

Takao Asano
Shin-ichi Nakano
Yoshio Okamoto
Osamu Watanabe (Eds.)

Algorithms and Computation

22nd International Symposium, ISAAC 2011
Yokohama, Japan, December 2011
Proceedings



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 Lecture 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*

Takao Asano Shin-ichi Nakano
Yoshio Okamoto Osamu Watanabe (Eds.)

Algorithms and Computation

22nd International Symposium, ISAAC 2011
Yokohama, Japan, December 5-8, 2011
Proceedings

Volume Editors

Takao Asano
Chuo University
Tokyo, 112-8551, Japan
E-mail: asano@ise.chuo-u.ac.jp

Shin-ichi Nakano
Gunma University
Kiryu-Shi, 376-8515, Japan
E-mail: nakano@cs.gunma-u.ac.jp

Yoshio Okamoto
Japan Advanced Institute of Science and Technology
Ishikawa, 923-1292, Japan
E-mail: okamotoy@jaist.ac.jp

Osamu Watanabe
Tokyo Institute of Technology
Tokyo, 152-8552, Japan
E-mail: watanabe@is.titech.ac.jp

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-25590-8 e-ISBN 978-3-642-25591-5
DOI 10.1007/978-3-642-25591-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011941283

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

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

© Springer-Verlag Berlin Heidelberg 2011

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.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typeetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The papers in this volume were presented at the 22nd International Symposium on Algorithms and Computation (ISAAC 2011), held in Yokohama, Japan, during December 5-8, 2011. In the past, ISAAC was held in Tokyo (1990), Taipei (1991), Nagoya (1992), Hong Kong (1993), Beijing (1994), Cairns (1995), Osaka (1996), Singapore (1997), Taejon (1998), Chennai (1999), Taipei (2000), Christchurch (2001), Vancouver (2002), Kyoto (2003), Hong Kong (2004), Hainan (2005), Kolkata (2006), Sendai (2007), Gold Coast (2008), Hawaii (2009), and Jeju (2010).

ISAAC is an annual international symposium that covers the very wide range of topics in algorithms and computation. The main purpose of the symposium is to provide a forum for researchers working in algorithms and the theory of computation where they can exchange ideas in this active research community. In response to the call for papers, ISAAC 2011 received 187 submissions from 42 countries. Each submission was reviewed by at least three Program Committee members with the assistance of external referees. Since there were many high-quality papers, the Program Committee’s task was extremely difficult. Through an extensive discussion, the Program Committee selected 76 papers. Two special issues, one of *Algorithmica* and one of the *International Journal of Computational Geometry and Applications*, were prepared with selected papers from ISAAC 2011.

The best paper award was given to “Linear-Time Algorithms for Hole-Free Rectilinear Proportional Contact Graph Representations” by Muhammad Jawaherul Alam, Therese Biedl, Stefan Felsner, Andreas Gerasch, Michael Kaufmann and Stephen G. Kobourov. The best student paper award was given to “Fixed-Parameter Complexity of Feedback Vertex Set in Bipartite Tournaments” by Sheng-Ying Hsiao. Two eminent invited speakers, Sanjeev Arora from Princeton University, USA, and Dorothea Wagner from Karlsruhe Institute of Technology, Germany, also contributed to this volume.

We would like to thank all Program Committee members and external referees for their excellent work in the reviewing process. We would like to thank all authors who submitted papers for consideration; they all contributed to the high quality of the symposium. We would also like to thank the Organizing Committee members for their dedicated contribution that made the symposium possible and enjoyable. Finally, we would like to thank our sponsors and supporting organizations for their assistance and support.

December 2011

Takao Asano
Shin-ichi Nakano
Yoshio Okamoto
Osamu Watanabe

Organization

Symposium Chair

Osamu Watanabe

Tokyo Institute of Technology, Japan

Program Committee

Kazuyuki Amano	Gunma University, Japan
Takao Asano	Chuo University, Japan, Co-chair
Timothy M. Chan	University of Waterloo, Canada
Kun-Mao Chao	National Taiwan University, Taiwan
Marek Chrobak	University of California, Riverside, USA
Kyung-Yong Chwa	KAIST, Korea
Richard Cole	New York University, USA
Xiaotie Deng	University of Liverpool, UK
Lisa K. Fleischer	Dartmouth, USA
Magnús M. Halldórsson	Reykjavik University, Iceland
Wen-Lian Hsu	Academia Sinica, Taiwan
Hiroshi Imai	University of Tokyo, Japan
Tibor Jordán	Eötvös Loránd University, Hungary
Ming-Yang Kao	Northwestern University, USA
J. Mark Keil	University of Saskatchewan, Canada
Giuseppe Liotta	University of Perugia, Italy
Julian Mestre	University of Sydney, Australia
Shin-ichi Nakano	Gunma University, Japan, Co-chair
Mitsunori Ogihara	University of Miami, USA
Yoshio Okamoto	JAIST, Japan
Jung-Heum Park	Catholic University of Korea, Korea
Kunsoo Park	Seoul National University, Korea
Md. Saidur Rahman	Bangladesh University of Engineering and Technology, Bangladesh
Igor Shparlinski	Macquarie University, Australia
Robert E. Tarjan	Princeton University and HP Labs, USA
Takeaki Uno	National Institute of Informatics, Japan
Jens Vygen	University of Bonn, Germany
Dorothea Wagner	Karlsruhe Institute of Technology, Germany
Alexander Wolff	University of Würzburg, Germany
Ke Yi	Hong Kong University of Science and Technology, Hong Kong
Lisa Zhang	Bell Laboratories, USA
Xiao Zhou	Tohoku University, Japan

Organizing Committee

Takashi Horiyama	Saitama University, Japan
Akinori Kawachi	Tokyo Institute of Technology, Japan
Ken-ichi Kawarabayashi	NII, Japan
Yoshio Okamoto	JAIST, Japan, Chair
Keisuke Tanaka	Tokyo Institute of Technology, Japan

Sponsors

ISAAC 2011 was supported by:

- “Global COE: Computationism as a Foundation for the Sciences” of Tokyo Institute of Technology
- Inoue Foundation for Science
- Kayamori Foundation of Informational Science Advancement
- Support Center for Advanced Telecommunications Technology Research, Foundation
- The Telecommunications Advancement Foundation

It was held in cooperation with:

- IEICE Information and Systems Society, Technical Committee on Theoretical Foundations of Computing (COMP)
- Special Interest Group on Algorithms (SIGAL) of IPSJ

Industrial Sponsors of ISAAC 2011 were:

- IBM Japan, Ltd.
- Nippon Telegraph and Telephone Corporation (NTT)

External Reviewers

Achlioptas, Dimitris	Bhaskar, Umang	Chun, Jinhee
Akutsu, Tatsuya	Bhattacharya, Binay	Cormode, Graham
Alt, Helmut	Binucci, Carla	Coudert, David
Araki, Toru	Buchin, Kevin	Cygan, Marek
Aronov, Boris	Buchin, Maike	Demaine, Erik D.
Arya, Sunil	Butler, Steve	Di Giacomo, Emilio
Ásgeirsson, Eyjólfur Ingi	Caragiannis, Ioannis	Dibbelt, Julian
Averbakh, Igor	Castelli Aleardi, Luca	Doerr, Benjamin
Azaron, Amir	Chang, Chia-Jung	Dumitrescu, Adrian
Bae, Sang Won	Chen, Kuan-Yu	Durocher, Stephane
Bampis, Evripidis	Cheng, Siu-Wing	Etesami, Omid
Bauer, Reinhard	Choi, Joonsoo	Fan, Jianxi
Bérczi, Kristóf	Christ, Tobias	Fekete, Sándor
Berenbrink, Petra	Chu, An-Chiang	Fink, Martin

