdot)$  and  $K$  as follows. We maintain a point  $\xi \in L_f$ . Initially,  $\xi$  is set to  $p$ . Suppose we have processed  $b_1, \dots, b_{i-1}$ . Let  $\eta \in L_f$  be the current  $\varphi(b_i)$ . If  $\|\hat{\xi} - b_i\| \leq \|\hat{\eta} - b_i\|$ , then we swap  $\xi$  and  $\varphi(b_i)$ , otherwise neither  $\xi$  nor  $\varphi(b_i)$  is updated. We then process  $b_{i+1}$ . After processing all points of  $B_f$  if  $\xi = p$ , i.e., no  $\varphi(b_i)$  is updated, we stop. Otherwise, we add  $p$  to  $K_f$  and delete  $\xi$  from  $K_f$ . The insertion procedure makes at most two changes in  $K_f$ , and it can be verified that (P1)–(P3) are maintained.

*Deleting a point.* Suppose  $p$  is deleted from  $L_f$ . Suppose  $p \in \mathbb{H}_J$ . If  $p \notin K_f$ , then  $p \in X$ . We delete  $p$  from  $X$  and  $X_J$  and recompute  $\psi(X_J, B_f)$ . If  $p \in K_f$ , i.e., there is a  $b_i \in B_f$  with  $p = \varphi(b_i)$ , then  $p \notin X$ . We delete  $p$  from  $K_f$  and  $K$ , recompute  $\varphi(b_i)$ , and add the new  $\varphi(b_i)$  to  $K_f$ . Let  $\varphi(b_i) \in \mathbb{H}_J$ ; we remove  $\varphi(b_i)$  from  $X_J$  and recompute  $\psi(X_J, B_f)$ . We modify the ordering of  $B_f$  by moving  $b_i$  from its current position to the end. This is the only place where the ordering

of  $B_f$  is modified. Since  $b_i$  is now the last point in the ordering of  $B_f$ , the new  $\varphi(b_i)$  does not affect any other  $\varphi(b_j)$ . The deletion procedure also makes at most two changes in  $K_f$  and maintains (P1)–(P3).

Finally, insertion or deletion of a point in  $P$  causes at most one insertion plus one deletion in  $L_f$ , therefore we can conclude the following:

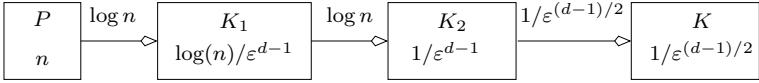
**Lemma 6.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$ ,  $A$  a set of anchor points of  $P$ , and  $0 < \varepsilon < 1$  a parameter.  $P$  can be preprocessed in  $O(n + 1/\varepsilon^{d-1})$  time into a data structure so that a stable  $\varepsilon$ -kernel of  $P$  with respect to  $A$  of size  $O(1/\varepsilon^{(d-1)/2})$  can be maintained in  $O(1/\varepsilon^{(d-1)/2})$  time under insertion and deletion, provided that  $A$  remains an anchor set of  $P$ .*

**Updating anchors.** We now describe the algorithm for maintaining a stable  $\varepsilon$ -kernel when anchors of  $P$  are no longer fixed and need to be updated dynamically. Roughly speaking, we divide  $P$  into *inner* and *outer* subsets of points. The outer subset acts as a *shield* so that a stable kernel of the inner subset with respect to a fixed anchor can be maintained using Lemma 4 or 6. When the outer subset can no longer act as a shield, we reconstruct the inner and outer sets and start the algorithm again. We refer to the duration between two consecutive reconstruction steps as an *epoch*. The algorithm maintains a stable kernel within each epoch, and the amortized number of changes in the kernel because of reconstruction at the beginning of a new epoch will be  $O(1)$ . We can use a de-amortization technique to make the  $\varepsilon$ -kernel stable across epochs. We now describe the algorithm in detail.

In the beginning of each epoch, we perform the following preprocessing. Set  $\alpha = 1/10$  and compute a  $\alpha$ -kernel  $\mathcal{L}$  of  $P$  of size  $O(\log n)$  using Chan's dynamic algorithm; we do not need the stable version of his algorithm advertised above.  $\mathcal{L}$  can be updated in  $O(\log n)$  time per insertion/deletion. We choose a parameter  $m$ , which is set to  $1/\varepsilon^{d-1}$  or  $1/\varepsilon^{(d-1)/2}$ . We create the outer subset of  $P$  by peeling off  $m$  “layers” of anchor points  $A_1, \dots, A_m$ . Initially, we set  $P_0 = P$ . Suppose we have constructed  $A_0, \dots, A_{i-1}$ . Set  $P_{i-1} = P \setminus \bigcup_{j=1}^{i-1} A_j$ , and  $\mathcal{L}$  is an  $\alpha$ -kernel of  $P_{i-1}$ . Next, we construct the anchor set  $A_i$  of  $\mathcal{L}$  as described earlier in this section. We set  $P_i = P_{i-1} \setminus A_i$  and update  $\mathcal{L}$  so that it is an  $\alpha$ -kernel of  $P_i$ . Let  $\mathcal{A} = \bigcup_i A_i$ ,  $A = A_m$ , and  $P_I = P \setminus \mathcal{A}$ . Let  $\mathbb{H} = (1+\alpha)I_A$ . By construction  $P_I \subset \mathbb{H}$ .  $\mathcal{A}$  forms the outer subset and acts as a shield for  $P_I$ , which is the inner subset. Set  $\delta = \varepsilon/(2(1+\alpha)(\beta_d)^2)$ , where  $\beta_d$  is the constant in Lemma 3.  $\square$

If  $m = 1/\varepsilon^{d-1}$  (resp.  $1/\varepsilon^{(d-1)/2}$ ), we maintain a stable  $\delta$ -kernel  $K_I$  of  $P_I$  with respect to  $A$  of size  $O(m)$  using Lemma 4 (resp. Lemma 6). Set  $K = K_I \cup \mathcal{A}$ ;  $|K| = O(m)$ . We prove below that  $K$  is an  $\varepsilon$ -kernel of  $P$ . Let  $p$  be a point that is inserted into or deleted from  $P$ . If  $p \in \mathbb{H}$ , then we update  $K_I$  using Lemma 4 or 6. On the other hand, if  $p$  lies outside  $\mathbb{H}$ , we insert it into or delete it from  $\mathcal{A}$ . Once  $\mathcal{A}$  has been updated  $m$  times, we end the current epoch and discard the current  $K$ . We begin a new epoch and reconstruct  $\mathcal{A}$ ,  $P_I$ , and  $K_I$  as described above.

The preprocessing step at the beginning of a new epoch causes  $O(m)$  changes in  $K$  and there are at least  $m$  updates in each epoch, therefore the algorithm maintains a stable kernel in the amortized sense. Again, using a de-amortization

**Fig. 2.** Composing stable  $\varepsilon$ -kernel algorithms

technique, we can ensure that  $K$  is stable. The correctness of the algorithm follows from the following lemma (proved in full version [3]).

**Lemma 7.**  *$K$  is always an  $\varepsilon$ -kernel of  $P$ .*

Using Lemmas 4 and 6, we can bound the amortized update time and conclude the following.

**Lemma 8.** *For a set  $P$  of  $n$  points in  $\mathbb{R}^d$  and a parameter  $0 < \varepsilon < 1$ , there is a data structure that can maintain a stable  $\varepsilon$ -kernel of  $P$  of size  $O(1/\varepsilon^{(d-1)/2})$  under insertions and deletions in amortized time  $O(n\varepsilon^{(d-1)/2} + 1/\varepsilon^{(d-1)/2} + \log n)$ , or of size  $O(1/\varepsilon^{d-1})$  in amortized time  $O(n\varepsilon^{d-1} + \log n + \log(1/\varepsilon))$ .*

**Putting it together.** For a point set  $P \subset \mathbb{R}^d$  of size  $n$ , we can produce the best size and update time tradeoff for stable  $\varepsilon$ -kernels by invoking Lemma 2 to compose three stable  $\varepsilon$ -kernel algorithms, as illustrated in Figure 2. We first apply Lemma 1 to maintain a stable  $(\varepsilon/3)$ -kernel  $K_1$  of  $P$  of size  $O(\min\{n, (1/\varepsilon^{d-1}) \cdot \log n\})$  with update time  $O(\log n)$ . We then apply Lemma 8 to maintain a stable  $(\varepsilon/3)$ -kernel  $K_2$  of  $K_1$  of size  $O(1/\varepsilon^{d-1})$  with update time  $O(|K_1|\varepsilon^{d-1} + \log |K_1| + \log(1/\varepsilon)) = O(\log n + \log(1/\varepsilon))$ . Finally we apply Lemma 8 again to maintain a stable  $(\varepsilon/3)$ -kernel  $K$  of  $K_2$  of size  $O(1/\varepsilon^{(d-1)/2})$  with update time  $O(|K_2|\varepsilon^{(d-1)/2} + 1/\varepsilon^{(d-1)/2} + \log |K_2|) = O(1/\varepsilon^{(d-1)/2})$ .  $K$  is a stable  $\varepsilon$ -kernel of  $P$  of size  $O(1/\varepsilon^{(d-1)/2})$  with update time  $O(\log n + 1/\varepsilon^{(d-1)/2})$ . This completes the proof of Theorem 1.

### 3 Approximation Stability

In this section we prove the upper bound in Theorem 2. Due to lack of space we only prove the upper bound for  $d = 2, 3$ ; the remainder is in the full version [3].

By 1, it suffices to consider the case in which  $P$  is fat and the diameter of  $P$  is normalized to 1. Let  $K$  be an  $\varepsilon$ -kernel of  $P$  of the smallest size. Let  $\mathcal{P} = \text{conv}(K)$ , and  $\mathcal{P}_\varepsilon = \mathcal{P} \oplus \varepsilon\mathbb{B}^d$ . We have  $\mathcal{P} \subseteq \text{conv}(P) \subseteq \mathcal{P}_\varepsilon$  by the definition of  $\varepsilon$ -kernels. It suffices to show that there is a set  $K' \subseteq P$  such that for  $\mathcal{P}' = \text{conv}(K')$ ,  $\mathcal{P}' \subseteq \text{conv}(P) \subseteq \mathcal{P}_{\varepsilon/2}$ , and  $|K'| = O(|K|^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon))$  1.

For convenience, we assume that  $K'$  is not necessarily a subset of points in  $P$ ; instead, we only require  $K'$  to be a subset of points in  $\text{conv}(P)$ . By Caratheodory's theorem, for each point  $x \in K$ , we can choose a set  $P_x \subseteq P$  of at most  $d+1$  points such that  $x \in \text{conv}(P_x)$ . We set  $\bigcup_{x \in K'} P_x$  as the desired  $(\varepsilon/2)$ -kernel of  $P$ ;  $|\bigcup_{x \in K'} P_x| \leq (d+1)|K'| = O(\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor} \log^{d-2}(1/\varepsilon))$ .

Initially, we add a point into  $K'$  for each point in  $K$ . If  $p \in K$  lies on  $\partial \text{conv}(P)$ , we add  $p$  to  $K'$ . Otherwise we project  $p$  onto  $\partial \text{conv}(P)$  in a direction in which  $p$  is maximal in  $K$  and add the projected point to  $K'$ . Abusing the notation

slightly, we use  $\mathcal{P}$  to denote hull of these initial points. For simplicity, we assume  $\mathcal{P}$  to be a simplicial polytope.

**Decomposition of  $\mathcal{P}_\varepsilon \setminus \text{intr } \mathcal{P}$ .** There are  $d$  types of simplices on  $\partial\mathcal{P}$ . In  $\mathbb{R}^2$  these are points and edges. In  $\mathbb{R}^3$  these are points, edges, and triangles. We can decompose  $\mathcal{P}_\varepsilon \setminus \text{intr } \mathcal{P}$  into a set of regions, each region  $\sigma(f)$  corresponding to a simplex  $f$  in  $\mathcal{P}$ . For each simplex  $f$  in  $\mathcal{P}$  let  $f^* \subseteq \mathbb{S}^{d-1}$  denote the dual of  $f$  in the Gaussian diagram of  $\mathcal{P}$ . Recall that if  $f$  has dimension  $k$  ( $0 \leq k \leq d-1$ ), then  $f^*$  has dimension  $d-1-k$ . The region  $\mathcal{P}_\varepsilon \setminus \text{intr } \mathcal{P}$  is partitioned into a collection of  $|\mathcal{P}|$  regions (where  $|\mathcal{P}|$  is the number of faces of all dimensions in  $\mathcal{P}$ ). Each simplex  $f$  in  $\mathcal{P}$  corresponds to a region defined

$$\sigma(f) = \{f + zu \mid 0 \leq z \leq \varepsilon, u \in f^*\}.$$

For a subsimplex  $\tau \in f$ , we can similarly define a region  $\sigma(\tau) = \{\tau + zu \mid 0 \leq z \leq \varepsilon, u \in f^*\}$ . In  $\mathbb{R}^2$ , there are two types of regions: point regions and edge regions. In  $\mathbb{R}^3$ , there are three types of regions: point regions (see Figure 4(a)), edge regions (see Figure 4(b)), and triangle regions (see Figure 4(c)).

For convenience, for any point  $q = \bar{q} + z \cdot u \in \sigma(f)$ , where  $\bar{q} \in f$ ,  $0 \leq z \leq \varepsilon$ , and  $u \in f^*$ , we write  $q = \bar{q}[u, z]$  (which intuitively reads, the point whose projection onto  $f$  is  $\bar{q}$  and which is at a distance  $z$  above  $f$  in direction  $u$ ). We also write  $q[v] = \bar{q} + z \cdot v$  (intuitively,  $q[v]$  is obtained by rotating  $q$  w.r.t.  $f$  from direction  $u$  to direction  $v$ ). Similarly, we write a simplex  $\bar{\Delta}[u, z] = \bar{\Delta} \oplus z \cdot u$ , where  $\bar{\Delta}$  is a simplex inside  $f$ ,  $0 \leq z \leq \varepsilon$ , and  $u \in f^*$ , and write  $\Delta[v] = \bar{\Delta} \oplus z \cdot v$ .

We will proceed to prove the upper bound as follows. For each type of region  $\sigma(f)$  we place a bounded number of points from  $\sigma(f) \cap \text{conv}(P)$  into  $K'$  and then prove that all points in  $\sigma(f) \cap \text{conv}(P)$  are within a distance  $\varepsilon/2$  from some point in  $\mathcal{P}' = \text{conv}(K')$ . We begin by introducing three ways of “gridding”  $\sigma(f)$  and then use these techniques to directly prove results for several base cases, which illustrate the main conceptual ideas. These base cases will already be enough to prove the results in  $\mathbb{R}^2$  and  $\mathbb{R}^3$ . In the full version [3] we generalize this to  $\mathbb{R}^d$  using an involved recursive construction. We set a few global values:  $\delta = \varepsilon/12d$ ,  $\theta = 2 \arcsin(\delta/2\varepsilon)$ , and  $\rho = \delta/\varepsilon$ .

**1: Creating layers.** For a point  $q = \bar{q}[u, z] \in \sigma(f)$  we classify it depending on the value  $z = |q - \bar{q}|$ . If  $z \leq \varepsilon/2$ , then  $q$  is already within  $\varepsilon/2$  of  $\mathcal{P}$ . We then divide the range  $[\varepsilon/2, \varepsilon]$  into a constant  $H = (\varepsilon/2)/\delta$  number of cases using  $\mathcal{H} = \{h_1 = \varepsilon/2, h_2 = \varepsilon/2 + \delta, \dots, h_H = \varepsilon - \delta\}$ . If  $z \in [h_i, h_{i+1})$ , then we set  $q_{h_i} = \bar{q}[u, h_i]$ . We define  $\Psi_{f,h_i} \subset \sigma(f) \cap \text{conv}(P)$  to be the set of points that are a distance exactly  $h_i$  from  $f$ .

**2: Discretize angles.** We create a constant size  $\theta$ -net  $U_{f,h} = \{u_1, u_2, \dots\} \subset f^*$  of directions with the following properties. (1) For each  $q = \bar{q}[u, h] \in \Psi_{f,h}$  there is a direction  $u_i \in U_{f,h}$  such that the angle between  $u$  and  $u_i$  is at most  $\theta$ . (2) For each  $u_i \in U_{f,h}$  there is a point  $p_i = \bar{p}_i[u_i, h] \in \Psi_{f,h}$ ; let  $N_{f,h} = \{p_i \mid i \geq 1\}$ .  $U_{f,h}$  is constructed by first taking a  $(\theta/2)$ -net  $U_f$  of  $f^*$ , then for each  $u'_i \in U_f$  choosing a point  $p_i = \bar{q}_i[u_i, h] \in \Psi_{f,h}$  where  $u_i$  is within an angle  $\theta/2$  of  $u'_i$  (if one exists), and finally placing  $u_i$  in  $U_{f,h}$ .

**3: Exponential grid.** Define a set  $\mathcal{D} = \{d_0, d_1 = (1 + \rho)d_0, \dots, d_m = (1 + \rho)^m d_0\}$  of distances where  $d_m < 1$  and  $d_0 = \delta$ , so  $m = O(\log 1/\varepsilon)$ . For a face  $f \in \mathcal{P}$ , let any  $r \in \sigma(f)$  be called a *support point* of  $f$ . Let  $p_1, \dots, p_k$  be the vertices of the  $k$ -simplex  $f$ . For each  $p_j$ , and each  $d_i \in \mathcal{D}$  (where  $d_i < \|p_j - \bar{r}\|$ ), let  $p_{j,i}$  be the point at distance  $d_i$  from  $p_j$  on the segment  $p_j \bar{r}$ . For each boundary facet  $F$  of  $f$ , define a sequence of at most  $m$  simplices  $F_0, F_1, \dots \in \text{conv}(F \cup \bar{r})$ , each a homothet of  $F$ , so the vertices of  $F_i$  lie on segments  $p_j \bar{r}$  where  $p_j \in \partial F$  (see Figure 5(a)). The translation of each  $F_i$  is defined so it intersects a point  $p_{j,i}$  (where  $p_j \in \partial F$ ) and is as close to  $F$  as possible. This set of  $(k-1)$ -simplices for each  $F$  defines the exponential grid  $G_{r,f}$ . The full grid structure is revealed as this is applied recursively on each  $F_i$ .

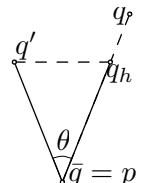
The exponential grid  $G_{r,\Delta}$  on a simplex  $\Delta$  has two important properties for a point  $q \in \Delta$ :

- (G1) If  $q \in \text{conv}(F \cap \bar{r})$  lies between boundary facet  $F$  and  $F_0$ , let  $q_0$  be the intersection of the line segment  $q\bar{r}$  with  $F_0$ ; then  $\|q - q_0\| \leq d_0 = \delta$ .
- (G2) If  $q \in \text{conv}(F \cap \bar{r})$  lies between  $F_{i-1}$  and  $F_i$  and the segment  $q\bar{r}$  intersects  $F_i$  at  $q_i$ , let  $q_F$  be the intersection of  $F$  with the ray  $\bar{r}q$ ; then  $\|q_i - q\| / \|q_i - q_F\| \leq \rho = \delta/\varepsilon$ .

We now describe how to handle certain simple types of regions: where  $f$  is a point or an edge. These will be handled the same regardless of the dimension of the problem, and they (the edge case in particular) will be used as important base cases for higher dimensional problems.

**Point regions.** Consider a point region  $\sigma(p)$ . For each  $h \in \mathcal{H}$  create  $\theta$ -net  $U_{p,h}$  for  $\Psi_{p,h}$ , so  $N_{p,h}$  are the corresponding points where each  $p_i = p[h, u_i] \in N_{p,h}$  has  $u_i \in U_{p,h}$ . Put each  $N_{p,h}$  in  $K'$ .

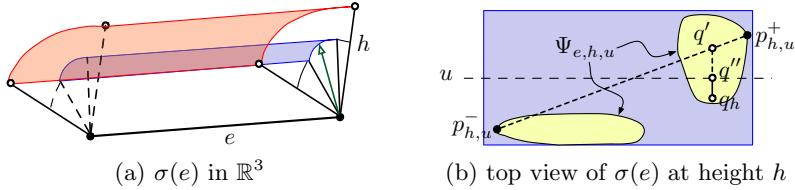
For any point  $q = \bar{q}[u', z] \in \sigma(p) \cap \text{conv}(P)$ , let  $q' = \bar{q}[u, h]$  where  $h \in \mathcal{H}$  is the largest value such that  $h \leq z$  and  $u \in U_{p,h}$  is the closest direction to  $u'$ ; set  $q_h = \bar{q}[u', h] = q'[u']$ . First  $\|q - q_h\| \leq \delta$  because  $z - h \leq \delta$ . Second  $\|q_h - q'\| \leq \delta$  because the angle between  $u'$  and  $u$  is at most  $\theta$ , and they are rotated about the point  $p$ . Thus  $\|q - q'\| \leq \|q - q_h\| + \|q_h - q'\| \leq 2\delta \leq \varepsilon/2$ .



**Lemma 9.** For a point region  $\sigma(p)$ , there exists a constant number of points  $K_p \subset \sigma(p) \cap \text{conv}(P)$  such that all points  $q \in \sigma(p) \cap \text{conv}(P)$  are within a distance  $\varepsilon/2$  of  $\text{conv}(K_p)$ .

**Edge regions.** Consider an edge region  $\sigma(e)$  for an edge  $e$  of  $\mathcal{P}$ . Orient  $e$  along the  $x$ -axis. For each  $h \in \mathcal{H}$  and  $u \in U_{e,h}$ , let  $\Psi_{e,h,u}$  be the set of points in  $\Psi_{e,h}$  within an angle  $\theta$  of  $u$ . For each  $\Psi_{e,h,u}$ , we add to  $K_e$  the (two) points of  $\Psi_{e,h,u}$  with the largest and smallest  $x$ -coordinates, denoted by  $p_{h,u}^+$  and  $p_{h,u}^-$ .

For any point  $q = \bar{q}[v, z] \in \sigma(e) \cap \text{conv}(P)$ , there is a point  $q'' = \bar{q}[u, h]$  such that  $h \in \mathcal{H}$  is the largest value less than  $z$  and  $u \in U_{e,h}$  is the closest direction to  $v$ . Furthermore,  $\|q - q''\| \leq \|q - q_h\| + \|q_h - q''\| \leq (z - h) + 2\varepsilon \sin(\theta/2) = \delta + \delta = 2\delta$ . We can also argue that there is a point  $q' = \bar{q}[u', z'] \in p_{h,u}^- p_{h,u}^+$ , because if  $\bar{q}$  has



**Fig. 3.** Illustration of 2 points in  $K'$  for edge case with specific  $h \in \mathcal{H}$  and  $u \in U_{e,h,\theta}$

smaller  $x$ -coordinate than  $p_{h,u}^-$  or larger  $x$ -coordinate than  $p_{h,u}^+$ , then  $q'$  cannot be in  $\Psi_{e,h,u}$ . Clearly the angle between  $u$  and  $u'$  is less than  $\theta$ . This also implies that  $h - z' < \delta$ . Thus  $\|q'' - q'\| \leq 2\delta$ , implying  $\|q - q'\| \leq 4\delta \leq \varepsilon/2$ .

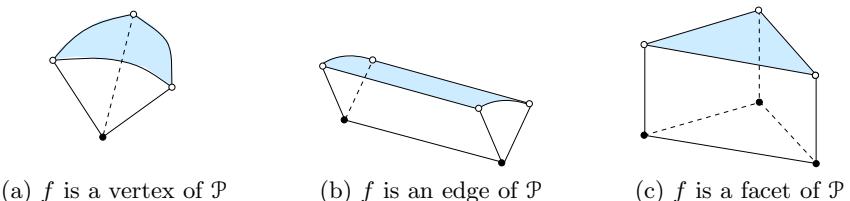
**Lemma 10.** *For an edge region  $\sigma(e)$ , there exists  $O(1)$  points  $K_e \subset \sigma(e) \cap \text{conv}(P)$  such that for any point  $q = \bar{q}[z, v] \in \sigma(e) \cap \text{conv}(P)$  there is a point  $p = \bar{q}[h, u] \in \text{conv}(K_e)$  such that  $z - h \leq 2\delta$ ,  $\|v - u\| \leq 2\delta$ , and, in particular,  $\|q - p\| \leq 4\delta \leq \varepsilon/2$ .*

For  $K \subset P \in \mathbb{R}^2$  there are  $|K|$  points and edges in  $\mathcal{P}$ . Thus combining Lemmas 9 and 10  $|K'|/|K| = O(1)$  and we have proven Theorem 2 for  $d = 2$ . Next, we prove the theorem for  $d = 3$ .

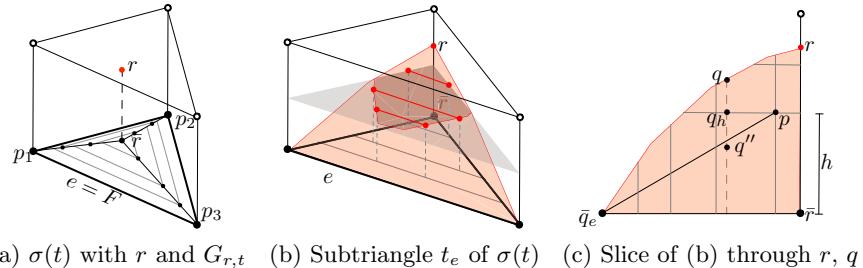
**Construction of  $K'$ .** Now consider  $K \subset P \in \mathbb{R}^3$  and the point regions, edge regions, and triangle regions in the decomposition of  $\mathcal{P}_\varepsilon \setminus \text{intr } \mathcal{P}$  (see Figure 4). By Lemmas 9 and 10 we can add  $O(|K|)$  points to  $K'$  to account for all point and edge regions. We can now focus on the  $O(|K|)$  triangle regions.

Consider a triangle region  $\sigma(t)$  for a triangle  $t$  in  $\mathcal{P}$  (see Figure 5(a)),  $t^*$  consists of a single direction, the one normal to  $t$ . Let  $r$  be the highest point of  $\sigma(t) \cap \text{conv}(P)$  in direction  $t^*$ . We add  $r$  to  $K'$  and we create an exponential grid  $G_{r,t}$  with  $r$  as the support point. For each edge  $e \in G_{r,t}$  and  $h \in \mathcal{H}$  we add the intersection of  $e[t^*, h]$  with the boundary of  $\sigma(t) \cap \text{conv}(P)$  to  $K'$ , as shown in Figure 5(b). Thus, in total we add  $O(|K| \log(1/\varepsilon))$  points to  $K'$ .

**Proof of correctness.** Consider any point  $q = \bar{q}[t^*, z] \in \sigma(t) \cap \text{conv}(P)$  and associate it with a boundary edge  $e$  of  $t$  such that  $\bar{q} \in \text{conv}(e \cup \bar{r})$ . Let  $q_h = \bar{q}[t^*, h]$  where  $h \in \mathcal{H}$  is the largest height such that  $h \leq z$ . If segment  $\bar{q}\bar{r}$  does not



**Fig. 4.** Illustration of regions in the partition of  $\mathcal{P}_\varepsilon \setminus \text{intr } \mathcal{P}$  in three dimensions



**Fig. 5.** Illustration to aid correctness of approximation of triangle regions in  $\mathbb{R}^3$

intersect any edge  $e_i$  parallel to  $e$  in  $G_{r,t}$ , let  $\bar{p} = \bar{r}$ . Otherwise, let  $e_i$  be the first segment parallel to  $e$  in  $G_{r,t}$  intersected by the ray  $\vec{qr}$ , and let  $\bar{p}$  be the intersection. Let  $p = \bar{p}[t^*, h]$  which must be in  $\text{conv}(K')$  by construction. If  $e_i = e_0$ , then by (G1) we have  $\|q_h - p\| = \|\bar{q} - \bar{p}\| \leq \delta$ , thus  $\|q - p\| \leq 2\delta \leq \varepsilon/2$  and we are done. Otherwise, let  $\bar{q}_e$  be the intersection of  $e$  with ray  $\vec{rq}$ . By (G2)  $\|\bar{p} - \bar{q}\| / \|\bar{p} - \bar{q}_e\| \leq \rho = \delta/\varepsilon$ . Thus,  $q'' = \bar{q}[t^*, h - \varepsilon\rho]$  is below the segment  $\bar{q}_e p$  (see Figure 5(c)) and thus  $q'' \in \text{conv}(K')$  since triangle  $p\bar{p}\bar{q}_e$  is in  $\text{conv}(K')$ . Finally,  $\|q - q''\| = \|q - q_h\| + \|q_h - q''\| \leq 2\delta \leq \varepsilon/2$ . This proves Theorem 2 for  $d = 3$ .

### 3.1 Remarks

- (1) For  $d = 2, 3$ ,  $\kappa(P, \varepsilon/2)$  is only a factor of  $O(1)$  and  $O(\log(1/\varepsilon))$ , respectively, larger than  $\kappa(P, \varepsilon)$ ; therefore, the sizes of optimal  $\varepsilon$ -coresets in these dimensions are relative stable. However, for  $d \geq 4$ , the stability drastically reduces in the worst case because of the superlinear dependency on  $\kappa(P, \varepsilon)$ .
- (2) Neither the upper nor the lower bound in the theorem is tight. For  $d = 3$ , we can prove a tighter lower bound of  $\Omega(\kappa(P, \varepsilon) \log(1/(\varepsilon \cdot \kappa(P, \varepsilon))))$ . We conjecture in  $\mathbb{R}^d$  that

$$\kappa(P, \varepsilon/2) = \Theta(\kappa(P, \varepsilon)^{\lfloor d/2 \rfloor} \log^{d-2}(1/(\varepsilon^{(d-1)/2} \cdot \kappa(P, \varepsilon)))).$$

## References

1. Agarwal, P.K., Har-Peled, S., Varadarajan, K.: Approximating extent measure of points. *Journal of ACM* 51(4), 606–635 (2004)
2. Agarwal, P.K., Har-Peled, S., Varadarajan, K.: Geometric approximations via coresets. In: *Combinatorial and Computational Geometry*, pp. 1–31 (2005)
3. Agarwal, P.K., Phillips, J.M., Yu, H.: Stability of  $\varepsilon$ -coresets. arXiv:1003.5874
4. Agarwal, P.K., Yu, H.: A space-optimal data-stream algorithm for coresets in the plane. In: *SoCG*, pp. 1–10 (2007)
5. Barequet, G., Har-Peled, S.: Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journ. of Algs.* 38, 91–109 (2001)
6. Chan, T.: Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry: Theory and Applications* 35, 20–35 (2006)
7. Chan, T.: Dynamic coresets. In: *SoCG*, pp. 1–9 (2008)

8. Har-Peled, S.: Approximation Algorithm in Geometry, ch. 22 (2010),  
<http://valis.cs.uiuc.edu/~sariel/teach/notes/aprx/>
9. Hershberger, J., Suri, S.: Adaptive sampling for geometric problems over data streams. Computational Geometry: Theory and Applications 39, 191–208 (2008)
10. Yu, H., Agarwal, P.K., Poreddy, R., Varadarajan, K.: Practical methods for shape fitting and kinetic data structures using coresets. Algorithmica 52, 378–402 (2008)
11. Zarrabi-Zadeh, H.: An almost space-optimal streaming algorithm for coresets in fixed dimensions. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 817–829. Springer, Heidelberg (2008)

# The Geodesic Diameter of Polygonal Domains\*

Sang Won Bae<sup>1</sup>, Matias Korman<sup>2</sup>, and Yoshio Okamoto<sup>3</sup>

<sup>1</sup> Department of Computer Science, Kyonggi University, Korea  
[swbae@kgu.ac.kr](mailto:swbae@kgu.ac.kr)

<sup>2</sup> Computer Science Department, Université Libre de Bruxelles (ULB), Belgium  
[mkormanc@ulb.ac.be](mailto:mkormanc@ulb.ac.be)

<sup>3</sup> Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Tokyo, Japan  
[okamoto@is.titech.ac.jp](mailto:okamoto@is.titech.ac.jp)

**Abstract.** This paper studies the geodesic diameter of polygonal domains having  $h$  holes and  $n$  corners. For simple polygons (i.e.,  $h = 0$ ), it is known that the geodesic diameter is determined by a pair of corners of a given polygon and can be computed in linear time. For general polygonal domains with  $h \geq 1$ , however, no algorithm for computing the geodesic diameter was known prior to this paper. In this paper, we present the first algorithm that computes the geodesic diameter of a given polygonal domain in worst-case time  $O(n^{7.73})$  or  $O(n^7(\log n + h))$ . Among other results, we show the following geometric observation: the geodesic diameter can be determined by two points in its interior. In such a case, there are at least five shortest paths between the points.

## 1 Introduction

In this paper, we address the geodesic diameter problem in polygonal domains. Intuitively, given a polygonal domain  $\mathcal{P}$  with holes, the geodesic distance  $d(p, q)$  between points  $p$  and  $q$  of  $\mathcal{P}$  is defined as the length of a shortest path between them, among all the paths that stay within  $\mathcal{P}$ . The geodesic diameter  $\text{diam}(\mathcal{P})$  is defined as the largest geodesic distance between any two points of  $\mathcal{P}$ .

For simple polygons (i.e., domains with no holes), the geodesic diameter has been extensively studied. Chazelle [7] provided the first  $O(n^2)$ -time algorithm computing the geodesic diameter of a simple polygon, where  $n$  is the number of corners of  $\mathcal{P}$ . Afterwards, Suri [19] presented an  $O(n \log n)$ -time algorithm that solves the all-geodesic-farthest neighbors problem, computing the farthest neighbor of every corner and thus finding the geodesic diameter. At last, Hershberger and Suri [12] showed that the diameter can be computed in linear time using their fast matrix search technique.

---

\* Work by S.W. Bae was supported by National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0005974). Work by Y. Okamoto was supported by Global COE Program “Computationism as a Foundation for the Sciences” and Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

On the other hand, the geodesic diameter of a domain with holes is less understood. Mitchell [15] has posed an open problem asking an algorithm for computing the geodesic diameter of a polygonal domain. Moreover, even for the corner-to-corner diameter  $\max_{u,v \in V} d(u, v)$ , we know nothing but the brute-force algorithm that takes  $O(n^2 \log n)$  time, checking all the geodesic distances between every pair of corners.<sup>1</sup>

This fairly wide gap between simple polygons and polygonal domains is seemingly due to the uniqueness of the shortest path between any two points. When no holes exist, it is well known that there is a unique shortest path between any two points [10]. Using this uniqueness, one can show that the diameter is realized by a pair of corners [12, 19]. For general polygonal domains, however, this is not the case. In this paper, we exhibit several examples where the diameter is realized by non-corner points on  $\partial\mathcal{P}$  or even by interior points of  $\mathcal{P}$ . (See Fig. 1) This observation also shows an immediate difficulty in devising any exhaustive algorithm since the search space is not discrete.

The status of the geodesic center problem is also similar. A point in  $\mathcal{P}$  is defined as a *geodesic center* if it minimizes the maximum geodesic distance from it to any other point of  $\mathcal{P}$ . Asano and Toussaint [3] introduced the first  $O(n^4 \log n)$ -time algorithm for computing the geodesic center of a simple polygon (i.e., when  $h = 0$ ), and Pollack, Sharir and Rote [18] improved it to  $O(n \log n)$  time. As with the diameter problem, there is no known algorithm for domains with holes. See O'Rourke and Suri [17] and Mitchell [15] for more references on the geodesic diameter/center problem.

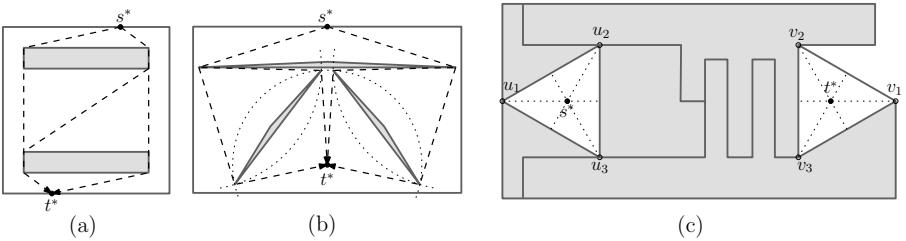
Since the geodesic diameter/center of a simple polygon is determined by its corners, one can exploit the *geodesic farthest-site Voronoi diagram* of the corners  $V$  to compute the diameter/center, which can be built in  $O(n \log n)$  time [2]. Recently, Bae and Chwa [4] presented an  $O(nk \log^3(n+k))$ -time algorithm for computing the geodesic farthest-site Voronoi diagram of  $k$  sites in polygonal domains with holes. This result can be used to compute the geodesic diameter  $\max_{p,q \in S} d(p, q)$  of a *finite* set  $S$  of points in  $\mathcal{P}$ . However, this approach cannot be directly used for computing  $\text{diam}(\mathcal{P})$  without any characterization of the diameter. Moreover, when  $S = V$ , this approach is no better than the brute-force  $O(n^2 \log n)$ -time algorithm for computing the corner-to-corner diameter  $\max_{u,v \in V} d(u, v)$ .

In this paper, we present the first algorithms that compute the geodesic diameter of a given polygonal domain in  $O(n^{7.73})$  or  $O(n^7(\log n + h))$  time in the worst case. Our new geometric results underlying the algorithms show that the existence of any diametral pair consisting of non-corner points implies multiple shortest paths between the pair; Among other results, we show that *if  $(s, t)$  is a diametral pair and both  $s$  and  $t$  lie in the interior of  $\mathcal{P}$ , then there are at least five shortest paths between  $s$  and  $t$* .

Some analogies between polygonal domains and convex polytopes in  $\mathbb{R}^3$  can be seen. O'Rourke and Schevon [16] proved that if the geodesic diameter on a convex 3-polytope is realized by two non-corner points, at least five shortest

---

<sup>1</sup> Personal communication with Mitchell.



**Fig. 1.** Three polygonal domains where the geodesic diameter is determined by a pair  $(s^*, t^*)$  of non-corner points; Gray-shaded regions depict the interior of the holes and dark gray segments depict the boundary  $\partial\mathcal{P}$ . Recall that  $\mathcal{P}$ , as a set, contains its boundary  $\partial\mathcal{P}$ . (a) Both  $s^*$  and  $t^*$  lie on  $\partial\mathcal{P}$ . There are three shortest paths between  $s^*$  and  $t^*$ . In this polygonal domain, there are two (symmetric) diametral pairs. (b)  $s^* \in \partial\mathcal{P} \setminus V$  and  $t^* \in \text{int}\mathcal{P}$ . Three triangular holes are placed in a symmetric way. There are four shortest paths between  $s^*$  and  $t^*$ . (c) Both  $s^*$  and  $t^*$  lie in the interior  $\text{int}\mathcal{P}$ . Here, the five holes are packed like jigsaw puzzle pieces, forming narrow corridors (dark gray paths) and two empty, regular triangles. Observe that  $d(u_1, v_1) = d(u_1, v_2) = d(u_2, v_2) = d(u_2, v_3) = d(u_3, v_3) = d(u_3, v_1)$ . The points  $s^*$  and  $t^*$  lie at the centers of the triangles formed by the  $u_i$  and the  $v_i$ , respectively. There are six shortest paths between  $s^*$  and  $t^*$ .

paths exist between the two (a simpler proof of the same fact was later shown by Zalgaller [20]). Based on this observation, they presented an  $O(n^{14} \log n)$ -time algorithm for computing the geodesic diameter on a convex 3-polytope. Afterwards, the time bound was improved to  $O(n^8 \log n)$  by Agarwal et al. [1] and recently to  $O(n^7 \log n)$  by Cook IV and Wenk [9]. Although our algorithms seemingly require fairly large amount of time, notice that they are comparable with the runtimes for convex polytopes.

Due to the space constraints, some of the discussions and examples are omitted in this version. Details can be found in the arXiv version [5].

## 2 Preliminaries

A *polygonal domain*  $\mathcal{P}$  with  $h$  holes and  $n$  corners  $V$  is a connected and closed subset of  $\mathbb{R}^2$  with  $h$  holes whose boundary  $\partial\mathcal{P}$  consists of  $h + 1$  simple closed polygonal chains of  $n$  total line segments. The holes and the outer boundary of  $\mathcal{P}$  are regarded as *obstacles* so that any feasible path in  $\mathcal{P}$  is not allowed to cross the boundary  $\partial\mathcal{P}$ . The geodesic distance  $d(p, q)$  between any two points  $p, q$  in a polygonal domain  $\mathcal{P}$  is defined as the length of a shortest obstacle-avoiding path between  $p$  and  $q$ , where the *length* of a path is the sum of the Euclidean lengths of its segments. It is well known from earlier work that there always exists a *shortest feasible path* between any two points  $p, q \in \mathcal{P}$  [14]. A pair  $(s, t)$  of points in  $\mathcal{P}$  that realizes the geodesic diameter  $\text{diam}(\mathcal{P})$  is called a *diametral pair*.