Fleiner, Tamás	Kiyomi, Masashi	Nöllenburg, Martin
Fujisaki, Eiichiro	Knauer, Christian	Onak, Krzysztof
Fujita, Ken-Etsu	Ko, Ming-Tat	Ono, Hirotaka
Fujita, Satoshi	Kortsarz, Guy	Pajor, Thomas
Fukunaga, Takuro	Kovács, Erika	Paku, Daichi
Gemsa, Andreas	Kratsch, Stefan	Pap, Júlia
Ghosh, Subir	Krug, Marcus	Parhami, Behrooz
Görke, Robert	Krumke, Sven	Pisanti, Nadia
Goldreich, Oded	Kusakari, Yoshiyuki	Rahman, M. Sohel
Gouveia, Luis	Kwon, Oh-Heum	Raichel, Benjamin
Grigoriev, Alexander	Lall, Ashwin	Raman, Venkatesh
Grilli, Luca	Lerner, Jürgen	Rutter, Ignaz
Gu, Qianping	Leung, Henry C.M.	Sadakane, Kunihiko
Gupta, Prosenjit	Levin, Asaf	Saitoh, Toshiki
Gutin, Gregory	Liao, Chung-Shou	Sakai, Yoshifumi
Halldórsson, Bjarni V.	Lim, Hyeong-Seok	Sauerwald, Thomas
Halman, Nir	Lin, Rung-Ren	Schlotter, Ildikó
Hartmann, Tanja	Lin, Wei-Yin	Schumm, Andrea
Hassin, Refael	Liu, Tian	Segev, Danny
Haunert, Jan-Henrik	Lu, Chi-Jen	Shin, Chan-Su
Hirata, Tomio	Lu, Pinyan	Silveira, Rodrigo
Hsieh, Sun-Yuan	Lubiw, Anna	Smorodinsky, Shakhar
Hsu, Tsan-Sheng	Lübbecke, Marco	Spoerhase, Joachim
Hüffner, Falk	Lyu, Yu-Han	Srivastav, Anand
Imai, Keiko	Maßberg, Jens	Sun, Xiaoming
Imamichi, Takashi	Matsui, Tetsushi	Suzuki, Koutarou
Inenaga, Shunsuke	Matsuura, Akihiro	Suzuki, Taiji
Irani, Sandy	McGregor, Andrew	Tabei, Yasuo
Isobe, Shuji	Meister, Daniel	Takahashi, Toshihiko
Ito, Takehiro	Melsted, Pál	Tamaki, Suguru
Iwamoto, Chuzo	Miklós Zoltán	Tanigawa, Shin-Ichi
Jin, Jiongxin	Mitra, Pradipta	Tao, Yufei
Jung, Hyunwoo	Miyamoto, Yuichiro	Teranishi, Isamu
Kamiyama, Naoyuki	Miyashiro, Ryuhei	Thomborson, Clark
Kaneta, Yusaku	Miyazaki, Shuichi	Tokuyama, Takeshi
Karim, Md. Rezaul	Mizuki, Takaaki	Tóth, Csaba
Karloff, Howard	Mnich, Matthias	Uchizawa, Kei
Katayama, Kengo	Mohar, Bojan	Ueno, Kenya
Kida, Takuya	Mondal, Debajyoti	Ueno, Shuichi
Kikuchi, Yosuke	Montecchiani, Fabrizio	Vassilvitskii, Sergei
Kim, Jae-Hoon	Morihata, Akimasa	Vigneron, Antoine
Kim, Soo-Hwan	Morita, Kenichi	Villanger, Yngve
Kim, Sook-Yeon	Nakamura, Akira	Wada, Koichi
Kim, Sung Kwon	Nandy, Subhas	Wang, Haitao
Kirkpatrick, David	Narisawa, Kazuyuki	Wang, Hung-Lung

X Organization

Weibel, Christophe	Yamamoto, Go	Yuan, Hao
Widmayer, Peter	Yamamoto, Hiroaki	Zhang, Shengyu
Wilkinson, Bryan	Yamanaka, Katsuhisa	Zhu, Binhai
Wismath, Stephen	Yamashita, Masafumi	Zhu, Yongding
Xu, Jinhui	Yamazaki, Koichi	
Yamakami, Tomoyuki	Yang, Wu Lung R.	

Table of Contents

Invited Talk I

Algorithm Engineering for Route Planning: An Update	1
<i>Dorothea Wagner</i>	

Invited Talk II

Semidefinite Programming and Approximation Algorithms: A Survey ...	6
<i>Sanjeev Arora</i>	

Approximation Algorithms I

The School Bus Problem on Trees	10
<i>Adrian Bock, Elyot Grant, Jochen Könemann, and Laura Sanità</i>	

Improved Approximations for Buy-at-Bulk and Shallow-Light k -Steiner Trees and $(k, 2)$ -Subgraph	20
<i>M. Reza Khani and Mohammad R. Salavatipour</i>	

Improved Approximation Algorithms for Routing Shop Scheduling	30
<i>Wei Yu and Guochuan Zhang</i>	

Contraction-Based Steiner Tree Approximations in Practice	40
<i>Markus Chimani and Matthias Woste</i>	

Computational Geometry I

Covering and Piercing Disks with Two Centers	50
<i>Hee-Kap Ahn, Sang-Sub Kim, Christian Knauer, Lena Schlipf, Chan-Su Shin, and Antoine Vigneron</i>	

Generating Realistic Roofs over a Rectilinear Polygon	60
<i>Hee-Kap Ahn, Sang Won Bae, Christian Knauer, Mira Lee, Chan-Su Shin, and Antoine Vigneron</i>	

Computing the Visibility Polygon Using Few Variables.....	70
<i>Luis Barba, Matias Korman, Stefan Langerman, and Rodrigo I. Silveira</i>	

Minimizing Interference in Ad-Hoc Networks with Bounded Communication Radius	80
<i>Matias Korman</i>	

Graph Algorithms

Hamiltonian Paths in the Square of a Tree	90
<i>Jakub Radoszewski and Wojciech Rytter</i>	
Dominating Induced Matchings for P_7 -free Graphs in Linear Time	100
<i>Andreas Brandstädt and Raffaele Mosca</i>	
Finding Contractions and Induced Minors in Chordal Graphs via Disjoint Paths	110
<i>Rémy Belmonte, Petr A. Golovach, Pinar Heggernes, Pim van 't Hof, Marcin Kamiński, and Daniël Paulusma</i>	
Recognizing Polar Planar Graphs Using New Results for Monopolarity	120
<i>Van Bang Le and Ragnar Nevries</i>	
Robustness of Minimum Cost Arborescences	130
<i>Naoyuki Kamiyama</i>	

Data Structures I

Path Queries in Weighted Trees	140
<i>Meng He, J. Ian Munro, and Gelin Zhou</i>	
Dynamic Range Majority Data Structures	150
<i>Amr Elmasry, Meng He, J. Ian Munro, and Patrick K. Nicholson</i>	
Dynamic Range Selection in Linear Space	160
<i>Meng He, J. Ian Munro, and Patrick K. Nicholson</i>	
A Dynamic Stabbing-Max Data Structure with Sub-Logarithmic Query Time	170
<i>Yakov Nekrich</i>	
Encoding 2D Range Maximum Queries	180
<i>Mordecai Golin, John Iacono, Danny Krizanc, Rajeev Raman, and S. Srinivasa Rao</i>	

Distributed Systems

Diameter and Broadcast Time of Random Geometric Graphs in Arbitrary Dimensions	190
<i>Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer</i>	
Broadcasting in Heterogeneous Tree Networks with Uncertainty	200
<i>Cheng-Hsiao Tsou, Gen-Huey Chen, and Ching-Chi Lin</i>	

Optimal File Distribution in Peer-to-Peer Networks	210
<i>Kai-Simon Goetzmann, Tobias Harks, Max Klimm, and Konstantin Miller</i>	

Computational Geometry II

Animal Testing	220
<i>Adrian Dumitrescu and Evan Hilscher</i>	
Cutting Out Polygons with a Circular Saw	230
<i>Adrian Dumitrescu and Masud Hasan</i>	
Fast Fréchet Queries	240
<i>Mark de Berg, Atlas F. Cook IV, and Joachim Gudmundsson</i>	

Graph Drawing and Information Visualization

Angle-Restricted Steiner Arborescences for Flow Map Layout	250
<i>Kevin Buchin, Bettina Speckmann, and Kevin Verbeek</i>	
Treemaps with Bounded Aspect Ratio	260
<i>Mark de Berg, Bettina Speckmann, and Vincent van der Weele</i>	
Simultaneous Embedding of Embedded Planar Graphs	271
<i>Patrizio Angelini, Giuseppe Di Battista, and Fabrizio Frati</i>	
Linear-Time Algorithms for Hole-Free Rectilinear Proportional Contact Graph Representations	281
<i>Muhammad Jawaherul Alam, Therese Biedl, Stefan Felsner, Andreas Gerasch, Michael Kaufmann, and Stephen G. Kobourov</i>	

Data Structures II

Fully Retroactive Approximate Range and Nearest Neighbor Searching	292
<i>Michael T. Goodrich and Joseph A. Simons</i>	
Compact Representation of Posets	302
<i>Arash Farzan and Johannes Fischer</i>	
Explicit Array-Based Compact Data Structures for Triangulations	312
<i>Luca Castelli Aleardi and Olivier Devillers</i>	
Space-Efficient Data-Analysis Queries on Grids	323
<i>Gonzalo Navarro and Luís M.S. Russo</i>	

Parameterized Algorithms I

A Polynomial Kernel for FEEDBACK ARC SET on Bipartite Tournaments	333
<i>Pranabendu Misra, Venkatesh Raman, M.S. Ramanujan, and Saket Saurabh</i>	

Fixed-Parameter Complexity of Feedback Vertex Set in Bipartite Tournaments	344
<i>Sheng-Ying Hsiao</i>	

Parameterized Algorithms for Inclusion of Linear Matchings	354
<i>Sylvain Guillemot</i>	

Computational Study on Bidimensionality Theory Based Algorithm for Longest Path Problem	364
<i>Chunhao Wang and Qian-Ping Gu</i>	

Parallel and External Memory Algorithms

Sorting, Searching, and Simulation in the MapReduce Framework	374
<i>Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang</i>	

External-Memory Multimaps	384
<i>Elaine Angelino, Michael T. Goodrich, Michael Mitzenmacher, and Justin Thaler</i>	

External Memory Orthogonal Range Reporting with Fast Updates	395
<i>Yakov Nekrich</i>	

Analysis of Speedups in Parallel Evolutionary Algorithms for Combinatorial Optimization (Extended Abstract)	405
<i>Jörg Lässig and Dirk Sudholt</i>	

Game Theory and Internet Algorithms

Verifying Nash Equilibria in PageRank Games on Undirected Web Graphs	415
<i>David Avis, Kazuo Iwama, and Daichi Paku</i>	

Improved Collaborative Filtering	425
<i>Aviv Nisgav and Boaz Patt-Shamir</i>	

Asymptotic Modularity of Some Graph Classes	435
<i>Fabien de Montgolfier, Mauricio Soto, and Laurent Viennot</i>	

Computational Complexity

Program Size and Temperature in Self-assembly	445
<i>Ho-Lin Chen, David Doty, and Shinnosuke Seki</i>	
Optimization, Randomized Approximability, and Boolean Constraint Satisfaction Problems	454
<i>Tomoyuki Yamakami</i>	
Lower Bounds for Myopic DPLL Algorithms with a Cut Heuristic	464
<i>Dmitry Itsykson and Dmitry Sokolov</i>	

Approximation Algorithms II

Algorithm for Single Allocation Problem on Hub-and-Spoke Networks in 2-Dimensional Plane	474
<i>Ryuta Ando and Tomomi Matsui</i>	
Packing-Based Approximation Algorithm for the k -Set Cover Problem	484
<i>Martin Fürer and Huiwen Yu</i>	
Capacitated Domination: Constant Factor Approximations for Planar Graphs	494
<i>Mong-Jen Kao and D.T. Lee</i>	

Randomized Algorithms

On Power-Law Distributed Balls in Bins and Its Applications to View Size Estimation	504
<i>Ioannis Atsonios, Olivier Beaumont, Nicolas Hanusse, and Yusik Kim</i>	
A Randomized Algorithm for Finding Frequent Elements in Streams Using $O(\log \log N)$ Space	514
<i>Masatora Ogata, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita</i>	

A Nearly-Quadratic Gap between Adaptive and Non-adaptive Property Testers (Extended Abstract)	524
<i>Jeremy Hurwitz</i>	

Online and Streaming Algorithms

Online Linear Optimization over Permutations	534
<i>Shota Yasutake, Kohei Hatano, Shuji Kijima, Eiji Takimoto, and Masayuki Takeda</i>	

On the Best Possible Competitive Ratio for Multislope Ski Rental	544
<i>Hiroshi Fujiwara, Takuma Kitano, and Toshihiro Fujito</i>	

Input-Thrifty Extrema Testing	554
<i>Kuan-Chieh Robert Tseng and David Kirkpatrick</i>	

Edit Distance to Monotonicity in Sliding Windows	564
<i>Ho-Leung Chan, Tak-Wah Lam, Lap-Kei Lee, Jiangwei Pan, Hing-Fung Ting, and Qin Zhang</i>	

Computational Geometry III

Folding Equilateral Plane Graphs	574
<i>Zachary Abel, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Jayson Lynch, Tao B. Schardl, and Isaac Shapiro-Ellowitz</i>	

Efficient Algorithms for the Weighted k -Center Problem on a Real Line	584
<i>Danny Z. Chen and Haitao Wang</i>	

Outlier Respecting Points Approximation	594
<i>Danny Z. Chen and Haitao Wang</i>	

An Improved Algorithm for Reconstructing a Simple Polygon from the Visibility Angles	604
<i>Danny Z. Chen and Haitao Wang</i>	

Parameterized Algorithms II

The Parameterized Complexity of Local Search for TSP, More Refined	614
<i>Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondřej Suchý</i>	

On the Parameterized Complexity of Consensus Clustering	624
<i>Martin Dörfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller</i>	

Two Fixed-Parameter Algorithms for the Cocompling Problem	634
<i>Victor Campos, Sulamita Klein, Rudini Sampaio, and Ana Silva</i>	

Parameterized Complexity of the Firefighter Problem	643
<i>Cristina Bazgan, Morgan Chopin, and Michael R. Fellows</i>	

String Algorithms

Faster Approximate Pattern Matching in Compressed Repetitive Texts	653
<i>Travis Gagie, Paweł Gawrychowski, and Simon J. Puglisi</i>	
A New Algorithm for the Characteristic String Problem under Loose Similarity Criteria	663
<i>Yoshifumi Sakai</i>	
Succinct Indexes for Circular Patterns	673
<i>Wing-Kai Hon, Chen-Hua Lu, Rahul Shah, and Sharma V. Thankachan</i>	
Range LCP	683
<i>Amihood Amir, Alberto Apostolico, Gad M. Landau, Avivit Levy, Moshe Lewenstein, and Ely Porat</i>	

Optimization

Computing Knapsack Solutions with Cardinality Robustness	693
<i>Naonori Kakimura, Kazuhisa Makino, and Kento Seimi</i>	
Max-Throughput for (Conservative) k -of- n Testing	703
<i>Lisa Hellerstein, Özgür Özkan, and Linda Sellie</i>	
Closest Periodic Vectors in L_p Spaces	714
<i>Amihood Amir, Estrella Eisenberg, Avivit Levy, and Noa Lewenstein</i>	
Maximum Weight Digital Regions Decomposable into Digital Star-Shaped Regions	724
<i>Matt Gibson, Dongfeng Han, Milan Sonka, and Xiaodong Wu</i>	

Computational Biology

Finding Maximum Sum Segments in Sequences with Uncertainty	734
<i>Hung-I Yu, Tien-Ching Lin, and D.T. Lee</i>	
Algorithms for Building Consensus MUL-trees	744
<i>Yun Cui, Jesper Jansson, and Wing-Kin Sung</i>	
Adaptive Phenotype Testing for AND/OR Items	754
<i>Francis Y.L. Chin, Henry C.M. Leung, and S.M. Yiu</i>	
An Index Structure for Spaced Seed Search	764
<i>Taku Onodera and Tetsuo Shibuya</i>	
Author Index	773

Algorithm Engineering for Route Planning

– An Update –

Dorothea Wagner

Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Germany
dorothea.wagner@kit.edu

Abstract. Nowadays, route planning systems belong to the most frequently used information systems. The algorithmic core problem of such systems, i.e., the fast computation of shortest paths is a classical problem that can be solved by Dijkstra's shortest paths algorithm. However, for the huge datasets that frequently appear in route planning the algorithm is far too slow. Recently, algorithms for route planning in transportation networks have undergone a rapid development, leading to methods that are more than a million times faster than Dijkstra's algorithm. In particular, computing shortest paths in huge networks has become a showpiece of algorithm engineering demonstrating the engineering cycle that consists of design, analysis, implementation and experimental evaluation of practicable algorithms. In this talk, we provide a condensed overview of the techniques enabling this development. In particular, new theoretical insights on when and why those techniques work so well will be discussed. The main part of the talk will focus on variants of the problem that occur in more realistic traffic scenarios.

Computing the shortest path (or quickest path with respect to travel times) in graphs modelling transportation networks is one of the showpieces of algorithm engineering. In general, Dijkstra's algorithm [23] finds a shortest path between a given source s and target t efficiently. But unfortunately, the algorithm is far too slow to be used on huge datasets which appear frequently in route planning. Thus, several speed-up techniques have been developed that compute additional data during a preprocessing step in order to speed-up queries during the online phase.

Classical speed-up techniques are *bidirectional search* and *goal-directed search* (also called A^*) [31]. The last twelve years, more sophisticated techniques were presented, starting from *geometric containers* [47][48][50] and *multilevel search* [48][49] to reach [30], *landmarks* [28], *arc-flags* [37][36][38][32][33], *multilevel overlay graphs* [34], and *highway hierarchies* [45][46] and *contraction hierarchies* [26][27], respectively combinations of those [35][10][29][11]. With the availability of huge road networks, the whole topic was undergoing a rapid development resulting in techniques that answer a shortest paths query in a static road network a million times faster than Dijkstra's algorithm [6][5][2]. For an overview see also [20].

On basis of those techniques, more advanced scenarios have been considered, like dynamic networks [21], time-dependent travel times [7][13][14][42], multiple

criteria [11][22][25], or arbitrary metrics [15]. While only some of the early papers [47][48][49] considered route planning in public transportation networks, the other speed-up techniques mentioned so far focused on road networks. Speed-ups obtained for time table information in public transportation networks are still comparably small [43][39][44][12][24]; see also [40] for an overview. Only recently, also public transportation networks [19][4][17] and multi-modal transportation networks [18] were reconsidered. However, there is still a lot of work to be done for such scenarios.

Another line of research considers theoretical aspects of route planning. While research on speed-up techniques so far mostly asked for techniques that guarantee to compute *shortest* paths, the speed-ups obtained are only ensured empirically, i.e. on basis of extensive experiments. So on one hand, theoretical insights on why and when such techniques work so well are of interest [31]. See also [16] for a nice survey illustrating the interplay between practical and theoretical results on route planning. On the other hand, theoretical results on the algorithmic complexity of optimally preprocessing speed-up techniques [9][8] add to the understanding of the field.

In this talk, we provide a condensed overview of the development of the last twelve years in the field. The main part of the talk will focus on recent results, in particular on new theoretical insights and variants of the problem that occur in more realistic traffic scenarios.