Throughout the paper, we frequently use several topological concepts such as open and closed subsets, neighborhoods, and the boundary  $\partial A$  and the interior

$\text{int}A$  of a set  $A$ ; all of them are derived with respect to the standard topology on  $\mathbb{R}^d$  with the Euclidean norm  $\|\cdot\|$  for fixed  $d \geq 1$ . We call a  $k$ -dimensional affine subspace of  $\mathbb{R}^d$  a  $k$ -flat and denote the straight line segment joining two points  $a, b$  by  $ab$ .

Let  $V$  be the set of all corners of  $\mathcal{P}$  and  $\pi(s, t)$  be a shortest path between  $s \in \mathcal{P}$  and  $t \in \mathcal{P}$ . Such a path is represented as a sequence  $\pi(s, t) = (s, v_1, \dots, v_k, t)$  for some  $v_1, \dots, v_k \in V$ ; that is, a polygonal chain through a sequence of corners [14]. Note that we may have  $k = 0$  when  $d(s, t) = \|s - t\|$ . If two paths (with possibly different endpoints) induce the same sequence of corners, then they are said to have the same *combinatorial structure*.

The *shortest path map*  $\text{SPM}(s)$  for a fixed  $s \in \mathcal{P}$  is a decomposition of  $\mathcal{P}$  into cells such that every point in a common cell can be reached from  $s$  by shortest paths of the same combinatorial structure. Each cell  $\sigma_s(v)$  of  $\text{SPM}(s)$  is associated with a corner  $v \in V$  which is the last corner of  $\pi(s, t)$  for any  $t$  in the cell  $\sigma_s(v)$ . We also define the cell  $\sigma_s(s)$  as the set of points  $t \in \mathcal{P}$  in which  $\pi(s, t)$  passes through no corner of  $\mathcal{P}$ . Each edge of  $\text{SPM}(s)$  is an arc on the boundary of two incident cells  $\sigma_s(v_1)$  and  $\sigma_s(v_2)$  determined by two corners  $v_1, v_2 \in V \cup \{s\}$ . Similarly, each vertex of  $\text{SPM}(s)$  is determined by at least three corners  $v_1, v_2, v_3 \in V \cup \{s\}$ . Note that for fixed  $s \in \mathcal{P}$  a point whose distance to  $s$  is the largest lies at either (1) a vertex of  $\text{SPM}(s)$ , (2) an intersection between the boundary  $\partial\mathcal{P}$  and an edge of  $\text{SPM}(s)$ , or (3) a corner in  $V$  (otherwise a farther point can be found). The shortest path map  $\text{SPM}(s)$  has  $O(n)$  complexity and can be computed in  $O(n \log n)$  time using  $O(n \log n)$  working space [13]. For more details on shortest path maps, see [14, 13, 15].

*Path-length function.* If  $\pi(s, t) \neq \overline{st}$ , let  $u$  and  $v$  be the first and last corners of  $V$  visited in  $\pi(s, t)$ , respectively. The path  $\pi(s, t)$  is formed as the union of  $\overline{su}$ ,  $\overline{vt}$  and  $\pi(u, v)$ . Note that  $u$  and  $v$  are not necessarily distinct. In order to realize such a path, we assert that  $s$  is visible from  $u$  and  $t$  is visible from  $v$ . That is,  $s \in \text{VR}(u)$  and  $t \in \text{VR}(v)$ , where  $\text{VR}(p)$  for any  $p \in \mathcal{P}$  is defined to be the set of all points  $q \in \mathcal{P}$  such that  $\overline{pq} \subset \mathcal{P}$ . The set  $\text{VR}(p)$  is called the *visibility region* of  $p \in \mathcal{P}$  [8].

We now define the *path-length function*  $\text{len}_{u,v}: \text{VR}(u) \times \text{VR}(v) \rightarrow \mathbb{R}$  for any fixed pair of corners  $u, v \in V$  to be  $\text{len}_{u,v}(s, t) := \|s - u\| + d(u, v) + \|v - t\|$ . That is,  $\text{len}_{u,v}(s, t)$  represents the length of the path from  $s$  to  $t$  that has the fixed combinatorial structure, entering  $u$  from  $s$  and exiting  $v$  to  $t$ . Also, unless  $d(s, t) = \|s - t\|$  (equivalently,  $s \in \text{VR}(t)$ ), the geodesic distance  $d(s, t)$  can be expressed as the pointwise minimum of some path-length functions:

$$d(s, t) = \min_{u \in \text{VR}(s), v \in \text{VR}(t)} \text{len}_{u,v}(s, t).$$

Consequently, we have two possibilities for a diametral pair  $(s^*, t^*)$ ; either we have  $d(s^*, t^*) = \|s^* - t^*\|$  or the pair  $(s^*, t^*)$  is a local maximum of the lower envelope of several path-length functions. We will mainly study the latter case, since the former can be easily handled.

## 2.1 Local Maxima of the Lower Envelope of Convex Functions

In this section, we give an interesting property of the lower envelope of a family of convex functions.

**Theorem 1.** *Let  $\mathcal{F}$  be a finite family of real-valued convex functions defined on a convex subset  $C \subseteq \mathbb{R}^d$  and  $g(x) := \min_{f \in \mathcal{F}} f(x)$  be their lower envelope. Suppose that  $g$  attains a local maximum at an interior point  $x^* \in C$  and there are exactly  $m$  functions  $f_1, \dots, f_m \in \mathcal{F}$  such that  $m \leq d$  and  $f_i(x^*) = g(x^*)$  for all  $i = 1, \dots, m$ . If none of the  $f_i$  attains a local minimum at  $x^*$ , then there exists a  $(d+1-m)$ -flat  $\varphi \subset \mathbb{R}^d$  through  $x^*$  such that  $g$  is constant on  $\varphi \cap U$  for some neighborhood  $U \subset \mathbb{R}^d$  of  $x^*$  with  $U \subset C$ .*

Here, we give a sketchy idea of our proof for the theorem. Consider a local maximum  $x^*$  of the lower envelope  $g$  of a family of functions. Let  $m$  be the number of functions  $f \in \mathcal{F}$  such that  $f(x^*) = g(x^*)$ . Since  $g$  is the lower envelope, all other functions  $f' \in \mathcal{F}$  must satisfy  $f'(x^*) > g(x^*)$ . In particular, the function  $g$  is the lower envelope of the  $m$  convex functions in a small neighborhood of  $x^*$ . Since no function attains a local minimum at  $x^*$ , for each function there exists at least one direction in which the function value does not decrease. We show something stronger: Using convexity of the  $f_i$ , we will show that there is a flat of dimension  $d+1-m$  through  $x^*$  on which the  $m$  functions that define  $g$  must either increase or stay constant. Since  $x^*$  is a local maximum, the only possibility is that  $g$  remains constant in the flat.

## 3 Properties of Geodesic-Maximal Pairs

We call a pair  $(s^*, t^*) \in \mathcal{P} \times \mathcal{P}$  *maximal* if  $(s^*, t^*)$  is a local maximum of the geodesic distance function  $d$ . That is,  $(s^*, t^*)$  is maximal if and only if there are two neighborhoods  $U_s, U_t \subset \mathbb{R}^2$  of  $s^*$  and of  $t^*$ , respectively, such that for any  $s \in U_s \cap \mathcal{P}$  and any  $t \in U_t \cap \mathcal{P}$  we have  $d(s^*, t^*) \geq d(s, t)$ .

Let  $E$  be the set of all sides of  $\mathcal{P}$  without their endpoints and  $\mathcal{B}$  be their union. Note that  $\mathcal{B} = \partial\mathcal{P} \setminus V$  is the boundary of  $\mathcal{P}$  except the corners  $V$ . The goal of this section is to prove the following theorem, which is the main geometric result of this paper.

**Theorem 2.** *Suppose that  $(s^*, t^*)$  is a maximal pair in  $\mathcal{P}$  that are not visible from each other, and  $\Pi(s^*, t^*)$ ,  $V_{s^*}$ , and  $V_{t^*}$  are defined as above. We have the following implications.*

- (V-V)  $s^* \in V, t^* \in V$  implies  $|\Pi(s^*, t^*)| \geq 1, |V_{s^*}| \geq 1, |V_{t^*}| \geq 1$ ;
- (V-B)  $s^* \in V, t^* \in \mathcal{B}$  implies  $|\Pi(s^*, t^*)| \geq 2, |V_{s^*}| \geq 1, |V_{t^*}| \geq 2$ ;
- (V-I)  $s^* \in V, t^* \in \text{int}\mathcal{P}$  implies  $|\Pi(s^*, t^*)| \geq 3, |V_{s^*}| \geq 1, |V_{t^*}| \geq 3$ ;
- (B-B)  $s^* \in \mathcal{B}, t^* \in \mathcal{B}$  implies  $|\Pi(s^*, t^*)| \geq 3, |V_{s^*}| \geq 2, |V_{t^*}| \geq 2$ ;
- (B-I)  $s^* \in \mathcal{B}, t^* \in \text{int}\mathcal{P}$  implies  $|\Pi(s^*, t^*)| \geq 4, |V_{s^*}| \geq 2, |V_{t^*}| \geq 3$ ;
- (I-I)  $s^* \in \text{int}\mathcal{P}, t^* \in \text{int}\mathcal{P}$  implies  $|\Pi(s^*, t^*)| \geq 5, |V_{s^*}| \geq 3, |V_{t^*}| \geq 3$ .

Moreover, each of the above bounds is tight.

We first give an idea of the proof. The general reasoning is roughly the same for all the different scenarios, and thus we focus on the case in which  $(s^*, t^*)$  is a maximal pair and both  $s^*$  and  $t^*$  are interior points. Consider the geodesic distance as a four-variate function in a neighborhood of  $(s^*, t^*)$ . As mentioned in Section 2, the geodesic distance is the pointwise minimum of the family of path-length functions. Since the pair  $(s^*, t^*)$  is maximal, we want to apply Theorem 1. We will show that all the requirements are satisfied, which implies that the geodesic distance is constant in a flat of dimension  $d + 1 - m = 5 - m$ . However, we will also show that the geodesic distance function can only remain constant in a zero-dimensional flat (i.e., a point), hence  $m \geq 5$ . The main technical difficulty of the proof is the fact that the  $\text{len}_{u,v}$  are not globally defined. Thus, we must slightly redefine them in a way that all conditions of Theorem 1 are satisfied. In the other cases (boundary-interior, boundary-boundary, etc.) the boundary of  $\mathcal{P}$  introduces additional constraints that reduce the degrees of freedom of the geodesic distance function. Hence, fewer paths are enough to pin the solution.

Throughout this section, for easy discussion, we assume that there is a *unique* shortest path between any two corners  $u, v \in V$ . This assumption does not affect Theorem 2 since multiple shortest paths between corners in  $V$  can only increase  $|\Pi(s^*, t^*)|$ . Note that this assumption implies that the pairs  $(u_i, v_i)$  are distinct, while the  $u_i$  (also the  $v_i$ ) are not necessarily distinct. We then have  $|V_{s^*}| \leq m$ ,  $|V_{t^*}| \leq m$ , and  $|\{(u_i, v_i) \mid 1 \leq i \leq m\}| = m$ , where  $m = |\Pi(s^*, t^*)|$ .

The following lemma proves the bounds on  $|V_{s^*}|$  and  $|V_{t^*}|$  of Theorem 2.

**Lemma 1.** *Let  $(s^*, t^*)$  be a maximal pair.*

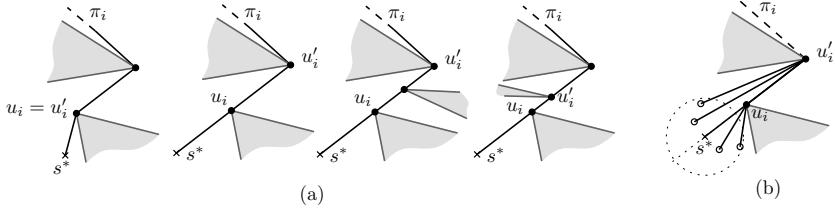
1. *If  $t^* \in \mathcal{B}$ , then  $|V_{t^*}| \geq 2$ . Moreover, if  $t^* \in e \in E$ , then there exists  $v \in V_{t^*}$  such that  $v$  is off the line supporting  $e$ .*
2. *If  $t^* \in \text{int}\mathcal{P}$ , then  $|V_{t^*}| \geq 3$  and  $t^*$  lies in the interior of the convex hull of  $V_{t^*}$ .*

Lemma 1 immediately implies the lower bound on  $|\Pi(s^*, t^*)|$  when  $s^* \in V$  or  $t^* \in V$  since  $|\Pi(s^*, t^*)| \geq \max\{|V_{s^*}|, |V_{t^*}|\}$ . This finishes the proof for Cases (V- $*$ ). Note that the bounds for Case (V-V) are trivial.

From now on, we assume that both  $s^*$  and  $t^*$  are not corners of  $\mathcal{P}$ . This assumption, together with Lemma 1, implies multiple shortest paths between  $s^*$  and  $t^*$ , and thus  $d(s^*, t^*) > \|s^* - t^*\|$ . Hence, as discussed in Section 2, any maximal pair falling into one of Cases (B-B), (B-I), and (I-I) appears as a local maximum of the lower envelope of some path-length functions.

**Case (I-I): When both  $s^*$  and  $t^*$  lie in  $\text{int}\mathcal{P}$ .** We will apply Theorem 1 to prove Theorem 2 for Case (I-I). By definition of  $u_i$  and  $v_i$ , we have that the path-length function  $\text{len}_{u_i, v_i}(s^*, t^*)$  satisfies  $\text{len}_{u_i, v_i}(s^*, t^*) = d(s^*, t^*)$ . Thus, at least  $m$  pairs  $(u, v)$  of corners satisfy  $\text{len}_{u, v}(s^*, t^*) = d(s^*, t^*)$ . If there are exactly  $m$  such pairs, we can apply Theorem 1 directly.

Unfortunately, this is not always the case. A single shortest path  $\pi_i \in \Pi(s^*, t^*)$  may give additional pairs  $(u, v)$  of corners with  $u, v \in \pi_i$  such that  $(u, v) \neq (u_i, v_i)$  and  $\text{len}_{u, v}(s^*, t^*) = d(s^*, t^*)$ . This happens only when  $u, u_i, s^*$  or  $v, v_i, t^*$  are collinear. In order to resolve this problem, we define the *merged* path-length functions that satisfy all the requirements of Theorem 1 even in the degenerate case.



**Fig. 2.** (a) How to determine  $u'_i$ . (left to right)  $u_i = u'_i$ ;  $s^*$ ,  $u_i$ , and the second corner are collinear;  $s^*$  and the first three corners are collinear (b) For points in a small disk  $B$  centered at  $s^*$  with  $B \subset \text{VR}(u'_i) \cup \text{VR}(u_i)$ , the function  $\text{start}_i$  measures the length of the shortest path from  $u'_i$  to each.

Recall that the combinatorial structure of each shortest path  $\pi_i \in \Pi(s^*, t^*)$  can be represented by a sequence  $(u_i = u_{i,1}, \dots, u_{i,k} = v_i)$  of corners in  $V$  (see Fig. 2). We define  $u'_i$  to be one of the  $u_{i,j}$  as follows: If  $s^*$  does not lie on the line  $\ell$  through  $u_i$  and  $u_{i,2}$ , then  $u'_i := u_i$ ; otherwise, if  $s^* \in \ell$ , then  $u'_i := u_{i,j}$ , where  $j$  is the largest index such that for any open neighborhood  $N \subset \mathbb{R}^2$  of  $s^*$  there exists a point  $s \in (N \cap \text{VR}(u_{i,j})) \setminus \ell$ . Note that such  $u'_i$  always exists, and if no three of  $V$  are collinear, then we always have either  $u'_i = u_i$  or  $u'_i = u_{i,2}$ ; Fig. 2(a) illustrates how to determine  $u'_i$ . Also, we define  $v'_i$  in an analogous way. Let  $\text{start}_i: \text{VR}(u'_i) \cup \text{VR}(u_i) \rightarrow \mathbb{R}$  and  $\text{end}_i: \text{VR}(v'_i) \cup \text{VR}(v_i) \rightarrow \mathbb{R}$  be two functions defined as

$$\begin{aligned} \text{start}_i(s) &:= \begin{cases} \|s - u'_i\| & \text{if } s \in \text{VR}(u'_i), \\ \|s - u_i\| + \|u_i - u'_i\| & \text{if } s \in \text{VR}(u_i) \setminus \text{VR}(u'_i); \end{cases} \\ \text{end}_i(t) &:= \begin{cases} \|t - v'_i\| & \text{if } t \in \text{VR}(v'_i), \\ \|t - v_i\| + \|v_i - v'_i\| & \text{if } t \in \text{VR}(v_i) \setminus \text{VR}(v'_i). \end{cases} \end{aligned}$$

This allows us to define merged path-length function  $f_i: D_i \rightarrow \mathbb{R}$  as

$$f_i(s, t) := \text{start}_i(s) + d(u'_i, v'_i) + \text{end}_i(t),$$

where  $D_i := (\text{VR}(u'_i) \cup \text{VR}(u_i)) \times (\text{VR}(v'_i) \cup \text{VR}(v_i)) \subseteq \mathcal{P} \times \mathcal{P}$ ; see Fig. 2(b). We consider  $\mathcal{P} \times \mathcal{P}$  as a subset of  $\mathbb{R}^4$  and each pair  $(s, t) \in \mathcal{P} \times \mathcal{P}$  as a point in  $\mathbb{R}^4$ . Also, we denote by  $(s_x, s_y)$  the coordinates of a point  $s \in \mathcal{P}$  and we write  $s = (s_x, s_y)$  or  $(s, t) = (s_x, s_y, t_x, t_y)$  by an abuse of notation. Observe that  $f_i(s, t) = \min\{\text{len}_{u_i, v_i}(s, t), \text{len}_{u'_i, v_i}(s, t), \text{len}_{u_i, v'_i}(s, t), \text{len}_{u'_i, v'_i}(s, t)\}$  for any  $(s, t) \in D_i$  if we define  $\text{len}_{u, v}(s, t) = \infty$  when  $s \notin \text{VR}(u)$  or  $t \notin \text{VR}(v)$ .

**Lemma 2.** *The functions  $f_i$  satisfy the following conditions.*

- (i) *There exists an open convex neighborhood  $C \subset \mathbb{R}^4$  of  $(s^*, t^*)$  with  $C \subseteq \bigcap D_i$  such that  $d(s, t) = \min_{i \in \{1, \dots, m\}} f_i(s, t)$  for any  $(s, t) \in C$ .*
- (ii) *All functions  $f_i$  are convex on  $C$ .*

- (iii) None of the  $f_i$  attains a local minimum at  $(s^*, t^*)$ .
- (iv)  $f_i(s^*, t^*) = d(s^*, t^*)$  for any  $i \in \{1, \dots, m\}$ .
- (v) For any  $i \in \{1, \dots, m\}$  and any  $(s, t) \in \text{int}D_i$ , there exists a unique line  $\ell_i \subset \mathbb{R}^4$  through  $(s, t)$  such that  $f_i$  is constant on  $\ell_i \cap C$ . Moreover, there exists at most one other index  $j \neq i$  such that  $\ell_i = \ell_j$ .
- (vi) For any two indices  $i, j \in \{1, \dots, m\}$ , any  $(s, t) \in C$  and any small neighborhood  $U \subseteq C$  of  $(s, t)$ , there exists a pair  $(s', t') \in U$  such that  $f_i(s, t) < f_i(s', t')$  and  $f_j(s, t) < f_j(s', t')$ .

We take the convex neighborhood  $C$  of  $(s^*, t^*)$  of property (i) in Lemma 2 and apply Theorem 1 (note that properties (i)-(iv) ensure that the conditions of Theorem 1 are satisfied). Suppose that  $m < 5$ . Then, by Theorem 1, there exists at least one line  $\ell \in \mathbb{R}^4$  through  $(s^*, t^*)$  such that  $d$  is constant on  $\ell \cap C$ . In the following, we will use the other geometric properties (v)-(vi) of Lemma 2 to achieve a contradiction, finally showing that  $m \geq 5$  must hold.

Since  $(s^*, t^*)$  is a local maximum, there exists a small neighborhood  $U \subset C$  of  $(s^*, t^*)$  such that  $d(s, t) \leq d(s^*, t^*)$  for all  $(s, t) \in U$ . By property (v) of Lemma 2, at most two functions  $f_i$  are constant on  $\ell \cap U$ . Without loss of generality, we can assume that functions  $f_3, \dots, f_m$  are not constant. Since the geodesic distance is constant on  $\ell \cap U$  and  $d(s, t) = \min_{i \in \{1, \dots, m\}} f_i(s, t)$ , any function that does not remain constant must strictly increase in both directions along  $\ell$ .

That is, for any  $(s', t') \in \ell \cap U$  with  $(s', t') \neq (s^*, t^*)$  and for all  $i \geq 3$ , we have  $\min\{f_1(s', t'), f_2(s', t')\} < f_i(s', t')$ . Thus, there exists a small neighborhood  $U_2 \subseteq U$  of  $(s', t')$  such that  $d(s, t) = \min\{f_1(s, t), f_2(s, t)\}$  for all  $(s, t) \in U_2$ . However, by property (vi) of Lemma 2, there exists a pair  $(s'', t'') \in U_2$  such that  $f_1(s', t') < f_1(s'', t'')$  and  $f_2(s', t') < f_2(s'', t'')$ , contradicting the fact that  $(s^*, t^*)$  is maximal. Hence, we achieve a bound  $m = |\Pi(s^*, t^*)| \geq 5$ , as claimed in Case (I-I) of Theorem 2.

As aforementioned, the other cases (B-B) and (B-I) can be handled with analogous arguments. The claimed bounds on  $|V_{s^*}|$  and  $|V_{t^*}|$  are shown by Lemma 1 which completes the proof of Theorem 2.

## 4 Computing the Geodesic Diameter

Since a diametral pair is in fact maximal, it falls into one of the cases shown in Theorem 2. In order to find a diametral pair we examine all possible scenarios accordingly.

Cases (V-\*), where at least one point is a corner in  $V$ , can be handled in  $O(n^2 \log n)$  time by computing  $\text{SPM}(v)$  for every  $v \in V$  and traversing it to find the farthest point from  $v$ , as discussed in Section 2. We focus on Cases (B-B), (B-I), and (I-I), where a diametral pair consists of two non-corner points.

From the computational point of view, the most difficult case corresponds to Case (I-I) of Theorem 2. In particular, when  $|\Pi(s^*, t^*)| = 5$  and ten corners of  $V$  are involved, resulting in  $|V_{s^*}| = |V_{t^*}| = 5$ . Note that we do not need to take special care for the case of  $|\Pi(s^*, t^*)| > 5$ . By Theorem 2 and its proof, it is guaranteed that there are at least five distinct pairs  $(u_1, v_1), \dots, (u_5, v_5)$  of corners

in  $V$  such that  $\text{len}_{u_i, v_i}(s^*, t^*) = d(s^*, t^*)$  for any  $i \in \{1, \dots, 5\}$  and the system of equations  $\text{len}_{u_1, v_1}(s, t) = \dots = \text{len}_{u_5, v_5}(s, t)$  determines a 0-dimensional zero set, corresponding to a constant number of candidate pairs in  $\text{int}\mathcal{P} \times \text{int}\mathcal{P}$ . Moreover, each path-length function  $\text{len}_{u, v}$  is an algebraic function of degree at most 4. Given five distinct pairs  $(u_i, v_i)$  of corners, we can compute all candidate pairs  $(s, t)$  in  $O(1)$  time by solving the system<sup>2</sup>. For each candidate pair we compute the geodesic distance between the pair to check its validity. Since the geodesic distance between any two points  $s, t \in \mathcal{P}$  can be computed in  $O(n \log n)$  time [13], we obtain a brute-force  $O(n^{11} \log n)$ -time algorithm, checking  $O(n^{10})$  candidate pairs obtained from all possible combinations of 10 corners in  $V$ .

As a different approach, one can exploit the *SPM-equivalence decomposition* of  $\mathcal{P}$ , which subdivides  $\mathcal{P}$  into regions such that the shortest path map of any two points in a common region are *topologically equivalent* [8]. It is not difficult to see that if  $(s, t)$  is a pair of points that equalizes any five path-length functions, then both  $s$  and  $t$  appear as vertices of the decomposition. However, the current best upper bound on the complexity of the SPM-equivalence decomposition is  $O(n^{10})$  [8], and thus this approach hardly leads to a remarkable improvement.

Instead, we do the following for Case **(I-I)** with  $|V_{s^*}| = 5$ . We choose any five corners  $u_1, \dots, u_5 \in V$  (as a candidate for the set  $V_{s^*}$ ) and overlay their shortest path maps  $\text{SPM}(u_i)$ . Since each  $\text{SPM}(u_i)$  has  $O(n)$  complexity, the overlay consists of  $O(n^2)$  cells. Any cell of the overlay is the intersection of five cells associated with  $v_1, \dots, v_5 \in V$  in  $\text{SPM}(u_1), \dots, \text{SPM}(u_5)$ , respectively. Choosing a cell of the overlay, we get five (possibly, not distinct)  $v_1, \dots, v_5$  and a constant number of candidate pairs by solving the system  $\text{len}_{u_1, v_1}(s, t) = \dots = \text{len}_{u_5, v_5}(s, t)$ . We iterate this process for all possible tuples of five corners  $u_1, \dots, u_5$ , obtaining a total of  $O(n^7)$  candidate pairs in  $O(n^7 \log n)$  time. Note that the other subcases with  $|V_{s^*}| \leq 4$  can be handled similarly, resulting in  $O(n^6)$  candidate pairs.

For each candidate pair  $(s, t)$ , we must check their validity (that is, that the paths  $(s, u_i, \dots, v_i, t)$  are indeed shortest paths) and their geodesic distance using a two-point query structure of Chiang and Mitchell [8]. For a fixed parameter  $0 < \delta \leq 1$  and any fixed  $\epsilon > 0$ , we can construct, in  $O(n^{5+10\delta+\epsilon})$  time, a data structure that supports  $O(n^{1-\delta} \log n)$ -time two-point shortest path queries. The total running time is  $O(n^7 \log n) + O(n^{5+10\delta+\epsilon}) + O(n^7) \times O(n^{1-\delta} \log n)$ . We set  $\delta = \frac{3}{11}$  to optimize the running time to  $O(n^{7+\frac{8}{11}+\epsilon})$ .

Also, we can use an alternative two-point query data structure whose performance is sensitive to  $h$  [8]: after  $O(n^5)$  preprocessing time using  $O(n^5)$  storage, two-point queries can be answered in  $O(\log n + h)$  time<sup>3</sup>. Using this alternative

<sup>2</sup> Here, we assume that fundamental operations on a constant number of polynomials of constant degree with a constant number of variables can be performed in constant time.

<sup>3</sup> If  $h$  is relatively small, one could use the structure of Guo, Maheshwari and Sack [11] which answers a two-point query in  $O(h \log n)$  time after  $O(n^2 \log n)$  preprocessing time using  $O(n^2)$  storage, or another structure by Chiang and Mitchell [8] that supports a two-point query in  $O(h \log n)$  time, spending  $O(n + h^5)$  preprocessing time and storage.

structure, the total running time of our algorithm becomes  $O(n^7(\log n + h))$ . Note that this method outperforms the previous one when  $h = O(n^{\frac{8}{11}})$ .

The other cases can be handled analogously with strictly better time bound. For Case **(B-I)**, we handle only the case of  $|\Pi(s^*, t^*)| = 4$  with  $|V_{t^*}| = 3$  or 4. For the subcase with  $|V_{t^*}| = 4$ , we choose any four corners from  $V$  as  $v_1, \dots, v_4$  as a candidate for  $V_{t^*}$  and overlay their shortest path maps  $\text{SPM}(v_i)$ . The overlay, together with  $V$ , decomposes  $\partial\mathcal{P}$  into  $O(n)$  intervals. Each such interval determines  $u_1, \dots, u_4$  as above, and the side  $e_s \in E$  on which  $s^*$  should lie. Now, we have a system of four equations on four variables: three from the corresponding path-length functions  $\text{len}_{u_i, v_i}$  which should be equalized at  $(s^*, t^*)$  and the fourth from the supporting line of  $e_s$ . Solving the system, we get a constant number of candidate maximal pairs, again by Theorem 2 and its proof. In total, we obtain  $O(n^5)$  candidate pairs. The other subcase with  $|V_{t^*}| = 3$  can be handled similarly, resulting in  $O(n^4)$  candidate pairs. As above, we can exploit two different structures for two-point queries. Consequently, we can handle Case **(B-I)** in  $O(n^{5+\frac{10}{11}+\epsilon})$  or  $O(n^5(\log n + h))$  time.

In Case **(B-B)** when  $s^*, t^* \in \mathcal{B}$ , we handle the case of  $|\Pi(s^*, t^*)| = 3$  with  $|V_{s^*}| = 2$  or 3. For the subcase with  $|V_{s^*}| = 3$ , we choose three corners as a candidate of  $V_{s^*}$  and take the overlay of their shortest path maps  $\text{SPM}(u_i)$ . It decomposes  $\partial\mathcal{P}$  into  $O(n)$  intervals. Each such interval determines three corners  $v_1, v_2, v_3$  forming  $V_{t^*}$  and a side  $e_t \in E$  on which  $t^*$  should lie. Note that we have only three equations so far; two from the three path-length functions and the third from the line supporting to  $e_t$ . Since  $s^*$  also should lie on a side  $e_s \in E$  with  $e_s \neq e_t$ , we need to fix such a side  $e_s$  that  $\bigcap_{1 \leq i \leq 3} \text{VR}(u_i)$  intersects  $e_s$ . In the worst case, the number of such sides  $e_s$  is  $\Theta(n)$ . Thus, we have  $O(n^5)$  candidate pairs for Case **(B-B)**; again, the other subcase with  $|V_{s^*}| = 2$  contributes to a smaller number  $O(n^4)$  of candidate pairs. Testing each candidate pair can be done as above, resulting in  $O(n^{5+\frac{10}{11}+\epsilon})$  or  $O(n^5(\log n + h))$  total running time.

Alternatively, one can exploit a two-point query structure only for boundary points on  $\partial\mathcal{P}$  for Case **(B-B)**. The two-point query structure by Bae and Okamoto [6] builds an explicit representation of the graph of the lower envelope of the path-length functions  $\text{len}_{u, v}$  restricted on  $\partial\mathcal{P} \times \partial\mathcal{P}$  in  $O(n^5 \log n \log^* n)$  time.<sup>4</sup> Since  $|\Pi(s^*, t^*)| \geq 3$  in Case **(B-B)**, such a pair appears as a vertex on the lower envelope. Hence, we are done by traversing all the vertices of the lower envelope.

As Case **(I-I)** is the bottleneck, we conclude the following.

**Theorem 3.** *Given a polygonal domain having  $n$  corners and  $h$  holes, the geodesic diameter and a diametral pair can be computed in  $O(n^{7+\frac{8}{11}+\epsilon})$  or  $O(n^7(\log n + h))$  time in the worst case, where  $\epsilon$  is any fixed positive number.  $\square$*

## 5 Concluding Remarks

We have presented the first algorithms that compute the geodesic diameter of a given polygonal domain. As mentioned in the introduction, a similar result for

<sup>4</sup> More precisely, in  $O(n^4 \lambda_{65}(n) \log n)$  time, where  $\lambda_m(n)$  stands for the maximum length of a Davenport-Schinzel sequence of order  $m$  on  $n$  symbols.

convex 3-polytopes was shown in [16]. We note that, although the main result of this paper is similar, the techniques used in the proof are quite different. Indeed, the key requirement for our proof is the fact that shortest paths in our environment are polygonal chains whose vertices are in  $V$ , a claim that does not hold in higher dimensions (even in 2.5-D surfaces). It would be interesting to find other environments in which similar result holds.

Another interesting question would be finding out how many maximal pairs a polygonal domain can have. The analysis of Section 4 gives an  $O(n^7)$  upper bound. On the other hand, one can easily construct a simple polygon in which the number of maximal pairs is  $\Omega(n^2)$ . Any improvement on the  $O(n^7)$  upper bound would lead to an improvement in the running time of our algorithm.

Though in this paper we have focused on *exact* geodesic diameters only, an efficient algorithm for finding an *approximate* geodesic diameter would be also interesting. Notice that any point  $s \in \mathcal{P}$  and its farthest point  $t \in \mathcal{P}$  yield a 2-approximate diameter; that is,  $\text{diam}(\mathcal{P}) \leq 2 \max_{t \in \mathcal{P}} d(s, t)$  for any  $s \in \mathcal{P}$ . Also, based on a standard technique using a rectangular grid with a specified parameter  $0 < \epsilon < 1$ , one can obtain a  $(1 + \epsilon)$ -approximate diameter in  $O((\frac{n}{\epsilon^2} + \frac{n^2}{\epsilon}) \log n)$  time as follows. Scale  $\mathcal{P}$  so that  $\mathcal{P}$  can fit into a unit square, and partition  $\mathcal{P}$  with a grid of size  $\epsilon^{-1} \times \epsilon^{-1}$ . We define the set  $D$  as the point set that has the center of grid squares (that have a nonempty intersection with  $\mathcal{P}$ ) and intersection points between boundary edges and grid segments. We now can discretize the diameter problem by considering only geodesic distances between pairs of points of  $D$ . It turns out that, when  $\epsilon$  is small enough, the distance between any two points  $s$  and  $t$  in  $\mathcal{P}$  is within a  $1 + \epsilon$  factor of the distance between two points of  $D$ . Breaking the quadratic bound in  $n$  for the  $(1 + \epsilon)$ -approximate diameter seems a challenge at this stage.<sup>5</sup> We conclude by posing the following problem: for any or some  $0 < \epsilon < 1$ , is there any algorithm that finds a  $(1 + \epsilon)$ -approximate diametral pair in  $O(n^{2-\delta} \cdot \text{poly}(1/\epsilon))$  time for some positive  $\delta > 0$ ?

*Acknowledgements.* We thank Hee-Kap Ahn, Jiongxin Jin, Christian Knauer, and Joseph Mitchell for fruitful discussion, and Joseph O'Rourke for pointing out the reference [20].

## References

1. Agarwal, P.K., Aronov, B., O'Rourke, J., Schevon, C.A.: Star unfolding of a polytope with applications. *SIAM J. Comput.* 26(6), 1689–1713 (1997)
2. Aronov, B., Fortune, S., Wilfong, G.: The furthest-site geodesic Voronoi diagram. *Discrete Comput. Geom.* 9, 217–255 (1993)
3. Asano, T., Toussaint, G.: Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University (1985)
4. Bae, S.W., Chwa, K.-Y.: The geodesic furthest-site Voronoi diagram in a polygonal domain with holes. In: Proc. 25th Annu. Sympos. Comput. Geom. (SoCG), pp. 198–207 (2009)

---

<sup>5</sup> The idea of this approximation algorithm is due to Hee-Kap Ahn.

5. Bae, S.W., Korman, M., Okamoto, Y.: The geodesic diameter of polygonal domains. arXiv preprint (2010), arXiv:1001.0695
6. Bae, S.W., Okamoto, Y.: Querying two boundary points for shortest paths in a polygonal domain. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 1054–1063. Springer, Heidelberg (2009), arXiv:0911.5017
7. Chazelle, B.: A theorem on polygon cutting with applications. In: Proc. 23rd Annu. Sympos. Found. Comput. Sci. (FOCS), pp. 339–349 (1982)
8. Chiang, Y.-J., Mitchell, J.S.B.: Two-point Euclidean shortest path queries in the plane. In: Proc. 10th ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 215–224 (1999)
9. Cook IV, A.F., Wenk, C.: Shortest path problems on a polyhedral surface. In: Proc. 11th Internat. Sympos. Algo. Data Struct (WADS), pp. 156–167 (2009)
10. Guibas, L.J., Hershberger, J.: Optimal shortest path queries in a simple polygon. J. Comput. Syst. Sci. 39(2), 126–152 (1989)
11. Guo, H., Maheshwari, A., Sack, J.-R.: Shortest path queries in polygonal domains. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 200–211. Springer, Heidelberg (2008)
12. Hershberger, J., Suri, S.: Matrix searching with the shortest path metric. SIAM J. Comput. 26(6), 1612–1634 (1997)
13. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. SIAM J. Comput. 28(6), 2215–2256 (1999)
14. Mitchell, J.S.B.: Shortest paths among obstacles in the plane. Internat. J. Comput. Geom. Appl. 6(3), 309–331 (1996)
15. Mitchell, J.S.B.: Shortest paths and networks. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 27, pp. 607–641. CRC Press, Inc., Boca Raton (2004)
16. O'Rourke, J., Schevon, C.: Computing the geodesic diameter of a 3-polytope. In: Proc. 5th Annu. Sympos. Comput. Geom. (SoCG), pp. 370–379 (1989)
17. O'Rourke, J., Suri, S.: Polygons. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 26, pp. 583–606. CRC Press, Inc., Boca Raton (2004)
18. Pollack, R., Sharir, M., Rote, G.: Computing the geodesic center of a simple polygon. Discrete Comput. Geom. 4(6), 611–626 (1989)
19. Suri, S.: The all-geodesic-furthest neighbors problem for simple polygons. In: Proc. 3rd Annu. Sympos. Comput. Geom. (SoCG), pp. 64–75 (1987)
20. Zalgaller, V.A.: An isoperimetric problem for tetrahedra. Journal of Mathematical Sciences 140(4), 511–527 (2007)

# Polyhedral and Algorithmic Properties of Quantified Linear Programs\*

Ulf Lorenz, Alexander Martin, and Jan Wolf

Institute of Mathematics, Technische Universität Darmstadt, Germany

**Abstract.** Quantified linear programs (QLPs) are linear programs with variables being either existentially or universally quantified. The integer variant is PSPACE-complete, and the problem is similar to games like chess, where an existential and a universal player have to play a two-person-zero-sum game. At the same time, a QLP with  $n$  variables is a variant of a linear program living in  $\mathbb{R}^n$ , and it has strong similarities with multistage-stochastic programs with variable right-hand side. We show for the continuous case that the union of all winning policies of the existential player forms a polytope in  $\mathbb{R}^n$ , that its vertices are games of so called extremal strategies, and that these vertices can be encoded with polynomially many bits. The latter allows the conclusion that solving a QLP is in PSPACE. The hardness of the problem stays unknown.

## 1 Introduction

In the 1940s, linear programming arose as a mathematical planning model and rapidly found its daily use in many industries. However, integer programming, which was introduced in 1951, became dominant far later at the beginning the 1990s. Certainly, one reason for the delay of the integer programming success story stems from the fact that linear programming resides in the complexity class P, while integer programming is NP complete. Nowadays, we are able to solve very large mixed integer programs of practical size, but companies observe an increasing danger of disruptions, i.e., events occur which prevent companies from acting as planned. Therefore, there is a need for planning and deciding under uncertainty. Uncertainty, however, often pushes the complexity of traditional optimizations problems, which are in P or NP, to PSPACE. The quantified versions of linear integer programs cover the complexity class PSPACE. The relaxed versions, which we examine in this paper, additionally have remarkable polyhedral properties. The idea of our research is to explore the abilities of linear programming when applied to PSPACE-complete problems, similar as it was applied to NP-complete problems in the 1990s.

### 1.1 State-of-the-Art

For traditional deterministic optimization one assumes data for a given problem to be fixed and exactly known when the decisions have to be taken. However, data

---

\* Research partially supported by German Research Foundation (DFG) funded SFB 805.

are often afflicted with some kinds of uncertainties, and only estimations, maybe in form of probability distributions, are known. Examples are flight or travel times. Throughput-time, arrival times of externally produced goods, and scrap rate are subject to variations in production planning processes. One possibility to deal with these uncertainties is to aggregate a given probability distribution to a single estimated number. Then, the optimum concerning these estimated input data can be computed with the help of traditional optimization tools. In some fields of application, as e.g. the fleet assignment problem of airlines, this procedure was successfully established. In other fields, like production planning and control, this technique could not be successfully applied, although mathematical models do exist [17]. Because of the intuitive complexity of even some deterministic models, its stochastic counterparts are often not considered. Instead, safety stock is introduced, demanded quantities are overestimated, and buffers are oversized. However, this resource intensive behaviour is contrasted by a couple of publications within the last decade, indicating that stochastic problems are not necessarily out of scope [7,12,15,14,19].