References

1. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: VC-Dimension and Shortest Path Algorithms. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 690–699. Springer, Heidelberg (2011)
2. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 230–241. Springer, Heidelberg (2011)
3. Abraham, I., Fiat, A., Goldberg, A.V., Werneck, R.F.: Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In: Charikar, M. (ed.) Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 782–793. SIAM (2010)
4. Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V., Viger, F.: Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 290–301. Springer, Heidelberg (2010)
5. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Shortest-Path Queries in Road Networks. In: Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX 2007), pp. 46–59. SIAM (2007)
6. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast Routing in Road Networks with Transit Nodes. Science 316(5824), 566 (2007)
7. Batz, G.V., Geisberger, R., Neubauer, S., Sanders, P.: Time-Dependent Contraction Hierarchies and Approximation. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 166–177. Springer, Heidelberg (2010)

8. Bauer, R., Columbus, T., Katz, B., Krug, M., Wagner, D.: Preprocessing Speed-Up Techniques Is Hard. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 359–370. Springer, Heidelberg (2010)
9. Bauer, R., D’Angelo, G., Delling, D., Wagner, D.: The Shortcut Problem – Complexity and Approximation. In: Nielsen, M., Kučera, A., Miltersen, P.B., Palamidessi, C., Tůma, P., Valencia, F. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 105–116. Springer, Heidelberg (2009)
10. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. ACM Journal of Experimental Algorithms 14(2.4), 1–29 (2009); Special Section on Selected Papers from ALENEX 2008
11. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra’s Algorithm. ACM Journal of Experimental Algorithms 15(2.3), 1–31 (2010); Special Section devoted to WEA 2008
12. Bauer, R., Delling, D., Wagner, D.: Experimental Study on Speed-Up Techniques for Timetable Information Systems. In: Liebchen, C., Ahuja, R.K., Mesa, J.A. (eds.) Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS 2007). Open Access Series in Informatics (OASIcs), pp. 209–225 (2007)
13. Delling, D.: Time-Dependent SHARC-Routing. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 332–343. Springer, Heidelberg (2008)
14. Delling, D.: Time-Dependent SHARC-Routing. Algorithmica 60(1), 60–94 (2011); Special Issue: European Symposium on Algorithms 2008
15. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Customizable Route Planning. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 376–387. Springer, Heidelberg (2011)
16. Delling, D., Goldberg, A.V., Werneck, R.F.: Shortest Paths in Road Networks: From Practice to Theory and Back. It—Information Technology (2012) (to appear)
17. Delling, D., Katz, B., Pajor, T.: Parallel Computation of Best Connections in Public Transportation Networks. In: 24th International Parallel and Distributed Processing Symposium (IPDPS 2010). IEEE Computer Society (2010)
18. Delling, D., Pajor, T., Wagner, D.: Accelerating Multi-Modal Route Planning by Access-Nodes. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 587–598. Springer, Heidelberg (2009)
19. Delling, D., Pajor, T., Wagner, D.: Engineering Time-Expanded Graphs for Faster Timetable Information. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 182–206. Springer, Heidelberg (2009)
20. 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)
21. Delling, D., Wagner, D.: Landmark-Based Routing in Dynamic Graphs. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 52–65. Springer, Heidelberg (2007)
22. Delling, D., Wagner, D.: Pareto Paths with SHARC. In: Vahrenhold, J. (ed.) SEA 2009. LNCS, vol. 5526, pp. 125–136. Springer, Heidelberg (2009)
23. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik 1, 269–271 (1959)
24. 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)

25. Geisberger, R., Kobitzsch, M., Sanders, P.: Route Planning with Flexible Objective Functions. In: Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX 2010), pp. 124–137. SIAM (2010)
26. 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)
27. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using contraction hierarchies. Transportation Science (2011) (accepted for publication)
28. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Better Landmarks Within Reach. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 38–51. Springer, Heidelberg (2007)
29. Goldberg, A.V., Kaplan, H., Werneck, R.F.: Reach for A*: Shortest Path Algorithms with Preprocessing. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) The Shortest Path Problem: Ninth DIMACS Implementation Challenge. DIMACS Book, vol. 74, pp. 93–139. American Mathematical Society (2009)
30. Gutman, R.J.: Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), pp. 100–111. SIAM (2004)
31. Hart, P.E., Nilsson, N., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107 (1968)
32. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) 9th DIMACS Implementation Challenge - Shortest Paths (November 2006)
33. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In: Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.) The Shortest Path Problem: Ninth DIMACS Implementation Challenge. DIMACS Book, vol. 74, pp. 41–72. American Mathematical Society (2009)
34. Holzer, M., Schulz, F., Wagner, D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries. ACM Journal of Experimental Algorithms 13(2.5), 1–26 (2008)
35. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining Speed-up Techniques for Shortest-Path Computations. ACM Journal of Experimental Algorithms 10(2.5), 1–18 (2006)
36. Köhler, E., Möhring, R.H., Schilling, H.: Acceleration of Shortest Path and Constrained Shortest Path Computation. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 126–138. Springer, Heidelberg (2005)
37. Lauther, U.: An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In: Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, vol. 22, pp. 219–230. IfGI prints (2004)
38. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speedup Dijkstra's Algorithm. ACM Journal of Experimental Algorithms 11(2.8), 1–29 (2006)
39. Müller-Hannemann, M., Schnee, M.: Finding All Attractive Train Connections by Multi-criteria Pareto Search. In: Geraets, F., Kroon, L.G., Schoebel, A., Wagner, D., Zaroliagis, C.D. (eds.) Railway Optimization 2004. LNCS, vol. 4359, pp. 246–263. Springer, Heidelberg (2007)

40. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.D.: 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)
41. Müller-Hannemann, M., Weihe, K.: Pareto Shortest Paths is Often Feasible in Practice. In: Brodal, G.S., Frigioni, D., Marchetti-Spaccamela, A. (eds.) *WAE 2001*. LNCS, vol. 2141, pp. 185–197. Springer, Heidelberg (2001)
42. Nannicini, G., Delling, D., Liberti, L., Schultes, D.: Bidirectional A* Search on Time-Dependent Road Networks. *Networks* (2011); Journal version of *WEA 2008*
43. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Experimental Comparison of Shortest Path Approaches for Timetable Information. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX 2004), pp. 88–99. SIAM (2004)
44. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithms* 12(2.4), 1–39 (2007)
45. Sanders, P., Schultes, D.: Highway Hierarchies Hasten Exact Shortest Path Queries. In: Brodal, G.S., Leonardi, S. (eds.) *ESA 2005*. LNCS, vol. 3669, pp. 568–579. Springer, Heidelberg (2005)
46. Sanders, P., Schultes, D.: Engineering Highway Hierarchies. In: Azar, Y., Erlebach, T. (eds.) *ESA 2006*. LNCS, vol. 4168, pp. 804–816. Springer, Heidelberg (2006)
47. Schulz, F., Wagner, D., Weihe, K.: Dijkstra’s Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. In: Vitter, J.S., Zaroliagis, C.D. (eds.) *WAE 1999*. LNCS, vol. 1668, pp. 110–123. Springer, Heidelberg (1999)
48. Schulz, F., Wagner, D., Weihe, K.: Dijkstra’s Algorithm On-Line: An Empirical Case Study from Public Railroad Transport. *ACM Journal of Experimental Algorithms* 5(12), 1–23 (2000)
49. Schulz, F., Wagner, D., Zaroliagis, C.: Using Multi-Level Graphs for Timetable Information in Railway Systems. In: Mount, D.M., Stein, C. (eds.) *ALENEX 2002*. LNCS, vol. 2409, pp. 43–59. Springer, Heidelberg (2002)
50. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric Containers for Efficient Shortest-Path Computation. *ACM Journal of Experimental Algorithms* 10(1.3), 1–30 (2005)

Semidefinite Programming and Approximation Algorithms: A Survey

Sanjeev Arora

Computer Science, Princeton University,
Princeton NJ 08544, USA
arora@princeton.edu

Abstract. Computing approximate solutions for NP-hard problems is an important research endeavor. Since the work of Goemans-Williamson in 1993, semidefinite programming (a form of convex programming in which the variables are vector inner products) has been used to design the current best approximation algorithms for problems such as MAX-CUT, MAX-3SAT, SPARSEST CUT, GRAPH COLORING, etc. The talk will survey this area, as well as its fascinating connections with topics such as geometric embeddings of metric spaces, and Khot's unique games conjecture.

The talk will be self-contained.

Computing approximate solutions to NP-hard problems is an important endeavor in theoretical computer science, and the past two decades have seen great progress in our understanding of approximability. Thanks to PCP Theorems, we now know that certain approximations are NP-hard, in other words as hard as exact optimization. For an introduction to PCP Theorems see the relevant chapter in [1]; for a survey of inapproximability results see [3][13][9]. For a survey of progress in approximation algorithms see [15][12].

Semidefinite programming (SDP) is a form of convex programming that has been used to design the best approximation algorithms for a host of problems. An SDP is a linear program in n^2 variables that are constrained to define a positive semidefinite (psd) matrix. Since a matrix A is psd iff there exist n vectors v_1, v_2, \dots, v_n such that $A_{ij} = \langle v_i, v_j \rangle$ we can also think of SDP as a form of “vector programming.”

SDP is useful because it allows some weak simulation of a nonlinear program, which is a natural way to represent NP-hard optimization problems. For example the nonlinear program for the MAX-CUT problem is

$$\begin{aligned} \max \quad & \sum_{\{ij\} \in E} x_i(1 - x_j) \\ \text{s.t. } & x_i(1 - x_i) = 0 \quad \forall i \end{aligned}$$

whereas the corresponding SDP relaxation is:

$$\begin{aligned} \max \quad & \sum_{\{ij\} \in E} \langle v_i, v_0 - v_j \rangle \\ \text{s.t.} \quad & \langle v_i, v_0 - v_i \rangle = 0 \quad \forall i \\ & \langle v_0, v_0 \rangle = 1 \end{aligned}$$

Though the usefulness of SDPs had been known since around 1980, the first successful use in designing an interesting approximation algorithm was in the seminal work of Goemans and Williamson [8] that gave a 1.13 approximation for MAX CUT using the above relaxation. This algorithm, which relies on a simple *hyperplane rounding*, became the template for a host of other algorithms in the subsequent years. See [7] for a survey. In the talk we refer to this set of algorithms as “Generation 1.” There were also some algorithms that apply simple modifications of the GW analysis, which we refer to as Generation 1.5.

Then progress was stalled for a while. Then in 2004 in joint work with Rao, and Vazirani the author [4] designed a $O(\sqrt{\log n})$ -approximation for SPARSEST CUT and related graph partitioning problems. This algorithm used the SDP relaxation with triangle inequality constraints, and gave a very global analysis that led to new insights into metrics of *negative type* and also almost-optimal embeddings of l_1 metrics into l_2 [6, 2]. This analysis also inspired several other papers that used the triangle inequality constraints to design algorithms for other problems. In the talk we refer to this spate of work as “Generation 2” algorithms.

In recent years attention has focused on the issue of whether or not the simple SDP-based algorithms designed thus far can be improved. For example, can the simple 1.13-approximation algorithm of GW be improved for MAX-CUT? Recent work from computational complexity suggests that the answer in this and many other cases may be “No” if Khot’s unique games conjecture is true. For a survey of this work see the survey [10]. The most striking aspect of this work is that the instances on which the SDP integrality gap is large (e.g., 1.13 in case of MAX-CUT) is used as a gadget in the reduction that shows the UGC-hardness of MAX-CUT.

The work on UGC hardness also led to a new generic rounding algorithm for all MAX-CSPs called *Squish-n-solve* [11]. This algorithm gives the best-possible approximation for all MAX-CSPs if the UGC is true. In the survey this algorithm is called “Generation 2.5.”

The work on UGC motivated the discovery of a new subexponential approximation algorithm for the unique games problem [14]. The new insights obtained here were transferred to SDP in recent work of Barak, Raghavendra, and Steurer [5] to give a new rounding algorithm for SDP relaxations based upon

Lasserre relaxations. The level- r Lasserre relaxations involve introducing a new vector variable for each subset of up to r original variables. The following is the relaxation for MAX-CUT.

$$\begin{aligned} \max \quad & \sum_{\{ij\} \in E} \langle v_i, v_0 - v_j \rangle \\ \langle v_i, v_0 - v_i \rangle = 0 \quad & \forall i \\ \langle v_0, v_0 \rangle = 1 \quad & \\ (\text{here } v_i \text{ is shorthand for } v_{\{i\}} \text{ and } v_0 \text{ for } v_\emptyset) \quad & \\ \langle v_P, v_Q \rangle = \langle v_S, v_T \rangle \quad & \forall P, Q, S, T, \text{ s.t.} \\ P \cup Q = S \cup T, \text{ and } |P \cup Q| \leq r \quad & \end{aligned}$$

Understanding the strength of Lasserre relaxations has been an open problem for quite some time and a few results known suggest they do not help. (See a forthcoming survey by Chlamtac and Tulsiani.) The BRS rounding algorithm leads to new $(1+\epsilon)$ approximation for several problems on graphs of low *threshold rank*. The C -threshold rank of a graph is the number of eigenvalues greater than C . This new rounding algorithm perhaps gives a glimmer of hope that the Generation 3 of SDP-based algorithms is now here.

References

1. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
2. Arora, S., Lee, J.R., Naor, A.: Euclidean distortion and the sparsest cut. *J. Amer. Math. Soc.* 21(1), 1–21 (2008)
3. Arora, S., Lund, C.: Hardness of approximations. In: Approximation Algorithms for NP-hard Problems. Course Technology (1996)
4. Arora, S., Rao, S., Vazirani, U.V.: Expander flows, geometric embeddings and graph partitioning. *J. ACM* 56(2) (2009); Prelim version ACM STOC 2004
5. Barak, B., Raghavendra, P., Steurer, D.: Rounding semidefinite programming hierarchies via global correlation. *CoRR*, abs/1104.4680 (2011); To appear in IEEE FOCS 2011
6. Chawla, S., Gupta, A., Räcke, H.: Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In: SODA 2005: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 102–111. Society for Industrial and Applied Mathematics, Philadelphia (2005)
7. Goemans, M.X.: Semidefinite programming and combinatorial optimization. In: Proceedings of the International Congress of Mathematicians, Berlin, vol. III, pp. 657–666 (1998)
8. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.* 42(6), 1115–1145 (1995)
9. Harsha, P., Charikar, M., Andrews, M., Arora, S., Khot, S., Moshkovitz, D., Zhang, L., Aazami, A., Desai, D., Gorodezky, I., Jagannathan, G., Kulikov, A.S., Mir, D.J., Newman, A., Nikolov, A., Pritchard, D., Spencer, G.: Limits of approximation algorithms: Pcp's and unique games (dimacs tutorial lecture notes). *CoRR*, abs/1002.3864 (2010)

10. Khot, S.: On the unique games conjecture (invited survey). In: Annual IEEE Conference on Computational Complexity, pp. 99–121 (2010)
11. Raghavendra, P., Steurer, D.: Integrality gaps for strong SDP relaxations of Unique Games. In: FOCS, pp. 575–585 (2009)
12. Shmoys, D., Williamson, D.: Design of Approximation Algorithms. Cambridge University Press (2011)
13. Trevisan, L.: Inapproximability of combinatorial optimization problems. CoRR, cs.CC/0409043 (2004)
14. Trevisan, L.: Inapproximability of combinatorial optimization problems. CoRR, cs.CC/0409043 (2004)
15. Vazirani, V.: Approximation Algorithms. Springer, Heidelberg (2001)

The School Bus Problem on Trees

Adrian Bock¹, Elyot Grant², Jochen Könemann², and Laura Sanità¹

¹ EPFL, Lausanne, Switzerland

² University of Waterloo, Canada

Abstract. The School Bus Problem is an NP-hard vehicle routing problem in which the goal is to route buses that transport children to a school such that for each child, the distance travelled on the bus does not exceed the shortest distance from the child’s home to the school by more than a given *regret* threshold. Subject to this constraint and bus capacity limit, the goal is to minimize the number of buses required.

In this paper, we give a polynomial time 4-approximation algorithm when the children and school are located at vertices of a fixed *tree*. As a byproduct of our analysis, we show that the integrality gap of the natural set-cover formulation for this problem is also bounded by 4. We also present a constant approximation for the variant where we have a fixed number of buses to use, and the goal is to minimize the maximum regret.

1 Introduction

Vehicle routing is an important and active topic in computer science and operations research. In the literature, the objective is typically to find a minimum-cost set of routes in a network that achieve a certain objective subject to a set of constraints. The constraints and cost are often related to the distance travelled, number of routes or vehicles used, coverage of the network by the routes, and so on. Problems of this kind are frequent and crucial in areas such as logistics, distribution systems, and public transportation (see, e.g., the survey by [12]).

In vehicle routing problems relevant to public transportation, a secondary objective often must be taken into account beyond minimizing operation cost: namely, it is crucial to design routes so as to optimize *customer satisfaction* in order to motivate customers to use the service. This requirement is essential in the so-called *School Bus Problem* (SBP)—the focus of this paper.

In the SBP, we must route buses that pick up children and bring them from their homes to a school. However, parents do not want their children to spend too much time on the bus relative to the time required to transport them to school by car along a shortest path. In fact, if the additional distance travelled by the bus exceeds a certain *regret* threshold, the parents would rather drive their children to school by themselves, which is unacceptable. Subject to this, the goal is to cover all of the children using a minimum number of buses.

Formally, we are given an undirected network $G(V, E)$ with distances on the edges $d : E \rightarrow \mathbb{Z}_+$, a node $s \in V$ representing the school, and a set $W \subseteq V$ representing the houses of children. Additionally, we are given a bus capacity

$C \in \mathbb{Z}_+$, and a regret bound $R \in \mathbb{Z}_+$. The aim is to construct a minimum cardinality set \mathcal{P} of walks ending at s (bus routes) and assign each child $w \in W$ to be the responsibility of some bus $p(w) \in \mathcal{P}$ such that (i) for each walk $P \in \mathcal{P}$, the total number of children w with $p(w) = P$ is at most the capacity C ; (ii) for every child the regret bound is respected, that is: $d^P(w, s) \leq d(w, s) + R$, where $d^P(w, s)$ is the distance from the child w to the school s on the walk $p(w)$, and $d(w, s)$ is the shortest distance from w to s in the graph G .

In an additional variation of the problem, we have a fixed number N of buses we can use, and the goal is to minimize the maximum regret R . We call this variant the *School Bus Problem with Regret minimization* (SBP-R).