## 1.2 Complexity and Algorithmic Issues

From complexity theory, we know that many interesting optimization problems under uncertainty are PSPACE complete [16]. The negative results from complexity theory seem discouraging to find efficient (i.e. polynomial time) algorithms to solve optimization problems that are PSPACE complete. As a consequence, algorithmics mostly deal with restricted problems which allow finding a polynomial-time algorithm, or at least approximations [10,6]. The self-restriction to *efficient algorithms* (in a formal sense), however, is related to worst-case instances. This does not reflect the fact that the most astonishing and admired successes of computing intelligence are modeled as NP-complete problems (mixed integer linear programs) and PSPACE-complete problems (computer games like Chess [4]).

Contrasting the efficiency debate, we can interpret a set of expressions, which can encode a PSPACE-complete problem, as a very powerful modeling language: more powerful than necessary to encode any NP-complete problem. The fact that it is not possible to find polynomial time algorithms for all problems that are encoded with the help of such a powerful modeling language, leads to the consequence that research for new solutions must be driven from the application-side or even from the instances-side, as e.g. presented in [12]. Relatively unexplored are the abilities of linear programming extensions in the PSPACE-complete world. In this context, Subramani introduced the notion of quantified linear programs [20,21].

## 1.3 Solution Issues

Prominent solution paradigms for optimization under uncertainty are Dynamic Programming [2], Sampling [11], the exploration of Markov-Chains [22], Robust Optimization [13], and Stochastic Programming [3,7,18,5]. Markov-Chains and

Dynamic Programming are often used for problems from complexity class P, but for more complex problems, other algorithms are often faster, like e.g. the Alphabeta algorithm for two-person zero-sum games [4]. Stochastic Programming can be essentially divided into two-stage and multi-stage problems with so-called recourse. A set of initial decisions are taken first, followed by a random event. After this, recourse decisions are taken, which allow to compensate for events that have been observed in previous stages. The multi-stage problem, accordingly consists of multiple stages, with a random event occurring between each stage. Such a problem can be transformed into a so-called deterministic equivalent program (DEP) and then be solved with the help of linear programming. An appropriate procedure for solving multi-stage stochastic programs are the nested decomposition procedure and its variants [3]. A multi-stage stochastic integer problem with only discrete probabilities is also called a game against nature, cf. [16,8].

In Section 2 we formally describe the QLP-problem and present an illustrating example. In Section 3, we analyse the polyhedral properties of the QLP problem. We make intensive use of the problem's ambiguity, being a two-person zero-sum game between two players on the one side, and being a convex multi-stage decision problem on the other one. In the last section, we present a solution algorithm running in polynomial space, based on Nested Benders decomposition. Up to now, only Fourier-Motzkin elimination based procedures were known, which consume double exponential time and space in the worst case.

## 2 The Problem Statement: Quantified Linear Programs

Within this paper, we intend to concentrate on quantified linear programs, as they were introduced by Subramani [21,20], who also presented first analytical results.

**Given:** A vector of variables  $x = (x_1, \dots, x_n) \in \mathbb{Q}^n$ , upper and lower bounds  $u \in \mathbb{Z}^n$  and  $l \in \mathbb{Z}^n$  with  $l_i \leq x_i \leq u_i$ , a matrix  $A \in \mathbb{Q}^{m \times n}$ , a vector  $b \in \mathbb{Q}^m$  and a quantifier string  $Q = (q_1, \dots, q_n) \in \{\forall, \exists\}^n$ , where the quantifier  $q_i$  belongs to the variable  $x_i$ , for all  $1 \leq i \leq n$ . We denote a QLP as  $[Q : Ax \leq b]$ . A maximal subset of  $Q(x)$ , which contains a consecutive sequence of quantifiers of the same type, is called a (variable-) *block*. A full assignment of numbers to the variables is interpreted as a game between an existential player (fixing the existential variables) and a universal player (fixing the universally quantified variables). The variables are set in consecutive order, as determined by the quantifier string. Consequently, we say that a player makes the move  $x_k = z$ , if he fixes the variable  $x_k$  to the value  $z$ . At each such move, the corresponding player knows the settings of  $x_1, \dots, x_{i-1}$  before setting  $x_i$ . If, at the end, all constraints of  $Ax \leq b$  hold, the existential player wins the game. Otherwise the universal player wins. For ease of explanation, we sometimes rewrite the system as  $[Q(x, y) : A \cdot \begin{pmatrix} x \\ y \end{pmatrix} \leq b]$ ,  $y$  representing the universal variables,  $x$  the existential ones. Without loss of generality, we can assume that universally quantified variables are between zero and one.

$$\forall x_1 \forall x_2 \exists x_3 : \begin{pmatrix} 10 & -4 & 2 \\ 10 & 4 & -2 \\ -10 & 4 & 1 \\ -10 & -4 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 4 \\ 12 \\ 8 \end{pmatrix}, \text{with } x_1 \in [-1,0], x_2 \in [0,1], x_3 \in [-2,2]$$

**Fig. 1.** QLP instance in dimension 3

**Question:** Is there an algorithm that fixes a variable  $x_i$  with the knowledge, how  $x_1, \dots, x_{i-1}$  have been set before, such that the existential player wins the game, no matter, how the universal player acts when he has to move?

In [20, 21], this problem occurs in two variants: a) all variables are discrete (QIP) and b) all variables are continuous (QLP). It has been shown that the restricted QLP problem with only one quantifier-change is either in P (when the quantifier string begins with existential quantifiers and ends with universal ones) or is coNP-complete (when the quantifier string begins with universal quantifiers and ends with existential ones) [21].

**Definition 1.** (*Strategy*) A strategy  $(V_x \dot{\cup} V_y, E, L)$  for the existential player  $S$  is a labeled tree of depth  $n$ , with  $V_x$  and  $V_y$  being two disjoint sets of nodes, and a set  $L \in \mathbb{Q}^{|E|}$  of edge-labels. Nodes from  $V_x$  are called existential nodes, nodes from  $V_y$  are called universal nodes. Each tree level  $i$  consists either of only existential nodes or of only universal nodes, depending on the quantifier  $q_i$  of the variable  $x_i$ . Each edge of the set  $E$ , leaving a tree-node of level  $i$ , represents an assignment for the variable  $x_i$ .  $l_i \in L$  describes the value of variable  $x_i$  on edge  $e_i \in E$ . Existential nodes have exactly one successor, universal nodes have two successors, one representing  $x_i = 0$  and  $x_i = 1$  the other<sup>1</sup>. (Generally, the coefficients of a universal variable  $y$  of an arbitrary QLP-instance can be scaled in a way such that  $y \in [0, 1]$ .) A strategy is called a winning strategy, if all paths from the root to a leaf represent a vector  $x$  such that  $Ax \leq b$ .

**Definition 2.** (*Policy*) A policy is an algorithm that fixes a variable  $x_i$ , being the  $i^{th}$  component of the vector  $x$ , with the knowledge, how  $x_1, \dots, x_{i-1}$  have been set before.

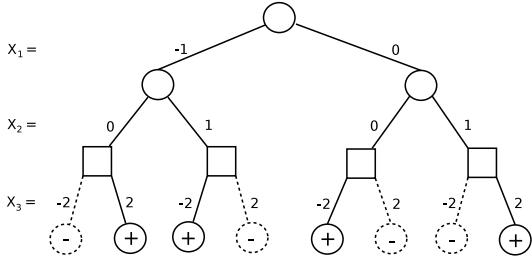
A 3-dimensional example of a QIP/QLP is shown in Fig. 1. If we restrict the variables to the integer bounds of their domains, we observe a winning strategy for the existential player as shown in Figure 2. In this example, a + in a tree leaf means that the existential player wins when this leaf is reached. A - marks a win for the universal player. Numbers at the edges mark the choices for variables. If the universal player moves to -1 and 0 (i.e., he sets  $x_1 = -1$  and  $x_2 = 0$ ) the existential player has to move to 2. If the universal player moves to -1 and 1, the existential player must set the variable  $x_3 = -2$  etc. We see that the existential player has to react carefully to former moves of the universal player.

<sup>1</sup> Later, in Lemma 2, we will show that by forcing universal variables to 0 and 1, we do not lose any generality.

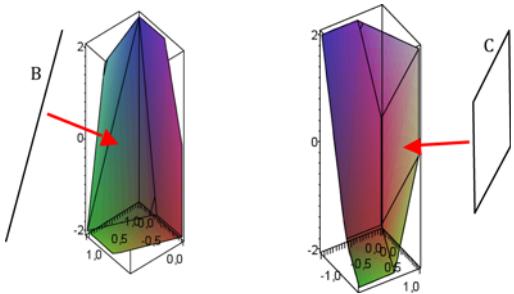
If we now relax the variables, allowing non-integral values for the corresponding domains, the resulting solution space of the corresponding QLP becomes polyhedral.

Moreover, the solution of the resulting problem when  $x_1 = -1$  is a line segment: B (cf. Fig. 3). On the left side of Figure 3, we can still see the corresponding partial strategy of the integer example in Figure 2. The strategy consists of the two end-points of B. On the right side of Figure 3, the same convex hull of the solution space is shown from another perspective.

We observe that the existential player has more freedom in the choice of  $x_3$ , when the universal player sets  $x_1 = 0$ . If  $x_1 = 0$ , the solution space of the rest-problem will just be the facet C. Interestingly, the line segment B and the facet C lie in the 3-dimensional vector space in such a way that there exists no linear description for a solution policy. The fact that a solution of a QLP has no linear description in general is not surprising, because the QLP problem is coNP-hard.



**Fig. 2.** The winning strategy (solid) for the integer QLP example in Fig. 1



**Fig. 3.** A visualization of the 3-d solution space of example of Fig. 1

### 3 QLP Analysis

In the following, we consider a QLP as defined in the previous section. Let existentially quantified variables be from  $\mathbb{Q}$  and universally quantified variables be from  $\{0, 1\}$ , and let  $n$  be the dimension of  $x$ .

**Definition 3.** (*convex combination of strategies*) Let  $S$  and  $S'$  be two strategies for the same QLP instance. We call  $S'' = (V, E, L'')$  a convex combination of  $S = (V, E, L)$  and  $S' = (V, E, L')$  if there is some  $\alpha \in [0, 1]$  such that for each edge  $e_i \in E$  it is  $l''_i = \alpha l'_i + (1 - \alpha)l_i$ . We write  $S'' = \alpha S' + (1 - \alpha)S$ .

**Lemma 1.** (*strategy-convexity*) If  $S$  and  $S'$  are winning strategies (for the existential player), and  $\alpha \in [0, 1]$ , also  $S'' = \alpha S' + (1 - \alpha)S$  will be a winning strategy.

*Proof. (sketched)* We apply conventional convexity to all paths of the strategies  $S, S'$  and  $S''$ .

Up to this point, we have restricted the choices of the universal player to 0 and 1 because we were interested in the existence of winning policies for the existential player. Now, we are interested in polyhedral properties. Therefore, the next step is to relax this integrality constraint. As a consequence, the solution of an existential strategy can no longer be described as a tree, and instead, we have to use the notion of a policy. This means, an existential variable, i.e. the component  $x_k$  of a solution vector  $x$  is determined as the value of a computable function:  $x_k = f_k(x_1, \dots, x_{k-1})$ .

**Lemma 2.** (*policy-convexity*) If  $P$  and  $P'$  are winning policies for the existential player, also  $P'' = \alpha P' + (1 - \alpha)P$  will be a winning policy for the existential player. Here,  $P''$  is an algorithm that picks the outputs of  $P$  and  $P'$  for arbitrary input and combines them accordingly.

*Proof. (sketched)* Conventional convexity arguments on paths of the policies  $P, P'$  and  $P''$  are combined with induction.

**Remark:** We can conclude that the existential player has a winning strategy against a two-value-restricted universal player if and only if he has a winning policy against an universal player without the restriction (also cf. [21]).

### 3.1 A Simple Solution Procedure for the QLP Problem ([21])

Let  $[Q : Ax \leq b]$  be a QLP instance. If the last variable  $x_n$  of  $x$  is existentially quantified, we can apply the Fourier-Motzkin elimination (FME) procedure. No matter how  $x_1, \dots, x_{n-1}$  are fixed, the remaining problem is a (parametrized) polytope. If, however,  $x_n$  is a universal quantified variable, there is a simpler algorithm to eliminate the last variable. Let us interpret the given instance as a game between the two players. Then fixing the last variable means that the universal player makes the last move. The existential player wants to achieve that  $Ax \leq b$ , but the universal player wants to hinder him, especially with his last move. If possible, the universal player will therefore choose a move which destroys at least one of the inequalities. Therefore, the existential player has a winning strategy if and only if he has a winning strategy against the  $(n-1)$ -dimensional problem which occurs if we assume the worst case universal choice, inequality by inequality. Let us call the procedure that eliminates existential variables with the help of the FME and universal variables with the help of the given gaming argument, the *QLP-elimination-procedure*. A detailed step by step proof for correctness can be found in [21].

**Definition 4.** (*extremal strategies (extremal policy)*) A strategy (policy)  $S$  is called extremal, if  $S$  cannot be described as a convex combination of two different strategies (policies). This includes that all sub-strategies / sub-policies, induced by the corresponding subtrees of  $S$ , are extremal, if and only if  $S$  is extremal itself.

### 3.2 The Solution Spaces of QLP Instances

In the following, we show that the solution space of a QLP instance, i.e., the union of all winning policies of the existential player, is a polytope. Certainly, there are several ways how to show the given fact, but we choose a quite elementary approach, because it makes the following detail visible: The vertices of the solution-space correspond to games of extremal winning strategies. Moreover, an extremal winning strategy is determined by a subset of polytope-vertices in  $\mathbb{R}^n$ . The proof proceeds in several steps.

**Lemma 3.** *The convex combination of all leaves of all extremal winning strategies against a  $\{0, 1\}$ -restricted universal player, is a polytope in  $\mathbb{R}^n$ .*

*Proof.* : Of course, the convex combination of the leaves of all winning strategies is a convex set. We now argue that the number of extreme points is finite. Let any QLP-instance  $I = [Q : Ax \leq b]$  be given, let  $S$  be an extremal winning strategy for the existential player. We eliminate the variables  $x_n, \dots, x_{n-k}$  such that only the first block of variables stays, with the help of the QLP-elimination-procedure. If the variables of this first block are  $m$  universally quantified variables, the first move of the universal player is an element of  $\{0, 1\}^m$ . If the variables of the first block are existential variables, the remaining problem is a conventional linear program with  $m$  variables. This remaining problem has finitely many extreme points, and  $S$  can only be extremal, if the first move of  $S$  corresponds to one of these extreme points, because otherwise  $S$  could be composed as a convex combination of other strategies. Therefore, there are only finitely many possible choices for the first player in any extremal strategy. Inductively, we see that there are only finitely many extremal strategies, and therefore there are only finitely many leaves within these strategies, and therefore the convex combination of all leaves of all extremal strategies contains only finitely many extreme points.

**Lemma 4.** *The convex combination of all outcomes of all possible winning policies is a polytope.*

*Proof.* Let  $P$  be a winning policy. If the universal player chooses moves only from  $\{0, 1\}$ , the winning policy is a winning strategy. This strategy is either an extremal strategy, or each game of the strategy can be expressed as the convex combination of two other strategies. Thus, every point, which is represented by an existential-player-winning game, is also a point in the convex hull of the leaves of all extremal winning strategies. If the universal player is allowed to choose his moves from  $\{0, \alpha, 1\}$ , the resulting game can be expressed as a convex combination of extremal strategies as well, for all  $\alpha \in [0, 1]$ . Again, every point, which is represented by such a game, where the existential player wins, is also a point in the convex hull of the leaves of all extremal winning strategies. Last but not least, it is obvious that there are no holes in the union of the winning policies. Let  $p$  be any point in the convex hull of the extremal winning strategies, and let us interpret this point as a game between the existential player and the universal player. Then, this game is part of a winning policy, because otherwise the point could not be part of the convex hull of the extremal winning strategies.

**Theorem 1.** *The solution space of all winning policies is a polytope (clear with Lemma 3 and Lemma 4)*

**Theorem 2.** *The vertices of a QLP-solution space can be described with polynomial many bits per vertex.* In order to prove this theorem, we present some preparations. Especially, we make use of a technique known from stochastic programming, where a deterministic equivalent program (DEP) is built for a multistage stochastic program. The general form of a multistage stochastic program with fixed recourse is as follows (cf. [3]), where the matrices  $T^i$  and the vectors  $h^i$  are possibly stochastic.

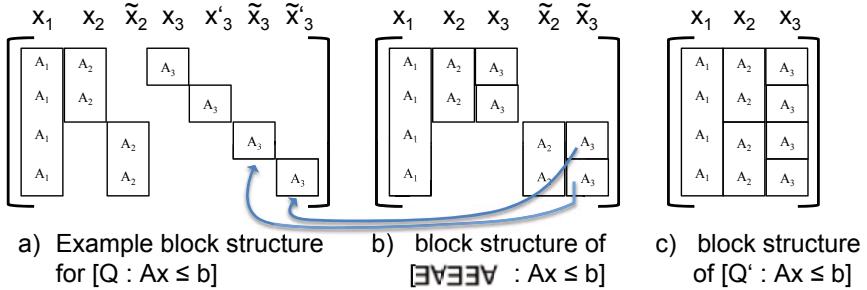
$$\begin{aligned} \min_{x^0} \quad & c^0 x^0 + E_{\xi^1} (\min_{x^1} c^1 x^1 + \dots + E_{\xi^H | \xi^1 \dots \xi^{H-1}} (\min_{x^H} c^H x^H)) \\ \text{s.t.} \quad & Ax^0 = b, \\ & \mathbf{T}^0 x^0 + W x^1 = h^1 \\ & \vdots \\ & \mathbf{T}^{H-1} x^{H-1} + W x^H = \mathbf{h}^H, \\ & l^t \leq x^t \leq u^t, \text{ for all } x \in \{0, \dots, H\} \end{aligned}$$

Usually, it is assumed that the stochastic elements are defined over a discrete probability space  $(\Xi, \sigma(\Xi), P)$ , where  $\Xi = \Xi^1 \otimes \dots \otimes \Xi^H$  is the support of the random data in each period, subject to  $\Xi^t = \{\xi_s^t = (T_s^t, W_s^t, h_s^t, c_s^t), s = 1, \dots, S^t\}$ . Because the probability space is discrete, a DEP can be defined by replicating the deterministic linear program for each possible scenario (possible path of events). Additionally, it is required that decisions do not depend on future events. We can construct a deterministic equivalent for QLP-instances, accordingly. Instead to encode the scenario tree, we encode the decision tree of the universal player into the deterministic equivalent by replicating the inequalities of the QLP-instance for each possible decision combination of the universal player. Let  $A_1, \dots, A_t$  be those blocks of columns of  $A$  that belong to the existential player and which are separated by the columns belonging to the universal player. If we bring the universal-player-columns to the right side, we will have created a multistage decision problem with variable right-hand side. Now, let  $[Q : Ax \leq b]$  be a QLP instance with  $t$  quantifier-changes, and, furthermore, let  $[Q' : Ax \leq b]$  be the QLP-instance where the sequence of quantifiers is modified to  $(\exists \dots \exists \forall \dots \forall)$ .

Figure 4a) shows the block structure of an example with four scenarios in block ladder matrix form. In the example, the universal player is allowed to make two moves, and there are two universally quantified variables and the quantifier-sequence is  $\exists \forall \exists \forall$ . Figure 4c) shows the block structure of the manipulated QLP  $Q'$ . It can be observed that the dimension of the deterministic equivalent in Fig. 4a) is higher, because the different choices of the existential player after a move of the universal player have to be considered. Figure 4a) and b) show, how the block structure of an instance changes, when the position of the last universal quantifier is changed.

**Definition 5.** *Let  $P \subset \mathbb{R}^n$  be a polytope and  $\phi$  and  $\nu$  be positive integers.*

- (i)  *$P$  has row-complexity (also called facet-complexity in [2]) of at most  $\phi$ , if there is a system of inequalities with rational coefficients  $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$*

Fig. 4. Block structures of  $Q$ ,  $Q'$ , and a medium instance

for some  $m \in \mathbb{N}$ , such that  $P$  is described with the help of a set of inequalities and the encoding length (i.e. the number of bits used) of each row is at most  $\phi$ . Each entry in  $A$  and  $b$  requires at least one bit.

- (ii)  $P$  has vertex-complexity of at most  $\nu$ , if there are finite sets  $V, E \subseteq \mathbb{IR}^n$ , such that  $P = \text{convexhull}(V)$  and the encoding length of each vector in  $V \cup E$  is at most  $\nu$ . We assume that  $\nu \geq n$ .

**Lemma 5.**  $\nu \leq 4\phi n^2$ , and  $\phi \leq 3\nu n^2$ . Proof and details can be found in [9].

**Lemma 6.** The vertex-complexity  $\nu'$  of the manipulated QLP-polytope  $P([Q' : Ax \leq b], A \in Q^{m \times n}, b \in Q^m)$  is in  $O(\phi n^6)$ ,  $\phi$  being the worst-case row-complexity of  $(A, b)$ .

*Proof.* Let the row-complexity of  $QLP' := [Q' : Ax + By \leq b]$  be  $\phi_{QLP'}$  with existential variables  $x$  and universal variables  $y$ , let the number of universal variables equal to  $k$ .  $QLP'$  has  $2^k$  scenarios. Its DEP, denoted by  $\text{detEq}(QLP')$ , or  $\text{detEq}'$  for short, lives in  $\mathbb{IR}^{n-k}$  and has row-complexity  $\phi_{\text{detEq}'} \leq \phi_{QLP'}$  because the DEP simply consists of a replication of  $A$  with various right hand sides  $b_i, i \in \{1, \dots, 2^k\}$ . Moreover, the encoding length  $\langle b_i \rangle$  is less or equal to  $\langle By \rangle$  for any  $y$ . Let  $D$  be the set of inequalities of  $\text{detEq}'$ . (q.e.d.)

As a consequence of Lemma 5, the vertex-complexity  $\nu_D \leq 4\phi_D n^2$ . Now we analyse the application of the following algorithm, which creates a game of an extremal winning strategy for the existential player and thus a vertex of an implicit QLP-polytope.

**Algorithm:** compute some QLP-vertex ( $Q$ )

- 1  $D^Q :=$  set of inequalities of deterministic equivalent of QLP-instance  $Q$
- 2 **for**  $i := 1$  **to**  $t-1$  // let  $t$  different blocks of existential variables in  $Q$  exist
- 3  $D_{\text{copy}} := D^Q$
- 4 eliminate the variables  $x_{(i+1,1)}, \dots, x_{(i+1,k(i))}, \dots, x_{(t,1)}, \dots, x_{(t,k(t))}$  of the variable blocks  $i+1, \dots, t$  in  $D_{\text{copy}}$ , with the help of Fourier-Motzkin elimination (FME), and store the new inequalities in  $D_i^Q$
- 5 choose the  $i^{\text{th}}$  move  $m_i$  of the universal player, i.e., delete those half of inequalities from  $D^Q$ , which do not belong to  $m_i$
- 6 solve  $D_1^Q \cup \dots \cup D_t^Q$

The elimination in line 4 does not effect the vertex-complexity of the given system in the remaining dimensions. Therefore, the row-complexity of  $D_i$  is  $\phi_{D_i} \leq 3\nu_D n^2 \leq 3 \cdot 4\phi_D n^2 \cdot n^2$ . The deletions in line 5 do not increase the row-complexity of  $D^Q$ . Therefore, the row-complexity of  $\phi_{D_1 \cup \dots \cup D_t} \leq 12\phi_D n^4$ . A last application of Lemma 5 leads us to  $\nu_{QLP'} = O(\nu_{D_1 \cup \dots \cup D_t}) = O(\phi_D n^6) = O(\phi_{QLP'} n^6)$ .

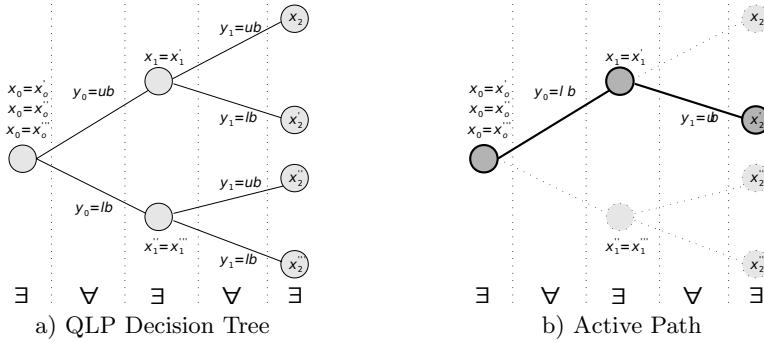
**Lemma 7.** *The vertex-complexity of the original QLP-polytope  $P([Q : Ax \leq b])$  is of order  $O(\phi_{QLP} n^6)$ .*

*Proof.* Let two QLP-instances  $Q, Q'$  be given, and let without loss of generality the quantifier string of  $Q$  end with the existential quantifiers  $(q_{j+1}, \dots, q_n)$ ,  $q_j$  being the universal quantifier with largest index. Let  $Q$  and  $Q'$  be equal except that in  $Q'$  the universal quantifier  $q_j$  is shifted to the end of the quantifier string, i.e.,  $Q' = (q_1, \dots, q_{j-1}, q_{j+1}, q_j)$ . Now, let us inspect the given algorithm, acting on  $Q'$  and  $Q$ . Assume that the sets  $D_1^{Q'} \cup \dots \cup D_t^{Q'}$  are already known. Moving the quantifier  $q_j$  from the last position to the position  $j$  in  $Q'$  has the following effect. In  $Q'$  the existential player must fix the variables  $x_{j+1}, \dots, x_n$  before the universal player decides  $x_j$ . In  $Q$ , this is not the case, and in order to express that the existential player has a choice after  $x_j$  is fixed, the variables  $x_{j+1}, \dots, x_n$  must be duplicated in the representation of the corresponding DEP, let us say to  $x'_{j+1}, \dots, x'_n$  (cf. Fig. 4). This increases the dimension of the deterministic equivalent and, at the same time, partitions the inequalities belonging to different scenarios, one induced by  $x_j = 0$  and the other induced by  $x_j = 1$ , whereas  $x_j$  is a universal variable. Thus, due to the nature of the Fourier-Motzkin elimination, every new inequality in  $D_i^Q$ , processed by line 4 of the above algorithm, is also contained in  $D_i^{Q'}$  as well. Inductively, it follows that  $D_1^Q \cup \dots \cup D_t^Q$  is a subset of  $D_1^{Q'} \cup \dots \cup D_t^{Q'}$ , and therefore with  $Q' := QLP'$  of Lemma 6:  $\phi_{D_1^Q \cup \dots \cup D_t^Q} \leq \phi_{D_1^{Q'} \cup \dots \cup D_t^{Q'}}$  and thus  $\nu_{QLP} = O(\phi_{D_1^Q \cup \dots \cup D_t^Q} n^6) = O(\phi_{D_1^{Q'} \cup \dots \cup D_t^{Q'}} n^6) = O(\phi_{QLP} n^6)$ . Thus, Theorem 2 is proven as well.

## 4 Algorithms Using Polynomial Space

The task of the following algorithm is to find suitable first-stage variables or the information that no solution exists. Because single paths of extremal strategies can be expressed with  $O(\phi n^6)$  bits, it is possible to find a winning strategy, if it exists, with the help of a depth first search (DFS). On each level of the DFS, one of the variables is fixed, and the choices on each level are all numbers of  $\mathbb{Q}$  that can be encoded with  $O(\phi n^6)$  many bits. This technique can be applied to the integer version of the QLP problem and to the mixed integer version, as well.

Another, more practical way, to organize the solution process in polynomial space is based upon the nested decomposition algorithm (NDA). Nested decomposition is based on the observation that the corresponding implicit scenario tree can be transformed to a DEP that can be solved by a recursive application of Benders decomposition. To solve a QLP with this algorithm, we construct a DEP as well. However, instead of encoding the scenario tree, we encode the decision tree of the universal player into the deterministic equivalent by replicating the inequalities of the QLP-instance for each possible decision combination of the



**Fig. 5.** QLP Decision Tree and Active Path

universal player. A mapping from a QLP with the quantifier sequence  $\exists \forall \exists \forall \exists$  to a decision tree is shown in Figure 5a). Due to the nature of QLPs, we can use the following simplifying assumptions for our variant of nested benders. First of all, since QLPs do not have an objective function, we only need to pass back Bender's feasibility cuts to parent nodes. Furthermore, because we only need to consider the upper and lower bound of each universally quantified variable, each node only has two successors. Since we only need a variable allocation for the first-stage variables, the only type of information that has to be stored in the tree, are feasibility cuts and the proposal for the current  $x$ -vector allocation, which then belong to a specific path as depicted in Figure 5b). As described in [1] we traverse the tree *depth-first* instead *breadth-first*. Thus, only nodes that are contained in the current path are *active nodes* and must therefore store cuts from their successors. When a node becomes a *passive node*, all cuts that are stored at the node can be removed, and recomputed later if needed.

To solve QLPs with polynomial space, we propose the following modification of the (implicit) decision tree, whose stages until now were determined by the quantifier changes, respectively the different quantifier blocks of existentially quantified variables. We change to a formulation where each stage consists of nodes where each master problem only corresponds to a single existential variable. The resulting decision tree now consists of nodes that either have two outgoing arcs in the case when the existentially quantified variable is the last one in the respective quantifier block, or one outgoing arc for inner variables of a block, since the successor node is not affected by an universal variable. The feasibility cuts that are passed back to the nodes at higher stages can be viewed as variable bounds. This implies that using simple preprocessing techniques each node of the current path will consist of at most two constraints, an upper and a lower bound. The polynomial space complexity directly follows.

## References

1. Altenstedt, F.: Memory consumption versus computational time in nested benders decomposition for stochastic linear programming. Tech. Rep., Chalmers University, Goteborg (2003)

2. Bellmann, R.: Dynamic programming. Princeton University Press, Princeton (1957)
3. Birge, J.R., Louveaux, F.: Intro. to Stochastic Programming. Springer, Heidelberg (1997)
4. Donninger, C., Kure, A., Lorenz, U.: Parallel brutus: The first distributed, fpga accelerated chess program. In: Proc. of 18th International Parallel & Distributed Processing Symposium (IPDPS), Santa Fe. IEEE Computer Society, Los Alamitos (2004)
5. Dyer, M.E., Stougie, L.: Computational complexity of stochastic programming problems. *Math. Program.* 106(3), 423–432 (2006)
6. Eisenbrand, F., Rothvoß, T.: A ptas for static priority real-time scheduling with resource augmentation. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 246–257. Springer, Heidelberg (2008)
7. Engell, S., Märkert, A.M., Sand, G., Schultz, R.: Aggregated scheduling of a multiproduct batch plant by two-stage stochastic integer programming. *Optimiz. and Engineering* 5 (2004)
8. Grothklags, S., Lorenz, U., Monien, B.: From state-of-the-art static fleet assignment to flexible stochastic planning of the future. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS, vol. 5515, pp. 140–165. Springer, Heidelberg (2009)
9. Grötschel, M., Lovasz, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd edn. Springer, Heidelberg (1993)
10. Hochbaum, D.S.: Approximation Algorithms for NP-hard Problems. PWS (1997)
11. Kleywegt, A.J., Shapiro, A., Homem-De-Mello, T.: The sample average approximation method for stochastic discrete optimization. *SIAM Jour. of Opt.*, 479–502 (2001)
12. König, F.G., Lübbcke, M.E., Möhring, R.H., Schäfer, G., Spenke, I.: Solutions to real-world instances of pspace-complete stacking. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 729–740. Springer, Heidelberg (2007)
13. Liebchen, C., Lübbcke, M.E., Möhring, R.H., Stiller, S.: The concept of recoverable robustness, linear programming recovery, and railway applications. In: Robust and online large-scale optimization, pp. 1–27 (2009)
14. Megow, N., Vredeveld, T.: Approximation results for preemptive stochastic online scheduling. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 516–527. Springer, Heidelberg (2006)
15. Möhring, R.H., Schulz, A.S., Uetz, M.: Approximation in stochastic scheduling: The power of lp-based priority schedules. *Journal of ACM* 46(6), 924–942 (1999)
16. Papadimitriou, C.H.: Games against nature. *J. of Comp. and Sys. Sc.*, 288–301 (1985)
17. Pochet, Y., Wolsey, L.A.: Production planning by mixed integer programming. Springer Series in Operations Research and Financial Engineering. Springer, New York (2006)
18. Schultz, R.: Stochastic programming with integer variables. *Math. Progr.* 97, 285–309 (2003)
19. Shmoys, D.B., Swamy, C.: Stochastic optimization is (almost) as easy as deterministic optimization. In: Proc. FOCS 2004, pp. 228–237 (2004)
20. Subramani, K.: Analyzing selected quantified integer programs. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 342–356. Springer, Heidelberg (2004)
21. Subramani, K.: On a decision procedure for quantified linear programs. *Annals of Mathematics and Artificial Intelligence* 51(1), 55–77 (2007)
22. Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: Efficient decision algorithms for probabilistic simulations. *Logical Methods in Computer Science* 4(4) (2008)

# Approximating Parameterized Convex Optimization Problems\*

Joachim Giesen<sup>1</sup>, Martin Jaggi<sup>2</sup>, and Sören Laue<sup>1</sup>

<sup>1</sup> Friedrich-Schiller-Universität Jena, Germany

<sup>2</sup> ETH Zürich, Switzerland

**Abstract.** We extend Clarkson’s framework by considering parameterized convex optimization problems over the unit simplex, that depend on one parameter. We provide a simple and efficient scheme for maintaining an  $\varepsilon$ -approximate solution (and a corresponding  $\varepsilon$ -coreset) along the entire parameter path. We prove correctness and optimality of the method. Practically relevant instances of the abstract parameterized optimization problem are for example regularization paths of support vector machines, multiple kernel learning, and minimum enclosing balls of moving points.

## 1 Introduction

We study convex optimization problems over the unit simplex that are parameterized by a single parameter. We are interested in optimal solutions of the optimization problem for all parameter values, i.e., the whole solution path in the parameter. Since the complexity of the exact solution path might be exponential in the size of the input [7], we consider approximate solutions with an approximation guarantee for all parameter values, i.e., approximate solutions along the whole path. We provide a general framework for computing approximate solution paths that has the following properties:

- (1) *Generality.* Apart from being specified over the unit simplex, we hardly make any assumptions on the optimization problem under consideration. Hence, the framework can be applied in many different situations.
- (2) *Simplicity.* The basic idea behind the framework is a very simple continuity argument.
- (3) *Practicality.* It has been shown that our framework works well for real world problems.
- (4) *Efficiency.* Although the framework is very simple it still gives improved theoretical bounds for known problems.
- (5) *Optimality.* We show that it is the best possible one can do up to a constant factor.

---

\* This work has been supported by the DFG under grant GI-711/3-1, by a Google Research Award, and by the Swiss National Science Foundation (SNF Project 20PA21-121957).

Let us explain the different aspects in more detail.

*Generality:* We build on the general primal-dual approximation criterion that has been introduced by Clarkson in his coresets framework [5] for convex optimization problems over the unit simplex. Among the many problems that fit into Clarkson's framework are for example the smallest enclosing ball problem, polytope distance problems, and binary classification support vector machines. For many of these problems parameterized versions are known, e.g., the smallest enclosing ball problem for points that move with time, or soft margin support vector machines that trade-off a regularization term and a loss term in the objective function of the corresponding optimization problem.

*Simplicity:* The basic algorithmic idea behind our framework is computing at some parameter value an approximate solution whose approximation guarantee holds for some sub-interval of the problem path. This solution is then updated at the boundary of the sub-interval to a better approximation that remains a good approximation for a consecutive sub-interval. For computing the initial approximation and the updates from previous approximations, any arbitrary (possibly problem specific) algorithm can be used, that ideally can be started from the previous solution (warm start). We provide a simple lemma that allows to bound the number of necessary parameter sub-intervals for a prescribed approximation quality. For interesting problems, the lemma also implies the existence of small coresets that are valid for the entire parameter path.

*Practicality:* Our work is motivated by several problems from machine learning and computational geometry that fit into the described framework, in particular, support vector machines and related classification methods, multiple kernel learning [3], and the smallest enclosing ball problem [4]. We have implemented the proposed algorithms and applied them to choose the optimal regularization parameter for a support vector machine, and to find the best combination of two kernels which is a special case of the multiple kernel learning problem.

*Efficiency:* Our framework gives a path complexity of  $O(\frac{1}{\varepsilon})$ , meaning that an  $\varepsilon$ -approximate solution needs to be updated only  $O(\frac{1}{\varepsilon})$  times along the whole path. This positively contrasts the complexity of exact solution paths.

*Optimality:* We provide lower bounds that show that one cannot do better, i.e., there exist examples where one needs essentially at least one fourth as many sub-intervals as predicted by our method.

*Related Work.* Many of the aforementioned problems have been recently studied intensively, especially machine learning methods such as computing exact solution paths in the context of support vector machines and related problems [10,12,13,18]. But exact algorithms can be fairly slow compared to approximate methods. To make things even worse, the complexity of exact solution paths can be very large, e.g., it can grow exponentially in the input size as it has been shown for support vector machines with  $\ell_1$ -loss term [7]. Hence, approximation algorithms have become popular also for the case of solution paths lately, see e.g. [6]. However, to the best of our knowledge, so far no approximation quality guarantees along the path could be given for any of these existing algorithms.

## 2 Clarkson's Framework

In [5] Clarkson considers convex optimization problems of the form

$$\begin{aligned} & \min_x f(x) \\ \text{s.t. } & x \in S_n \end{aligned} \tag{1}$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is convex and continuously differentiable, and  $S_n$  is the unit simplex, i.e.,  $S_n$  is the convex hull of the standard basis vectors of  $\mathbb{R}^n$ . We additionally assume that the function  $f$  is non-negative on  $S_n$ . A point  $x \in \mathbb{R}^n$  is called a feasible solution, if  $x \in S_n$ .

The Lagrangian dual of Problem 1 (sometimes also called Wolfe dual) is given by the unconstrained problem

$$\max_x \omega(x), \text{ where } \omega(x) := f(x) + \min_i (\nabla f(x))_i - x^T \nabla f(x).$$

In this framework Clarkson studies approximating the optimal solution. His measure of approximation quality is (up to a multiplicative positive constant) the primal-dual gap

$$g(x) := f(x) - \omega(x) = x^T \nabla f(x) - \min_i (\nabla f(x))_i.$$

Note that convexity of  $f$  implies the weak duality condition  $f(\hat{x}) \geq \omega(x)$ , for the optimal solution  $\hat{x} \in S_n$  of the primal problem and any feasible solution  $x$ , which in turn implies non-negativity of the primal-dual gap, i.e.,  $g(x) \geq 0$  for all feasible  $x$ , see [5]. A feasible solution  $x$  is an  $\varepsilon$ -approximation to Problem 1 if

$$g(x) \leq \varepsilon f(x).$$

Sometimes an  $\varepsilon$ -approximation is defined more restrictively as  $g(x) \leq \varepsilon f(\hat{x})$ , relative to the optimal value  $f(\hat{x})$  of the primal optimization problem. Note that this can directly be obtained from our slightly weaker definition by setting  $\varepsilon$  in the definition of an  $\varepsilon$ -approximation to  $\varepsilon' := \frac{\varepsilon}{1+\varepsilon}$ , because  $g(x) \leq \frac{\varepsilon}{1+\varepsilon} f(x) \Leftrightarrow (1+\varepsilon)(f(x) - \omega(x)) \leq \varepsilon f(x) \Leftrightarrow g(x) \leq \varepsilon \omega(x) \leq \varepsilon f(\hat{x})$ . A subset  $C \subseteq [n]$  is called an  $\varepsilon$ -coreset, if there exists an  $\varepsilon$ -approximation  $x$  to Problem 1 with  $x_i = 0, \forall i \in [n] \setminus C$ .

The case of *maximizing* a concave, continuously differentiable, non-negative function  $f$  over the unit simplex  $S_n$  can be treated analogously. The Lagrangian dual problem is given as

$$\min_x \omega(x), \text{ where } \omega(x) := f(x) + \max_i (\nabla f(x))_i - x^T \nabla f(x),$$

and the duality gap is  $g(x) := \omega(x) - f(x) = \max_i (\nabla f(x))_i - x^T \nabla f(x)$ . Again,  $x \in S_n$  is an  $\varepsilon$ -approximation if  $g(x) \leq \varepsilon f(x)$  (which immediately implies  $g(x) \leq \varepsilon f(\hat{x})$  for the optimal solution  $\hat{x}$  of the primal maximization problem).

Clarkson [5] showed that  $\varepsilon$ -coresets of size  $\lceil \frac{2C_f}{\varepsilon} \rceil$  do always exist, and that the sparse greedy algorithm [5, Algorithm 1.1] obtains an  $\varepsilon$ -approximation after at most  $2 \lceil \frac{4C_f}{\varepsilon} \rceil$  many steps. Here  $C_f$  is an absolute constant describing the “non-linearity” or “curvature” of the function  $f$ .