Like many vehicle routing problems, both SBP and SBP-R are strongly NP-hard, even on the simplest of graphs. To see this, consider a star centered at the school s with children located at all other vertices. If k of the edges are very long, then determining if k buses are sufficient with a regret bound R is precisely equivalent to solving the bin-packing decision problem with k bins and objects sized according to the distances from the remaining children to the school. It follows that bin-packing can be reduced to both SBP and SBP-R on stars.

Many variants of vehicle routing have been studied in the context of exact, approximate, and heuristic algorithms. For SBP and SBP-R, heuristic methods for practical applications have been examined (see the survey of [10]), but there is no literature concerning formal approximability and inapproximability results. Our goal is to advance the state of the art in this perspective.

To begin, it is easy to see that the SBP can be formulated as a set covering problem. With this observation, one can easily derive a logarithmic approximation as well as a logarithmic upper bound on the integrality gap of the natural set-cover formulation applied to the SBP (see Section 2 for more details). The SBP is closely related (but not equal) to the Distance Constrained Vehicle Routing Problem (DVRP), which is well known and widely studied in terms of approximation. The DVRP can also be approximated within a logarithmic factor in general graphs, but there is a better 2-approximation on *trees*, as shown by [9]. They also show that the set-cover formulation for the DVRP on trees has integrality gap of at most 20. A natural question is then whether or not the SBP also admits a constant approximation/integrality gap on such graphs. Unfortunately, a straightforward adaptation of their methods to the SBP does not work. Therefore, in order to develop improved approximation results for the SBP, we need to introduce some new ideas.

1.1 Our Results

We first give a simple combinatorial 4-approximation for the SBP on *trees*.

Theorem 1. *There exists a polynomial time 4-approximation for the SBP on trees. The approximation factor reduces to 3 in the case of unlimited capacity.*

In contrast to the results given in [9] for DVRP, our algorithm for SBP immediately yields an integrality gap bound matching the approximation factor:

Theorem 2. *The integrality gap of the natural set-cover formulation of the SBP on trees is at most 4. In case of unlimited capacity, the gap is at most 3.*

On the negative side, we can prove an inapproximability factor of $\frac{5}{4}$ for the SBP on trees. Due to lack of space, we refer to the full version of this paper for a proof.

Finally, we give a combinatorial 12.5-approximation for the SBP-R on trees.

Theorem 3. *There exists a polynomial time 12.5-approximation algorithm for the SBP-R on trees in the case of unlimited capacity.*

1.2 Related Work

There are an enormous number of results concerning vehicle routing problems; see the survey [12]. The Capacitated Vehicle Routing Problem (CVRP) enforces a limit C on the number of visited locations in each route, and the goal is the minimization of the total length of all the routes. The paper [5] established a strong link to the underlying Travelling Salesman Problem (TSP) by giving an approximation algorithm that relies on the approximation algorithms for TSP. Depending on the capacity bound C , it is possible to obtain a PTAS in the euclidean plane for some special cases (if the capacity is either small [1], or very large [2]). If we restrict the input to trees, there is a 2-approximation [6]. Another constraint considered in literature is a bound D on the length of a vehicle tour, under the objective of minimizing the number of routes. This is the Distance Constrained Vehicle Routing Problem (DVRP). It was raised and studied for applications in [7] and [8]. Routing problems like the DVRP can be directly encoded as instances of Minimum Set Cover, and thus often admit logarithmic approximations. The authors of [9] give a careful analysis of the set cover integer programming formulation of the DVRP and bound its integrality gap by $\mathcal{O}(\log D)$ on general graphs and by $\mathcal{O}(1)$ on a tree. They also obtain a constant approximation for the DVRP on a tree and a $\mathcal{O}(\log D)$ approximation in general.

Many practical problems involving school buses have been studied, but primarily within the context of *heuristic methods* for real-life instances. We refer to [10] for a thorough survey of possible formulations and heuristic solution methods. Our notion of regret was first introduced as a vehicle routing objective in [11]. They considered a more general problem involving timing windows for customers and applied metaheuristics to produce solutions to real-life instances.

2 Preliminaries

We first observe that the capacity bound can be neglected for a slight loss in the approximation factor for the SBP. The proof is essentially identical to that of a similar result proved in [9] for the DVRP, and therefore we omit it.

Lemma 1. *Given an α -approximation to the SBP with unlimited capacity for each bus, there is an $\alpha + 1$ -approximation to the SBP with capacity bound C .*

If P is a walk ending at s and covering a subset S of nodes, then we say that P has regret R if a regret bound of R is respected for all children in S . The following useful fact holds for both the SBP and the SBP-R (due to lack of space we omit its easy proof):

Proposition 1. *Let P be a walk starting at some node v and ending at s . For all nodes covered by P the regret bound R is respected if and only if it is respected for v .*

We next give a covering integer programming formulation of the SBP. Let \mathcal{S} be the family of all feasible sets of C or fewer children that can be covered by a single walk ending at s having regret at most R . We introduce a variable x_S for each $S \in \mathcal{S}$ and give the following formulation:

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} x_S \\ \text{s.t.} \quad & \sum_{S: w \in S} x_S \geq 1 \quad \forall w \in W \\ & x_S \in \{0, 1\} \quad \forall S \in \mathcal{S}. \end{aligned} \quad (\text{IP})$$

An $\mathcal{O}(\log |W|)$ -approximation algorithm easily follows from adapting the greedy strategy for set cover. Such a greedy algorithm, applied to an SBP instance, repeatedly searches for a feasible walk ending at s that picks up the maximum number of uncovered children, doing so until every child is picked up. At each iteration, we guess the starting point v^* (by trying all $|V| - 1$ possibilities). Using Proposition 1, the resulting problem we are left with is to find a $v^* - s$ walk in G of length at most $d(v^*, s) + R$ visiting the maximum number of uncovered nodes in W . Such a problem is well known in the literature as the *Orienteering Problem*, and can be approximated within a constant [3, 4]. Following standard methods (e.g., see [13]), we may then obtain an $\mathcal{O}(\log C)$ -approximation algorithm for the SBP and show that the integrality gap of (IP) is at most $\mathcal{O}(\log C)$ (we refer to the full version of this paper for a rigorous argument). However, these logarithmic results are not known to be tight; in particular, we show how to do better for the SBP on *trees*.

In the remainder of this paper, we will focus on the infinite capacity version of the SBP on a tree T with root s . We denote by $P(u, v)$ and $d(u, v)$ the unique path from u to v in T , and its corresponding length. For a subset of edges F , we let $d(F) := \sum_{e \in F} d(e)$. An Euler tour of a connected set of edges is a walk that visits each edge exactly twice. We note that subtrees of T that contain no vertices in W will never be visited by a bus in any optimal solution, and thus we can assume without loss of generality that all leaves of T contain children. In such an instance, a feasible solution will simply cover all of T with bus routes and thus is still feasible if every node of T contains a child (assuming infinite capacities). We thus will assume, without loss of generality, that $W = V$.

3 A 4-Approximation to the SBP on Trees

We prove Theorem 1 by first giving a combinatorial 3-approximation for the SBP with unlimited capacity on graphs that are trees, and subsequently applying Lemma 1. Our algorithm is based on the following intuitive observations:

- When the input tree is very short (say, of height at most $\frac{R}{2}$ on an instance with regret R), then it is relatively easy to obtain a 2-approximation for the SBP by simply cutting an Euler tour of the tree into short pieces and assigning each piece to a bus.
- General trees can be partitioned into smaller pieces (subtrees) such that at least one bus is required for each piece, but each piece can be solved almost optimally via a similar Euler tour method.

We begin with some definitions. We call a set of vertices $\{a_1, \dots, a_m\} \subseteq V$ *R-independent* if for all $a_i \neq a_j$, we have $d(a_i, \text{lca}(a_i, a_j)) > \frac{R}{2}$, where $\text{lca}(a_i, a_j)$ is the lowest common ancestor of the vertices a_i and a_j in T . By iteratively marking the leaf in T furthest from the root such that R -independence is maintained among marked leaves, we can obtain, in polynomial time, an inclusion-wise maximal R -independent set of leaves A such that all vertices in T are within a distance of $\frac{R}{2}$ from a path $P(s, a)$ for some $a \in A$. We shall call A a set of *anchors*. By construction, no two distinct anchors a_i and a_j can both be covered by a walk of regret at most R , immediately yielding the following lower bound:

Proposition 2. *The size $|A|$ of the set of anchors is a lower bound on the number of buses that is needed in any feasible solution.*

We now give a second useful lower bound. Let $Q := \bigcup_{a \in A} P(s, a)$. We call Q the *skeleton* of T , noting that Q is a subtree of T whose leaves are the anchors. Observe that all edges in the skeleton Q will automatically be covered if a bus visits each anchor. Since each anchor must be visited at least once, it suffices to only consider covering the anchors and the *non-skeletal edges* of T , i.e. the edges in $T \setminus Q$. The edges in $T \setminus Q$ form a collection of disjoint subtrees, each of which has height at most $\frac{R}{2}$. We call these *short subtrees*.