### 3 Optimizing Parameterized Functions

We extend Clarkson's framework and consider parameterized families of functions  $f_t(x) = f(x; t) : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$  that are convex and continuously differentiable in  $x$  and parameterized by  $t \in \mathbb{R}$ , i.e., we consider the following families of minimization problems

$$\begin{aligned} & \min_x f_t(x) \\ \text{s.t. } & x \in S_n \end{aligned} \tag{2}$$

Again, we assume  $f_t(x) \geq 0$  for all  $x \in S_n$  and  $t \in \mathbb{R}$ .

The following simple lemma is at the core of our discussion and characterizes how we can change the parameter  $t$  such that a given  $\frac{\varepsilon}{\gamma}$ -approximate solution  $x$  (for  $\gamma > 1$ ) at  $t$  stays an  $\varepsilon$ -approximate solution.

**Lemma 1.** *Let  $x \in S_n$  be an  $\frac{\varepsilon}{\gamma}$ -approximation to Problem 2 for some fixed parameter value  $t$ , and for some  $\gamma > 1$ . Then for all  $t' \in \mathbb{R}$  that satisfy*

$$\begin{aligned} & x^T \nabla(f_{t'}(x) - f_t(x)) - (\nabla(f_{t'}(x) - f_t(x)))_i - \varepsilon(f_{t'}(x) - f_t(x)) \\ & \leq \varepsilon \left(1 - \frac{1}{\gamma}\right) f_t(x), \quad \forall i \in [n], \end{aligned} \tag{3}$$

*the solution  $x$  is still an  $\varepsilon$ -approximation to Problem 2 at the changed parameter value  $t'$ .*

*Proof.* We have to show that  $g(x; t') \leq \varepsilon f_{t'}(x)$ , or in other words that

$$x^T \nabla f_{t'}(x) - (\nabla f_{t'}(x))_i \leq \varepsilon f_{t'}(x)$$

holds for all components  $i$ . We add to the Inequalities 3 for all components  $i$  the inequalities stating that  $x$  is an  $\frac{\varepsilon}{\gamma}$ -approximate solution at value  $t$ , i.e.

$$x^T \nabla f_t(x) - (\nabla f_t(x))_i \leq \frac{\varepsilon}{\gamma} f_t(x).$$

This gives for all  $i \in [n]$

$$x^T \nabla f_{t'}(x) - (\nabla f_{t'}(x))_i - \varepsilon(f_{t'}(x) - f_t(x)) \leq \varepsilon f_t(x),$$

which simplifies to the claimed bound  $x^T \nabla f_{t'}(x) - (\nabla f_{t'}(x))_i \leq \varepsilon f_{t'}(x)$ .  $\square$

The analogue of Lemma 1 for maximizing a concave function over the unit simplex is the following lemma whose proof follows along the same lines:

**Lemma 2.** *Let  $x \in S_n$  be an  $\frac{\varepsilon}{\gamma}$ -approximation to the maximization problem  $\max_{x \in S_n} f_t(x)$  at parameter value  $t$ , for some  $\gamma > 1$ . Here  $f_t(x)$  is a parameterized family of concave, continuously differentiable functions in  $x$  that are non-negative on  $S_n$ . Then for all  $t' \in \mathbb{R}$  that satisfy*

$$\begin{aligned} & (\nabla(f_{t'}(x) - f_t(x)))_i - x^T \nabla(f_{t'}(x) - f_t(x)) - \varepsilon(f_{t'}(x) - f_t(x)) \\ & \leq \varepsilon \left(1 - \frac{1}{\gamma}\right) f_t(x), \quad \forall i \in [n], \end{aligned} \tag{4}$$

*the solution  $x$  is still an  $\varepsilon$ -approximation at the changed parameter value  $t'$ .*

We define the  $\varepsilon$ -approximation path complexity for Problem 2 as the minimum number of sub-intervals of the parameter interval  $\mathbb{R}$  such that for each sub-interval there is a single solution of Problem 2 which is an  $\varepsilon$ -approximation for that entire sub-interval.

Lemma 1 and 2 imply upper bounds on the path complexity. Next, we will show that these upper bounds are tight up to a multiplicative factor of  $4 + 2\varepsilon$ .

### 3.1 Lower Bound

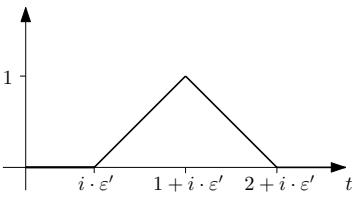
To see that the approximate path complexity bounds we get from Lemma 2 are optimal consider the following parameterized optimization problem:

$$\begin{aligned} \max_x f_t(x) &:= x^T f(t) \\ \text{s.t. } x &\in S_n \end{aligned} \tag{5}$$

where  $f(t) = (f_0(t), \dots, f_k(t))$  is a vector of functions and  $f_i(t)$  is defined as follows

$$f_i(t) = \begin{cases} 0, & \text{for } t < i\varepsilon' \\ t - i\varepsilon', & \text{for } i\varepsilon' \leq t < 1 + i\varepsilon' \\ -t + 2 + i\varepsilon', & \text{for } 1 + i\varepsilon' \leq t \leq 2 + i\varepsilon' \\ 0, & \text{for } 2 + i\varepsilon' < t \end{cases}$$

for some arbitrary fixed  $\varepsilon' > 0$  and  $n \geq 1/\varepsilon'$ . See Figure 1 for an illustration of the function  $f_i(t)$ .



**Fig. 1.** Function  $f_i(t)$

Each of the  $f_i(t)$  attains its maximum 1 at  $t = 1 + i\varepsilon'$ . Since  $f_t(x)$  is linear in  $x$  it is hence concave in  $x$  for every fixed  $t$ . Hence, it is an instance of Problem 2. Let us now consider the interval  $t \in [1, 2]$ . In this interval consider the points  $t_i := 1 + i\varepsilon'$ , for  $i = 0, \dots, \lfloor 1/\varepsilon' \rfloor$ . At each of these points it holds that  $f_i(t_i) = 1$  and all the other  $f_j(t_i) \leq 1 - \varepsilon'$  when  $j \neq i$ . Hence, the value of the optimal solution to Problem 5 at parameter value  $t_i$  is 1, and it is attained at  $x = e_i$ , where  $e_i$  is the  $i$ -th standard basis vector.

Furthermore, for all other  $x \in S_n$  that have an entry at the coordinate position  $i$  that is at most  $1/2$  it holds that  $f_{t_i}(x) \leq 1 - \varepsilon'/2$ .

Hence, in order to have an  $\varepsilon$ -approximation for  $\varepsilon < \varepsilon'/2$  the approximate solution  $x$  needs to have an entry of more than  $1/2$  at the  $i$ -th coordinate position. Since all entries of  $x$  sum up to 1, all the other entries are strictly less than  $1/2$  and hence this solution cannot be an  $\varepsilon$ -approximation for any other parameter value  $t = t_j$  with  $j \neq i$ . Thus, for all values of  $t \in [1, 2]$  one needs at least  $1/\varepsilon'$  different solutions for any  $\varepsilon < \varepsilon'/2$ .

Choosing  $\varepsilon'$  arbitrarily close to  $2\varepsilon$  this implies that one needs at least  $\frac{1}{2\varepsilon} - 1$  different solutions to cover the whole path for  $t \in [1, 2]$ .

Lemma 2 gives an upper bound of  $\frac{2+\varepsilon}{\varepsilon} \frac{\gamma}{\gamma-1} = (\frac{2}{\varepsilon} + 1) \frac{\gamma}{\gamma-1}$  different solutions, since  $\nabla f_t(x) = f(t) = (f_i(t))_{i \in [n]}$  and  $\left| \frac{\partial f_i}{\partial t} \right| \leq 1$ ,  $(\nabla(f_{t'}(x) - f_t(x)))_i \leq |t' - t|$ .

Hence, this is optimal up to a factor of  $4 + 2\varepsilon$ . Indeed, also the dependence on the problem specific constants in Lemma 2 is tight: ‘contracting’ the functions  $f_i(t)$  along the  $t$ -direction increases the Lipschitz constant of  $(\nabla f_t(x))_i$ , which can be shown to be an upper bound on the problem specific constants in Lemma 2.

### 3.2 The Weighted Sum of Two Convex Functions

We are particularly interested in a special case of Problem 2. For any two convex, continuously differentiable functions  $f^{(1)}, f^{(2)} : \mathbb{R}^n \rightarrow \mathbb{R}$  that are non-negative on  $S_n$ , we consider the weighted sum  $f_t(x) := f^{(1)}(x) + t f^{(2)}(x)$  for a real parameter  $t \geq 0$ . The parameterized optimization Problem 2 in this case becomes:

$$\begin{aligned} & \min_x f^{(1)}(x) + t f^{(2)}(x) \\ & \text{s.t. } x \in S_n \end{aligned} \tag{6}$$

For this optimization problem we have the following corollary of Lemma 1:

**Corollary 1.** *Let  $x \in S_n$  be an  $\frac{\varepsilon}{\gamma}$ -approximate solution to Problem 6 for some fixed parameter value  $t \geq 0$ , and for some  $\gamma > 1$ . Then for all  $t' \geq 0$  that satisfy*

$$(t' - t) \left( x^T \nabla f^{(2)}(x) - (\nabla f^{(2)}(x))_i - \varepsilon f^{(2)}(x) \right) \leq \varepsilon \left( 1 - \frac{1}{\gamma} \right) f_t(x), \quad \forall i \in [n] \tag{7}$$

*solution  $x$  is an  $\varepsilon$ -approximate solution to Problem 6 at the parameter value  $t'$ .*

*Proof.* Follows directly from Lemma 1, and  $f_{t'}(x) - f_t(x) = (t' - t)f^{(2)}(x)$ .  $\square$

This allows us to determine the entire interval of admissible parameter values  $t'$  such that an  $\frac{\varepsilon}{\gamma}$ -approximate solution at  $t$  is still an  $\varepsilon$ -approximate solution at  $t'$ .

**Corollary 2.** *Let  $x$  be an  $\frac{\varepsilon}{\gamma}$ -approximate solution to the Problem 6 for some fixed parameter value  $t \geq 0$ , for some  $\gamma > 1$ , and let*

$$\begin{aligned} u &:= x^T \nabla f^{(2)}(x) - \min_i (\nabla f^{(2)}(x))_i - \varepsilon f^{(2)}(x) \\ l &:= x^T \nabla f^{(2)}(x) - \max_i (\nabla f^{(2)}(x))_i - \varepsilon f^{(2)}(x), \end{aligned}$$

*then  $x$  remains an  $\varepsilon$ -approximate solution for all  $0 \leq t' = t + \delta$  for the following values of  $\delta$ :*

(i) *If  $l < 0$  or  $0 < u$ , then the respective admissible values for  $\delta$  are*

$$\varepsilon \left( 1 - \frac{1}{\gamma} \right) \frac{f_t(x)}{l} \leq \delta \leq \varepsilon \left( 1 - \frac{1}{\gamma} \right) \frac{f_t(x)}{u}$$

(ii) *If  $u \leq 0$ , then  $\delta$  (and thus  $t'$ ) can become arbitrarily large.*

(iii) *If  $l \geq 0$ , then  $\delta$  can become as small as  $-t$ , and thus  $t'$  can become 0.*

Note that the  $\varepsilon$ -approximation path complexity for Problem 6 for a given value of  $\gamma > 1$  can be upper bounded by the minimum number of points  $t_j \geq 0$  such that the admissible intervals of  $\frac{\varepsilon}{\gamma}$ -approximate solutions  $x_j$  at  $t_j$  cover the whole parameter interval  $[0, \infty)$ .

Corollary 2 immediately suggests two variants of an algorithmic framework (forward- and backwards version) maintaining  $\varepsilon$ -approximate solutions over the entire parameter interval or in other words, tracking a guaranteed  $\varepsilon$ -approximate solution path. Note that as the internal optimizer, any arbitrary approximation algorithm can be used here, as long as it provides an approximation guarantee on the relative primal-dual gap. For example the standard Frank-Wolfe algorithm [5, Algorithm 1.1] is particularly suitable as its resulting coresets solutions are also sparse. The forward version is depicted in Algorithm 1.

**Algorithm 1.** APPROXIMATIONPATH—FORWARDVERSION  $(\varepsilon, \gamma, t_{\min}, t_{\max})$

```

1  compute an  $\frac{\varepsilon}{\gamma}$ -approximation  $x$  for  $f_t(x)$  at  $t := t_{\min}$  using a standard optimizer.
2  do
3     $u := x^T \nabla f^{(2)}(x) - \min_i (\nabla f^{(2)}(x))_i - \varepsilon f^{(2)}(x)$ 
4    if  $u > 0$  then
5       $\delta := \varepsilon \left(1 - \frac{1}{\gamma}\right) \frac{f_t(x)}{u} > 0$ 
6       $t := t + \delta$ 
7      improve the (now still  $\varepsilon$ -approximate) solution  $x$  for  $f_t(x)$  to an at least
        $\frac{\varepsilon}{\gamma}$ -approximate solution by applying steps of any standard optimizer.
8    else
9       $t := t_{\max}$ 
10   while  $t < t_{\max}$ 
```

## 4 Applications

Special cases of Problem 6 or the more general Problem 2 have applications in computational geometry and machine learning. In the following we discuss two applications in more detail, namely, a parameterized polytope distance problem as an application of the special case of Problem 6, and smallest enclosing balls of linearly moving points as an application of the general Problem 2. The polytope distance problem itself can be specialized for computing regularization paths of support vector machines (SVMs), and for kernel learning in the context of SVMs.

### 4.1 A Parameterized Polytope Distance Problem

In the setting of Section 3.2 we consider the case  $f^{(1)}(x) := x^T K^{(1)}x$  and  $f^{(2)}(x) := x^T K^{(2)}x$ , for two positive semi-definite matrices  $K^{(1)}, K^{(2)} \in \mathbb{R}^{n \times n}$ , i.e.,

$$\begin{aligned} \min_x f^{(1)}(x) + t f^{(2)}(x) &= x^T (K^{(1)} + t K^{(2)}) x \\ \text{s.t. } x &\in S_n . \end{aligned} \tag{8}$$

The geometric interpretation of this problem is as follows: let  $A(t) \in \mathbb{R}^{n \times r}$ ,  $r \leq n$ , be the unique matrix such  $A(t)^T A(t) = K^{(1)} + tK^{(2)}$  (Cholesky decomposition). The solution  $\hat{x}$  to Problem 8 is the point in the convex hull of the column vectors of the matrix  $A(t)$  that is closest to the origin. Hence, Problem 8 is a parameterized polytope distance problem. For the geometric interpretation of an  $\varepsilon$ -approximation in this context we refer to [9]. In the following we will always assume that the two original polytope distance problems are separable, i.e.  $\min_{x \in S_n} x^T K^{(1)} x > 0$  and  $\min_{x \in S_n} x^T K^{(2)} x > 0$ .

For this parameterized problem, the two quantities  $u$  and  $l$  that determine the admissible parameter intervals in Corollary 2 and the step size in both approximate path algorithms take the simpler form

$$u = (2 - \varepsilon)x^T K^{(2)} x - 2 \min_i (K^{(2)} x)_i \text{ and } l = (2 - \varepsilon)x^T K^{(2)} x - 2 \max_i (K^{(2)} x)_i,$$

since  $\nabla f^{(2)}(x) = 2K^{(2)}x$ . We now use the following lemma to bound the path complexity for instances of Problem 8.

**Lemma 3.** *Let  $0 < \varepsilon \leq 1$  and  $\gamma > 1$ . Then for any parameter  $t \geq 0$ , the length of the interval  $[t - \delta, t]$  with  $\delta > 0$ , on which an  $\frac{\varepsilon}{\gamma}$ -approximate solution  $x$  to Problem 8 at parameter value  $t$  remains an  $\varepsilon$ -approximation, is at least*

$$l_f(\varepsilon, \gamma) := \frac{\varepsilon}{2} \left(1 - \frac{1}{\gamma}\right) \frac{\min_{x \in S_n} x^T K^{(1)} x}{\max_{x \in S_n} x^T K^{(2)} x} = \Omega(\varepsilon). \quad (9)$$

*Proof.* For  $l = (2 - \varepsilon)x^T K^{(2)} x - 2 \max_i (K^{(2)} x)_i < 0$ , we get from Corollary 2 that the length of the left interval at  $x$  is of length at least

$$\varepsilon \left(1 - \frac{1}{\gamma}\right) \frac{f_t(x)}{-l}.$$

For any  $t \geq 0$ , we can lower bound

$$f_t(x) \geq f^{(1)}(x) = x^T K^{(1)} x \geq \min_{x \in S_n} x^T K^{(1)} x,$$

and for  $\varepsilon \leq 1$  we can upper bound

$$-l = 2 \max_i (K^{(2)} x)_i - (2 - \varepsilon)x^T K^{(2)} x \leq 2 \max_i (K^{(2)} x)_i,$$

because  $f^{(2)}(x) \geq 0$ . The value  $\max_i (K^{(2)} x)_i = \max_i e_i^T K^{(2)} x$  is the inner product between two points in the convex hull of the columns of the square root of the positive semi-definite matrix  $K^{(2)}$  (see the discussion at the beginning of this section). Let these two points be  $u, v \in \mathbb{R}^n$ . Using the Cauchy-Schwarz inequality we get

$$\begin{aligned} \max_i (K^{(2)} x)_i &= u^T v \leq \sqrt{\|u\|^2 \|v\|^2} \leq \frac{1}{2} (\|u\|^2 + \|v\|^2) \\ &\leq \max\{\|u\|^2, \|v\|^2\} \leq \max_{x \in S_n} x^T K^{(2)} x, \end{aligned}$$

where the last expression gives the norm of the largest vector with endpoint in the convex hull of the columns of the square root of  $K^{(2)}$ . Hence,  $-l \leq 2 \max_{x \in S_n} x^T K^{(2)} x$ . Combining the lower bound for  $f_t(x)$  and the upper bound for  $-l$  gives the stated bound on the interval length.  $\square$

Now, to upper bound the approximation path complexity we split the interval  $[0, \infty]$  into two parts: the sub-interval  $[0, 1]$  can be covered by at most  $1/l_f(\varepsilon, \gamma)$  admissible left intervals, i.e., by at most  $1/l_f(\varepsilon, \gamma)$  many admissible intervals. We reduce the analysis for the sub-interval  $t \in [1, \infty]$  to the analysis for  $[0, 1]$  by interchanging the roles of  $f^{(1)}$  and  $f^{(2)}$ . For any  $t \geq 1$ ,  $x$  is an  $\varepsilon$ -approximate solution to  $\min_{x \in S_n} f_t(x) := f^{(1)}(x) + t f^{(2)}(x)$  if and only if  $x$  is an  $\varepsilon$ -approximate solution to  $\min_{x \in S_n} f'_{t'}(x) := t' f^{(1)}(x) + f^{(2)}(x)$  for  $t' = \frac{1}{t} \leq 1$ , because the definition of an  $\varepsilon$ -approximation is invariant under scaling the objective function. Note that by allowing  $t = \infty$  we just refer to the case  $t' = 0$  in the equivalent problem for  $f'_{t'}(x)$  with  $t' = \frac{1}{t} \in [0, 1]$ . Using the lower bounds on the interval lengths  $l_f(\varepsilon, \gamma)$  and  $l_{f'}(\varepsilon, \gamma)$  (for the problem for  $f'_{t'}(x)$  with  $t' \in [0, 1]$ ) on both sub-intervals we get an upper bound of  $\left\lceil \frac{1}{l_f(\varepsilon, \gamma)} \right\rceil + \left\lceil \frac{1}{l_{f'}(\varepsilon, \gamma)} \right\rceil$  on the path complexity as is detailed in the following theorem:

**Theorem 1.** *Given any  $0 < \varepsilon \leq 1$  and  $\gamma > 1$ , the  $\varepsilon$ -approximation path complexity of Problem 8 is at most*

$$\frac{\gamma}{\gamma - 1} \left( \frac{\max_{x \in S_n} x^T K^{(2)} x}{\min_{x \in S_n} x^T K^{(1)} x} + \frac{\max_{x \in S_n} x^T K^{(1)} x}{\min_{x \in S_n} x^T K^{(2)} x} \right) \frac{2}{\varepsilon} + 2 = O\left(\frac{1}{\varepsilon}\right).$$

This proof of the path complexity immediately implies a bound on the time complexity of our approximation path Algorithm 1. In particular we obtain a linear running time of  $O\left(\frac{n}{\varepsilon^2}\right)$  for computing the global solution path when using [5, Algorithm 1.1] as the internal optimizer.

There are interesting applications of this result, because it is known that instances of Problem 8 include for example computing the solution path of a support vector machine – as the regularization parameter changes – and also finding the optimal combination of two kernel matrices in the setting of kernel learning. For more details we refer the reader to the full version of this paper.

## 4.2 Minimum Enclosing Ball of Points under Linear Motion

Let  $P = \{p_1, \dots, p_n\}$  be a set of  $n$  points in  $\mathbb{R}^d$ . The minimum enclosing ball (MEB) problem asks to find the smallest ball containing all points of  $P$ . The dual of the problem can be written [11] as:

$$\begin{aligned} & \max_x x^T b - x^T A^T A x \\ \text{s.t. } & x \in S_n \end{aligned} \tag{10}$$

where  $b = (b_i) = (p_i^T p_i)_{i \in [n]}$  and  $A$  is the matrix whose columns are the  $p_i$ .

Now we assume that the points move with constant speed in a fixed direction, i.e., they move linearly as follows

$$p_i(t) = p_i + tv_i, \quad t \in [0, \infty)$$

where  $t$  can be referred to as time parameter. The MEB problem for moving points reads as:

$$\begin{aligned} & \max_x x^T b(t) - x^T (P + tV)^T (P + tV)x \\ & \text{s.t. } x \in S_n \end{aligned} \quad (11)$$

where  $b(t) = (b_i(t)) = ((p_i + tv_i)^T (p_i + tv_i))_{i \in [n]}$  and  $P$  is the matrix whose columns are the points  $p_i$  and  $V$  is the matrix whose columns are the vector  $v_i$ . Problem 11 is a special case of the maximization version of Problem 6. Again, we are interested in the whole solution path, i.e. we want to track the center and the radius

$$r(t) = \sqrt{\hat{x}^T b(t) - \hat{x}^T (P + tV)^T (P + tV)\hat{x}} \quad \text{with } \hat{x} \in S_n \text{ optimal}$$

of the MEB of the points  $p_i(t)$  for  $t \in [0, \infty)$  (or approximations of it). For an analysis of an approximate solution path we make use of the following observation.

**Observation 1.** *The interval  $[0, \infty)$  can be subdivided into three parts: on the first sub-interval  $r(t)$  is decreasing, on the second sub-interval, the radius  $r(t)$  is constant, and on the third sub-interval, the radius is increasing.*

This can be seen as follows: consider the time when the radius of the MEB reaches its global minimum, just before the ball is expanding again. This is the point between the second and the third sub-interval. The points that cause the ball to expand at this point in time will prevent the ball from shrinking again in the future since the points move linearly. Thus the radius of the MEB will increase on the third sub-interval. By reversing the direction of time the same consideration leads to the observation that the radius of the MEB is decreasing on the first sub-interval.

We will consider each of the three sub-intervals individually. The second sub-interval can be treated like the standard MEB problem of non-moving points. Hence we only have to consider the first and the third sub-interval. We will only analyze the third sub-interval since the first sub-interval can be treated analogously with the direction of time reversed, i.e., the parameter  $t$  decreasing instead of increasing.

For the third sub-interval we know that the radius is increasing with time. We can shift the time parameter  $t$  such that we start with the third sub-interval at time  $t = 0$ . Let  $r > 0$  be the radius  $r(0)$  at time zero, i.e., we assume that the radius of the MEB never becomes zero. The case where the radius reaches 0 at some point is actually equivalent to the standard MEB problem for non-moving points. Without loss of generality we can scale all the vectors  $v_i$  such that the MEB defined by the points  $v_i$  has radius  $r$  as well, because this just means scaling

time. Without loss of generality we can also assume that the center of the MEB of the point sets  $P$  and  $V$  are both the origin. That is,  $\|p_i\| \leq r$  and  $\|v_i\| \leq r$ . We have  $f_t(x) := x^T b(t) - x^T (P + tV)^T (P + tV)x$ . A short computation shows that

$$(\nabla f_{t+\delta}(x) - \nabla f_t(x))_i - x^T (\nabla f_{t+\delta}(x)) - \nabla f_t(x)) \leq 12r^2(1+t+\delta)\delta.$$

and

$$|f_{t+\delta}(x) - f_t(x)| \leq 4r^2(1+t+\delta)\delta.$$

Now we can apply Lemma 2. Inequality 4 here simplifies to

$$12r^2(1+t+\delta)\delta + \varepsilon 4r^2(1+t+\delta)\delta \leq \varepsilon \left(1 - \frac{1}{\gamma}\right) r^2$$

since  $f_t(x) \geq r^2$ . Assuming  $\varepsilon \leq 1$ , we can set  $\delta = \frac{\varepsilon}{32} \left(1 - \frac{1}{\gamma}\right)$  for  $t, t+\delta \in [0, 1]$ . For the interval of  $t, t+\delta = [1, \infty)$  we apply the same trick as before and reduce it to the case of  $t, t+\delta \in [0, 1]$  by interchanging the roles of  $P$  and  $V$ . A short computation shows that an  $\varepsilon$ -approximation  $x$  at time  $t \geq 1$  for the original optimization problem

$$\begin{aligned} & \max_x x^T b(t) - x^T (P + tV)^T (P + tV)x \\ & \text{s.t. } x \in S_n \end{aligned}$$

is an  $\varepsilon$ -approximation for the optimization problem

$$\begin{aligned} & \max_x x^T b'(t') - x^T (t'P + V)^T (t'P + V)x \\ & \text{s.t. } x \in S_n \end{aligned}$$

at time  $t' = 1/t$ , where  $b'(t') = (b'_i(t')) = ((t'p_i + v_i)^T (t'p_i + v_i))_{i \in [n]}$ , i.e., the roles of  $P$  and  $V$  have been interchanged. This is again due to the fact that the relative approximation guarantee is invariant under scaling. Hence, we conclude with the following theorem on the approximation path complexity for the minimum enclosing ball problem under linear motion:

**Theorem 2.** *The  $\varepsilon$ -approximation path complexity of the minimum enclosing ball Problem 11 for parameter  $t \in [0, \infty)$  is at most*

$$64 \frac{\gamma}{\gamma - 1} \frac{1}{\varepsilon} = O\left(\frac{1}{\varepsilon}\right).$$

Since for the static MEB Problem 10, coresets of size  $O(\frac{1}{\varepsilon})$  exist, see 4, we obtain the following corollary to Theorem 2.

**Corollary 3.** *There exists an  $\varepsilon$ -coreset of size  $O(\frac{1}{\varepsilon^2})$  for Problem 11 that is globally valid under the linear motion, i.e., valid for all  $t \geq 0$ .*

The only other result known in this context is the existence of coresets of size  $2^{O(\frac{1}{\varepsilon^2} \log \frac{1}{\varepsilon})}$  that remain valid under polynomial motions 2, and earlier, Agarwal et al. 11 have already proven the existence of coresets of size  $O(1/\varepsilon^{2d})$  for the extent problem for moving points, which includes the MEB problem as a special case.

## 5 Conclusion

We have presented a framework to optimize convex functions over the unit simplex that are parameterized in one parameter. The framework is very general, simple and has been proven to be practical on a number of machine learning problems. Although it is very simple it still provides improved theoretical bounds on known problems. In fact, we showed that our method is optimal up to a small constant factor.

## References

1. Agarwal, P., Har-Peled, S., Varadarajan, K.: Approximating extent measures of points. *Journal of the ACM* 51(4), 606–635 (2004)
2. Agarwal, P., Har-Peled, S., Yu, H.: Embeddings of surfaces, curves, and moving points in euclidean space. In: *SCG 2007: Proceedings of the Twenty-third Annual Symposium on Computational Geometry* (2007)
3. Bach, F., Lanckriet, G., Jordan, M.: Multiple kernel learning, conic duality, and the smo algorithm. In: *ICML 2004: Proceedings of the Twenty-first International Conference on Machine Learning* (2004)
4. Bădoiu, M., Clarkson, K.L.: Optimal core-sets for balls. *Computational Geometry: Theory and Applications* 40(1), 14–22 (2007)
5. Clarkson, K.L.: Coresets, sparse greedy approximation, and the frank-wolfe algorithm. In: *SODA 2008: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2008)
6. Friedman, J., Hastie, T., Höfling, H., Tibshirani, R.: Pathwise coordinate optimization. *The Annals of Applied Statistics* 1(2), 302–332 (2007)
7. Gärtner, B., Giesen, J., Jaggi, M.: An exponential lower bound on the complexity of regularization paths. arXiv, cs.LG (2009)
8. Gärtner, B., Giesen, J., Jaggi, M., Welsch, T.: A combinatorial algorithm to compute regularization paths. arXiv, cs.LG (2009)
9. Gärtner, B., Jaggi, M.: Coresets for polytope distance. In: *SCG 2009: Proceedings of the 25th Annual Symposium on Computational Geometry* (2009)
10. Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. *The Journal of Machine Learning Research* 5, 1391–1415 (2004)
11. Matousek, J., Gärtner, B.: *Understanding and Using Linear Programming* (Universitext). Springer, New York (2006)
12. Rosset, S., Zhu, J.: Piecewise linear regularized solution paths. *Annals of Statistics* 35(3), 1012–1030 (2007)
13. Wu, Z., Zhang, A., Li, C., Sudjianto, A.: Trace solution paths for svms via parametric quadratic programming. In: *KDD 2008 DMMT Workshop* (2008)

# Approximation Schemes for Multi-Budgeted Independence Systems<sup>\*</sup>

Fabrizio Grandoni<sup>1</sup> and Rico Zenklusen<sup>2, \*\*</sup>

<sup>1</sup> Computer Science Department, University of Rome Tor Vergata  
[grandoni@disp.uniroma2.it](mailto:grandoni@disp.uniroma2.it)

<sup>2</sup> Department of Mathematics, EPFL  
[rico.zenklusen@epfl.ch](mailto:rico.zenklusen@epfl.ch)

**Abstract.** A natural way to deal with multiple, partially conflicting objectives is turning all the objectives but one into budget constraints. Some classical optimization problems, such as spanning tree and forest, shortest path, (perfect) matching, independent set (basis) in a matroid or in the intersection of two matroids, become NP-hard even with one budget constraint. Still, for most of these problems efficient deterministic and randomized approximation schemes are known. For two or more budgets, typically only multi-criteria approximation schemes are available, which return slightly infeasible solutions. Not much is known however for strict budget constraints: filling this gap is the main goal of this paper.

It is not hard to see that the above-mentioned problems whose solution sets do not correspond to *independence systems* are inapproximable already for two budget constraints. For the remaining problems, we present approximation schemes for a constant number  $k$  of budget constraints using a variety of techniques: i) we present a simple and powerful mechanism to transform multi-criteria approximation schemes into *pure* approximation schemes. This leads to deterministic and randomized approximation schemes for various of the above-mentioned problems; ii) we show that points in low-dimensional faces of any matroid polytope are almost integral, an interesting result on its own. This gives a deterministic approximation scheme for  $k$ -budgeted matroid independent set; iii) we present a deterministic approximation scheme for 2-budgeted matching. The backbone of this result is a purely topological property of curves in  $\mathbb{R}^2$ .

## 1 Introduction

In many applications, one has to compromise between several, partially conflicting goals. *Multi-Objective Optimization* is a broad area of study in Operations Research, Economics and Computer Science (see [8][22] and references therein). A variety of approaches have been employed to formulate such problems. Here we

\* Partially developed while the first author was visiting EPFL.

\*\* Partially supported by the Swiss National Science Foundation, grant number: PBEZP2-129524.

adopt the Multi-Budgeted Optimization approach [22]: we cast one of the goals as the objective function, and the others as *budget constraints*. More precisely, we are given a (finite) set  $\mathcal{F}$  of solutions for the problem, where each solution is a subset  $S$  of elements from a given universe  $E$  (e.g., the edges of a graph). We are also given a weight function  $w : \mathcal{F} \rightarrow \mathbb{Q}_+$  and a set of  $k = O(1)^1$  length functions  $\ell_i : \mathcal{F} \rightarrow \mathbb{Q}_+$ ,  $1 \leq i \leq k$ , that assign a weight  $w(S) := \sum_{e \in S} w(e)$  and an  $i$ -th-length  $\ell_i(S) := \sum_{e \in S} \ell_i(e)$ ,  $1 \leq i \leq k$ , to every candidate solution  $S$ . For each length function  $\ell_i$ , there is a budget  $L_i \in \mathbb{Q}_+$ . The *multi-budgeted optimization problem* can then be formulated as follows:

$$\text{maximize/minimize } w(S) \text{ subject to } S \in \mathcal{F}, \ell_i(S) \leq L_i, 1 \leq i \leq k.$$

We next use  $OPT$  to denote an optimum solution.

Following the literature on the topic, we focused on the set of problems below:

- **$k$ -BUDGETED (PERFECT) MATCHING:**  $\mathcal{F}$  is given by the (perfect) matchings of an undirected graph  $G = (V, E)$ .
- **$k$ -BUDGETED SPANNING TREE (FOREST):**  $\mathcal{F}$  is given by the spanning trees (forests) of  $G$ .
- **$k$ -BUDGETED SHORTEST PATH:**  $\mathcal{F}$  is given by the paths connecting two given nodes  $s$  and  $t$  in  $G$ .
- **$k$ -BUDGETED MATROID INDEPENDENT SET (BASIS):**  $\mathcal{F}$  is given by the independent sets (bases) of a matroid  $M = (E, \mathcal{I})^2$ .
- **$k$ -BUDGETED MATROID INTERSECTION INDEPENDENT SET (BASIS):**  $\mathcal{F}$  is given by the independent sets (bases) in the intersection of two matroids  $M_1 = (E, \mathcal{I}_1)$  and  $M_2 = (E, \mathcal{I}_2)$ .

All the problems above are polynomial-time solvable (see, e.g., [24]) in their un-budgeted version ( $k = 0$ ), but become NP-hard [13, 7] even for a single budget constraint ( $k = 1$ ). For the case of one budget ( $k = 1$ ), polynomial-time approximation schemes (PTASs) are known for SPANNING TREE [21] (see also [11]), SHORTEST PATH [25] (see also [10, 14]), and MATCHING [3]. The approach in [21] easily generalizes to the case of MATROID BASIS. A PTAS is also known for MATROID INTERSECTION INDEPENDENT SET [3]. No approximation algorithm is known for the problems above in the case  $k \geq 2$  (excluding multi-criteria algorithms which provide slightly infeasible solutions): investigating the existence of such algorithms is the main goal of this paper.

## 1.1 Our Results

We start by observing that several of the mentioned problems are inapproximable already for two budget constraints. More precisely, the corresponding feasibility

---

<sup>1</sup> The assumption that  $k$  is a constant is crucial in this paper.

<sup>2</sup> We recall that  $E$  is a finite ground set and  $\mathcal{I} \subseteq 2^E$  is a nonempty family of subsets of  $E$  (*independent sets*) which have to satisfy the following two conditions: (i)  $I \in \mathcal{I}$ ,  $J \subseteq I \Rightarrow J \in \mathcal{I}$  and (ii)  $I, J \in \mathcal{I}$ ,  $|I| > |J| \Rightarrow \exists z \in I \setminus J : J \cup \{z\} \in \mathcal{I}$ . A *basis* is a maximal independent set. For all matroids used in this paper we make the usual assumption that independence of a set can be checked in polynomial time. For additional information on matroids, see e.g. [24].

problem is NP-complete. Due to space constraints, we omit the simple proof of the following theorem (which might be considered as part of folklore).

**Theorem 1.** *For  $k \geq 2$ , it is NP-complete to decide whether there is a feasible solution for  $k$ -BUDGETED SHORTEST PATH,  $k$ -BUDGETED PERFECT MATCHING and  $k$ -BUDGETED SPANNING TREE (and hence also for  $k$ -budgeted matroid basis and matroid intersection basis).*

The remaining problems in the above list have a common aspect: the set of solutions  $\mathcal{F}$  forms an *independence system*. In other terms, for  $S \in \mathcal{F}$  and  $S' \subseteq S$ , we have  $S' \in \mathcal{F}$ . For these problems, we present deterministic and randomized approximation schemes, based on a variety of techniques.

Our first result is a simple but powerful mechanism to transform a multi-criteria PTAS, i.e., a PTAS that might violate the budgets by a small multiplicative factor, into a *pure* PTAS, where no budget is violated. Similarly, a multi-criteria polynomial randomized-time approximation scheme (PRAS) can be transformed into a pure PRAS (see Section 2).

**Theorem 2. (Feasibilization)** *Let  $\mathcal{P}_{ind}$  be a  $k$ -budgeted problem where the set of solutions  $\mathcal{F}$  is an independence system. Suppose that we are given an algorithm  $\mathcal{A}$  which, for any constant  $\delta > 0$ , computes in polynomial time an  $(1 - \delta)$  (resp., expected  $(1 - \delta)$ ) approximate solution to  $\mathcal{P}_{ind}$  violating each budget by a factor at most  $(1 + \delta)$ . Then there is a PTAS (resp., PRAS) for  $\mathcal{P}_{ind}$ .*

The basic idea is as follows. We show that a *good* solution exists even if we scale down the budgets by a small factor. This is done by applying a greedy discarding strategy similar to the greedy algorithm for KNAPSACK. Applying a multi-criteria PTAS (given as a black box) to the scaled problem gives a feasible solution for the original one, of weight *close* to the optimal weight.

To the best of our knowledge, this simple result was never observed before. Indeed, it implies improved approximation algorithms for a number of problems. A general construction by Papadimitriou and Yannakakis [18] provides multi-criteria PTASs (resp., PRASs) for problems whose exact version admits a pseudo-polynomial-time (PPT) deterministic (resp., Monte-Carlo) algorithm. We recall that the *exact version* of a given optimization problem asks for a feasible solution of exactly a given *target* weight. Combining their approach with our mechanism one obtains approximation schemes for several problems. For example, using the PPT-algorithm for EXACT FOREST in [2], one obtains a PTAS for  $k$ -BUDGETED FOREST. Similarly, the Monte-Carlo PPT-algorithm for EXACT MATCHING in [17] gives a PRAS for  $k$ -BUDGETED MATCHING. The Monte-Carlo PPT-algorithms for EXACT MATROID INTERSECTION INDEPENDENT SET in [5], which works in the special case of representable matroids<sup>3</sup>, implies a PRAS for the corresponding budgeted problem.

Of course, one can also exploit multi-criteria approximation schemes obtained with different techniques. For example, exploiting the multi-criteria PTAS in

<sup>3</sup> A matroid  $M = (E, \mathcal{I})$  is representable if its ground set  $E$  can be mapped in a bijective way to the columns of a matrix over some field, and  $I \subseteq E$  is independent in  $M$  iff the corresponding columns are linearly independent.

**[8]** for  $k$ -BUDGETED MATCHING in bipartite graphs, which is based on iterative rounding, one obtains a PTAS for the same problem. Very recently [6], a multi-criteria PRAS for  $k$ -BUDGETED MATROID INDEPENDENT SET, based on dependent randomized rounding, has been presented. This implies a PRAS for  $k$ -BUDGETED MATROID INDEPENDENT SET.

**Corollary 1.** *There are PTASs for  $k$ -BUDGETED FOREST and  $k$ -BUDGETED MATCHING in bipartite graphs. There are PRASs for  $k$ -BUDGETED MATCHING,  $k$ -BUDGETED MATROID INDEPENDENT SET, and  $k$ -BUDGETED MATROID INTERSECTION in representable matroids.*

Based on a different, more direct approach, we are able to turn the PRAS for  $k$ -BUDGETED MATROID INDEPENDENT SET into a PTAS. The main insight is the following structural property of faces of the matroid polytope which might be of independent interest (proof in Section [3]).

**Theorem 3.** *Let  $M = (E, \mathcal{I})$  be a matroid and let  $F$  be a face of dimension  $d$  of the matroid polytope<sup>4</sup>  $P_{\mathcal{I}}$ . Then any  $x \in F$  has at most  $2d$  non-integral components. Furthermore, the sum of all fractional components of  $x$  is at most  $d$ .*

A PTAS can then be easily derived as follows. We first guess the  $k/\varepsilon$  elements  $E_H$  of largest weight in the optimum solution in a preliminary phase, and reduce the problem consequently. This guessing step guarantees that the maximum weight  $w_{\max}$  of an element in the reduced problem satisfies  $kw_{\max} \leq \varepsilon w(E_H)$ . For the reduced problem, we compute an optimal fractional vertex solution  $x^*$  to the LP which seeks to find a maximum weight point in the matroid polytope intersected with the  $k$  budget constraints. Since  $x^*$  is chosen to be a vertex solution, and only  $k$  linear constraints are added to the matroid polytope,  $x^*$  lies on a face of the matroid polytope of dimension at most  $k$ . We then round down the fractional components of  $x^*$  to obtain an incidence vector  $\bar{x}$  which corresponds to some independent set  $E_L$ . By Theorem [3],  $|x^* - \bar{x}| \leq k$ , and hence,  $w(E_L) \geq w(x^*) - kw_{\max}$ . Then, it is not hard to see that  $E_H \cup E_L$  is a  $(1 - \varepsilon)$ -approximate feasible solution for the starting problem.

**Corollary 2.** *There is a PTAS for  $k$ -BUDGETED MATROID INDEPENDENT SET.*

Eventually, we present a PTAS (rather than a PRAS as in Corollary [1]) for 2-BUDGETED MATCHING (see Section [4]).

**Theorem 4.** *There is a PTAS for 2-BUDGETED MATCHING.*

Our PTAS works as follows. Let us confuse a matching  $M$  with the associated incidence vector  $x_M$ . We initially compute an optimal fractional matching  $x^*$ , and express it as the convex combination  $x^* = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$  of three matchings  $x_1$ ,  $x_2$ , and  $x_3$ . Then we exploit a *patching procedure* which, given two matchings  $x'$  and  $x''$  with high Lagrangian weight and a parameter  $\mu \in [0, 1]$ , computes a matching  $z$  which is not longer than  $x_\mu := \mu x' + (1 - \mu)x''$  with

---

<sup>4</sup> For some given matroid  $M = (E, \mathcal{I})$ , the corresponding matroid polytope  $P_{\mathcal{I}}$  is the convex hull of the incidence vectors of all independent sets.

respect to both lengths, and has a comparable weight. This procedure is applied twice: first on the matchings  $x_1$  and  $x_2$  with parameter  $\mu = \alpha_1/(\alpha_1 + \alpha_2)$ , hence getting a matching  $z'$ . Second, on the two matchings  $z'$  and  $x_3$  with parameter  $\mu = (\alpha_1 + \alpha_2)/(\alpha_1 + \alpha_2 + \alpha_3)$ . The resulting matching  $z''$  is feasible and almost optimal (modulo a preliminary guessing step).

Our patching procedure relies on a topological property of curves in  $\mathbb{R}^2$ , that we prove via Jordan's curve theorem [15]. An extension of the property above to curves in  $\mathbb{R}^k$  would imply a PTAS for  $k$ -BUDGETED MATCHING: this is left as an interesting open problem (details are omitted for lack of space).

## 1.2 Related Work

There are a few general tools for designing approximation algorithms for budgeted problems. One basic approach is combining *dynamic programming* (which solves the problem for polynomial weights and lengths) with *rounding and scaling* techniques (to reduce the problem to the case of polynomial quantities). This leads for example to the FPTAS for 1-BUDGETED SHORTEST PATH [10, 14, 25]. Another fundamental technique is the *Lagrangian relaxation method*. The basic idea is relaxing the budget constraints, and lifting them into the objective function, where they are weighted by Lagrangian multipliers. Solving the relaxed problem, one obtains two or more solutions with optimal Lagrangian weight, which can - if needed - be patched together to get a good solution for the original problem. Demonstrating this method, Goemans and Ravi [21] gave a PTAS for 1-BUDGETED SPANNING TREE, which also extends to 1-BUDGETED MATROID BASIS. Using the same approach, with an involved patching step, Berger, Bonifaci, Grandoni, and Schäfer [3] obtained a PTAS for 1-BUDGETED MATCHING and 1-BUDGETED MATROID INTERSECTION INDEPENDENT SET. Their approach does not seem to generalize to the case of multiple budget constraints.

The techniques above apply to the case of one budget. Not much is known for problems with two or more budgets. However, often *multi-criteria* approximation schemes are known, which provide a  $(1 - \varepsilon)$ -approximate solution violating the budgets by a factor  $(1 + \varepsilon)$ . First of all, there is a very general technique by Papadimitriou and Yannakakis [18], based on the construction of  $\varepsilon$ -approximate Pareto curves. Given an optimization problem with multiple objectives, the *Pareto curve* consists of the set of solutions  $S$  such that there is no solution  $S'$  which is strictly better than  $S$  (in a vectorial sense). Papadimitriou and Yannakakis show that, for any constant  $\varepsilon > 0$ , there always exists a polynomial-size  $\varepsilon$ -approximate Pareto curve  $\mathcal{A}$ , i.e., a set of solutions such that every solution in the Pareto curve is within a factor of  $(1 + \varepsilon)$  from some solution in  $\mathcal{A}$  on each objective. Furthermore, this approximate curve can be constructed in polynomial time in the size of the input and  $1/\varepsilon$  whenever there exists a PPT algorithm for the associated exact problem. This implies multi-criteria FPTASs for  $k$ -BUDGETED SPANNING TREE and  $k$ -BUDGETED SHORTEST PATH. Furthermore, it implies a multi-criteria FPRAS for  $k$ -BUDGETED (PERFECT) MATCHING. The latter result exploits the Monte-Carlo PPT algorithm for EXACT MATCHING

in [17]. Our PRAS improves on these results, approximation-wise (the running time is larger in our case).

Recently, Grandoni, Ravi and Singh [8] showed that the *iterative rounding* technique is an alternative way to achieve similar (or better) results. Using this method they obtain a multi-criteria PTAS for  $k$ -BUDGETED SPANNING TREE, which computes a solution of optimal cost violating each budget by a factor  $(1 + \varepsilon)$ . This improves, approximation-wise, on the result in [18] for the same problem (where the solution returned is suboptimal). The authors also show how to obtain a deterministic (rather than randomized [18]) multi-criteria PTAS for  $k$ -BUDGETED MATCHING in bipartite graphs.

All mentioned problems are easy in the unbudgeted version. Given an NP-hard unbudgeted problem which admits a  $\rho$  approximation, the *parametric search* technique in [16] provides a multi-criteria  $k\rho$  approximation algorithm violating each budget by a factor  $k\rho$  for the corresponding problem with  $k$  budgets. Other techniques lead to logarithmic approximation factors (see, e.g., [4, 19, 20]).

## 2 A Feasibilization Mechanism

*Proof (Theorem 2).* Let  $\varepsilon \in (0, 1]$  be a given constant, with  $1/\varepsilon \in \mathbb{N}$ . Consider the following algorithm. Initially we guess the  $h = k/\varepsilon$  elements  $E_H$  of  $OPT$  of largest weight, and reduce the problem consequently<sup>5</sup>, hence getting a problem  $\mathcal{P}'$ . Then we scale down all the budgets by a factor  $(1 - \delta)$ , and solve the resulting problem  $\mathcal{P}''$  by means of  $\mathcal{A}$ , where  $\delta = \varepsilon/(k+1)$ . Let  $E_L$  be the solution returned by  $\mathcal{A}$ . We finally output  $E_H \cup E_L$ .

Let  $OPT'$  and  $OPT''$  be the optimum solution to problems  $\mathcal{P}'$  and  $\mathcal{P}''$ , respectively. We also denote by  $L'_i$  and  $L''_i$  the  $i$ th budget in the two problems, respectively. Let  $w_{max}$  be the largest weight in  $\mathcal{P}'$  and  $\mathcal{P}''$ . We observe that trivially: (a)  $w(OPT) = w(E_H) + w(OPT')$  and (b)  $w_{max} \leq w(E_H)/h$ .

Let us show that (c)  $w(OPT'') \geq w(OPT')(1 - k\delta) - kw_{max}$ . Consider the following process: for each length function  $i$ , we remove from  $OPT'$  the element  $e$  with smallest ratio  $w(e)/\ell_i(e)$  until  $\ell_i(OPT') \leq (1 - \delta)L'_i$ . Let  $E_i$  be the set of elements removed. It is not hard to see that  $w(E_i) \leq \delta w(OPT') + w_{max}$ . It follows that  $OPT' - \cup_i E_i$  is a feasible solution for  $\mathcal{P}''$  of weight at least  $w(OPT')(1 - \delta k) - kw_{max}$ , proving (c).

We observe that  $E_L$  is feasible for  $\mathcal{P}'$  since, for each  $i$ ,  $\ell_i(E_L) \leq (1 + \delta)L''_i = (1 + \delta)(1 - \delta)L'_i \leq L'_i$ . As a consequence, the returned solution  $E_H \cup E_L$  is feasible. Moreover, when  $\mathcal{A}$  is deterministic, we have

$$\begin{aligned} w(E_H) + w(E_L) &\geq w(E_H) + (1 - \delta)w(OPT'') \\ &\stackrel{(c)}{\geq} w(E_H) + (1 - \delta)(w(OPT')(1 - \delta k) - kw_{max}) \end{aligned}$$

<sup>5</sup> As usual, by reducing we mean decreasing each budget  $L_i$  by  $\ell_i(E_H)$  and removing all the elements of weight strictly larger than  $\min_{e \in E_H} w(e)$ . By guessing we mean trying all the  $O(m^h)$  subsets of  $h$  elements.

$$\begin{aligned} &\stackrel{(b)}{\geq} (1 - k/h)w(E_H) + (1 - \delta(k+1))w(OPT') \\ &\geq (1 - \varepsilon)(w(E_H) + w(OPT')) \stackrel{(a)}{=} (1 - \varepsilon)w(OPT). \end{aligned}$$

The same bound holds in expectation when  $\mathcal{A}$  is randomized.

### 3 A PTAS for $k$ -BUDGETED MATROID INDEPENDENT SET

It is convenient to consider weights  $w$  and lengths  $\ell_i$  as vectors in  $\mathbb{Q}^E$ . We denote by  $\ell$  the matrix whose  $i$ th column is  $\ell_i$ , and let  $L = (L_1, \dots, L_k)^T$ . A *rank function*  $r : 2^E \rightarrow \mathbb{N}$  is associated to every matroid  $M = (E, \mathcal{I})$ ; it is defined by  $r(S) = \max\{|J| \mid J \subseteq S, J \in \mathcal{I}\}$ . The *matroid polytope*  $P_{\mathcal{I}}$  is the convex hull of the characteristic vectors  $\chi_I$  of the independent sets  $I \in \mathcal{I}$  and is described by the following set of inequalities:  $P_{\mathcal{I}} = \text{conv}\{\chi_I : I \in \mathcal{I}\} = \{x \geq 0 : x(S) \leq r(S) \forall S \subseteq E\}$ . As usual,  $x(S) := \sum_{e \in S} x(e)$ <sup>6</sup>.

*Proof (Theorem 3).* Let  $m = |E|$ . We assume that the matroid polytope has full dimension, i.e.,  $\dim(P_{\mathcal{I}}) = m$ , or equivalently, every element  $e \in E$  is independent. This can be assumed wlog since if  $\{e\} \notin \mathcal{I}$  for some  $e \in E$ , then we can reduce the matroid by deleting element  $e$ . Since  $\dim(P_{\mathcal{I}}) = m$  and  $\dim(F) = d$ ,  $F$  can be described by the inequality system of  $P_{\mathcal{I}}$ , where  $m-d$  linearly independent inequalities used in the description of  $P_{\mathcal{I}}$  are turned into equalities. More precisely, there are  $N \subseteq E$  and  $A_1, \dots, A_k \subseteq E$  such that

$$F = \{x \in P_{\mathcal{I}} \mid x(e) = 0 \forall e \in N, x(A_i) = r(A_i) \forall i \in \{1, \dots, k\}\},$$

and  $|N| + k = m - d$ . By standard uncrossing arguments, we can assume that the sets  $A_i$  form a chain, i.e.,  $A_1 \subsetneq A_2 \subsetneq \dots \subsetneq A_k$  (see for example [912] for further information on combinatorial uncrossing). We prove the claim by induction on the number of elements of the matroid. The theorem clearly holds for matroids with a ground set of cardinality one. First assume  $N \neq \emptyset$  and let  $e \in N$ . Let  $M'$  be the matroid obtained from  $M$  by deleting  $e$ , and let  $F'$  be the projection of  $F$  onto the coordinates corresponding to  $N \setminus \{e\}$ . Since  $F'$  is a face of  $M'$ , the claim follows by induction. Henceforth, we assume  $N = \emptyset$  which implies  $k = m - d$ . Let  $A_0 = \emptyset$  and  $B_i = A_i \setminus A_{i-1}$  for  $i \in \{1, \dots, k\}$ . In the following we show that we can assume

$$0 < r(A_i) - r(A_{i-1}) < |B_i| \quad \forall i \in \{1, \dots, k\}. \tag{1}$$

Notice that  $0 \leq r(A_i) - r(A_{i-1}) \leq |B_i|$  clearly holds by standard properties of rank functions (see [24] for more details). Assume that there is  $i \in \{1, \dots, k\}$  with  $r(A_i) = r(A_{i-1})$ . Since all points  $x \in F$  satisfy  $x(A_i) = r(A_i)$  and  $x(A_{i-1}) = r(A_{i-1})$ , we have  $x(B_i) = 0$ . Hence for any  $e \in B_i$ , we have  $x(e) = 0$  for  $x \in F$ . Again, we can delete  $e$  from the matroid, hence obtaining a smaller matroid for which the claim holds by the inductive hypothesis. Therefore, we can assume  $r(A_i) > r(A_{i-1})$  which implies the left inequality in (1).

---

<sup>6</sup> See [24] for more details and omitted standard definitions.

For the right inequality assume that there is  $i \in \{1, \dots, k\}$  with  $r(A_i) - r(A_{i-1}) = |B_i|$ . Hence, every  $x \in F$  satisfies  $x(B_i) = |B_i|$ , implying  $x(e) = 1$  for all  $e \in B_i$ . Let  $e \in B_i$ , and let  $F'$  be the projection of the face  $F$  onto the components  $N \setminus \{e\}$ . Since  $F'$  is a face of the matroid  $M'$  obtained from  $M$  by contracting  $e$ , the result follows again by the inductive hypothesis.

Henceforth, we assume that (II) holds. This implies in particular that  $|B_i| > 1$  for  $i \in \{1, \dots, k\}$ . Since  $\sum_{i=1}^k |B_i| \leq m$ , we have  $k \leq m/2$ , which together with  $k = m - d$  implies  $d \geq m/2$ . The claim of the theorem that  $x \in F$  has at most  $2d$  non-integral components is thus trivial in this case.

To prove the second part of the theorem we show that if (II) holds then  $x(E) \leq d$  for  $x \in F$ . For  $x \in F$  we have

$$\begin{aligned} x(E) &= x(E \setminus A_k) + \sum_{i=1}^k x(B_i) \leq |E| - |A_k| + \sum_{i=1}^k (r(A_i) - r(A_{i-1})) \\ &\leq |E| - |A_k| + \sum_{i=1}^k (|A_i| - |A_{i-1}| - 1) = m - k = d, \end{aligned}$$

where the first inequality follows from  $x(E \setminus A_k) \leq |E \setminus A_k|$  and  $x(B_i) = r(A_i) - r(A_{i-1})$ , and the second inequality follows from (II).

## 4 A PTAS for 2-BUDGETED MATCHING

In this section we present our PTAS for 2-BUDGETED MATCHING. We denote by  $\mathcal{M}$  the set of incidence vectors of matchings. With a slight abuse of terminology we call the elements in  $\mathcal{M}$  matchings. Let  $P_{\mathcal{M}}$  be the matching polyhedron. Analogously to Section 3, let  $\ell = (\ell_1, \ell_2)$  and  $L = (L_1, L_2)^T$ . A feasible solution in this framework is a matching  $x \in \mathcal{M}$  such that  $\ell^T x \leq L$ . For two elements  $z', z'' \in [0, 1]^E$ , we define their *symmetric difference*  $z' \Delta z'' \in [0, 1]^E$  by  $(z' \Delta z'')(e) = |z'(e) - z''(e)|$  for all  $e \in E$ . In particular, if  $z'$  and  $z''$  are incidence vectors, then their symmetric difference as defined above corresponds indeed to the symmetric difference in the usual sense. Recall that, when  $z'$  and  $z''$  are matchings,  $z' \Delta z''$  consists of a set of node-disjoint paths and cycles.

We start by presenting a property of curves in  $\mathbb{R}^2$ . This property is used to derive the mentioned *patching procedure*. Eventually, we describe and analyze our PTAS.

**A Property of Curves in  $\mathbb{R}^2$ .** We next describe a topological property of polygonal curves in  $\mathbb{R}^2$ , which will be crucial in our proof<sup>7</sup>. A *curve* in  $\mathbb{R}^2$  is a continuous function  $f : [0, \tau] \rightarrow \mathbb{R}^2$  for some  $\tau \in \mathbb{R}_+$ . A curve is called *polygonal* if it is piecewise linear. For  $a \in [0, \tau]$ , let  $f^a : [0, \tau] \rightarrow \mathbb{R}^2$  be the following curve.

$$f^a(t) = \begin{cases} f(t+a) - f(a) + f(0) & \text{if } t+a < \tau, \\ f(\tau) - f(a) + f(a+t-\tau) & \text{if } t+a \geq \tau. \end{cases}$$

<sup>7</sup> The lemma even holds for general (non-polygonal) curves. However, since we only need polygonal curves in our setting we restrict ourselves to this case since it simplifies the exposition.

Observe that  $f^a(0) = f(0)$  and  $f^a(\tau) = f(\tau)$  for any  $a \in [0, \tau]$ . The next lemma shows that any point  $x$  on the segment between  $f(0)$  and  $f(\tau)$  is contained in some curve  $f^a$ .

**Lemma 1.** *Let  $f : [0, \tau] \rightarrow \mathbb{R}^2$  be a polygonal curve, and let  $\mu \in [0, 1]$ . Then there are  $a, t \in [0, \tau]$  such that  $f^a(t) = \mu f(0) + (1 - \mu)f(\tau)$ .*

We next give an intuitive description of the proof of the lemma: a formal proof is postponed to the journal version of the paper. Let  $f = (f_1, f_2)$ . Since the statement of the lemma is independent of changes in the coordinate system (and the claim is trivial for  $f(0) = f(\tau)$ ), we can assume that  $f(0) = (0, 0)$  and  $f(\tau) = (r, 0)$  for some  $r > 0$ . The Gasoline Lemma [3] states that there is  $a_1 \in [0, \tau]$  such that  $f_2^{a_1}(t) \geq 0 \forall t \in [0, \tau]$ . In particular, this condition is satisfied by choosing  $a_1 \in \arg \min\{f_2(t) \mid t \in [0, \tau]\}$ . Analogously, for  $a_2 \in \arg \max\{f_2(t) \mid t \in [0, \tau]\}$ ,  $f_2^{a_2}(t) \leq 0 \forall t \in [0, \tau]$ . Hence, we have two curves,  $f^{a_1}$  and  $f^{a_2}$ , one above and the other below the x-axis, both with the same endpoints  $(0, 0)$  and  $(r, 0)$ . Furthermore, for  $a$  ranging from  $a_1$  to  $a_2$  (in a circular sense), the curve  $f^a$  continuously transforms from  $f^{a_1}$  to  $f^{a_2}$ , always maintaining the same endpoints. Then it is intuitively clear that the union of the curves  $f^a$  spans all the points on the segment from  $(0, 0)$  to  $(r, 0)$ , hence proving the claim.

**The Patching Procedure.** In this section we describe a *patching procedure* which, given two matchings  $x'$  and  $x''$  and a parameter  $\mu \in [0, 1]$ , computes a matching  $z$  satisfying  $\ell^T z \leq \ell^T x_\mu$ , where  $x_\mu := \mu x' + (1 - \mu)x''$  is a convex combination of the first two matchings. Furthermore, the weight  $w^T z$  is *close* to  $w^T x_\mu$ , provided that  $x'$  and  $x''$  have a *sufficiently* large Lagrangian weight, which is defined as follows. Let  $\lambda_1^*, \lambda_2^* \in \mathbb{R}_+$  be a pair of optimal dual multipliers for the budgets in the linear program  $\max\{w^T x \mid x \in P_{\mathcal{M}}, \ell^T x \leq L\}$ . The *Lagrangian weight* of  $x \in [0, 1]^E$  is  $\mathcal{L}(x) = w^T x - (\lambda_1^*, \lambda_2^*)(\ell^T x - L)$ . Notice, that by the theory of Lagrangian duality we have  $w^* = \max\{\mathcal{L}(x) \mid x \in P_{\mathcal{M}}\}$ , where  $w^*$  is the weight of an optimal LP solution, i.e.,  $w^* = \max\{w^T x \mid x \in P_{\mathcal{M}}, \ell^T x \leq L\}$  (see [13] for more information on Lagrangian duality).

We need the following notion of almost matching.

**Definition 1.** *For  $r \in \mathbb{N}$ , an  $r$ -almost matching in  $G$  is a (possibly fractional) vector  $y \in [0, 1]^E$  such that it is possible to set at most  $r$  components of  $y$  to zero to obtain a matching.*

We denote by  $\mathcal{M}_r$  the set of all  $r$ -almost matchings in  $G$ . Given an  $r$ -almost matching  $y$ , we let a *corresponding matching*  $z \in \mathcal{M}$  be a matching obtained by setting to zero the fractional components of  $y$ , and then computing a maximal matching in the resulting set of edges (in particular, we might need to set to 0 some 1 entries of  $y$  to obtain  $z$ ). Notice that  $w^T z \geq w^T y - rw_{\max}$ , where  $w_{\max}$  is the largest weight.

Our patching procedure first constructs a 2-almost matching  $y$ , and then returns a corresponding matching  $z$ . We next show how to compute  $y$ . Let us restrict our attention to the following set of candidate 2-almost matchings. Recall that  $s = x' \Delta x''$  is a set of paths and cycles. We construct an auxiliary graph

$C$ , consisting of one cycle  $(e_0, e_1, \dots, e_{\tau-1})$ , with the following property: there is a bijective mapping between the edges of  $C$  and the edges of  $s$  such that two consecutive edges of  $C$  are either consecutive in some path/cycle or belong to different paths/cycles. This can be easily achieved by cutting each cycle, appending the resulting set of paths one to the other, and gluing together the endpoints of the obtained path. For  $t \in [0, \tau]$ , we define  $s(t) \in [0, 1]^E$  as

$$(s(t))(e) = \begin{cases} 1 & \text{if } e = e_i, i < \lfloor t \rfloor; \\ t - \lfloor t \rfloor & \text{if } e = e_i, i = \lfloor t \rfloor; \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, for  $a, t \in [0, \tau]$ , we define

$$[0, 1]^E \ni s^a(t) = \begin{cases} s(a+t) - s(a) & \text{if } a+t \leq \tau; \\ s(a+t-\tau) + s(\tau) - s(a) & \text{if } a+t > \tau. \end{cases}$$

Intuitively,  $a$  and  $(a+t) \pmod{\tau}$  define a (fractional) subpath of  $C$ , and  $s^a(t)$  is the (fractional) incidence vector corresponding to that subpath. Additionally we define  $y^a(t) := x' \Delta s^a(t)$ . Note that  $y^a(t)$  is equal to  $x'$  and  $x''$  for  $t = 0$  and  $t = \tau$ , respectively.

**Lemma 2.** *For any  $a, t \in [0, \tau]$ ,  $y^a(t)$  is a 2-almost matching.*

*Proof.* One can easily observe that a matching can be obtained by setting the two components of  $y^a(t)$  to zero that correspond to the edges  $e_{\lfloor a \rfloor}$  and  $e_{\lfloor (a+t) \pmod{\tau} \rfloor}$ .

The following lemma shows that, in polynomial time, one can find a 2-almost matching  $y$  with lengths  $\ell^T y$  equal to the lengths of any convex combination of the two matchings  $x'$  and  $x''$ .

**Lemma 3.** *Let  $\mu \in [0, 1]$  and  $x_\mu = \mu x' + (1 - \mu)x''$ . In polynomial time,  $a, t \in [0, \tau]$  can be determined such that  $\ell^T y^a(t) = \ell^T x_\mu$ .*

*Proof.* Let  $f : [0, \tau] \rightarrow \mathbb{R}^2$  be the polygonal curve defined by  $f(t) = \ell^T y^0(t)$ . Since  $f(0) = \ell^T x'$  and  $f(\tau) = \ell^T x''$ , we have by Lemma 1 that there exists  $a, t \in [0, \tau]$  such that  $f^a(t) = \ell^T x_\mu$ . Since  $f^a(t) = \ell^T y^a(t)$ ,  $y := y^a(t)$  satisfies the claim. The values of  $\lfloor a \rfloor$  and  $\lfloor a+t \rfloor$  can be guessed in polynomial time by considering  $O(n^2)$  possibilities. Given those two rounded values, the actual values of  $a$  and  $t$  can be obtained by solving a linear program with a constant number of variables and constraints.

Our patching procedure computes a 2-almost matching  $y = y^a(t)$  with  $\ell^T y = \ell^T x_\mu$ , exploiting the lemma above, and then returns a corresponding matching  $z$ , by applying the procedure explained in the proof of Lemma 2. Trivially,  $\ell^T z \leq \ell^T y = \ell^T x_\mu$ . We next show that, if  $x'$  and  $x''$  have sufficiently large Lagrangian weight, then the weight of  $z$  is close to the weight of  $x_\mu$ .

**Lemma 4.** *Assume  $\mathcal{L}(x') \geq w^* - \Gamma$  and  $\mathcal{L}(x'') \geq w^* - \Gamma$  for some  $\Gamma \in \mathbb{R}_+$ . Then the matching  $z$  returned by the patching procedure satisfies  $w^T z \geq w^T x_\mu - 2w_{\max} - \Gamma$  and  $\ell^T z \leq \ell^T x_\mu$ .*

*Proof.* By Lemma 3 we have  $\ell^T y = \ell^T x_\mu$ , and since  $z \leq y$ , we get  $\ell^T z \leq \ell^T x_\mu$ . Let  $\bar{x}_\mu = x' + x'' - x_\mu = (1 - \mu)x' + \mu x''$ . Since  $\mathcal{L}(x') \geq w^* - \Gamma$ ,  $\mathcal{L}(x'') \geq w^* - \Gamma$  and  $\mathcal{L}$  is linear, we have  $\mathcal{L}(x_\mu) \geq w^* - \Gamma$  and  $\mathcal{L}(\bar{x}_\mu) \geq w^* - \Gamma$ . Recall that  $y = y^a(t)$  for a proper choice of  $a, t \in [0, \tau]$ . Let  $\bar{y} := x' + x'' - y$ . Notice that  $\bar{y} = y^{a'}(\tau - t)$  where  $a' = (a + t) \pmod{\tau}$ , and hence,  $\bar{y}$  is also a 2-almost matching by Lemma 2. Let  $\bar{z}$  be the matching corresponding to  $\bar{y}$  obtained by applying the procedure explained in the proof of Lemma 2 to  $\bar{y}$ . Notice that the pairs  $(z, y)$  and  $(\bar{z}, \bar{y})$  differ on the same two (or less) components. Hence

$$w^T z + w^T \bar{z} + 2w_{\max} \geq w^T y + w^T \bar{y} = w^T x_\mu + w^T \bar{x}_\mu. \quad (2)$$

Since  $y + \bar{y} = x_\mu + \bar{x}_\mu$  and  $\ell^T y = \ell^T x_\mu$ , we get  $\ell^T \bar{y} = \ell^T \bar{x}_\mu$ . Thus,  $\ell^T \bar{z} \leq \ell^T \bar{x}_\mu$  since  $\bar{z} \leq \bar{y}$ . This can be rewritten as  $\mathcal{L}(\bar{z}) - w^T \bar{z} \geq \mathcal{L}(\bar{x}_\mu) - w^T \bar{x}_\mu$ . Since  $\mathcal{L}(\bar{x}_\mu) \geq w^* - \Gamma$  and  $\mathcal{L}(\bar{z}) \leq w^*$ , we obtain  $w^T \bar{z} \leq w^T \bar{x}_\mu + \Gamma$ . Combining this result with (2) implies  $w^T z \geq w^T x_\mu - 2w_{\max} - \Gamma$ .

**The Algorithm.** Our PTAS works as follows. First it guesses the  $6/\varepsilon$  heaviest edges  $E_H$  in the optimum solution, and reduces the problem consequently. Then it computes a vertex  $x^* \in P_M$  of the polytope  $\{x \in P_M \mid \ell^T x \leq L\}$  of maximum weight  $w^* := w^T x^*$ . As  $x^*$  is a vertex solution of the polytope  $P_M$  with two additional constraints, it lies on a face of  $P_M$  of dimension at most two. Hence, by Carathéodory's Theorem,  $x^*$  can be expressed as a convex combination  $x^* = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3$  of three matchings  $x_1, x_2, x_3 \in P_M$ . Let  $\mu' = \alpha_1/(\alpha_1 + \alpha_2)$  and  $\mu'' = (\alpha_1 + \alpha_2)/(\alpha_1 + \alpha_2 + \alpha_3)$ . Applying Lemma 4 to  $x_1$  and  $x_2$  with  $\mu = \mu'$ , a matching  $z'$  is obtained. Applying Lemma 4 to  $z'$  and  $x_3$  with  $\mu = \mu''$ , we obtain a matching  $z''$ . The algorithm returns  $z''$  and  $E_H$ .

*Proof (Theorem 4).* Consider the algorithm above. The initial guessing can be performed in  $O(|E|^{6/\varepsilon})$  time. Since it is possible to efficiently separate over  $P_M$ ,  $x^*$  can be computed in polynomial time [24]. The same holds for the decomposition of  $x^*$  into three matchings by standard techniques (see for example [23]). Lemma 3 implies that the patching can be done in polynomial time. Hence the proposed algorithm runs in polynomial time as claimed.

Since  $\mathcal{L}(x^*) = w^*$  and  $\mathcal{L}(x) \leq w^*$  for  $x \in P_M$ , we get  $\mathcal{L}(x_1) = \mathcal{L}(x_2) = \mathcal{L}(x_3) = w^*$ . Let  $u := \mu' x_1 + (1 - \mu') x_2$  and  $v := \mu'' z' + (1 - \mu'') x_3$ . By Lemma 4, matching  $z'$  satisfies  $\ell^T z' \leq \ell^T u$  and  $w^T z' \geq w^T u - 2w_{\max}$ . Since  $u$  is a convex combination of  $x_1$  and  $x_2$ , we have  $\mathcal{L}(u) = w^*$ . Furthermore, by the relations between the lengths and weight of  $z'$  and  $u$ , we get  $\mathcal{L}(z') \geq \mathcal{L}(u) - 2w_{\max} = w^* - 2w_{\max}$ .

By Lemma 4, matching  $z''$  satisfies  $\ell^T z'' \leq \ell^T v$  and  $w^T z'' \geq w^T v - 4w_{\max}$ . We observe that  $z''$  satisfies the budget constraints since

$$\ell^T z'' \leq \ell^T v = \ell^T((\alpha_1 + \alpha_2)z' + \alpha_3 x_3) \leq \ell^T((\alpha_1 + \alpha_2)u + \alpha_3 x_3) = \ell^T x^* \leq L.$$

Furthermore,

$$\begin{aligned} w^T z'' &\geq w^T v - 4w_{\max} = w^T((\alpha_1 + \alpha_2)z' + \alpha_3 x_3) - 4w_{\max} \\ &\geq w^T((\alpha_1 + \alpha_2)u + \alpha_3 x_3) - 6w_{\max} = w^* - 6w_{\max}. \end{aligned}$$

Let  $OPT'$  be an optimum solution to the reduced problem. Of course,  $w^* \geq w(OPT')$ . Furthermore, the weight of the guessed edges  $E_H$  is at least  $6/\varepsilon w_{max}$ . Since  $w(OPT) = w(E_H) + w(OPT')$ , we can conclude that the solution returned by the algorithm has weight at least  $w(E_H)(1 - \varepsilon) + w(OPT') \geq (1 - \varepsilon)w(OPT)$ .

## References

1. Aggarwal, V., Aneja, Y.P., Nair, K.P.K.: Minimal spanning tree subject to a side constraint. *Computers & Operations Research* 9, 287–296 (1982)
2. Barahona, F., Pulleyblank, W.R.: Exact arborescences, matchings and cycles. *Discrete Applied Mathematics* 16(2), 91–99 (1987)
3. Berger, A., Bonifaci, V., Grandoni, F., Schäfer, G.: Budgeted matching and budgeted matroid intersection via the gasoline puzzle. To appear in *Mathematical Programming*, Preliminary version in IPCO 2008
4. Bilu, V., Goyal, V., Ravi, R., Singh, M.: On the crossing spanning tree problem. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) *RANDOM 2004 and APPROX 2004*. LNCS, vol. 3122, pp. 51–64. Springer, Heidelberg (2004)
5. Camerini, P., Galbiati, G., Maffioli, F.: Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms* 13, 258–273 (1992)
6. Chekuri, C., Vondrák, J., Zenklusen, R.: Dependent randomized rounding for matroid polytopes and applications (2009), <http://arxiv.org/abs/0909.4348>
7. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman, New York (1979)
8. Grandoni, F., Ravi, R., Singh, M.: Iterative rounding for multi-objective optimization problems. In: Fiat, A., Sanders, P. (eds.) *ESA 2009*. LNCS, vol. 5757, pp. 95–106. Springer, Heidelberg (2009)
9. Hurkens, C.A., Lovász, L., Schrijver, A., Tardos, E.: How to tidy up your set system. *Combinatorics*, 309–314 (1988)
10. Hassin, R.: Approximation schemes for the restricted shortest path problem. *Mathematics of Operation Research* 17(1), 36–42 (1992)
11. Hassin, R., Levin, A.: An efficient polynomial time approximation scheme for the constrained minimum spanning tree problem using matroid intersection. *SIAM Journal on Computing* 33(2), 261–268 (2004)
12. Jain, K.: A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica* 21, 39–60 (2001)
13. Korte, B., Vygen, J.: Combinatorial optimization. Springer, Heidelberg (2008)
14. Lorenz, D., Raz, D.: A simple efficient approximation scheme for the restricted shortest paths problem. *Operations Research Letters* 28, 213–219 (2001)
15. Munkres, J.R.: Topology, 2nd edn. Prentice-Hall, Englewood Cliffs (2000)
16. Marathe, M.V., Ravi, R., Sundaram, R., Ravi, S.S., Rosenkrantz, D.J., Hunt III, H.B.: Bicriteria network design problems. In: Fülpö, Z., Gecseg, F. (eds.) *ICALP 1995*. LNCS, vol. 944, pp. 487–498. Springer, Heidelberg (1995)
17. Mulmuley, K., Vazirani, U., Vazirani, V.: Matching is as easy as matrix inversion. *Combinatorica* 7(1), 101–104 (1987)
18. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of Web sources. In: *FOCS*, pp. 86–92 (2000)
19. Ravi, R.: Rapid rumor ramification: Approximating the minimum broadcast time. In: *FOCS*, pp. 202–213 (1994)

20. Ravi, R.: Matching based augmentations for approximating connectivity problems. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 13–24. Springer, Heidelberg (2006) (invited lecture)
21. Ravi, R., Goemans, M.X.: The constrained minimum spanning tree problem (extended abstract). In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 66–75. Springer, Heidelberg (1996)
22. Ravi, R., Marathe, M.V., Ravi, S.S., Rosenkrantz, D.J., Hunt, H.B.: Many birds with one stone: Multi-objective approximation algorithms. In: STOC, pp. 438–447 (1993)
23. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Chichester (1998)
24. Schrijver, A.: Combinatorial optimization, polyhedra and efficiency. Springer, Heidelberg (2003)
25. Warburton, A.: Approximation of Pareto optima in multiple-objective, shortest path problems. Operations Research 35, 70–79 (1987)

# Algorithmic Meta-theorems for Restrictions of Treewidth

Michael Lampis

Computer Science Department,  
Graduate Center, City University of New York  
[mlampis@gc.cuny.edu](mailto:mlampis@gc.cuny.edu)

**Abstract.** Possibly the most famous algorithmic meta-theorem is Courcelle’s theorem, which states that all MSO-expressible graph properties are decidable in linear time for graphs of bounded treewidth. Unfortunately, the running time’s dependence on the formula describing the problem is in general a tower of exponentials of unbounded height, and there exist lower bounds proving that this cannot be improved even if we restrict ourselves to deciding FO logic on trees.

We investigate whether this parameter dependence can be improved by focusing on two proper subclasses of the class of bounded treewidth graphs: graphs of bounded vertex cover and graphs of bounded max-leaf number. We prove stronger algorithmic meta-theorems for these more restricted classes of graphs. More specifically, we show it is possible to decide any FO property in both of these classes with a singly exponential parameter dependence and that it is possible to decide MSO logic on graphs of bounded vertex cover with a doubly exponential parameter dependence. We also prove lower bound results which show that our upper bounds cannot be improved significantly, under widely believed complexity assumptions. Our work addresses an open problem posed by Michael Fellows.

## 1 Introduction

Algorithmic metatheorems are general statements of the form “*All problems sharing property P, restricted to a class of inputs I can be solved efficiently*”. The archetypal, and possibly most celebrated, such metatheorem is Courcelle’s theorem which states that every graph property expressible in monadic second-order ( $\text{MSO}_2$ ) logic is decidable in linear time if restricted to graphs of bounded treewidth [3]. Metatheorems have been a subject of intensive research in the last years producing a wealth of interesting results. Some representative examples of metatheorems with a flavor similar to Courcelle’s can be found in the work of Frick and Grohe [12], where it is shown that all properties expressible in first order (FO) logic are solvable in linear time on planar graphs, and the work of Dawar et al. [5], where it is shown that all FO-definable optimisation problems admit a PTAS on graphs excluding a fixed minor (see [14] and [15] for more results on the topic).

Many interesting extensions have followed Courcelle's seminal result: for instance, Courcelle's theorem has been extended to logics more suitable for the expression of optimisation problems [1]. It has also been investigated whether it's possible to obtain similar results for larger graph classes (see [4] for a metatheorem for bounded cliquewidth graphs, [11] for corresponding hardness results and [17] for hardness results for graphs of small but unbounded treewidth). Finally, lower bound results have been shown proving that the running times predicted by Courcelle's theorem can not be improved significantly in general [13].

This lower bound result is one of the main motivations of this work, because in some ways it is quite devastating. Though Courcelle's theorem shows that a vast class of problems is solvable in linear time on graphs of bounded treewidth, the “hidden constant” in this running time, that is, the running time's dependence on the input's other parameters, which are the graph's treewidth and the formula describing the problem, is in fact (in the worst case) a tower of exponentials. Unfortunately, in [13] it is shown that this tower of exponentials is unavoidable even if we restrict ourselves to deciding FO logic on trees.

In this paper our aim is to investigate if it is possible to go around this harsh lower bound by restricting the considered class of input graphs further. In other words, we are looking for meta-theorems which would imply that all of FO or MSO logic can be solved in time not only linear in the size of the graph, but also depending more reasonably on the secondary parameters, if we are willing to give up some of the generality of the class of bounded-treewidth graphs. We concentrate on two graph classes: graphs of bounded vertex cover and graphs of bounded max-leaf number. We note that the investigation of the existence of stronger meta-theorems for these classes has been posed explicitly as an open problem by Fellows in [7].

Though graphs of bounded vertex cover or max-leaf number are considerably more restricted than bounded treewidth graphs, these classes are still interesting from the algorithmic point of view and the complexity of hard problems parameterized by vertex cover or max-leaf number has been investigated in the past ([9], [8]). Furthermore, as mentioned, strong lower bounds are known to apply to slightly more general classes: for bounded feedback vertex set and bounded pathwidth graphs even FO logic is non-elementary, while even for binary trees (thus for graphs of bounded treewidth and max degree) FO logic is at least triply exponential (again by [13]). Bounded vertex cover and bounded max-leaf number evade all these lower bound arguments so it's natural to ask what is exactly the complexity of FO and MSO logic for these classes of graphs?

The main results of this paper show that meta-theorems stronger than Courcelle's can indeed be shown for these classes of graphs. In addition, we show that our meta-theorems for vertex cover cannot be significantly improved under standard complexity assumptions.

Specifically, for the class of graphs of vertex cover bounded by  $k$  we show that

- All graph problems expressible with an FO formula  $\phi$  can be solved in time linear in the graph size and singly exponential in  $k$  and  $|\phi|$ .

- All graph problems expressible with an  $\text{MSO}_1$  formula  $\phi$  can be solved in time linear in the graph size and doubly exponential in  $k$  and  $|\phi|$ .
- Unless  $n$ -variable 3SAT can be solved in time  $2^{o(n)}$  (that is, unless the Exponential Time Hypothesis fails), then no  $f(k, \phi) \cdot \text{poly}(|G|)$  algorithm exists to decide  $\text{MSO}$  logic on graphs for any  $f(k, \phi) = 2^{2^{o(k+|\phi|)}}$ .
- Unless  $\text{FPT}=\text{W}[1]$ , there is no algorithm which can decide if an  $\text{FO}$  formula  $\phi$  with  $q$  quantifiers holds in a graph  $G$  of vertex cover  $k$  in time  $f(k, q)n^c$ , for any  $f(k, q) = 2^{O(k+q)}$ .