Suppose that a feasible walk starts at a vertex v in a short subtree \mathcal{T} . It will cover all the edges in $P(s, v)$, and may possibly cover some additional detour edges having total length at most $\frac{R}{2}$. Since \mathcal{T} is a short subtree, the non-skeletal edges in $P(s, v)$ have total length at most $\frac{R}{2}$. It follows that:

Observation 1. *The set of non-skeletal edges covered by any feasible walk P must have total length at most R : at most $\frac{R}{2}$ in length along the path from its starting vertex to the root, and at most $\frac{R}{2}$ length in edges covered by detours.*

From this, we can observe the following lower bound on the number of buses:

Proposition 3. *The number $\frac{1}{R} \sum_{e \in T \setminus Q} d(e)$ is a lower bound on the number of buses that are needed in any feasible solution.*

We build our 3-approximation from these two lower bounds by partitioning the edges of T into a family of subtrees each containing a single anchor, and approximating the optimal solution well on each of these subtrees. For anchors $A = \{a_1, \dots, a_m\}$, we define associated paths of edges $\{P_1, \dots, P_m\}$ as follows: (i) $P_1 = P(s, a_1)$, and (ii) $P_i = P(s, a_i) \setminus \left(\bigcup_{j=1}^{i-1} \{P_j\} \right)$ for $2 \leq i \leq m$. The edges in $\{P_1, \dots, P_m\}$ form a partition of the skeleton Q into paths, each of which starts at a different anchor.

We then let T_i be the set of all edges in both the path P_i and the set of all short subtrees attached to P_i . If a short subtree is attached to a junction point where two paths P_i and P_j meet, we arbitrarily assign it to either P_i or P_j so that the sets $\{T_1, \dots, T_m\}$ form a partition of all of the edges of T into a collection of subtrees, each containing a single anchor.

For each $1 \leq i \leq m$ we define a directed walk W_i that starts at the anchor a_i , proceeds along P_i in the direction toward the root s , and collects every edge in T_i by tracing out an Euler tour around each of the short subtrees in T_i that are attached to P_i . One may easily verify that it is always possible to quickly find such a walk such that the following properties are satisfied:

- W_i contains each edge in P_i exactly once and always proceeds in the direction toward s when collecting each edge in P_i .
- W_i contains each edge in $T_i \setminus P_i$ exactly twice: once proceeding in the direction away from s , and once in the direction toward s .

We now greedily assign the edges in the short subtrees in T_i to buses by simply adding edges to buses in the order in which they are visited by W_i . We first initialize a bus β_1 at the anchor a_i and have it travel along W_i until the total length of all of the edges it has traversed in the *downward* direction (away from the root s) is exactly $\frac{R}{2}$. At this point, we assume it lies on some vertex v_1 (if not, we may imagine adding v_1 to the middle of an existing edge in T_i , although this will not be relevant to our solution as there are then no children at v_1). We send bus β_1 from v_1 immediately back to the root s and create a new bus β_2 that starts at v_1 and continues to follow W_i until it too has traversed exactly $\frac{R}{2}$ length in edges of T_i in the downward direction. We assume it then lies at a vertex v_2 , create a new bus β_3 that starts at v_2 and continues to follow W_i , and so on. Eventually, some bus β_k will pick up the last remaining children and proceed to the root s , possibly with leftover detour to spare. We observe that the number of buses used is exactly $\lceil \frac{2 \sum_{e \in T_i \setminus P_i} d(e)}{R} \rceil$ since each bus other than the last one consumes exactly $\frac{R}{2}$ of the downward directed edges in W_i , and W_i proceeds downward along each edge in $T_i \setminus P_i$ exactly once. We also note that this is a feasible solution since a bus travelling a total downward direction of $\frac{R}{2}$ must make a detour no greater than R .

Doing this for each edge set T_i yields a feasible solution to the original instance using exactly $\sum_{i=1}^m \lceil \frac{2 \sum_{e \in T_i \setminus P_i} d(e)}{R} \rceil$ buses. This is at most

$$m + \frac{2}{R} \sum_{i=1}^m \sum_{e \in T_i \setminus P_i} d(e) = m + 2 \frac{\sum_{e \in T \setminus Q} d(e)}{R} \leq 3OPT$$

by Proposition 2 and Proposition 3, where OPT is the optimal number of buses required in any feasible solution. Together with Lemma 1, this proves Theorem 1.

One may notice that the bounds given in Propositions 2 and 3 are necessarily also respected by fractional solutions to the LP relaxation of (IP). Together with the argument above, this immediately implies that (IP) has an integrality gap of at most 4 (and 3 in the case of infinite capacities), proving Theorem 2. We refer to the full version of this paper for more details.

4 A 12.5-Approximation to the Uncapacitated SBP-R on Trees

In this section, the School Bus Problem with Regret Minimization (SBP-R) is considered. In SBP-R, the number of routes is bounded by a given parameter $N \in \mathbb{N}$ while the maximum regret is to be minimized. We prove Theorem 3 by giving a polynomial time 12.5-approximation algorithm for SBP-R.

Without loss of generality, we may assume the tree T to be binary. Suppose we can fix a value R for the regret. We will develop an algorithm that, given an instance and the value R , either outputs a set of at most N bus routes, with a maximum regret of $12.5R$, or asserts that every solution with at most N buses must have a regret value $> R$. Then, we can do binary search on the regret values and output the best solution found.

Suppose we have guessed a value for R . First, we find a set of anchors A with respect to R as described in section 3. Now, we look for a set of routes which only start at the anchors. Using the notion introduced for SBP in section 3, our strategy is to cluster and cut the short subtrees into suitable pieces (called *tickets*) that can be collected efficiently by buses. In order to explain the cutting technique for the short subtrees, we need to introduce some more definitions.

Every vertex $v \in Q$ of the skeleton is called a *junction point* if either it is the root s or v has degree more than 2 in Q . Let J be the set of junction points. The skeleton Q can be split at its junction points into a set of edge-disjoint paths, which we will call *core segments*. Formally, a path in Q is a core segment if and only if its endpoints are anchors or junction points and it contains no junction points in its interior. The next lemma shows how we obtain suitable tickets from a collection of short subtrees.

Lemma 2. *There exists a polynomial-time algorithm that, given a path P and a collection \mathcal{C} of short subtrees whose roots lie on P , produces a partition of the edges of \mathcal{C} into tickets $E_0^{\mathcal{C}}, E_1^{\mathcal{C}}, \dots, E_k^{\mathcal{C}}$ with $k \leq \left\lfloor \frac{\sum_{T \in \mathcal{C}} d(T)}{R} \right\rfloor$ such that:*

- (P1) *All of the edges in $E_0^{\mathcal{C}}$ can be collected with an additional regret $\leq 2.5R$ by a single bus whose route contains P .*
- (P2) *For all $1 \leq i \leq k$, all the edges in $E_i^{\mathcal{C}}$ can be collected with an additional regret at most $3R$ by a single bus whose route contains P .*

Proof. Since tree T is binary it follows that each short subtree in \mathcal{C} has a single root edge that connects it to the skeleton Q . Shift each of the short subtrees to the lowest node v of path P , and subsequently compute the Euler tour τ of all short subtrees with root v . Tour τ has the following important *contiguity property*: the edges of each of the short trees in \mathcal{C} form a contiguous sub-path.

We now cut τ into suitable pieces: Starting at v , walk along τ , and cut it at the first node such that the resulting piece has length $> 2R$. Continue like this to obtain $k + 1$ tickets. Denote by $E_1^{\mathcal{C}}, \dots, E_k^{\mathcal{C}}$ all but the last piece. The last part of the Euler tour defines $E_0^{\mathcal{C}}$. Note that $k \leq \left\lfloor \frac{\sum_{T \in \mathcal{C}} d(T)}{R} \right\rfloor$, and that every cutting point is covered by exactly two tickets. Since every node needs to be

covered only once in a solution, we can remove the last edge from each set E_i^C ($1 \leq i \leq k$); note that the resulting ticket has length at most $2R$.

The construction together with the contiguity property of τ implies that the walk corresponding to E_i^C partially visits at most two short subtrees (at the beginning and end); all other short subtrees are either fully contained in E_i^C , or not at all. Start and end vertex of the ticket are also at distance at most $\frac{R}{2}$ from v (cf. the definition of a short subtree); it follows that a bus whose root walk contains P can cover all edges corresponding to a ticket with regret at most $3R$.

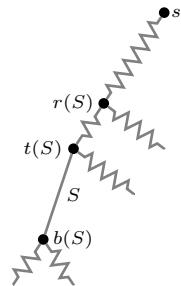
The last piece E_0^C of the Euler tour remaining from the cutting procedure certainly has length $\leq 2R$. Since the Euler tour goes back to v , only the distance to the starting point of the last piece has to be connected to v in order to obtain a set of tours. As in the previous case, one bus can cover these tours with regret $2.5R$, since the height of each subtree is at most $\frac{R}{2}$.

This strategy fulfills the properties (P1) and (P2) and runs in polynomial time.

The next simple lemma (whose proof is omitted) will be useful later.

Lemma 3. *One can find a mapping $\phi : J \longrightarrow A$ from junction points to anchors in polynomial time with the following properties:*

- (i) *For all $j \in J$, the junction point j lies on the path $P(s, \phi(j))$;*
- (ii) *For all $a \in A$, there is at most one junction point $j \in J$ with $\phi(j) = a$.*



For every core segment S , let $t(S)$ and $b(S)$ be the top and the bottom junction points in S . Let further $r(S)$ be the highest junction point at distance at most $R/2$ from $t(S)$ (see Fig. at side). Our algorithm works as follows.