Furthermore, for the class of graphs of max-leaf number bounded by  $k$  we show that

- All graph problems expressible with an  $\text{FO}$  formula  $\phi$  can be solved in time linear in the graph size and singly exponential in  $k$  and  $|\phi|$ .

Our upper bounds rely on techniques different from the standard dynamic programming on decompositions usually associated with treewidth. For max-leaf number we rely on the characterization of bounded max-leaf number graphs from [16] also used heavily in [8] and the fact that  $\text{FO}$  logic has limited counting power in paths. For vertex cover we exploit an observation that for  $\text{FO}$  logic two vertices that have the same neighbors are “equivalent” in a sense we will make precise. We state our results in this case in terms of a new graph “width” parameter that captures this graph property more precisely than bounded vertex cover. We call the new parameter neighborhood diversity, and the upper bounds for vertex cover follow by showing that bounded vertex cover is a special case of bounded neighborhood diversity. Our essentially matching lower bounds on the other hand are shown for vertex cover. In the last section of this paper we prove some additional results for neighborhood diversity, beyond the algorithmic meta-theorems of the rest of the paper, which we believe indicate that neighborhood diversity might be a graph structure parameter of independent interest and that its algorithmic and graph-theoretic properties may merit further investigation.

Due to space constraints, some of the proofs have been omitted and will appear in the full version of this paper.

## 2 Definitions and Preliminaries

**Model Checking,  $\text{FO}$  and  $\text{MSO}$  Logic.** In this paper we will describe algorithmic meta-theorems, that is, general methods for solving all problems belonging in a class of problems. However, the presentation is simplified if one poses this approach as an attack on a single problem, the model checking problem. In the model checking problem we are given a logic formula  $\phi$ , expressing a graph property, and a graph  $G$ , and we must decide if the property described by  $\phi$  holds in  $G$ . In that case, we write  $G \models \phi$ . Clearly, if we can describe an efficient algorithm for model checking for a specific logic, this will imply the existence of efficient algorithms for all problems expressible in this logic. Let us now give

more details about the logics we will deal with and the graphs which will be our input instances.

Our universe of discourse will be labeled, colored graphs. Specifically, we assume that the first part of the input is an undirected graph  $G(V, E)$ , a set of labels  $L$ , each associated with a vertex of  $V$  and a set of subsets of  $V$ ,  $\mathcal{C} = \{C_1, C_2, \dots, C_c\}$ , which we refer to as color classes. The interesting case here is unlabeled, uncolored graphs (that is,  $L = \mathcal{C} = \emptyset$ ), but the additional generality in the definition of the problem makes it easier to describe a recursive algorithm.

The formulas of FO logic are those which can be constructed using vertex variables, denoted usually by  $x_i, y_i, \dots$ , vertex labels denoted by  $l_i$ , color classes denoted by  $C_i$ , the predicates  $E(x_i, x_j)$ ,  $x_i \in C_j$ ,  $x_i = x_j$  operating on vertex variables or labels, standard propositional connectives and the quantifiers  $\exists, \forall$  operating on vertex variables. The semantics are defined in the usual way, with the  $E()$  predicate being true if  $(x_i, x_j) \in E$ .

For MSO logic the additional property is that we now introduce set variables denoted by  $X_i$  and allow the quantifiers and the  $\in$  predicate to operate on them. The semantics are defined in the obvious way.

If the set variables are allowed to range over sets of vertices only then the logic is sometimes referred to as  $\text{MSO}_1$ . A variation is  $\text{MSO}_2$  logic, where one is also allowed to use set variables that range over sets of edges. To accommodate for this case one also usually modifies slightly the definition of FO formulas to allow edge variables and an incidence predicate  $I(v, e)$  which is true if edge  $e$  is incident on vertex  $v$ .

**Bounded Vertex Cover and neighborhood diversity.** We will work extensively with graphs of bounded vertex cover, that is, graphs for which there exists a small set of vertices whose removal also removes all edges. We will usually denote the size of a graph's vertex cover by  $k$ . Note that there exist linear-time FPT algorithms for finding an optimal vertex cover in graphs where  $k$  is small (see e.g. [2]).

Our technique relies on the fact that in a graph of vertex cover  $k$ , the vertices outside the vertex cover can be partitioned into at most  $2^k$  sets, such that all the vertices in each set have exactly the same neighbors outside the set and each set contains no edges inside it. Since we do not make use of any other special property of graphs of small vertex cover, we are motivated to define a new graph parameter, called neighborhood diversity, which intuitively seems to give the largest graph family to which we can apply our method in a straightforward way.

**Definition 1.** *We will say that two vertices  $v, v'$  of a graph  $G(V, E)$  have the same type iff they have the same colors and  $N(v) \setminus \{v'\} = N(v') \setminus \{v\}$ .*

**Definition 2.** *A colored graph  $G(V, E)$  has neighborhood diversity at most  $w$ , if there exists a partition of  $V$  into at most  $w$  sets, such that all the vertices in each set have the same type.*

**Lemma 1.** *If an uncolored graph has vertex cover at most  $k$ , then it has neighborhood diversity at most  $2^k + k$ .*

In Section 7 we will show that neighborhood diversity can be computed in polynomial time and also prove some results which indicate it may be an interesting parameter in its own right. However, until then our main focus will be graphs of bounded vertex cover. We will prove most of our algorithmic results in terms of neighborhood diversity and then invoke Lemma II to obtain our main objective. We will usually assume that a partition of the graph into sets with the same neighbors is given to us, because otherwise one can easily be found in linear time by using the mentioned linear-time FPT algorithm for vertex cover and Lemma II.

**Bounded Max-Leaf Number.** We say that a connected graph  $G$  has max-leaf number at most  $l$  if no spanning tree of  $G$  has more than  $l$  leaves. The algorithmic properties of this class of graphs have been investigated in the past [6,10,8]. In this paper we rely heavily on a characterization of bounded max-leaf graphs by Kleitman and West [16] which is also heavily used in [8].

**Theorem 1.** [16] *If a graph  $G$  has max-leaf number at most  $l$ , then  $G$  is a subdivision of a graph on  $O(l)$  vertices.*

What this theorem tells us intuitively is that in a graph  $G(V, E)$  with max-leaf number  $l$  there exists a set  $S$  of  $O(l)$  vertices such that  $G[V \setminus S]$  is a collection of  $O(l^2)$  paths. Furthermore, only the endpoints of the paths can be connected to vertices of  $S$  in  $G$ .

It is well-known that a graph of max-leaf number at most  $l$  has a path decomposition of width at most  $2l$ . Furthermore, it must have maximum degree at most  $l$ . Bounded max-leaf number graphs are therefore a subclass of the intersection of bounded pathwidth and bounded degree graphs (in fact, they are a proper subclass, as witnessed by the existence of say  $2 \times n$  grids). Let us mention again that deciding FO logic on binary trees has at least a triply exponential parameter dependence, so the results we present for graphs of bounded max-leaf number can also be seen as an improvement on the currently known results for FO logic on bounded degree graphs, for this more restricted case.

### 3 FO Logic for Bounded Vertex Cover

In this Section we show how any FO formula can be decided on graphs of bounded vertex cover, with a singly exponential parameter dependence. Our main argument is that for FO logic, two vertices which have the same neighbors are essentially equivalent. We will state our results in the more general case of bounded neighborhood diversity and then show the corresponding result for bounded vertex cover as a corollary.

**Lemma 2.** *Let  $G(V, E)$  be a graph and  $\phi(x)$  a FO formula with one free variable. Let  $v, v' \in V$  be two distinct unlabeled vertices of  $G$  that have the same type. Then  $G \models \phi(v)$  iff  $G \models \phi(v')$ .*

*Proof.* (Sketch) Recall that the standard way of deciding an FO formula on a graph is, whenever we encounter an existential quantifier to try all possible choices of a vertex for that variable. This creates an  $n$ -ary decision tree with height equal to the number of quantifiers in  $\phi(x)$ . Every leaf corresponds to a choice of vertices for the  $q$  quantified variables, which makes the formula true or false. Internal nodes are evaluated as the disjunction (for existential quantifiers) or conjunction (for universal quantifiers) of their children.

It is possible to create a one-to-one correspondence between the trees for  $\phi(v)$  and  $\phi(v')$ , by essentially exchanging  $v$  and  $v'$ , showing that  $\phi(v)$  and  $\phi(v')$  are equivalent.  $\square$

**Theorem 2.** *Let  $\phi$  be a FO sentence of quantifier depth  $q$ . Let  $G(V, E)$  be a labeled colored graph with neighborhood diversity at most  $w$  and  $l$  labeled vertices. Then, there is an algorithm that decides if  $G \models \phi$  in time  $O((w + l + q)^q \cdot |\phi|)$ .*

**Corollary 1.** *There exists an algorithm which, given a FO sentence  $\phi$  with  $q$  variables and an uncolored, unlabeled graph  $G$  with vertex cover at most  $k$ , decides if  $G \models \phi$  in time  $2^{O(kq+q \log q)}$ .*

Thus, the running time is (only) singly exponential in the parameters, while a straightforward observation that bounded vertex cover graphs have bounded treewidth and an application of Courcelle's theorem would in general have a non-elementary running time. Of course, a natural question to ask now is whether it is possible to do even better, perhaps making the exponent linear in the parameter, which is  $(k + q)$ . As we will see later on, this is not possible if we accept some standard complexity assumptions.

## 4 FO Logic for Bounded Max-Leaf Number

In this section we describe a model checking algorithm for FO logic on graphs of small max-leaf number. Our main tool is the mentioned observation that all but a small fraction of the vertices have degree 2, and therefore (since we assume without loss of generality that the graph is connected) induce paths. We call a maximal set of connected vertices of degree 2 a topo-edge.

Our main argument is that when a topo-edge is very long (exponentially long in the number of quantifiers of the first-order sentence we are model checking) its precise length does not matter.

To make this more precise, let us first define a similarity relation on graphs.

**Definition 3.** *Let  $G_1, G_2$ , be two graphs. For a given  $q$  we will say that  $G_1$  and  $G_2$  are  $q$ -similar and write  $G_1 \sim_q G_2$  iff  $G_1$  contains a topo-edge of order at least  $2^{q+1}$  consisting of unlabeled vertices, call it  $P$ , and  $G_2$  can be obtained from  $G_1$  by contracting one of the edges of  $P$ . We denote the transitive closure of the relation  $\sim_q$  as  $\sim_q^*$ .*

Our main technical tool is now the following lemma.

**Lemma 3.** Let  $\phi$  be a FO formula with  $q$  quantifiers. Then, for any two graphs  $G_1, G_2$  if  $G_1 \sim_q G_2$  then  $G_1 \models \phi$  iff  $G_2 \models \phi$ . Therefore, if  $G_1 \sim_q^* G_2$  then  $G_1 \models \phi$  iff  $G_2 \models \phi$ .

Now we are ready to state our main result of this section.

**Theorem 3.** Let  $G$  be a graph on  $n$  vertices with max-leaf number  $k$  and  $\phi$  a FO formula with  $q$  quantifiers. Then, there exists an algorithm for deciding if  $G \models \phi$  running in time  $\text{poly}(n) + 2^{O(q^2+q\log k)}$ .

*Proof.* By applying Theorem 1 we know that  $G$  can be partitioned into a set of at most  $O(k)$  vertices of degree at least 3 and a collection of paths. By applying Lemma 3 we know that there exists a  $G'$  such that  $G \sim_q^* G'$  and  $G'$  consists of the same  $O(k)$  vertices of degree at least 3 and at most  $O(k^2)$  paths whose length is at most  $2^{q+1}$ . Of course,  $G'$  can be found in time polynomial in  $n$ .

Now, we can apply the straightforward algorithm to model check  $\phi$  on  $G'$ . For every quantifier we have at most  $O(k+q) + O(k(k+q)2^{q+1})$  choices, which is  $2^{O(q+\log k)}$ . Exhausting all possibilities for each vertex gives the promised running time.  $\square$

## 5 MSO Logic for Bounded Vertex Cover

First, let us state a helpful extension of the results of the Section 3. From the following Lemma it follows naturally that the model checking problem for  $\text{MSO}_1$  logic on bounded vertex cover graphs is in XP, that is, solvable in polynomial time for constant  $\phi$  and  $k$ , but our objective later on will be to do better.

**Lemma 4.** Let  $\phi(X)$  be an  $\text{MSO}_1$  formula with a free set variable  $X$ . Let  $G$  be a graph and  $S_1, S_2$  two sets of vertices of  $G$  such that all vertices of  $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$  are unlabeled and have the same type and furthermore  $|S_1 \setminus S_2| = |S_2 \setminus S_1|$ . Then  $G \models \phi(S_1)$  iff  $G \models \phi(S_2)$ .

Our main tool in this section is the following lemma.

**Lemma 5.** Let  $\phi(X)$  be an  $\text{MSO}_1$  formula with one free set variable  $X$ ,  $q_V$  quantified vertex variables and  $q_S$  quantified set variables. Let  $G$  be a graph and  $S_1, S_2$  two sets of vertices of  $G$  such that all vertices of  $(S_1 \setminus S_2) \cup (S_2 \setminus S_1)$  are unlabeled and belong in the same type  $T$ . Suppose that both  $|S_1 \cap T|$  and  $|S_2 \cap T|$  fall in the interval  $[2^{q_S} q_V, |T| - 2^{q_S} q_V - 1]$ . Then  $G \models \phi(S_1)$  iff  $G \models \phi(S_2)$ .

*Proof.* (Sketch) We can assume without loss of generality that  $S_1 \subseteq S_2$ , thanks to Lemma 4. In fact, we may assume that  $S_2 = S_1 \cup \{u\}$  for some vertex  $u$  and repeated applications of the same argument yield the claimed result.

Our argument is that the truth of  $\phi(S_1)$  and  $\phi(S_2)$  can be decided by an algorithm which checks for each set variable every combination of sizes that its intersection has with each type and for each vertex variable one representative of each type. If  $u$  is never picked as a representative then the algorithm must

give the same answer for  $\phi(S_1)$  and  $\phi(S_2)$ . This can be guaranteed if the type  $u$  belongs in always has more than  $q_V$  vertices. Every time the algorithm picks a value for a set variable, the type  $u$  belongs in is partitioned in two. Since the only thing that matters to the algorithm is the size of the set's intersection with  $u$ 's type, we are free to select a set that puts  $u$  in the larger of the two new sub-types. Because  $S_1 \cap T$  and  $S_2 \cap T$  have constrained sizes, we can always guarantee that  $u$  is never selected.  $\square$

**Theorem 4.** *There exists an algorithm which, given a graph  $G$  with  $l$  labels, neighborhood diversity at most  $w$  and an  $\text{MSO}_1$  formula  $\phi$  with at most  $q_S$  set variables and  $q_V$  vertex variables, decides if  $G \models \phi$  in time  $2^{O(2^{q_S}(w+l)q_S^2q_V \log q_V)} \cdot |\phi|$ .*

**Corollary 2.** *There exists an algorithm which, given an  $\text{MSO}_1$  sentence  $\phi$  with  $q$  variables and an uncolored, unlabeled graph  $G$  with vertex cover at most  $k$ , decides if  $G \models \phi$  in time  $2^{2^{O(k+q)}}$ .*

Again, this gives a dramatic improvement compared to Courcelle's theorem, though exponentially worse than the case of FO logic. This is an interesting point to consider because for treewidth there does not seem to be any major difference between the complexities of model checking FO and  $\text{MSO}_1$  logic.

The natural question to ask here is once again, can we do significantly better? For example, perhaps the most natural question to ask is, is it possible to solve this problem in  $2^{2^{o(k+q)}}$ ? As we will see later on, the answer is no, if we accept some standard complexity assumptions.

Finally, let us briefly discuss the case of  $\text{MSO}_2$  logic. In general this logic is more powerful than  $\text{MSO}_1$ , so it is not straightforward to extend Theorem 4 in this case. However, if we are not interested in neighborhood diversity but just in vertex cover we can observe that all edges in a graph with vertex cover of size  $k$  have one of their endpoints in one of the  $k$  vertices of the vertex cover. Thus, any edge set  $X$  can be written as the union of  $k$  edge sets. In turn, each of these  $k$  edge sets can easily be replaced by vertex sets, without loss of information, since we already know one of the endpoints of each of these edges. Using this trick we can replace every edge set variable in an  $\text{MSO}_2$  sentence with  $k$  vertex set variables, thus obtaining a  $2^{2^{O(kq)}}$  algorithm for  $\text{MSO}_2$  logic on graphs of bounded vertex cover.

## 6 Lower Bounds

In this Section we will prove some lower bound results for the model checking problems we are dealing with. Our proofs rely on a construction which reduces SAT to a model checking problem on a graph with small vertex cover.

Given a propositional 3-CNF formula  $\phi_p$  with  $n$  variables and  $m$  clauses, we want to construct a graph  $G$  that encodes its structure, while having a small vertex cover. The main problem is encoding numbers up to  $n$  with graphs of small vertex cover but this can easily be achieved by using the binary representation of numbers.

We begin constructing a graph by adding  $7 \log n$  vertices, call them  $u_{(i,j)}$ ,  $1 \leq i \leq 7, 1 \leq j \leq \log n$ . Add all edges of the form  $(u_{(i,j)}, u_{(k,j)})$  (so we now have  $\log n$  disjoint copies of  $K_7$ ). Let  $N_i = \{u_{(i,j)} \mid 1 \leq j \leq \log n\}$ .

For every variable  $x_i$  in  $\phi_p$  add a new vertex to the graph, call it  $v_i$ . Define for every number  $i$  the set  $X(i) = \{j \mid \text{the } j\text{-th bit of the binary representation of } i \text{ is } 1\}$ . Add the edges  $(v_i, u_{(1,j)}), j \in X(i)$ , that is connect every variable vertex with the vertices of  $N_1$  that correspond to the binary representation of its index. Let  $U = \{v_i \mid 1 \leq i \leq n\}$  be the vertices corresponding to variables.

For every clause  $c_i$  in  $\phi_p$  add a new vertex to the graph, call it  $w_i$ . If the first literal in  $c_i$  is a positive variable  $x_k$  then add the edges  $(w_i, u_{(2,j)}), j \in X(k)$ . If the first literal is a negated variable  $\neg x_k$ , add the edges  $(w_i, u_{(3,j)}), j \in X(k)$ . Proceed in a similar way for the second and third literal, that is, if the second literal is positive connect  $w_i$  with the vertices that correspond to the binary representation of the variable in  $N_4$ , otherwise in  $N_5$ . For the third literal do the same with  $N_6$  or  $N_7$ . Let  $W = \{w_i \mid 1 \leq i \leq m\}$  be the vertices corresponding to clauses.

Finally, set the color classes to be  $\{N_1, N_2, \dots, N_7, U, W\}$ .

Now, looking at the graph it is easy to see if a vertex  $v_i$  corresponds to a variable that appears positive in the clause represented by a vertex  $w_i$ . They must satisfy the formula

$$pos(v_i, w_j) = \bigvee_{k=2,4,6} \forall x(x \in N_1 \rightarrow \exists y((E(v_i, x) \leftrightarrow E(w_j, y)) \wedge y \in N_k \wedge E(x, y)))$$

It is not hard to define  $neg(v_i, w_j)$  in a similar way. Now it is straight-forward to check if  $\phi_p$  was satisfiable:

$$\begin{aligned} \phi = \exists S (\forall xx \in S \rightarrow x \in U) \wedge (\forall ww \in W \rightarrow \exists xx \in U \wedge \\ ((pos(x, w) \wedge x \in S) \vee (neg(x, w) \wedge x \notin S))) \end{aligned}$$

Clearly,  $\phi$  holds in the constructed graph iff  $\phi_p$  is satisfiable.  $S$  corresponds to the set of variables set to true in a satisfying assignment. Let us also briefly remark that it is relatively easy to eliminate the colors and labels from the construction above, therefore the lower bounds given below apply to the natural form of the problem.

**Lemma 6.**  $G \models \phi$  iff  $\phi_p$  is satisfiable. Furthermore,  $\phi$  has size  $O(1)$  and  $G$  has a vertex cover of size  $O(\log n)$ .

**Theorem 5.** Let  $\phi$  be a MSO formula with  $q$  quantifiers and  $G$  a graph with vertex cover  $k$ . Then, unless 3-SAT can be solved in time  $2^{o(n)}$ , there is no algorithm which decides if  $G \models \phi$  in time  $O(2^{2^{o(k+q)}} \cdot \text{poly}(n))$ .

**Theorem 6.** Let  $\phi$  be a FO formula with  $q_v$  vertex quantifiers and  $G$  a graph with vertex cover  $k$ . Then, unless FPT=W[1], there is no algorithm which decides if  $G \models \phi$  in time  $O(2^{O(k+q_v)} \cdot \text{poly}(n))$ .

## 7 Neighborhood Diversity

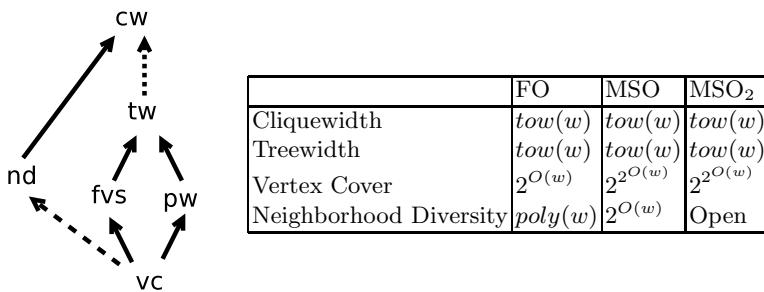
In this Section we give some general results on the new graph parameter we have defined, neighborhood diversity. We will use  $nd(G)$ ,  $tw(G)$ ,  $cw(G)$  and  $vc(G)$  to denote the neighborhood diversity, treewidth, cliquewidth and minimum vertex cover of a graph  $G$ . We will call a partition of the vertex set of a graph  $G$  into  $w$  sets such that all vertices in every set share the same type a neighborhood partition of width  $w$ .

First, some general results

- Theorem 7.** 1. Let  $V_1, V_2, \dots, V_w$  be a neighborhood partition of the vertices of a graph  $G(V, E)$ . Then each  $V_i$  induces either a clique or an independent set. Furthermore, for all  $i, j$  the graph either includes all possible edges from  $V_i$  to  $V_j$  or none.
2. For every graph  $G$  we have  $nd(G) \leq 2^{vc(G)} + vc(G)$  and  $cw(G) \leq nd(G) + 1$ . Furthermore, there exist graphs of constant treewidth and unbounded neighborhood diversity and vice-versa.
3. There exists an algorithm which runs in polynomial time and given a graph  $G(V, E)$  finds a neighborhood partition of the graph with minimum width.

Taking into account the observations of Theorem 7 we summarize what we know about the graph-theoretic and algorithmic properties of neighborhood diversity and related measures in Figure 1.

There are several interesting points to make here. First, though our work is motivated by a specific goal, beating the lower bounds that apply to graphs of bounded treewidth by concentrating on a special case, it seems that what we have achieved is at least somewhat better; we have managed to improve the algorithmic meta-theorems that were known by focusing on a class which is not



**Fig. 1.** A summary of the relations between neighborhood diversity and other graph widths. Included are cliquewidth, treewidth, pathwidth, feedback vertex set and vertex cover. Arrows indicate generalization, for example bounded vertex cover is a special case of bounded feedback vertex set. Dashed arrows indicate that the generalization may increase the parameter exponentially, for example treewidth  $w$  implies cliquewidth at most  $2^w$ . The table summarizes the best known model checking algorithm's dependence on each width for the corresponding logic.

necessarily smaller than bounded treewidth, only different. However, our class is a special case of another known width which generalizes treewidth as well, namely cliquewidth. Since the lower bound results which apply to treewidth apply to cliquewidth as well, this work can perhaps be viewed more appropriately as an improvement on the results of [4] for bounded cliquewidth graphs.

Second, is the case of  $\text{MSO}_2$  logic. The very interesting hardness results shown in [11] demonstrate that the tractability of  $\text{MSO}_2$  logic is in a sense the price one has to pay for the additional generality that cliquewidth provides over treewidth. It is natural to ask if the results of [11] can be strengthened to apply to neighborhood diversity or  $\text{MSO}_2$  logic can be shown tractable parameterized by neighborhood diversity.

Though we cannot yet fully answer the above question related to  $\text{MSO}_2$ , we can offer some first indications that this direction might merit further investigation. In [11] it is shown that  $\text{MSO}_2$  model checking is not fixed-parameter tractable when the input graph's cliquewidth is the parameter by considering three specific  $\text{MSO}_2$ -expressible problems and showing that they are W-hard. The problems considered are Hamiltonian cycle, Graph Chromatic Number and Edge Dominating Set. We can show that these three problems can be solved efficiently on graphs of small neighborhood diversity. Since small neighborhood diversity is a special case of small cliquewidth, where these problems are hard, this result could be of independent interest.

**Theorem 8.** *Given a graph  $G$  whose neighborhood diversity is  $w$ , there exist algorithms running in time  $O(f(w) \cdot \text{poly}(|G|))$  that decide Hamiltonian cycle, Graph Chromatic Number and Edge Dominating Set.*

## 8 Conclusions and Open Problems

In this paper we presented algorithmic meta-theorems which improve the running times implied by previously known meta-theorems for more restricted inputs. In this way we have partially explored the trade-off which can be achieved between running time and generality. This is an interesting area for further investigations and much more can be done.

For bounded max-leaf number the complexity of  $\text{MSO}$  logic is unknown. Quite likely, it is possible to improve upon the Courcelle's theorem for this case as well, but the problem remains open. Also, it would be nice to obtain a lower bound for  $\text{FO}$  logic in this case showing that it is impossible to achieve  $2^{o(q^2)}$ , i.e. that the exponent must be quadratic. For neighborhood diversity the most interesting open problem is the complexity of  $\text{MSO}_2$ .

Going further, it would also make sense to investigate whether restricting the model checking problem to graphs of bounded vertex cover or max-leaf number can also allow us to solve logics wider than  $\text{MSO}_2$ . Some indications that this may be possible are given in [9].

## References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *J. Algorithms* 12(2), 308–340 (1991)
2. Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 238–249. Springer, Heidelberg (2006)
3. Courcelle, B.: The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. *Inf. Comput.* 85(1), 12–75 (1990)
4. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.* 33(2), 125–150 (2000)
5. Dawar, A., Grohe, M., Kreutzer, S., Schweikardt, N.: Approximation schemes for first-order definable optimisation problems. In: LICS, pp. 411–420. IEEE Computer Society, Los Alamitos (2006)
6. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-Time Extremal Structure I. In: Broersma, H., Johnson, M., Szeider, S. (eds.) ACiD. Texts in Algorithmics, vol. 4, pp. 1–41. King’s College, London (2005)
7. Fellows, M.R.: Open problems in parameterized complexity. In: AGAPE spring school on fixed parameter and exact algorithms (2009)
8. Fellows, M.R., Lokshtanov, D., Misra, N., Mnich, M., Rosamond, F.A., Saurabh, S.: The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. *Theory Comput. Syst.* 45(4), 822–848 (2009)
9. Fellows, M.R., Lokshtanov, D., Misra, N., Rosamond, F.A., Saurabh, S.: Graph layout problems parameterized by vertex cover. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 294–305. Springer, Heidelberg (2008)
10. Fellows, M.R., Rosamond, F.A.: The Complexity Ecology of Parameters: An Illustration Using Bounded Max Leaf Number. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 268–277. Springer, Heidelberg (2007)
11. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Clique-width: on the price of generality. In: Mathieu, C. (ed.) SODA, pp. 825–834. SIAM, Philadelphia (2009)
12. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. *J. ACM* 48(6), 1184–1206 (2001)
13. Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic* 130(1-3), 3–31 (2004)
14. Grohe, M.: Logic, graphs, and algorithms. *Electronic Colloquium on Computational Complexity (ECCC)* 14(091) (2007)
15. Hlinený, P., il Oum, S., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. *Comput. J.* 51(3), 326–362 (2008)
16. Kleitman, D., West, D.: Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics* 4, 99 (1991)
17. Kreutzer, S., Tazari, S.: On brambles, grid-like minors, and parameterized intractability of monadic second order logic. In: SODA (2010)

# Determining Edge Expansion and Other Connectivity Measures of Graphs of Bounded Genus

Viresh Patel\*

School of Engineering and Computing Sciences, Durham University,  
Science Laboratories, South Road, Durham DH1 3LE, U.K.  
`vireshe.patel@dur.ac.uk`

**Abstract.** In this paper, we show that for an  $n$ -vertex graph  $G$  of genus  $g$ , the edge expansion of  $G$  can be determined in time  $n^{O(g^2)}$ . We show that the same is true for various other similar measures of edge connectivity.

## 1 Introduction

### 1.1 Background and Motivation

Edge expansion (known also as the minimum cut quotient, the isoperimetric number, or the flux of a graph) is a well-studied notion in graph theory and arises in several contexts of discrete mathematics and theoretical computer science. These include the explicit construction of expander graphs, the analysis of certain randomised algorithms, and graph partitioning problems. In this paper, we are concerned with giving an exact algorithm for determining the edge expansion (and other similar measures) of graphs embedded on surfaces.

Throughout, we use the term graph to mean multigraph without loops, unless otherwise stated. For a graph  $G = (V, E)$  and  $e \in E$ , we write  $e = ab$  to mean that the vertices  $a$  and  $b$  are the end points of  $e$ .

For  $S$  a nonempty proper subset of  $V$  and  $\bar{S}$  its complement, we define

$$[S, \bar{S}]_G = \{e \in E : e = ab, a \in S, b \in \bar{S}\},$$

which we call an *edge-cut* of  $G$ . (The subscript is dropped when it clear which graph we are referring to.) For a cut  $[S, \bar{S}]$  of a graph  $G = (V, E)$ , define the *balance* of the cut to be  $b(S, \bar{S}) := \min(|S|, |\bar{S}|)/|V|$ . Note that the balance of a cut is a real number in the interval  $(0, \frac{1}{2}]$ . Two well-known graph cut problems, which take into account the balance of cuts, are the *minimum quotient cut* problem and the *sparsest cut* problem. These ask respectively to minimize the *cut quotient*  $q(S, \bar{S})$  and the *cut density*  $d(S, \bar{S})$  over all cuts  $[S, \bar{S}]$  of a graph  $G$ , where

$$q(S, \bar{S}) = \frac{|[S, \bar{S}]|}{b(S, \bar{S})} \quad \text{and} \quad d(S, \bar{S}) = \frac{|[S, \bar{S}]|}{b(S, \bar{S})(1 - b(S, \bar{S}))}.$$

---

\* Supported by EPSRC grant EP/F064551/1.

Both  $q$  and  $d$  penalise unbalanced cuts, although  $q$  does so to a greater extent than  $d$ . Note also that  $q$  and  $d$  differ by at most a factor of 2.

Problems such as the minimum quotient cut problem and the sparsest cut problem underlie many divide and conquer algorithms [16], and find applications in VLSI layout problems, packet routing in distributed networking, clustering, and so on. Unfortunately, for general graphs, finding a minimum quotient cut or a sparsest cut is known to be NP-hard [6, 10]. Thus there are two possible ways of developing efficient algorithms for these problems: either by considering approximation algorithms or by restricting attention to certain graph classes. There has been much research done in finding approximation algorithms for these problems. Here, we mention only the seminal paper of Leighton and Rao [9] giving a polynomial-time  $O(\log n)$ -approximation algorithm for the minimum quotient cut problem, and the significant improvement in the approximation factor to  $O(\sqrt{\log n})$  in a paper of Arora, Rao, and Vazirani [2]. On the hardness side, Ambühl et al. [1] proved that the sparsest cut problem admits no polynomial-time approximation scheme unless NP-hard problems can be solved in randomized subexponential time.

We approach the problems of minimum quotient cut and sparsest cut from the perspective of developing exact polynomial-time algorithms for restricted graph classes. Such approaches have not received as much attention in recent years as the development of approximation algorithms, but we hope this paper will take a step towards sparking interest.

Bonsma [3] gave polynomial-time algorithms for finding sparsest cuts of unit circular graphs and cactus graphs. Park and Phillips [14], building on the work of Rao [15], gave a polynomial-time algorithm for determining the minimum quotient cut (as it has been defined above) of planar graphs. Given that many planar-graph algorithms have been adapted for generalizations of planar graphs – see for example the introduction to [4] and the references therein – surprisingly little is known about the complexity of computing minimum quotient cuts or sparsest cuts for generalizations of planar graphs. Here, we generalize the algorithm of Park and Phillips to give the first exact polynomial-time algorithm for determining minimum quotient cuts and sparsest cuts of bounded-genus graphs.

## 1.2 Results

Before we state our result precisely, we give a generalization of the minimum quotient cut and sparsest cut. Notice that the denominators for both the cut quotient  $q$  and the cut density  $d$  are concave and increasing functions of  $b(S, \bar{S})$  on the interval  $[0, \frac{1}{2}]$ . For any concave, increasing function  $f : [0, \frac{1}{2}] \rightarrow [0, \infty)$  and a cut  $[S, \bar{S}]$  of a graph  $G$ , we define

$$d_G^f(S, \bar{S}) = \frac{|[S, \bar{S}]|}{f(b(S, \bar{S}))},$$

and we define

$$d^f(G) = \min d_G^f(S, \bar{S}),$$

where the minimum is taken over all cuts  $[S, \bar{S}]$  of  $G$ . Any cut  $[S, \bar{S}]$  that minimizes  $d_G^f$  is referred to as an  $f$ -sparsest cut of  $G$ .

Let  $f : [0, \frac{1}{2}] \rightarrow [0, \infty)$  be a fixed concave, increasing function that is computable in polynomial time on the rationals, and let  $g$  be a fixed non-negative integer. The input for our algorithm is an  $n$ -vertex undirected multigraph  $G$  of genus  $g$ . Our algorithm computes an  $f$ -sparsest cut of  $G$  in time  $O(n^{2g^2+4g+7})$ .

### 1.3 Overview and Techniques

In this section we give an informal overview of our methods. Our methods extend those of Park and Phillips [14] and combine them with surface homology techniques, used for example in [4].

A generalisation of an averaging argument given by Mohar [11] shows that, given a graph  $G$ , there exists a sparsest cut  $[S, \bar{S}]$  of  $G$  that is *minimal*, i.e. a cut where the graphs induced by  $G$  on  $S$  and  $\bar{S}$  are both connected. This extends easily to  $f$ -sparsest cuts, where  $f$  is a concave increasing function.

There is a standard correspondence between the cuts of a planar graph  $G$  and the cycles of its dual  $D(G)$ : the minimal cuts of  $G$  correspond precisely to the cycles of  $D(G)$ , and the size of a cut in  $G$  is equal to the length of its corresponding cycle in  $D(G)$ . One can similarly construct a dual graph  $D(G)$  for a graph  $G$  embedded on a surface; however the correspondence between minimal cuts of  $G$  and cycles of  $D(G)$  is not quite so simple. Roughly, for a graph  $G$  of genus  $g$ , a minimal cut of  $G$  corresponds to a union of at most  $g + 1$  cycles of  $D(G)$ , but the reverse does not hold: a union of at most  $g + 1$  cycles in  $D(G)$  does not necessarily correspond to a cut in  $G$ . Using the surface embedding of  $G$ , we construct a function  $\Theta$  from the set of oriented edges of  $D(G)$  to  $\mathbb{Z}^{2g}$  with the following property: summing  $\Theta$  around the oriented edges of a union of cycles of  $D(G)$  gives the zero vector if and only if that union of cycles corresponds to a (certain generalization of a) cut of  $G$ .

Extending and simplifying an idea from [14], we also construct a weighting  $\hat{w}$  on the set of oriented edges of  $D(G)$  with the following property: if a union of cycles in  $D(G)$  corresponds to a cut in  $G$ , then summing  $\hat{w}$  around the oriented edges of cycles in the union essentially gives the balance of  $[S, \bar{S}]$ .

Using  $D(G)$ ,  $\Theta$ , and  $\hat{w}$ , we construct a type of covering graph  $H$ , again extending an idea in [14]. For each fixed value  $\mathbf{v}$  and  $k$  of  $\Theta$  and  $\hat{w}$ , we can use  $H$  to find a shortest cycle in  $D(G)$  whose  $\Theta$ -value is  $\mathbf{v}$  and whose  $\hat{w}$ -value is  $k$ . Such a shortest cycle of  $D(G)$  corresponds to a shortest path in  $H$  between suitable vertices.

By repeatedly applying a shortest-path algorithm to  $H$ , we obtain, for every  $\mathbf{v}$  and  $k$  (in a suitable range), a shortest cycle of  $D(G)$  whose  $\Theta$ -value is  $\mathbf{v}$  and whose  $\hat{w}$ -value is  $k$ . We construct the set  $X$  of every union of at most  $g + 1$  of these shortest cycles. The size of  $X$  is  $n^{O(g^2)}$ , and we show that at least one element of  $X$  corresponds to an  $f$ -sparsest cut of  $G$ .

## 2 Preliminaries

Due to limited space, we only state the various lemmas we require for our algorithm; however, complete proofs will be given in the full version of this paper.

Throughout, rather than working with functions  $f : [0, \frac{1}{2}] \rightarrow [0, \infty)$  that are concave and increasing, we work instead with functions  $f : [0, 1] \rightarrow [0, \infty)$  that are concave and increasing on  $[0, \frac{1}{2}]$  and have the property that  $f(x) = f(1-x)$  for all  $x \in [0, 1]$ . We work with these functions purely for the convenience of having, for any cut  $[S, \bar{S}]$  of  $G$ , that

$$f(b(S, \bar{S})) = f(|S|/|V|) = f(|\bar{S}|/|V|).$$

For the algorithm, we assume that  $f$  can be computed in polynomial time on the rationals.

The description and the proof of correctness of our algorithm is most conveniently and naturally expressed in the language of surface homology. Through the remainder of this section, we introduce the necessary concepts, keeping our treatment as simple and self-contained as possible. One can find more comprehensive treatments in e.g. [78].

Although we are only concerned with undirected graphs when determining  $f$ -sparsest cuts, we will need to orient edges of our graphs through the course of our proofs and algorithm. Each edge  $e = ab$  of a graph  $G = (V, E)$  has two orientations, namely  $(a, e, b)$  and  $(b, e, a)$ . The two orientations are denoted  $\vec{e}$  and  $\overleftarrow{e}$ , although we cannot say which is which in general. Given a set  $E$  of edges, we write  $\vec{E}$  for the set of their orientations, two for each edge. The *edge space* of  $G$ , denoted  $\mathcal{E}(G)$ , is the free abelian group over  $\vec{E}$  modulo the relation that  $\overleftarrow{e} = -\vec{e}$ . Thus, for each  $\rho \in \mathcal{E}(G)$ , we can express  $\rho$  uniquely as

$$\rho = \sum_{\vec{e} \in \vec{E}} \lambda_{\vec{e}} \vec{e},$$

where  $\lambda_{\vec{e}} \in \mathbb{Z}$  for all  $\vec{e} \in \vec{E}$  and  $\min(\lambda_{\vec{e}}, \lambda_{\overleftarrow{e}}) = 0$ . We define

$$|\rho| = \sum_{\vec{e} \in \vec{E}} \lambda_{\vec{e}}.$$

Thus  $|\rho|$  in a sense counts the number of edges in  $\rho$ .

The *cut space*  $\mathcal{T}(G)$  of  $G = (V, E)$ , which is a subgroup of  $\mathcal{E}(G)$ , is defined as follows. For a cut  $[S, \bar{S}]$  of  $G$ , define

$$[\overrightarrow{S, \bar{S}}] = \sum_{\substack{ab=e \in E \\ a \in S, b \notin S}} (a, e, b).$$

Then  $\mathcal{T}(G)$  is the subgroup of  $\mathcal{E}(G)$  generated by  $\{\overrightarrow{[S, \bar{S}]} : S \subseteq V\}$ .

The next lemma shows that, by suitably assigning weights to oriented edges of a graph, we can determine the balance of a cut simply by summing the weights of the edges in the (oriented) cut.

**Lemma 1.** Let  $G = (V, E)$  be a connected graph and let  $v \in V$  be some fixed vertex. There exists a function  $w : \overrightarrow{E} \rightarrow \mathbb{Z}$  with the following properties.

- (i) For every  $\overrightarrow{e} \in \overrightarrow{E}$ , we have  $w(\overleftarrow{e}) = -w(\overrightarrow{e})$ .
- (ii) For every  $\overrightarrow{e} \in \overrightarrow{E}$ , we have  $|w(\overrightarrow{e})| \leq |V|$ .
- (iii) For every  $S$  satisfying  $v \in S \subseteq V$ , we have that

$$w([\overline{S}, \overrightarrow{\bar{S}}]) := \sum_{(a,e,b) \in [\overline{S}, \overrightarrow{\bar{S}}]} w(a, e, b) = |\bar{S}|.$$

Furthermore, a function satisfying the above properties can be constructed in linear time.

*Remark 1.* The function  $w$  described in Lemma 1 can be extended to a homomorphism  $w : \mathcal{E}(G) \rightarrow \mathbb{Z}$  because of property (i).

Throughout,  $w$  will be a homomorphism from  $\mathcal{E}(G)$  to  $\mathbb{Z}$  satisfying the properties of Lemma 1.

Recall that the domain for the function  $d_G^f$  is the set of all cuts of  $G$ . It turns out that it is necessary to extend  $d_G^f$  in a natural way to a function on  $\mathcal{T}(G)$ . Minimizing  $d_G^f$  over  $\mathcal{T}(G)$  will be equivalent to minimizing it over the set of cuts of  $G$ . Although  $\mathcal{T}(G)$  is an infinite set, we will eventually look to minimize  $d_G^f$  over a suitable finite subset of  $\mathcal{T}(G)$ .

For each  $\phi \in \mathcal{T}(G)$ , we define

$$d_G^f(\phi) := \frac{|\phi|}{f(|w(\phi)|/n)}; \quad (1)$$

if  $f(|w(\phi)|) = 0$  or  $|w(\phi)| > n$ , we define  $d_G^f(\phi) = \infty$ . Note that if  $\phi = [\overline{S}, \overrightarrow{\bar{S}}]$ , then  $|\phi| = |[\overline{S}, \overrightarrow{\bar{S}}]|$  and  $|w(\phi)|$  is either  $|S|$  or  $|\bar{S}|$ . Hence the function  $d_G^f$  defined above extends the definition of  $d_G^f$  given in the introduction.

Our next lemma shows that minimizing  $d_G^f$  over  $\mathcal{T}(G)$  is equivalent to minimizing  $d_G^f$  over simple cuts  $[\overline{S}, \overrightarrow{\bar{S}}]$  of  $G$ . The proof uses the concavity of  $f$  together with a straightforward averaging argument.

**Lemma 2.** Let  $G = (V, E)$  be a graph. For every  $\phi \in \mathcal{T}(G)$ , there exists  $S \subseteq V$  such that

$$d_G^f(S, \bar{S}) \leq d_G^f(\phi),$$

so in particular

$$\min_{\phi \in \mathcal{T}(G)} d_G^f(\phi) = d^f(G).$$

A walk  $w$  in a graph  $G = (V, E)$  is specified by giving an alternating sequence of vertices and edges  $w = (x_1, e_1, x_2, e_2, \dots, x_{k-1}, e_{k-1}, x_k)$ , where  $(x_i, e_i, x_{i+1}) \in \overrightarrow{E}$  for all  $i$ . We write  $|w|$  for the number of oriented edges (with multiplicity) traversed in  $w$  (which in this case is  $k - 1$ ). If  $x_1 = x_k$  then  $w$  is called a *closed*

*walk.* If all (non-oriented) edges of a closed walk  $w$  are distinct, then  $w$  is called a *circuit* of  $G$ . If all the vertices of a closed walk  $w$  are distinct (except  $x_1 = x_k$ ), then  $w$  is called a *cycle*. A walk in which all vertices are distinct is called a *path*. We write  $\vec{w}$  for the element of  $\mathcal{E}(G)$  given by

$$\vec{w} = \sum_{i=1}^{k-1} (x_i, e_i, x_{i+1});$$

we refer to  $\vec{w}$  as an *oriented walk*, (circuit, etc). Note that for a walk  $w$ , we have  $|w| \geq |\vec{w}|$  with equality when  $w$  is a circuit.

The *cycle space*  $\mathcal{C}(G)$  is the subgroup of  $\mathcal{E}(G)$  (redundantly) generated by the oriented cycles  $\vec{c}$  of  $G$ . Note that  $\mathcal{C}(G)$  contains all oriented closed walks of  $G$ .

We now turn our attention to graphs embedded on closed orientable surfaces. Formally, a surface is a compact, connected topological space in which every point of the surface has an open neighbourhood homeomorphic to  $\mathbb{R}^2$  or the closed halfplane  $\{(x, y) \in \mathbb{R}^2 : y \geq 0\}$ . The set of points having halfplane open neighbourhoods is called the boundary of the surface. Every component of the boundary is homeomorphic to the circle  $S^1$ . A closed surface is one without boundary. A surface is called orientable if it does not contain a subset (with the subset topology) homeomorphic to the möbius band.

The genus of a connected, orientable surface is the maximum number of cuttings along non-intersecting, closed, simple curves that can be made without disconnecting the surface. It is well known from the classification of surfaces that every closed orientable surface of genus  $g$  is homeomorphic to a sphere with  $g$  handles. Informally, a graph  $G$  can be embedded on a surface  $\Sigma$  if  $G$  can be drawn on  $\Sigma$  in such a way that no edge crosses a vertex or another edge, except possibly at its end points. For example, all planar graphs can be embedded on the sphere.

From an algorithmic point of view, one can use *rotation systems* to input or output embeddings of graphs on surfaces. We do not formally define embeddings or rotation systems here because we shall only use graph embeddings indirectly when applying existing algorithms to our problem; instead we refer the reader to [13].

Throughout, we shall only consider cellular embeddings. An embedding of  $G$  on  $\Sigma$  is called *cellular* if removing the image of  $G$  from  $\Sigma$  leaves a set of topological disks called the *faces* of  $G$ . The *genus* of a graph  $G$  is defined to be the smallest integer  $g$  such that  $G$  can be embedded on a closed orientable surface of genus  $g$ . If  $G$  is a graph of genus  $g$ , then every embedding of  $G$  on a closed orientable surface of genus  $g$  is cellular (Proposition 3.4.1 [13]), and for fixed  $g$ , such an embedding can be found in linear time [12]. Euler's Theorem gives the following relationship between the number of vertices  $n$ , the number of edges  $m$ , the number of faces  $\ell$ , and the number of boundary components  $b$  in a cellular embedding of a graph  $G$  on a surface  $\Sigma$  of genus  $g$ :

$$n - m + \ell + b = 2 - 2g.$$

We now define the *boundary space* of a graph  $G$  embedded on  $\Sigma$ . Each oriented edge  $\vec{e}$  of  $G$  separates two (possibly equal) faces of  $G$  denoted  $\text{left}(\vec{e})$  and  $\text{right}(\vec{e})$ . (The notion of left and right with respect to an oriented edge is well defined for orientable surfaces.) For a face  $F$ , the oriented edges  $\vec{e}$  for which  $\text{left}(\vec{e}) = F$  taken in order (with appropriate intervening vertices) form a closed walk  $f$  around  $F$ , which we call the facial walk of  $F$ . Let  $F_1, \dots, F_\ell$  be the faces of the embedding and let  $f_i$  be the facial walk of  $F_i$ . The oriented facial walk  $\vec{f}_i$  of  $F_i$  is given by

$$\vec{f}_i = \sum_{\vec{e}: \text{left}(\vec{e}) = F_i} \vec{e}.$$

Notice that  $f_i$  may contain two oppositely oriented edges, but such edges cancel in  $\vec{f}_i$ , leaving the sum of oriented edges that form the boundary of  $F_i$ . The boundary space  $\mathcal{B}(G)$  is defined to be the group generated by  $\vec{f}_1, \dots, \vec{f}_\ell$  and is easily seen to be a subgroup of  $\mathcal{C}(G)$ . Note that the boundary space of  $G$ , in contrast to the cycle space and cut space, depends on the embedding of  $G$ .

The geometric dual of a graph  $G$  cellularly embedded on  $\Sigma$  is denoted by  $D(G) = (V', E')$  and is the graph (with cellular embedding on  $\Sigma$ ) constructed from  $G$  as follows. For each face  $F$  of  $G$ , a vertex  $D(F)$  is placed inside  $F$ : these are the vertices of  $D(G)$ . Each edge  $e$  of  $G$  has a corresponding edge  $D(e)$  in  $D(G)$ :  $D(e) = D(F_1)D(F_2)$ , where  $e$  is the edge separating the (possibly indistinct) faces  $F_1$  and  $F_2$  (and  $D(e)$  crosses  $e$  and no other edge of  $G$  in the embedding of  $D(G)$ ). Note that  $G$  (with its embedding) is a dual of  $D(G)$  (with its embedding). Therefore, each vertex  $v$  of  $G$  corresponds to a face  $D(v)$  of  $D(G)$ . Thus  $D$  maps vertices, edges, and faces of  $G$  bijectively to faces, edges, and vertices of  $D(G)$  respectively. We extend  $D$  to map oriented edges of  $G$  to oriented edges of  $D(G)$  as follows. Given an oriented edge  $\vec{e}$ , we set  $D(\vec{e}') = (D(\text{left}(\vec{e})), D(e), D(\text{right}(\vec{e})))$ . This, however, reverses the sense of left and right so that  $D(D(\vec{e})) = -\vec{e}'$ .

*Remark 2.* Although  $G$  is a loopless graph,  $D(G)$  may not be. Nonetheless, all notions introduced so far carry through naturally for loops of embedded graphs.

Since  $D$  bijectively maps edges of  $G$  to edges of  $D(G)$ , we see that  $D$  can be extended to an isomorphism  $D : \mathcal{E}(G) \rightarrow \mathcal{E}(D(G))$ . We have the following well-known correspondence.

**Proposition 1.** *The restriction of  $D$  to  $\mathcal{T}(G)$  gives an isomorphism  $\mathcal{T}(G) \rightarrow \mathcal{B}(D(G))$ .*

Rather than working with  $\mathcal{T}(G)$ , we can work instead with  $\mathcal{B}(D(G))$  using the isomorphism  $D$ . Since all boundaries are sums of oriented cycles, we can use shortest-path algorithms to find shortest boundaries in  $D(G)$ , which if done suitably, can give us  $f$ -sparsest cuts in  $G$ .

For  $\sigma \in \mathcal{E}(D(G))$ , define  $\hat{w}(\sigma) := \hat{w}(D^{-1}(\sigma))$ . Notice that for all  $\sigma \in \mathcal{B}(D(G))$ , we have  $|D^{-1}(\sigma)| = |\sigma|$ . Thus, defining

$$\hat{d}_{D(G)}^f(\sigma) = \frac{|\sigma|}{f(|\hat{w}(\sigma)|)}, \quad (2)$$

we have that

$$\min_{\sigma \in \mathcal{B}(D(G))} \hat{d}_{D(G)}^f(\sigma) = \min_{\phi \in \mathcal{T}(G)} d_G^f(\phi) = d^f(G).$$

We now set about trying to minimize  $\hat{d}_{D(G)}^f$ . Our next lemma says that when minimizing  $\hat{d}_{D(G)}^f$ , we can restrict attention to elements of  $\mathcal{B}(D(G))$  that are the sum of at most  $g + 1$  oriented circuits (where  $g$  is the genus of  $G$ ). The lemma essentially follows from Euler's Theorem and the fact that there exists an  $f$ -sparsest cut that is minimal.

**Lemma 3.** *Suppose  $G = (V, E)$  is a graph cellularly embedded on a surface  $\Sigma$  of genus  $g$ . There exists  $\sigma \in \mathcal{B}(D(G))$  that minimizes  $\hat{d}_{D(G)}^f$  such that  $\sigma = \vec{w}_1 + \dots + \vec{w}_r$ , where  $\vec{w}_1, \dots, \vec{w}_r$  are disjoint oriented circuits of  $D(G)$  and  $r \leq g + 1$ . Furthermore  $m \geq |\sigma| = |\vec{w}_1| + \dots + |\vec{w}_r|$ , where  $m = |E|$ .*

We can easily use shortest-path algorithms to find shortest cycles in a graph. However in this situation, we are required to find a shortest boundary (loosely speaking). We require a simple way of testing whether a cycle is a boundary. This is accomplished using the ideas of homology.

Given a closed, orientable surface  $\Sigma$  of genus  $g$  and a point  $x_0$  on  $\Sigma$ , a simple loop at  $x_0$  is a curve  $\gamma : [0, 1] \rightarrow \Sigma$  that is injective except that  $\gamma(0) = \gamma(1) = x_0$ . A *system of loops* at  $x_0$  is a set of simple loops at  $x_0$  that are pairwise disjoint (except at  $x_0$ ) with the property that cutting  $\Sigma$  along these loops yields a topological disk. By Euler's formula, every system of loops must consist of  $2g$  loops. Let  $G$  be an  $n$ -vertex graph cellularly embedded on  $\Sigma$  and let  $x_0$  be a vertex. A *combinatorial system of loops* at  $x_0$  is a set of  $2g$  closed walks of  $G$  through  $x_0$  such that an infinitesimal perturbation of these walks gives a system of loops. Erickson and Whittlesey [5] give a greedy algorithm, which, given a cellular embedding of  $G$  on  $\Sigma$ , finds a combinatorial system of loops  $c(1), \dots, c(2g)$  in  $O(n \log n + gn)$  time.

We define a homomorphism  $\Theta_i : \mathcal{C}(D(G)) \rightarrow \mathbb{Z}$ , where, for every  $\sigma \in \mathcal{C}(D(G))$ , the integer  $\Theta_i(\sigma)$  measures the net number of times  $\sigma$  crosses  $c(i)$  (here sign indicates the direction of crossings). Let us define  $\Theta_i$  formally.

For each oriented edge  $\vec{e}$  of  $G$  and each oriented edge  $\vec{e}^*$  of  $D(G)$ , define

$$\Theta_{\vec{e}}(\vec{e}^*) = \begin{cases} 1 & \text{if } D(\vec{e}) = \vec{e}^*; \\ -1 & \text{if } D(\vec{e}) = -\vec{e}^*; \\ 0 & \text{otherwise.} \end{cases}$$

For every  $\phi = \sum_i \lambda_i \vec{e}_i \in \mathcal{E}(G)$  and every  $\sigma = \sum_j \mu_j \vec{e}_j^* \in \mathcal{E}(D(G))$ , we define

$$\Theta_\phi(\sigma) = \sum_i \sum_j \lambda_i \mu_j \Theta_{\vec{e}_i}(\vec{e}_j^*),$$

which counts the directed number of times  $\phi$  and  $\pi$  cross each other. In algebraic topology,  $\Theta_\phi(\sigma)$  is referred to as a *cap product*. It is easy to check that the above is

well defined. We write  $\Theta_i$  as a shorthand for  $\Theta_{\overrightarrow{c(i)}}$ . Finally, define  $\Theta : \mathcal{E}(D(G)) \rightarrow \mathbb{Z}^{2g}$  by setting

$$\Theta(\sigma) = (\Theta_1(\sigma), \dots, \Theta_{2g}(\sigma))$$

for every  $\sigma \in \mathcal{E}(D(G))$ .

We have the following proposition which effectively says that any cycle of  $D(G)$  that intersects each  $\overrightarrow{c(i)}$  a net number of zero times is a boundary. It is very much what we expect from the properties of homology.

**Lemma 4.** *The function  $\Theta$  defined above is a surjective homomorphism from  $\mathcal{C}(D(G))$  to  $\mathbb{Z}^{2g}$  whose kernel is  $\mathcal{B}(D(G))$ .*

For convenience, we combine some of the results we have so far to give the following proposition.

**Proposition 2.** *There exists an element  $\sigma \in \mathcal{B}(D(G))$  that minimizes  $\hat{d}_{D(G)}^f$  and satisfies the following properties.*

- (a) *We can write  $\sigma$  as  $\overrightarrow{w_1} + \dots + \overrightarrow{w_r}$  where the  $\overrightarrow{w_i}$  are oriented circuits in  $D(G)$  and  $r \leq g+1$ .*
- (b) *We have  $|\sigma| = |\overrightarrow{w_1}| + \dots + |\overrightarrow{w_r}|$  with  $|\overrightarrow{w_i}| \leq m$  for all  $i$ .*
- (c)  $\Theta(\sigma) = \Theta(\overrightarrow{w_1}) + \dots + \Theta(\overrightarrow{w_r}) = \mathbf{0}$ .

*Proof.* Statements (a) and (b) follow from Lemma 3, and statement (c) follows from Lemma 4.  $\square$

Next we define a type of *covering graph* of  $D(G)$  in which certain shortest paths will correspond to elements of  $\mathcal{C}(D(G))$  whose sum will minimize  $\hat{d}_{D(G)}^f$ . The general construction we use is often referred to as the *derived graph of a voltage graph*.

We construct an infinite multigraph  $H = (V_H, E_H)$  from  $D(G) = (V', E')$  as follows. We set  $V_H = V' \times \mathbb{Z} \times \mathbb{Z}^{2g}$ . For convenience, we describe the oriented edges of  $H$  before describing its edges. For each oriented edge  $\overrightarrow{e} = (u_1, e, u_2)$  of  $D(G)$  and each  $(k, \mathbf{v}) \in \mathbb{Z} \times \mathbb{Z}^{2g}$ , we have an oriented edge of  $H$ , denoted  $(\overrightarrow{e}, k, \mathbf{v})^*$ , from

$$(u_1, k, \mathbf{v}) \quad \text{to} \quad (u_2, k + \hat{w}(\overrightarrow{e}), \mathbf{v} + \Theta(\overrightarrow{e})).$$

The same edge oriented oppositely is given by  $(\overleftarrow{e}, k + \hat{w}(\overrightarrow{e}), \mathbf{v} + \Theta(\overrightarrow{e}))^*$ . We write  $(\overrightarrow{e}, k, \mathbf{v})$  for the edge of  $H$  corresponding to the oriented edge  $(\overrightarrow{e}, k, \mathbf{v})^*$ ; thus every edge of  $H$  has two labels.

Walks of  $D(G)$  naturally correspond to walks of  $H$  as follows. Let  $w = (u_0, e_1, u_1, \dots, e_{t-1}, u_t)$  be a walk of  $D(G)$ . Let  $w_i = (u_0, e_1, u_1, \dots, e_{i-1}, u_i)$  be the same walk up to the  $i$ th vertex, and let  $\overrightarrow{e}_i = (u_{i-1}, e_i, u_i)$ . Let  $H(w)$  be the walk in  $H$  given by  $H(w) = (u_0^H, e_1^H, u_1^H, \dots, e_{t-1}^H, u_t^H)$ , where  $u_0^H = (u, 0, \mathbf{0})$  and

$$u_i^H = (u_i, \hat{w}(\overrightarrow{w_i}), \Theta(\overrightarrow{w_i})) \quad \text{and} \quad e_i^H = (\overrightarrow{e}_i, \hat{w}(\overrightarrow{w_{i-1}}), \Theta(\overrightarrow{w_{i-1}})).$$

It is easy to check that  $w \mapsto H(w)$  is a bijective correspondence between walks of  $D(G)$  and walks of  $H$  that start at  $(u, 0, \mathbf{0})$  for some  $u \in V'$ . Furthermore,  $w$

is a walk of  $D(G)$  from  $u$  to  $u'$  and satisfies  $\hat{w}(\vec{w}) = k$  and  $\Theta(\vec{w}) = \mathbf{v}$  if and only if  $H(w)$  is a walk of  $H$  from  $(u, 0, \mathbf{0})$  to  $(u', k, \mathbf{v})$ .

Defining  $V_H^* = V' \times \{-mn, \dots, mn\} \times \{-m, \dots, m\}^{2g} \subset V_H$ , let  $H^*$  be the finite graph induced by  $H$  on  $V_H^*$ . For  $(u, k, \mathbf{v}) \in V_H^*$ , define  $p(u, k, \mathbf{v})$  to be a shortest path in  $H^*$  from  $(u, 0, \mathbf{0})$  to  $(u, k, \mathbf{v})$  (if it exists); this path corresponds to a closed walk in  $D(G)$ . For fixed  $k$  and  $\mathbf{v}$ , let  $p(k, \mathbf{v})$  be the path of minimum length in  $\{p(u, k, \mathbf{v}) : u \in V'\}$  (if it exists). Let  $w(k, \mathbf{v})$  be the (closed) walk of  $D(G)$  corresponding to the path  $p(k, \mathbf{v})$  in  $H^*$ . We have the following lemma.

**Lemma 5.** *Suppose  $w$  is a circuit of  $D(G)$  such that  $\hat{w}(\vec{w}) = k$  and  $\Theta(\vec{w}) = \mathbf{v}$  and  $|\vec{w}| \leq m$ . Then  $|\vec{w}| \geq |\overrightarrow{w(k, \mathbf{v})}|$ .*

Let  $W$  be the set of all the  $\overrightarrow{w(k, \mathbf{v})}$ . Let  $X = \{\overrightarrow{w_1} + \dots + \overrightarrow{w_r} : \overrightarrow{w_i} \in W \forall i$  and  $r \leq g+1\}$ , and let  $Y = \{\sigma \in X : \Theta(\sigma) = \mathbf{0}\}$ . We have the following corollary.

**Corollary 1.** *There exists an element of  $Y$  that minimizes  $\hat{d}_{D(G)}^f$ .*

We are now ready to present our algorithm and to prove its correctness.

### 3 The Algorithm

In this section, we present the basic steps of our algorithm and compute its running time. In order to keep the presentation simple, we do not optimize the running time. Our algorithm runs in time  $O(n^{2g^2+4g+7})$ .

Let  $f : [0, \frac{1}{2}] \rightarrow [0, \infty)$  be a fixed concave, increasing function that is computable in polynomial time on the rationals, and let  $g$  be a fixed non-negative integer. The input for our algorithm is an  $n$ -vertex undirected graph  $G$  of genus  $g$ . Since  $n = |V|$  then  $m = |E| = O(n)$  from Euler's formula. Using the result of Mohar [12] mentioned earlier, we can find an embedding of  $G$  on a surface  $\Sigma$  of genus  $g$  in  $O(n)$  time (where the constant factor strongly depends on  $g$ ). From the embedding we can construct (in  $O(n)$  time) the dual graph  $D(G) = (V', E')$  together with the function  $D$  which maps each oriented edge  $\vec{e} \in \vec{E}$  of  $G$  to its dual  $D(\vec{e}) \in \vec{E}'$  in  $D(G)$ . We have  $|E'| = |E| = O(n)$ , and from Euler's formula, we have  $|V'| = O(n)$ . By the result of Erickson and Whittlesey [5] mentioned earlier, we can find a combinatorial system of loops  $c(1), \dots, c(2g)$  of  $G$  in time  $O(n)$ . At this point the algorithm no longer requires the embedding of  $G$ .

Next we construct and store the (restricted) functions  $\hat{w} : \vec{E}' \rightarrow \mathbb{Z}$  and  $\Theta : \vec{E}' \rightarrow \mathbb{Z}^{2g}$ . The computation and storage of these functions imposes an insignificant time cost in the final analysis, so any crude bound on the running time is sufficient. It is easy to check that both functions can be computed and stored in time  $O(n^2)$ .

From the (restricted) functions  $\hat{w}$  and  $\Theta$ , we can construct the graph  $H^*$  directly from its definition (given in the previous section). Observe that  $H^*$  has  $O(n) \cdot (2mn+1) \cdot (2m+1)^{2g} = O(n^{2g+3})$  vertices and  $O(n^{2g+3})$  edges. Thus it takes  $O(n^{2g+3})$  time to construct  $H^*$ . For each  $(u, k, \mathbf{v}) \in V_{H^*}$ , we compute and store

the shortest path  $p(u, k, \mathbf{v})$  from  $(u, 0, \mathbf{0})$  to  $(u, k, \mathbf{v})$  in  $H^*$ . Finding each shortest path requires  $O(|V_{H^*}| \log(|V_{H^*}|)) = O(n^{2g+4})$  time using Dijkstra's algorithm, and so, computing all the  $p(u, k, \mathbf{v})$  requires  $|V_{H^*}|O(n^{2g+4}) = O(n^{4g+7})$  time. For each fixed  $k \in \{-mn, \dots, mn\}$  and  $\mathbf{v} \in \{-n, \dots, n\}^{2g}$ , we compute and store  $p(k, \mathbf{v})$ , the path of minimum length amongst the  $p(u, k, \mathbf{v})$ , and we use  $p(k, \mathbf{v})$  to compute and store  $w(k, \mathbf{v})$  and  $\overrightarrow{w(k, \mathbf{v})}$  (recall that  $w(k, \mathbf{v})$  is the closed walk in  $D(G)$  corresponding to  $p(k, \mathbf{v})$  as described in the previous section). The time cost so far is  $O(n^{4g+7})$ .

Recall the sets  $W$ ,  $X$ , and  $Y$  from the previous section. Having stored the set  $W$  of all walks  $\overrightarrow{w(k, \mathbf{v})}$ , we compute and store the set

$$X = \{\overrightarrow{w_1} + \cdots + \overrightarrow{w_r} : \overrightarrow{w_i} \in W \forall i \text{ and } r \leq g+1\}.$$

Adding elements of  $W$  together requires  $O(n)$  time; hence computing and storing  $X$  requires  $O(n|X|) = O(n|W|^{g+1}) = O(n \cdot n^{2(g+1)^2})$  time. We compute  $\Theta(\sigma)$  for every  $\sigma \in X$  and store the set  $Y = \{\sigma \in X : \Theta(\sigma) = \mathbf{0}\}$ , which takes  $O(n|X|)$  time. Finally we find an element  $\sigma^*$  of  $Y$  that minimizes  $\hat{d}_{D(G)}^f$ , which takes  $O(n|Y|)$  time, and from Corollary 11 we know  $\sigma^*$  minimizes  $\hat{d}_{D(G)}^f$  over all elements of  $\mathcal{B}(D(G))$ . Thus  $d^f(G)$  is given by  $\hat{d}_{D(G)}^f(\sigma^*)$ , and the total time taken to find  $\sigma^*$  is dominated by  $\max(O(n^{2(g+1)^2+1}), O(n^{4g+7})) = O(n^{2g^2+4g+7})$ .

In order to find an  $f$ -sparsest cut of  $G$ , we compute  $\phi^* = D^{-1}(\sigma^*)$  and decompose it into a sum of cuts to give

$$\phi = \sum_{i=1}^k \overrightarrow{[S_i, \bar{S}_i]},$$

where  $k = O(mn)$  from the bound on the size of  $H^*$ . Now one of the cuts  $[S_i, \bar{S}_i]$  is an  $f$ -sparsest cut by the proof of Lemma 2, and such a cut can be found in  $O(n^2)$  time once  $\sigma^*$  has been found.

## 4 Open Problems

An obvious question that arises from this work is whether the running time of our algorithm can be improved. Specifically, it would be interesting to know if the problem of finding the edge expansion of a graph is fixed parameter tractable with respect to genus.

Our algorithm crucially relies on our graph being embedded on an orientable surface. In particular, we use the fact that a graph embedded on an orientable surface has a *directed* dual; this is not the case for graphs embedded on non-orientable surfaces. It would be interesting to develop methods for finding edge expansion of graphs embedded on non-orientable surfaces.

## Acknowledgements

The author thanks the anonymous reviewer whose detailed comments and suggested references have improved the paper.

## References

1. Ambuhl, C., Mastrolilli, M., Svensson, O.: Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In: FOCS 2007: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 329–337. IEEE Computer Society, Los Alamitos (2007)
2. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. *J. ACM* 56(2), Art. 5, 37 (2009)
3. Bonsma, P.S.: Linear time algorithms for finding sparsest cuts in various graph classes. In: 6th Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications. *Electron. Notes Discrete Math.*, vol. 28, pp. 265–272. Elsevier, Amsterdam (2007)
4. Chambers, E.W., Erickson, J., Nayyeri, A.: Minimum cuts and shortest homologous cycles. In: SCG 2009: Proceedings of the 25th annual symposium on Computational geometry, pp. 377–385. ACM, New York (2009)
5. Erickson, J., Whittlesey, K.: Greedy optimal homotopy and homology generators. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1038–1046. ACM, New York (2005) (electronic)
6. Garey, M.R., Johnson, D.S.: Computers and intractability. In: A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. W. H. Freeman and Co, San Francisco (1979)
7. Giblin, P.J.: Graphs, surfaces and homology, 2nd edn. Chapman & Hall, London (1981); An introduction to algebraic topology, Chapman and Hall Mathematics Series
8. Gross, J.L., Tucker, T.W.: Topological graph theory. Dover Publications Inc., Mineola (2001); Reprint of the 1987 original [Wiley, New York; MR0898434 (88h:05034)] with a new preface and supplementary bibliography
9. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM* 46(6), 787–832 (1999)
10. Matula, D.W., Shahrokhi, F.: Sparsest cuts and bottlenecks in graphs. *Discrete Appl. Math.* 27(1-2), 113–123 (1990); Computational algorithms, operations research and computer science, Burnaby, BC (1987)
11. Mohar, B.: Isoperimetric numbers of graphs. *J. Combin. Theory Ser. B* 47(3), 274–291 (1989)
12. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.* 12(1), 6–26 (1999) (electronic)
13. Mohar, B., Thomassen, C.: Graphs on surfaces. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore (2001)
14. Park, J.K., Phillips, C.A.: Finding minimum-quotient cuts in planar graphs. In: STOC, pp. 766–775 (1993)
15. Rao, S.B.: Faster algorithms for finding small edge cuts in planar graphs. In: STOC 1992: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing, pp. 229–240. ACM, New York (1992)
16. Shmoys, D.M.: Approximation algorithms for NP-hard problems, ch. 5. PWS Publishing Co., Boston (1997)

# Constructing the R\* Consensus Tree of Two Trees in Subcubic Time

Jesper Jansson<sup>1,\*</sup> and Wing-Kin Sung<sup>2,3</sup>

<sup>1</sup> Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan  
[Jesper.Jansson@ocha.ac.jp](mailto:Jesper.Jansson@ocha.ac.jp)

<sup>2</sup> School of Computing, National University of Singapore, 3 Science Drive 2,  
Singapore 117543  
[ksung@comp.nus.edu.sg](mailto:ksung@comp.nus.edu.sg)

<sup>3</sup> Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672

**Abstract.** The previously fastest algorithms for computing the R\* consensus tree of two given (rooted) phylogenetic trees  $T_1$  and  $T_2$  with a leaf label set of cardinality  $n$  run in  $\Theta(n^3)$  time [3,8]. In this manuscript, we describe a new  $O(n^2 \log n)$ -time algorithm to solve the problem. This is a significant improvement because the R\* consensus tree is defined in terms of a set  $\mathcal{R}_{maj}$  which may contain  $\Omega(n^3)$  elements, so any direct approach which explicitly constructs  $\mathcal{R}_{maj}$  requires  $\Omega(n^3)$  time.

## 1 Introduction

Phylogenetic trees are leaf-labeled trees which are commonly used to describe the evolutionary history of a set of objects such as biological species or languages [5,6,10]. Typically, in a phylogenetic tree, each leaf represents one of the objects being studied and the branching structure of the tree indicates the assumed evolutionary relationships among the objects. A *consensus tree* is a single phylogenetic tree which summarizes the branching information contained in an input collection of phylogenetic trees with identical leaf label sets. Consensus trees are useful when different data sets or different tree inference methods have produced a set of trees with the same leaf label set and slightly conflicting structures, yet a single tree is required to represent all of them [5]. Also, by exclusion, a consensus tree indicates areas of conflict in the input trees [2]. Furthermore, consensus trees are sometimes used as a basis for new phylogenetic inferences [2].

Many types of consensus trees have been defined and studied in detail. For a survey, see, e.g., [2]. Different types of consensus trees use different criteria to resolve conflicts among the input trees, so their mathematical properties vary. Therefore, the most suitable type of consensus tree to use in practice depends on the particular application. In this paper, we consider the so-called *R\* consensus tree*. One advantage of the R\* consensus tree is that it provides a statistically consistent estimator of the species tree topology when combining a set of gene

\* Funded by the Special Coordination Funds for Promoting Science and Technology, Japan.

trees, as recently demonstrated by Degnan *et al.* [4]; moreover, R\* consensus outperformed other methods such as majority-rule consensus in the study conducted by [4]. For the case of two input trees, it is known that the R\* consensus tree is equivalent to the *RV-III tree* introduced in [8].

The R\* consensus tree is defined formally in Section 2 below. In short, given a set of input trees on a leaf label set  $L$ , if the rooted triplet  $xy|z$  for any  $\{x, y, z\} \subseteq L$  is consistent with more input trees than each of the two rooted triplets  $xz|y$  and  $yz|x$  is, then  $xy|z$  belongs to a set named  $\mathcal{R}_{maj}$ . The R\* consensus tree is the tree  $\tau$  having the largest possible number of internal nodes such that every rooted triplet consistent with  $\tau$  belongs to  $\mathcal{R}_{maj}$ .

The goal of this paper is to develop a fast algorithm for constructing the R\* consensus tree for two input trees  $T_1$  and  $T_2$  with a leaf label set  $L$  of cardinality  $n$ . The previous algorithms for this problem require  $\Theta(n^3)$  time [38]. Our main result is to reduce the time complexity to subcubic. When  $T_1$  and  $T_2$  have similar branching structures,  $|\mathcal{R}_{maj}| = \Omega(n^3)$ . Hence, in order to obtain a fast algorithm, we have to avoid explicitly constructing the set  $\mathcal{R}_{maj}$ . For this purpose, we use an alternative formulation based on distances from leaves to lowest common ancestors of pairs of leaves to compute the Apresjan clusters of a function  $s_{\mathcal{R}_{maj}}$  associated to  $\mathcal{R}_{maj}$ . Then, we find the strong clusters of  $\mathcal{R}_{maj}$  which are subsequently used to build the R\* consensus tree. The running time of our new algorithm `R*_consensus_tree` (described in Section 3) is  $O(n^2 \log n)$ .

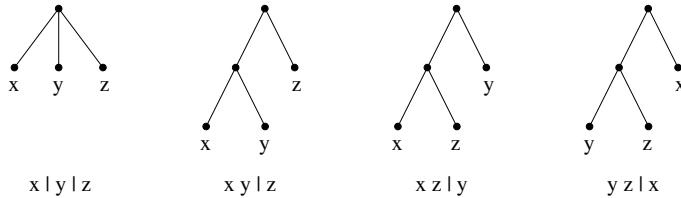
## 2 Preliminaries

### 2.1 Basic Definitions

A *phylogenetic tree* is a rooted, unordered, distinctly leaf-labeled tree in which every internal node has at least two children. From here on, “tree” means “phylogenetic tree”, and every leaf in a tree is identified with its label.

We shall use the following terminology and notation. For any tree  $T$  and any internal node  $u$  of  $T$ , the subtree of  $T$  rooted at  $u$  is denoted by  $T[u]$ . The set of all leaves in a tree  $T$  is written as  $\Lambda(T)$ . For any two nodes  $u$  and  $v$  in a tree  $T$ , the *lowest common ancestor of  $u$  and  $v$  in  $T$*  is denoted by  $\text{lca}^T(u, v)$ . A *triplet* is a tree with exactly three leaves. Any *non-binary* tree with exactly three leaves  $\{x, y, z\}$  is called a *fan triplet* and is written as  $x|y|z$ . On the other hand, any *binary* tree with exactly three leaves  $\{x, y, z\}$  is called a *rooted triplet*, and is denoted by  $xy|z$  if the lowest common ancestor of  $x$  and  $y$  is a proper descendant of the lowest common ancestor of  $x$  and  $z$ . Note that there are precisely four different triplets for any set of three leaf labels  $\{x, y, z\}$ , namely  $x|y|z$ ,  $xy|z$ ,  $xz|y$ , and  $yz|x$  (see Fig. 1).

For any tree  $T$  and  $\{x, y, z\} \subseteq \Lambda(T)$ , the fan triplet  $x|y|z$  is said to be *consistent with  $T$*  if  $\text{lca}^T(x, y) = \text{lca}^T(x, z) = \text{lca}^T(y, z)$ . Similarly, the rooted triplet  $xy|z$  is *consistent with  $T$*  if  $\text{lca}^T(x, y)$  is a proper descendant of  $\text{lca}^T(x, z) = \text{lca}^T(y, z)$ . Let  $T||_{\{x, y, z\}}$  denote the unique fan triplet or rooted triplet with leaf label set  $\{x, y, z\}$  which is consistent with  $T$ . Finally, for any tree  $T$ , let  $r(T)$  be the set of all rooted triplets which are consistent with  $T$ , i.e., define



**Fig. 1.** The four different triplets  $x|y|z$ ,  $xy|z$ ,  $xz|y$ , and  $yz|x$  leaf-labeled by  $\{x, y, z\}$

$r(T) = \{T||_{\{x, y, z\}} : \{x, y, z\} \subseteq \Lambda(T) \text{ and } T||_{\{x, y, z\}} \text{ is not a fan triplet}\}$ , and define  $t(T)$  as the set of *all* triplets (rooted triplets as well as fan triplets) consistent with  $T$ , i.e.,  $t(T) = \{T||_{\{x, y, z\}} : \{x, y, z\} \subseteq \Lambda(T)\}$ . It follows that  $|t(T)| = \Theta(|\Lambda(T)|^3)$  for any tree  $T$ , and  $|r(T)| = \Theta(|\Lambda(T)|^3)$  when  $T$  is a binary tree because  $|r(T)| = |t(T)|$  in this case.

## 2.2 Strong Clusters and Apresjan Clusters

Below, let  $\mathcal{R}$  be a given set of triplets over a leaf label set  $L = \bigcup_{r \in \mathcal{R}} \Lambda(r)$  such that for each  $\{x, y, z\} \subseteq L$ , at most one of  $x|y|z$ ,  $xy|z$ ,  $xz|y$ , and  $yz|x$  belongs to  $\mathcal{R}$ . A *cluster* of  $L$  is any subset of  $L$ . We define two special types of clusters:

- A cluster  $A$  of  $L$  is called a *strong cluster* of  $\mathcal{R}$  if  $aa'|x \in \mathcal{R}$  for all  $a, a' \in A$  and  $x \in L \setminus A$ . Furthermore,  $L$  as well as every singleton set of  $L$  is also defined to be a strong cluster of  $\mathcal{R}$ .
- For each  $a, b \in L$ , define  $s_{\mathcal{R}}(a, b) = |\{ab|y : ab|y \in \mathcal{R}\}|$ . A cluster  $A$  of  $L$  is called an *Apresjan cluster* of  $s_{\mathcal{R}}$  if  $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$  for all  $a, a' \in A$  and  $x \in L \setminus A$ .

Write  $n = |L|$ . By Theorem 2.3 and Corollary 2.1 of [3], the following holds:

**Lemma 1.** (*Bryant and Berry [3]*)

1. There are  $O(n)$  Apresjan clusters of  $s_{\mathcal{R}}$ .
2. Given the values of  $s_{\mathcal{R}}(a, b)$  for all  $a, b \in L$ , the Apresjan clusters of  $s_{\mathcal{R}}$  can be computed in  $O(n^2)$  time.

There is a relationship between the strong clusters of  $\mathcal{R}$  and the Apresjan clusters of  $s_{\mathcal{R}}$ :

**Lemma 2.** Every strong cluster of  $\mathcal{R}$  is an Apresjan cluster of  $s_{\mathcal{R}}$ .

*Proof.* Let  $C$  be a strong cluster of  $\mathcal{R}$ . Consider any fixed  $a, a' \in C$  and  $x \in L \setminus C$ . For every  $y \in L \setminus C$ , we have  $aa'|y \in \mathcal{R}$  by the definition of a strong cluster, which gives  $s_{\mathcal{R}}(a, a') = |\{aa'|y : aa'|y \in \mathcal{R}\}| \geq |L \setminus C|$ . In the same way,  $ab|x \in \mathcal{R}$  holds for every  $b \in C$ , which (along with the requirement that for each  $\{x, y, z\} \subseteq L$ , at most one of  $x|y|z$ ,  $xy|z$ ,  $xz|y$ , and  $yz|x$  belongs to  $\mathcal{R}$ ) implies that  $ax|b \notin \mathcal{R}$ . Thus,  $s_{\mathcal{R}}(a, x) = |\{ax|y : ax|y \in \mathcal{R}\}| \leq |(L \setminus C) \setminus \{x\}| < |L \setminus C|$ . We have just shown that  $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$ , so  $C$  is an Apresjan cluster of  $s_{\mathcal{R}}$ .  $\square$

### 2.3 R\* Consensus Trees

Let  $T_1$  and  $T_2$  be two given trees with  $\Lambda(T_1) = \Lambda(T_2) = L$ . For any  $\{a, b, c\} \subseteq L$ , define  $\#ab|c$  as the number of trees  $T_i$  for which  $ab|c \in r(T_i)$ . The set of “majority rooted triplets”  $\mathcal{R}_{maj}$  is defined as  $\{ab|c : a, b, c \in L \text{ and } \#ab|c > \#ac|b, \#bc|a\}$ . An  $R^*$  consensus tree of  $T_1$  and  $T_2$  is a tree  $\tau$  with  $\Lambda(\tau) = L$  which satisfies  $r(\tau) \subseteq \mathcal{R}_{maj}$  and which maximizes the number of internal nodes.

The next two lemmas describe some useful properties of the strong clusters of  $\mathcal{R}_{maj}$ .

**Lemma 3.** *Let  $T$  be a tree with  $\Lambda(T) = L$  and  $r(T) \subseteq \mathcal{R}_{maj}$ . For any node  $u$  of  $T$ ,  $\Lambda(T[u])$  is a strong cluster of  $\mathcal{R}_{maj}$ .*

*Proof.* If  $u$  is a leaf or the root of  $T$  then  $\Lambda(T[u])$  is trivially a strong cluster of  $\mathcal{R}_{maj}$ . If  $u$  is an internal node then for any two  $a, a' \in \Lambda(T[u])$  and any  $x \notin \Lambda(T[u])$ , the triplet  $aa'|x$  belongs to  $\mathcal{R}_{maj}$ , so  $\Lambda(T[u])$  is a strong cluster of  $\mathcal{R}_{maj}$ .  $\square$

**Lemma 4.** *There exists a tree  $\tau$  such that the set  $\{\Lambda(\tau[u]) : u \text{ is a node in } \tau\}$  equals the set of strong clusters of  $\mathcal{R}_{maj}$ .*

*Proof.* First observe that for any two strong clusters  $A$  and  $B$  of  $\mathcal{R}$ , either  $A \subsetneq B$ ,  $B \subsetneq A$ , or  $A \cap B = \emptyset$ . To prove this claim, suppose for the sake of contradiction that there are  $x, y, z \in L$  such that  $x \in A \setminus B$ ,  $y \in B \setminus A$ , and  $z \in A \cap B$ . Since  $A$  is a strong cluster,  $xz|y \in \mathcal{R}$ . Since  $B$  is a strong cluster,  $yz|x \in \mathcal{R}$ . By the definition of  $\mathcal{R}_{maj}$ , we cannot have both of  $xz|y$  and  $yz|x$  in  $\mathcal{R}_{maj}$ . The claim follows. This implies that the strong clusters of  $\mathcal{R}_{maj}$  form a hierarchy (laminar family) on  $L$ .

Next, if  $A$  and  $B$  are strong clusters of  $\mathcal{R}_{maj}$  with  $A \subsetneq B$  then there must exist some strong cluster  $C$  of  $\mathcal{R}_{maj}$  such that  $C \subsetneq B$  and  $C \cap A = \emptyset$ . (To see this, take any  $c \in B \setminus A$  and recall that  $\{c\}$  is by definition a strong cluster of  $\mathcal{R}_{maj}$ .)

By these two observations, there exists a tree  $\tau$  leaf-labeled by  $L$  such that: (1) each strong cluster  $A$  of  $\mathcal{R}_{maj}$  corresponds to a node in  $\tau$  whose set of descendants is precisely  $A$ ; (2)  $A \subsetneq B$ , where  $A$  and  $B$  are strong clusters of  $\mathcal{R}_{maj}$ , implies that the node corresponding to  $A$  is a proper descendant of the node corresponding to  $B$ ; and (3) every internal node of  $\tau$  has at least two children. Hence, the lemma follows.  $\square$

Say that a tree  $T$  includes a cluster  $A$  of  $L$  if  $T$  contains a node  $u$  such that  $\Lambda(T[u]) = A$ .

**Theorem 1.** *The  $R^*$  consensus tree of  $T_1$  and  $T_2$  exists and is unique. In particular, it includes every strong cluster of  $\mathcal{R}_{maj}$  and no other clusters of  $L$ .*

*Proof.* By Lemma 4, there exists a (unique) tree  $\tau$  that includes all the strong clusters of  $\mathcal{R}_{maj}$  and does not include any other clusters. For any  $aa'|x \in r(\tau)$ , there exists a node  $u$  in  $\tau$  such that  $a, a' \in \Lambda(\tau[u])$  and  $x \notin \Lambda(\tau[u])$ , i.e.,  $a, a' \in A$

and  $x \notin A$  for some strong cluster  $A$  of  $\mathcal{R}_{maj}$ , which implies that  $ad'|x \in \mathcal{R}_{maj}$ . Thus,  $r(\tau) \subseteq \mathcal{R}_{maj}$ .

To prove the optimality of  $\tau$ , suppose that there exists a tree  $\tau'$  which satisfies  $r(\tau') \subseteq \mathcal{R}_{maj}$  and has more internal nodes than  $\tau$ . By Lemma 3, the set of leaves in each rooted subtree of  $\tau'$  forms a strong cluster of  $\mathcal{R}_{maj}$ . Since  $\tau'$  has more internal nodes than  $\tau$ , it follows that  $\tau'$  includes some strong cluster of  $\mathcal{R}_{maj}$  which  $\tau$  does not include. This contradicts Lemma 4. Hence,  $\tau$  is an R\* consensus tree of  $T_1$  and  $T_2$ .  $\square$

### 3 Constructing the R\* Consensus Tree

Based on the discussion in Sections 2.2 and 2.3, we can construct the R\* consensus tree of two given trees  $T_1$  and  $T_2$  as follows: First compute  $s_{\mathcal{R}_{maj}}$  and all the Apresjan clusters of  $s_{\mathcal{R}_{maj}}$ .<sup>1</sup> Then, check each Apresjan cluster to see if it is a strong cluster of  $\mathcal{R}_{maj}$  (by Lemma 2, the set of strong clusters of  $\mathcal{R}_{maj}$  is a subset of the set of Apresjan clusters of  $s_{\mathcal{R}_{maj}}$ ). Finally, build a tree which includes all the strong clusters of  $\mathcal{R}_{maj}$  in accordance with Theorem 1. The algorithm is named `R*_consensus_tree` and is outlined in Fig. 2.

**Algorithm R\*\_consensus\_tree**

**Input:** Two trees  $T_1, T_2$  with  $\Lambda(T_1) = \Lambda(T_2)$ .  
**Output:** The R\* consensus tree of  $T_1$  and  $T_2$ .

- 1: Define  $L = \Lambda(T_1) = \Lambda(T_2)$  and compute  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$  as described in Section 4.
- 2: Compute the Apresjan clusters of  $s_{\mathcal{R}_{maj}}$ .
- 3: **for** each Apresjan cluster  $A$  of  $s_{\mathcal{R}_{maj}}$  **do**
- 4:   Determine if  $A$  is a strong cluster of  $\mathcal{R}_{maj}$  as described in Section 5.
- 5: **end for**
- 6: Construct the R\* consensus tree using all the strong clusters of  $\mathcal{R}_{maj}$ .

**Fig. 2.** Algorithm `R*_consensus_tree`

We now analyze the time complexity of Algorithm `R*_consensus_tree`. In Section 4 below, we shall describe how to implement step 1 in  $O(n^2 \log n)$  time. Next, in step 2, the Apresjan clusters of  $s_{\mathcal{R}_{maj}}$  can be computed by running the algorithm of Bryant and Berry [3], which takes  $O(n^2)$  time according to Lemma 1 (see also Corollary 2.1 in [3]). There are  $O(n)$  Apresjan clusters in the loop of step 3 by Lemma 1, and each one is checked in  $O(n)$  time in step 4, as detailed in Section 5 below. Finally, the last step builds the R\* consensus tree from the strong clusters of  $\mathcal{R}_{maj}$  in  $O(n^2)$  time. In summary, we have the following theorem.

<sup>1</sup> Recall from Section 2.2 that for a set  $\mathcal{R}$  of triplets over a leaf label set  $L$ , the function  $s_{\mathcal{R}}$  is defined on each  $a, b \in L$  by  $s_{\mathcal{R}}(a, b) = |\{ab|y : ab|y \in \mathcal{R}\}|$ .

**Theorem 2.** Algorithm `R*_consensus_tree` constructs the  $R^*$  consensus tree of  $T_1$  and  $T_2$  in  $O(n^2 \log n)$  time.

The following two sections are devoted to implementing steps 1 and 4 of Algorithm `R*_consensus_tree` efficiently.

## 4 Computing $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$

From the definition of  $\mathcal{R}_{maj}$ , we have:

**Lemma 5.** For any  $a, b, c \in L$ ,  $ab|c \in \mathcal{R}_{maj}$  if and only if either

1.  $ab|c \in t(T_1) \cap t(T_2)$ ; or
2.  $ab|c \in t(T_1)$  and  $a|b|c \in t(T_2)$ ; or
3.  $a|b|c \in t(T_1)$  and  $ab|c \in t(T_2)$ .

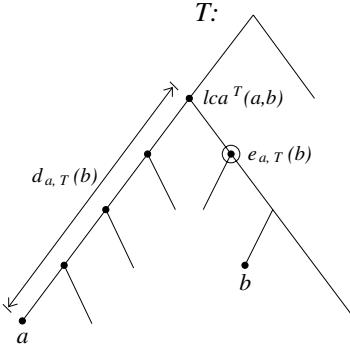
We introduce the following three auxiliary functions:

$$\begin{cases} count_1(a, b) = |\{w \in L \setminus \{a, b\} : ab|w \in t(T_1) \cap t(T_2)\}| \\ count_2(a, b) = |\{w \in L \setminus \{a, b\} : ab|w \in t(T_1), a|b|w \in t(T_2)\}| \\ count_3(a, b) = |\{w \in L \setminus \{a, b\} : a|b|w \in t(T_1), ab|w \in t(T_2)\}| \end{cases}$$

Then, Lemma 5 immediately gives  $s_{\mathcal{R}_{maj}}(a, b) = count_1(a, b) + count_2(a, b) + count_3(a, b)$  for any  $a, b \in L$ .

To compute  $count_1$ ,  $count_2$ , and  $count_3$ , we could preprocess  $T_1$  and  $T_2$  in  $O(n)$  time so that lowest common ancestor queries in  $T_1$  and  $T_2$  can be answered in  $O(1)$  time [147]. Then, for any given  $a, b \in L$ , a brute-force solution would easily obtain all of  $count_1(a, b)$ ,  $count_2(a, b)$ , and  $count_3(a, b)$  in  $O(n)$  time by checking  $T_1||_{\{a, b, w\}}$  and  $T_2||_{\{a, b, w\}}$  for each  $w \in L \setminus \{a, b\}$ . This approach would therefore compute  $count_i(a, b)$  for all  $a, b \in L$  and all  $i = 1, 2, 3$  in  $O(n^3)$  time. In the rest of this section, we show how to obtain  $count_i(a, b)$  for all  $a, b \in L$  and  $i = 1, 2, 3$  more efficiently.

First, in Section 4.1, we reformulate the  $count_i(a, b)$ -functions in terms of distances from leaves to lowest common ancestors of leaves. All of these distances may be computed in  $O(n^2)$  time. Then, based on this alternative formulation, for any fixed leaf label  $a \in L$ , Section 4.2 describes an  $O(n \log n)$ -time method for computing  $count_1(a, b)$  for all  $b \in L \setminus \{a\}$ , and Section 4.3 describes an  $O(n)$ -time method for computing  $count_2(a, b)$  for all  $b \in L \setminus \{a\}$ . (By symmetry, we can obtain  $count_3(a, b)$  for all  $b \in L \setminus \{a\}$  in  $O(n)$  time with the same technique as in Section 4.3.) To summarize, after  $O(n^2)$  time preprocessing,  $O(n \log n)$  time is enough to compute  $count_1(a, b)$ ,  $count_2(a, b)$ , and  $count_3(a, b)$  for all  $b \in L \setminus \{a\}$  for any fixed  $a \in L$ . Therefore, we only need  $O(n^2 \log n)$  time in total to obtain  $s_{\mathcal{R}_{maj}}(a, b)$  for all  $a, b \in L$ .



**Fig. 3.** Illustrating the definitions of  $d_{a,T}(b)$  and  $e_{a,T}(b)$

#### 4.1 Distances from Leaves to Lowest Common Ancestors

For any tree  $T$  and any  $a, b \in \Lambda(T)$ , let  $d_{a,T}(b)$  be the distance between  $a$  and  $\text{lca}^T(a, b)$  in  $T$ , and let  $e_{a,T}(b)$  be the child of  $\text{lca}^T(a, b)$  which is an ancestor of  $b$ . (W.l.o.g., define  $e_{a,T}(a) = \emptyset$ .) See Fig. 3 for an example. Note that the values of  $d_{a,T}(b)$  and  $e_{a,T}(b)$  for all  $a, b \in \Lambda(T)$  can be computed in  $O(n^2)$  time by a bottom-up traversal. The next lemma states the connection between  $d$  and  $e$  and the triplets consistent with  $T$ .

**Lemma 6.** *For any tree  $T$  and any  $a, x, w \in \Lambda(T)$ , it holds that:*

- If  $d_{a,T}(x) < d_{a,T}(w)$  then  $ax|w \in t(T)$ .
- If  $d_{a,T}(x) = d_{a,T}(w)$  and  $e_{a,T}(x) \neq e_{a,T}(w)$  then  $a|x|w \in t(T)$ .

*Proof.* If  $d_{a,T}(x) < d_{a,T}(w)$ , the  $\text{lca}^T(a, w)$  is an ancestor of  $\text{lca}^T(a, x)$ . Thus,  $ax|w \in t(T)$ .

If  $d_{a,T}(x) = d_{a,T}(w)$  then  $v = \text{lca}^T(a, w) = \text{lca}^T(a, x)$ . Since  $e_{a,T}(x) \neq e_{a,T}(w)$ , we know that  $\text{lca}^T(x, w) = v$ . Hence,  $a|x|w \in t(T)$ .  $\square$

We remark that the first part of Lemma 6 is a special case of Theorem 1 in [9], which was derived by Lee *et al.* [9] to solve a different problem known as *the maximum agreement subtree problem*.

Next, consider two trees  $T_1$  and  $T_2$  with  $\Lambda(T_1) = \Lambda(T_2) = L$ . We have:

**Lemma 7.** *For any  $a, b, w \in L$ :*

- $\text{count}_1(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w) \text{ and } d_{a,T_2}(b) < d_{a,T_2}(w)\}|$ .
- $\text{count}_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), d_{a,T_2}(b) = d_{a,T_2}(w), \text{ and } e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$ .
- $\text{count}_3(a, b) = |\{w : d_{a,T_1}(b) = d_{a,T_1}(w), e_{a,T_1}(b) \neq e_{a,T_1}(w), \text{ and } d_{a,T_2}(b) < d_{a,T_2}(w)\}|$ .

*Proof.* By Lemma 6,  $d_{a,T_1}(b) < d_{a,T_1}(w)$  and  $d_{a,T_2}(b) < d_{a,T_2}(w)$  mean that  $ab|w \in t(T_1) \cap t(T_2)$ . Hence,  $\text{count}_1(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w) \text{ and } d_{a,T_2}(b) < d_{a,T_2}(w)\}|$ .

By Lemma 6 again,  $d_{a,T_1}(b) < d_{a,T_1}(w)$ ,  $d_{a,T_2}(b) = d_{a,T_2}(w)$ , and  $e_{a,T_2}(b) \neq e_{a,T_2}(w)$  mean that  $ab|w \in t(T_1)$  and  $a|b|w \in t(T_2)$ . Hence,  $\text{count}_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), d_{a,T_2}(b) = d_{a,T_2}(w), \text{ and } e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$ .

The formula for  $\text{count}_3(a, b)$  follows by symmetry.  $\square$

## 4.2 Computing $\text{count}_1(a, b)$ for All $b \in L \setminus \{a\}$

This section describes how to compute  $\text{count}_1(a, b)$  for all  $b \in L \setminus \{a\}$  in  $O(n \log n)$  time for any fixed  $a \in L$ , assuming the values of  $d_{a,T_1}(b)$ ,  $d_{a,T_2}(b)$  for all  $b \in L \setminus \{a\}$  have been precomputed.

By applying a stable sorting, all elements in  $L \setminus \{a\}$  can be ordered using  $O(n)$  time so that  $x$  is before  $y$  if either: (1)  $d_{a,T_1}(x) < d_{a,T_1}(y)$ ; or (2)  $d_{a,T_1}(x) = d_{a,T_1}(y)$  and  $d_{a,T_2}(x) < d_{a,T_2}(y)$ . Suppose the resulting ordering of  $L \setminus \{a\}$  is  $x_{(1)}, x_{(2)}, \dots, x_{(n-1)}$ . For any  $x_{(i)}$ , let  $i'$  be the smallest integer which is larger than  $i$  such that  $d_{a,T_1}(x_{(i')}) \neq d_{a,T_1}(x_{(i)})$ . Then,  $\text{count}_1$  obeys the following:

**Lemma 8.** *For any  $i \in \{1, 2, \dots, n-1\}$ ,  $\text{count}_1(a, x_{(i)})$  equals the number of elements in  $\{d_{a,T_2}(x_{(j)}) : j \geq i'\}$  which are strictly larger than  $d_{a,T_2}(x_{(i)})$ .*

*Proof.* From Lemma 7,  $\text{count}_1(a, x_{(i)}) = |\{x_{(j)} : d_{a,T_1}(x_{(i)}) < d_{a,T_1}(x_{(j)}) \text{ and } d_{a,T_2}(x_{(i)}) < d_{a,T_2}(x_{(j)})\}|$ . By the definition of  $i'$ ,  $\text{count}_1(a, x_{(i)}) = |\{x_{(j)} : j \geq i' \text{ and } d_{a,T_2}(x_{(i)}) < d_{a,T_2}(x_{(j)})\}|$ . The lemma follows.  $\square$

We compute  $\text{count}_1(a, x_{(i)})$  for all  $i$  in decreasing order from  $n-1$  to 1 iteratively, as illustrated in Algorithm Compute\\_count<sub>1</sub> in Fig. 4. We maintain the invariant

Algorithm Compute\_count<sub>1</sub>

**Input:**  $a \in L$ .

**Output:** The values of  $\text{count}_1(a, b)$  for all  $b \in L \setminus \{a\}$ .

- 1: Perform a stable sorting to rank the elements of  $L \setminus \{a\}$  according to  $d_{a,T_1}$  and  $d_{a,T_2}$ , and obtain the ordering  $x_{(1)}, x_{(2)}, \dots, x_{(n-1)}$ .
- 2: Let  $D$  be an empty binary search tree.
- 3: Let  $i' = n$  and define  $d_{a,T_1}(x_{(n)}) = d_{a,T_2}(x_{(n)}) = \infty$ .
- 4: **for**  $i = n-1$  down to 1 **do**
- 5:   **if**  $d_{a,T_1}(x_{(i)}) < d_{a,T_1}(x_{(i+1)})$  **then**
- 6:     Insert  $d_{a,T_2}(x_{(j)})$  into  $D$  for  $j = i+1, \dots, i'-1$ .
- 7:     Let  $i' = i+1$ .
- 8:   **end if**
- 9:   Set  $\text{count}_1(a, x_{(i)})$  to the number of elements in  $D$  which are strictly greater than  $d_{a,T_2}(x_{(i)})$ .
- 10: **end for**

Fig. 4. Algorithm Compute\_count<sub>1</sub>

that in every iteration  $i$ , the value  $i'$  equals the smallest integer larger than  $i$  such that  $d_{a,T_1}(x_{(i')}) \neq d_{a,T_1}(x_{(i)})$ , and keep a binary search tree  $D$  for  $\{d_{a,T_2}(x_{(j)}) : j \geq i'\}$  so that by Lemma 8,  $count_1(a, x_{(i)})$  equals the number of elements in  $D$  strictly greater than  $d_{a,T_2}(x_{(i)})$ . Each operation on a binary search tree takes  $O(\log n)$  time, so in total, `Compute_count1` runs in  $O(n \log n)$  time.

### 4.3 Computing $count_2(a, b)$ for All $b \in L \setminus \{a\}$

Here, we show how to compute  $count_2(a, b)$  (and by symmetry,  $count_3(a, b)$ ) for all  $b \in L \setminus \{a\}$  in  $O(n)$  time for any fixed  $a \in L$ . We assume that the values of  $d_{a,T_1}(b), d_{a,T_2}(b), e_{a,T_1}(b), e_{a,T_2}(b)$  for all  $b \in L \setminus \{a\}$  have been precomputed.

The algorithm is named `Compute_count2` and is listed in Fig. 5. It first partitions  $L \setminus \{a\}$  into disjoint subsets  $C_1, C_2, \dots, C_p$  in such a way that for every pair of elements  $x, y$  in the same  $C_i$ , it holds that  $d_{a,T_2}(x) = d_{a,T_2}(y)$ . Then, the algorithm further partitions each  $C_i$  into  $C_{ij}, \dots, C_{iq}$  so that for every pair of elements  $x, y$  in the same  $C_{ij}$ , it holds that  $e_{a,T_2}(x) = e_{a,T_2}(y)$ . Finally, the algorithm obtains  $count_2(a, x)$  for all  $x \in L \setminus \{a\}$  in  $O(n)$  time based on Lemma 9.

**Lemma 9.** *Let  $b \in L \setminus \{a\}$  and suppose  $b \in C_{ij}$ . Then,  $count_2(a, b) = |\{w \in C_i : d_{a,T_1}(w) > d_{a,T_1}(b)\}| - |\{w \in C_{ij} : d_{a,T_1}(w) > d_{a,T_1}(b)\}|$ .*

*Proof.* By Lemma 7,  $count_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), d_{a,T_2}(b) = d_{a,T_2}(w), \text{ and } e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$ . Since  $b \in C_{ij}$ , we have: (1)  $w \in C_i$  means  $d_{a,T_2}(b) = d_{a,T_2}(w)$ ; and (2)  $w \in C_{ij}$  means  $e_{a,T_2}(b) = e_{a,T_2}(w)$ . Therefore,  $count_2(a, b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), w \in C_i, \text{ and } w \notin C_{ij}\}|$ .  $\square$

## 5 Determining If a Cluster Is a Strong Cluster

This section shows how to check whether a given cluster  $A$  of  $L$  is a strong cluster of  $\mathcal{R}_{maj}$  in  $O(n)$  time. Our solution depends on the following lemma.

**Lemma 10.** *Let  $u_1$  and  $u_2$  be the lowest common ancestor of  $A$  in  $T_1$  and  $T_2$ , respectively.  $A$  is a strong cluster of  $\mathcal{R}_{maj}$  if and only if:*

- (1) *For  $i = 1, 2$ , each subtree  $B$  attached to  $u_i$  in  $T_i$  satisfies either  $\Lambda(B) \subseteq A$  or  $\Lambda(B) \subseteq L \setminus A$ ; and*
- (2)  *$X_1 \cap X_2 = \emptyset$ , where  $X_i = (L \setminus A) \cap \Lambda(T_i[u_i])$ .*

*Proof.* ( $\Rightarrow$ ) Let  $A$  be a strong cluster of  $\mathcal{R}_{maj}$ . We need to prove that both conditions (1) and (2) hold.

For the sake of obtaining a contradiction, suppose that for some  $i \in \{1, 2\}$ , there exist  $a \in A, x \in L \setminus A$  where both  $a$  and  $x$  are in the same subtree attached to  $u_i$ . Then,  $lca^{T_i}(a, x)$  is a proper descendant of  $u_i$ . The node  $u_i$  is defined as the lowest common ancestor of the leaves in  $A$ , so  $lca^{T_i}(a, a') = u_i$  for some  $a' \in A$ . However, then  $lca^{T_i}(a, x)$  is a proper descendant of  $lca^{T_i}(a, a')$ , giving  $ax|a' \in t(T_i)$ . This implies that  $aa'|x$  cannot belong to  $\mathcal{R}_{maj}$ , which contradicts the fact that  $A$  is a strong cluster of  $\mathcal{R}_{maj}$ . Thus, condition (1) must hold.

**Algorithm Compute\_count<sub>2</sub>**
**Input:**  $a \in L$ .

**Output:** The values of  $count_2(a, b)$  for all  $b \in L \setminus \{a\}$ .

```

1: Partition  $L \setminus \{a\}$  into  $C_1, \dots, C_p$  so that for every pair of elements  $x, y$  in the
   same  $C_i$ , it holds that  $d_{a,T_2}(x) = d_{a,T_2}(y)$ .
2: for every  $C_i$  do
3:   Perform a stable sorting to rank the elements of  $C_i$  and obtain the ordering
       $b_1, \dots, b_{|C_i|}$  such that  $d_{a,T_1}(b_1) \leq d_{a,T_2}(b_2) \leq \dots \leq d_{a,T_2}(b_{|C_i|})$ .
4:   { Note that  $s_i(b) = |\{w \in C_i : d_{a,T_1}(w) > d_{a,T_1}(b)\}|$ . }
5:    $s_i(b_1) = 0$ .
6:   for  $j = 2, 3, \dots, |C_i|$  do
7:     if  $d_{a,T_1}(b_j) > d_{a,T_1}(b_{j-1})$  then
8:        $s_i(b_j) = s_i(b_{j-1}) + 1$ .
9:     else
10:     $s_i(b_j) = s_i(b_{j-1})$ .
11:   end if
12: end for
13: Partition  $C_i$  into  $C_{i1}, \dots, C_{iq}$  so that for every pair of elements  $x, y$  in the
   same  $C_{ij}$ , it holds that  $e_{a,T_2}(x) = e_{a,T_2}(y)$ .
14: for every  $C_{ij}$  do
15:   Perform a stable sorting to rank the elements of  $C_{ij}$  and obtain the or-
      dering  $b_1, \dots, b_{|C_{ij}|}$  such that  $d_{a,T_1}(b_1) \leq d_{a,T_2}(b_2) \leq \dots \leq d_{a,T_2}(b_{|C_{ij}|})$ .
16:   { Note that  $s_{ij}(b) = |\{w \in C_{ij} : d_{a,T_1}(w) > d_{a,T_1}(b)\}|$ . }
17:    $s_{ij}(b_1) = 0$ .
18:   for  $k = 2, 3, \dots, |C_{ij}|$  do
19:     if  $d_{a,T_1}(b_k) > d_{a,T_1}(b_{k-1})$  then
20:        $s_{ij}(b_k) = s_{ij}(b_{k-1}) + 1$ .
21:     else
22:        $s_{ij}(b_k) = s_{ij}(b_{k-1})$ .
23:     end if
24:   end for
25:   for every  $b \in C_{ij}$  do
26:     Let  $count_2(a, b)$  equal  $s_i(b) - s_{ij}(b)$ .
27:   end for
28: end for
29: end for

```

**Fig. 5.** Algorithm Compute\_count<sub>2</sub>

Next, we prove by contradiction that condition (2) must hold. If  $X_1 \cap X_2 \neq \emptyset$ , where  $X_i = (L \setminus A) \cap \Lambda(T_i[u_i])$ , is true then there exists an  $x \in X_1 \cap X_2$ . We claim that there exist  $a, a' \in A$  with  $\text{lca}^{T_1}(a, a') = u_1$  and  $\text{lca}^{T_2}(a, a') = u_2$ . This yields  $x|a|a' \in t(T_1)$  and  $x|a|a' \in t(T_2)$  because the subtree attached to  $u_i$  for  $i = 1, 2$  which contains  $x$  cannot contain  $a$  or  $a'$  by condition (1) above. Thus,  $aa'|x \notin \mathcal{R}_{maj}$ , contradicting the fact that  $A$  is a strong cluster of  $\mathcal{R}_{maj}$ .

It remains to prove the claim that there exist  $a, a' \in A$  such that  $\text{lca}^{T_1}(a, a') = u_1$  and  $\text{lca}^{T_2}(a, a') = u_2$ . Assume on the contrary that for each pair  $a, a' \in A$ ,

```
Algorithm Check_if_strong_cluster
```

**Input:** A cluster  $A$  of  $L$ .

**Output:** “Yes”, if  $A$  is a strong cluster of  $\mathcal{R}_{maj}$ ; “no”, otherwise.

```

1: Let  $u_1$  and  $u_2$  be the lowest common ancestor of  $A$  in  $T_1$  and  $T_2$ , respectively.
2: for  $i = 1, 2$  do
3:   if there exists a subtree in  $T_i$  attached to  $u_i$  containing leaves from  $A$  as well
      as from  $L \setminus A$  then
4:     return “no”
5:   end if
6: end for
7: Let  $X_i = (L \setminus A) \cap \Lambda(T_i[u_i])$  for  $i = 1, 2$ .
8: if  $X_1 \cap X_2 = \emptyset$  then
9:   return “yes”
10: else
11:   return “no”
12: end if
```

Fig. 6. Algorithm Check\_if\_strong\_cluster

there exists an  $i \in \{1, 2\}$  such that  $\text{lca}^{T_i}(a, a')$  is a proper descendant of  $u_i$ . Note that  $A$  is located in at least two subtrees attached to  $u_i$  for  $i = 1, 2$ . For every pair of subtrees attached to  $u_1$  that contain elements from  $A$ , suppose the sets of leaves in the two subtrees are  $A^1$  and  $A^2$ . To ensure that  $\text{lca}^{T_2}(x, y)$  is a proper descendant of  $u_2$  for every  $x \in A^1$  and  $y \in A^2$ , we need  $A^1 \cup A^2$  to appear in one subtree attached to  $u_2$ . By this argument, all elements of  $A$  appear in one subtree attached to  $u_2$ . Then,  $u_2$  cannot be the lowest common ancestor of  $A$  in  $T_2$ , and we arrive at a contradiction.

( $\Leftarrow$ ) Suppose  $A$  satisfies the two conditions stated in the lemma. We need to prove that for every  $x, y \in A$  and  $z \in L \setminus A$ , it holds that  $xy|z \in \mathcal{R}_{maj}$ . Note that  $z$  belongs to either  $X_1$ ,  $X_2$ , or  $((L \setminus A) \setminus X_1) \setminus X_2$ , which gives three cases:

- If  $z \in ((L \setminus A) \setminus X_1) \setminus X_2$ : Then  $\text{lca}^{T_i}(x, z)$  is a proper ancestor of  $\text{lca}^{T_i}(x, y)$  for  $i = 1, 2$ . Thus,  $xy|z \in \mathcal{R}_{maj}$ .
- If  $z \in X_1$ : Then  $xy|z \in t(T_2)$  and there are two cases depending on whether or not  $\text{lca}^{T_1}(x, y)$  is a proper descendant of  $u_1$ . If yes, then  $xy|z \in t(T_1)$ , and thus  $xy|z \in \mathcal{R}_{maj}$ . If no, then  $\text{lca}^{T_1}(x, y) = u_1$ , and thus  $x|y|z \in t(T_1)$ , which also yields  $xy|z \in \mathcal{R}_{maj}$ .
- If  $z \in X_2$ : Then it follows that  $xy|z \in \mathcal{R}_{maj}$  in the same way as for the case  $z \in X_1$  above.  $\square$

We can use Lemma 10 to check if a given cluster  $A$  of  $L$  is a strong cluster of  $\mathcal{R}_{maj}$  in  $O(n)$  time, as shown in Algorithm Check\_if\_strong\_cluster in Fig. 6.

## 6 Concluding Remarks

In this paper, we have described how to compute the R\* consensus tree of two input trees in  $O(n^2 \log n)$  time. The definition of the set of majority rooted

triplets  $\mathcal{R}_{maj}$  as well as the definition of an  $R^*$  consensus tree can be naturally extended to the case of  $k > 2$  input trees; see [2]. The currently fastest algorithm for the case  $k > 2$  runs in  $O(kn^3)$  time and is outlined in [2]. It can be implemented by explicitly constructing the sets  $r(T_i)$  for every input tree  $T_i$  using  $O(kn^3)$  total time, then obtaining  $\mathcal{R}_{maj}$  in  $O(kn^3)$  time by finding the most frequently occurring rooted triplet (if it exists) for each  $\{x, y, z\}$  in the leaf label set in  $O(k)$  time, and finally applying the  $O(n^3)$ -time strong clustering algorithm from Corollary 2.2 in [3] to  $\mathcal{R}_{maj}$ . An open problem is to reduce the time complexity to  $o(kn^3)$ . It seems difficult to extend the approach used in this paper because it would yield an exponential number (in  $k$ ) of cases in Lemma 5 and thus express  $s_{\mathcal{R}_{maj}}(a, b)$  as the sum of an exponential number of auxiliary *count*-functions, causing the total running time to be exponential in  $k$ .

## Acknowledgments

The authors would like to thank David Bryant for some clarifications and the anonymous referees for their helpful comments.

## References

1. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
2. Bryant, D.: A classification of consensus methods for phylogenetics. In: Janowitz, M.F., Lapointe, F.-J., McMorris, F.R., Mirkin, B., Roberts, F.S. (eds.) Bioconsensus. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 61, pp. 163–184. American Mathematical Society, Providence (2003)
3. Bryant, D., Berry, V.: A structured family of clustering and tree construction methods. *Advances in Applied Mathematics* 27(4), 705–732 (2001)
4. Degnan, J.H., DeGiorgio, M., Bryant, D., Rosenberg, N.A.: Properties of consensus methods for inferring species trees from gene trees. *Systematic Biology* 58(1), 35–54 (2009)
5. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates, Inc., Sunderland (2004)
6. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
7. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing* 13(2), 338–355 (1984)
8. Kannan, S., Warnow, T., Yooseph, S.: Computing the local consensus of trees. *SIAM Journal on Computing* 27(6), 1695–1724 (1998)
9. Lee, C.-M., Hung, L.-J., Chang, M.-S., Shen, C.-B., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. *Information Processing Letters* 94(5), 211–216 (2005)
10. Nakhleh, L., Warnow, T., Ringe, D., Evans, S.N.: A comparison of phylogenetic reconstruction methods on an Indo-European dataset. *Transactions of the Philosophical Society* 103(2), 171–192 (2005)

# Author Index

- Abraham, Ittai II-87  
Adler, Isolde I-97  
Agarwal, Pankaj K. I-487  
Ajwani, Deepak II-75  
Anschelevich, Elliot I-158  
Arya, Sunil I-374  
Azar, Yossi II-51
- Bae, Sang Won I-500  
Bahmani, Bahman I-170  
Bansal, Nikhil II-218  
Bartal, Yair II-87  
Bar-Yehuda, Reuven I-255  
Bast, Hannah I-290  
Belazzougui, Djamal I-427  
Bendich, Paul I-1  
Berenbrink, Petra I-134  
Bhawalkar, Kshipra II-17  
Boldi, Paolo I-427  
Bonifaci, Vincenzo II-230  
Brodal, Gerth Stølting II-171  
Buchbinder, Niv II-51  
Buchin, Kevin I-110, I-463, II-63  
Buchin, Maike I-463, II-63  
Büsing, Christina I-350
- Carlsson, Erik I-290  
Chan, Sze-Hang I-23  
Chao, Kun-Mao I-415  
Chechik, Shiri I-84  
Chen, Kuan-Yu I-415  
Chen, Ning II-147  
Chrobak, Marek I-195  
Cicalese, Ferdinando I-207  
Colin de Verdière, Éric II-100  
Cormode, Graham I-231  
Crescenzi, Pierluigi I-302  
Culpepper, J. Shane II-194  
Cygan, Marek I-72  
Czumaj, Artur I-410
- da Fonseca, Guilherme D. I-374  
Das, Abhimanyu I-219  
Davoodi, Pooya II-171
- Dickerson, Matthew T. I-362  
Dorn, Frederic I-97
- Edelsbrunner, Herbert I-1  
Eigenwillig, Arno I-290  
Eisenbrand, Friedrich I-11  
Elkin, Michael I-48  
Elmasry, Amr II-183  
Elsässer, Robert I-134  
Eppstein, David I-362
- Feldman, Jon I-182  
Ferragina, Paolo II-1  
Fomin, Fedor V. I-97  
Friedler, Sorelle A. I-386  
Fujita, Ryo II-123  
Fujiwara, Hiroshi I-439
- Gairing, Martin II-17  
Galli, Laura I-338  
Gaspers, Serge I-267  
Geisberger, Robert I-290  
Ghosh, Arpita II-147  
Gibson, Matt I-243  
Giesen, Joachim I-524  
Goodrich, Michael T. I-362  
Grandoni, Fabrizio I-536  
Grossi, Roberto I-302  
Gupta, Anupam II-218  
Gutin, Gregory I-326
- Hajiaghayi, MohammadTaghi I-314  
Halperin, Dan I-398  
Harks, Tobias II-29  
Harrelson, Chris I-290  
Hellweg, Frank I-60  
Hemmer, Michael I-398  
Henzinger, Monika I-182  
Hermelin, Danny I-255  
Hoefer, Martin I-158, II-29
- Iersel, Leo van I-326  
Imbrenda, Claudio I-302  
Iwama, Kazuo II-135

- Jacobs, Tobias I-439  
 Jaggi, Martin I-524  
 Jain, Kamal II-51  
 Jansson, Jesper I-573
- Kamiński, Marcin I-122  
 Kang, Ross J. II-112  
 Kaplan, Haim I-475  
 Kapralov, Michael I-170  
 Kärkkäinen, Juha I-451  
 Katz, Matthew J. I-475  
 Kempe, David I-219  
 Kesavan, Karthikeyan I-11  
 Khandekar, Rohit I-314  
 Klimm, Max II-29  
 Kobayashi, Yusuke II-123  
 Korman, Matias I-500  
 Kortsarz, Guy I-314  
 Korula, Nitish I-182  
 Kowalik, Lukasz I-72  
 Kreveld, Marc van I-463  
 Krysta, Piotr II-39
- Lampis, Michael I-549  
 Lam, Tak-Wah I-23  
 Langberg, Michael I-84  
 Lanzi, Leonardo I-302  
 Laue, Sören I-524  
 Lee, Lap-Kei I-23  
 Li, Jian II-218  
 Löffler, Maarten I-463  
 Lorenz, Ulf I-512
- Makino, Kazuhisa I-195, II-123  
 Marchetti-Spaccamela, Alberto II-230  
 Marino, Andrea I-302  
 Martin, Alexander I-512  
 Mattikalli, Raju S. I-11  
 Maue, Jens I-350  
 Mestre, Julián II-218  
 Mirrokni, Vahab S. I-182  
 Mitzenmacher, Michael I-231  
 Miyazaki, Shuichi II-135  
 Mnich, Matthias I-267, I-326, II-112  
 Morgenstern, Gila I-475  
 Morozov, Dmitriy I-1  
 Mount, David M. I-374, I-386  
 Mozes, Shay II-206  
 Mucha, Marcin I-72  
 Müller, Tobias II-112
- Nagarajan, Viswanath II-218  
 Navarro, Gonzalo II-194  
 Neiman, Ofer II-87  
 Niemeier, Martin I-11  
 Nordsieck, Arnold W. I-11
- Okamoto, Yoshio I-500  
 Osipov, Vitaly I-278
- Pagh, Rasmus I-427  
 Patel, Amit I-1  
 Patel, Viresh I-561  
 Paulusma, Daniël I-122  
 Peleg, David I-84  
 Phillips, Jeff M. I-487  
 Pilipczuk, Marcin I-72  
 Pirwani, Imran A. I-243  
 Pountourakis, Emmanouil I-146  
 Puglisi, Simon J. I-451, II-194
- Radhakrishnan, Jaikumar II-159  
 Rao, S. Srinivasa II-171  
 Rawitz, Dror I-255  
 Raychev, Veselin I-290  
 Roditty, Liam I-84  
 Roughgarden, Tim II-17  
 Rudra, Atri II-218
- Sanders, Peter I-278  
 Sankowski, Piotr I-72  
 Sauerwald, Thomas I-134  
 Sau, Ignasi I-97  
 Schmidt, Melanie I-60  
 Schulman, Leonard J. II-87  
 Schulz, André I-110, II-63  
 Setter, Ophir I-398  
 Shah, Smit II-159  
 Shannigrahi, Saswata II-159  
 Sharir, Micha I-475  
 Silveira, Rodrigo I. I-463  
 Sitchinava, Nodari II-75  
 Skopalik, Alexander II-29  
 Skutella, Martin I-11, I-36  
 Sohler, Christian I-60  
 Solomon, Shay I-48  
 Stein, Cliff I-182  
 Stiller, Sebastian I-338  
 Sung, Wing-Kin I-573

- Thaler, Justin I-231  
Thilikos, Dimitrios M. I-97, I-122  
Turpin, Andrew II-194  
Vaccaro, Ugo I-207  
Ventre, Carmine II-39  
Verschae, José I-11, I-36  
Vidali, Angelina I-146  
Viger, Fabien I-290  
Vigna, Sebastiano I-427  
Wenk, Carola I-463  
Wiese, Andreas I-11  
Wiratma, Lionov I-463  
Woeginger, Gerhard J. I-195  
Wolf, Jan I-512  
Wulff-Nilsen, Christian II-206  
Xu, Haifeng I-195  
Yanagisawa, Hiroki II-135  
Yeo, Anders I-326  
Yu, Hai I-487  
Zeh, Norbert II-75  
Zenklusen, Rico I-536