Algorithm 1

1. Find a maximal R -independent set of anchors A .
2. Initialize a default bus at each anchor $a \in A$.
3. For each junction point $j \in J$ in bottom-up order do:

Assign an arbitrary left-to-right ordering of the two segments S_l, S_r with $t(S_l) = j = t(S_r)$.

- (a) Let \mathcal{C}_1 be the collection of short subtrees whose root node lies in the left core segment S_l at distance $\leq R/2$ from j . Let \mathcal{C}_2 be the collection of short subtrees whose root node lies in the core segment S_j with $b(S_j) = j$ at distance $> R/2$ from $t(S_j)$. Note that $\mathcal{C}_2 = \emptyset$ is possible.
- (b) Apply Lemma 2 to $\mathcal{C}_1 \cup \mathcal{C}_2$ on $P = S_l \cup S_j$, and obtain tickets E_0, E_1, \dots, E_y .
- (c) Let \mathcal{C}_3 be the collection of short subtrees whose root node lies in the right core segment S_r at distance $\leq R/2$ from j .
- (d) Apply Lemma 2 to \mathcal{C}_3 on path $P = S_r$, and obtain tickets F_0, F_1, \dots, F_z .

- (e) Assign the tickets E_0, F_0 to the default bus at $\phi(j)$; remove these tickets.
 - (f) Place the y tickets E_1, \dots, E_y and z tickets F_1, \dots, F_z at $r(S_l) = r(S_r)$.
4. For each anchor $a \in A$ do:
- Let \mathcal{C}_a be the collection of short subtrees whose root node lies in the core segment S_a with $b(S_a) = a$ at distance $> R/2$ from $t(S_a)$.
 - (a) Apply Lemma 2 to \mathcal{C}_a on $P = S_a$, and obtain tickets K_0, K_1, \dots, K_w .
 - (b) Assign K_0 to the default bus at a and remove this ticket.
 - (c) Place the w tickets K_1, \dots, K_w at a .
5. Assign every bus greedily the lowest ticket available on its path to the root.
6. Add a new bus for each unmatched ticket.

Lemma 4. *Algorithm 1 outputs a set of buses with maximum regret $12.5R$.*

Proof. Any bus starting at an anchor a collects at most 4 different regret amounts:

- at most one ticket from the matching in step 5. In the worst case, the ticket is placed at a junction point $r(S)$ where S denotes the segment where the subtrees contributing to the ticket are rooted. The top junction point $t(S)$ is at distance at most $\frac{R}{2}$ from $r(S)$, by definition. Since the subtrees considered for the ticket are rooted on S at distance at most $\frac{R}{2}$ from $t(S)$, every tour of a ticket is rooted at the skeleton at distance $\leq R$ from $P(s, a)$. Together with (P2), we can cover a ticket with a walk of regret $\leq 5R$.
- at most two remaining pieces of the Euler tour in step 3(e). There is at most one junction point j with $\phi(j) = a$ by (ii) of lemma 3. For a junction point j , each part E_0 and F_0 can be covered with regret $\leq 2.5R$ by (P2).
- at most one remaining piece of the Euler tour at a assigned in step 4(b). This ticket K_0 can be covered with regret $\leq 2.5R$ by (P2).

In total, the bus from anchor a collects regret of at most $5R + 5R + 2.5R = 12.5R$.

The following observation (proof omitted) is used to prove the next lemma.

Observation 2. *Any edge $e \in T \setminus Q$ that is contained in a ticket T placed at a junction node j must be covered by a bus starting from an anchor in the subtree rooted at j .*

Also if A is a set of anchors in T , then $A \cap F$ is a set of anchors of any subtree $F \subseteq T$. Let B be the number of bus routes output by the algorithm. We obtain:

Lemma 5. *B is a lower bound on the number of buses with regret at most R needed to cover all points.*

Proof. Observe that to cover all points, the number of arbitrary buses with regret $\leq R$ is at least the minimum number of buses from anchors that cover $\leq R$ non-skeletal edges each. To show that B is a lower bound on the latter number, we apply Observation 1 and 2 locally and show how to combine the lower bounds from two disjoint subtrees. We can interpret B as the number of anchors plus the number of tickets that are not assigned to a bus in step 5 of the algorithm. Note that at every anchor requires a bus by Proposition 2. We traverse the tree

now bottom up from the anchors and inductively compute at each anchor or junction point a lower bound on the number of buses. We distinguish two cases at a node j that is either an anchor or a junction point:

Case 1: There is no unmatched ticket placed at j . If j is an anchor, Proposition 2 applies. Otherwise, since all tickets at j are matched, we can focus on the lower bounds at the two successor junction points j_1, j_2 below j . It follows from Observation 2 that the lower bound at j is the sum of the buses needed at each of the disjoint subtrees rooted at j_1 and j_2 .

Case 2: There is at least one unmatched ticket placed at j . Due to Observation 2, we look at the subtree below j . By step 5 of the algorithm, every bus starting in this subtree is full. Thus Observation 1 proves that we need an additional bus for each ticket. This yields that the lower bound is the sum of the number of buses used below j plus the number of unmatched tickets.

References

- [1] Adamaszek, A., Czumaj, A., Lingas, A.: PTAS for k -Tour Cover Problem on the Plane for Moderately Large Values of k . In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 994–1003. Springer, Heidelberg (2009)
- [2] Asano, T., Katoh, N., Tamaki, H., Tokuyama, T.: Covering points in the plane by k -tours: towards a polynomial time approximation scheme for general k . In: STOC 1997 (1997)
- [3] Blum, A., Chawla, S., Karger, D.R., Lane, T., Meyerson, A., Minkoff, M.: Approximation Algorithms for Orienteering and Discounted-Reward TSP. SIAM Journal on Computing 37(2), 653–670 (2007)
- [4] Chekuri, C., Korula, N., Pal, M.: Improved Algorithms for Orienteering and Related Problems. In: SODA, pp. 661–670 (2008)
- [5] Haimovich, M., Rinnoy Kan, A.H.G.: Bounds and heuristic for capacitated routing problems. Mathematics of OR 10(4), 527–542 (1985)
- [6] Labbe, M., Laporte, G., Mercure, H.: Capacitated Vehicle Routing on Trees. Operations Research 39(4), 616–622 (1991)
- [7] Laporte, G., Desrochers, M., Norbert, Y.: Two exact algorithms for the Distance Constrained Vehicle Routing Problem. Networks 14, 47–61 (1984)
- [8] Li, C.-L., Simchi-Levi, S., Desrochers, M.: On the distance constrained vehicle routing problem. Operations Research 40, 790–799 (1992)
- [9] Nagarajan, V., Ravi, R.: Approximation Algorithms for Distance Constrained Vehicle Routing Problems. Tepper School of Business, Carnegie Mellon University, Pittsburgh (2008)
- [10] Park, J., Kim, B.-I.: The school bus routing problem: A review. European Journal of Operational Research 202, 311–319 (2010)
- [11] Spada, M., Bierlaire, M., Liebling, T.M.: Decision-Aiding Methodology for the School Bus Routing and Scheduling Problem. Transportation Science 39(4), 477–490 (2005)
- [12] Toth, P., Vigo, D.: The Vehicle Routing Problem (2001)
- [13] Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)

Improved Approximations for Buy-at-Bulk and Shallow-Light k -Steiner Trees and ($k, 2$)-Subgraph

M. Reza Khani^{1,*} and Mohammad R. Salavatipour^{2,**}

¹ Dept. of Computing Science, Univ. of Alberta
khani@ualberta.ca

² Toyota Tech. Inst. at Chicago, and Dept. of Computing Science, Univ. of Alberta
mreza@cs.ualberta.ca

Abstract. In this paper we give improved approximation algorithms for some network design problems. In the Bounded-Diameter or Shallow-Light k -Steiner tree problem (SLkST), we are given an undirected graph $G = (V, E)$ with terminals $T \subseteq V$ containing a root $r \in T$, a cost function $c : E \rightarrow \mathbb{R}^+$, a length function $\ell : E \rightarrow \mathbb{R}^+$, a bound $L > 0$ and an integer $k \geq 1$. The goal is to find a minimum c -cost r -rooted Steiner tree containing at least k terminals whose diameter under ℓ metric is at most L . The input to the Buy-at-Bulk k -Steiner tree problem (BBkST) is similar: graph $G = (V, E)$, terminals $T \subseteq V$, cost and length functions $c, \ell : E \rightarrow \mathbb{R}^+$, and an integer $k \geq 1$. The goal is to find a minimum total cost r -rooted Steiner tree H containing at least k terminals, where the cost of each edge e is $c(e) + \ell(e) \cdot f(e)$ where $f(e)$ denotes the number of terminals whose path to root in H contains edge e . We present a bicriteria $(O(\log^2 n), O(\log n))$ -approximation for SLkST: the algorithm finds a k -Steiner tree of diameter at most $O(L \cdot \log n)$ whose cost is at most $O(\log^2 n \cdot \text{OPT}^*)$ where OPT^* is the cost of an LP relaxation of the problem. This improves on the algorithm of [9] with ratio $(O(\log^4 n), O(\log^2 n))$. Using this, we obtain an $O(\log^3 n)$ -approximation for BBkST, which improves upon the $O(\log^4 n)$ -approximation of [9]. We also consider the problem of finding a minimum cost 2-edge-connected subgraph with at least k vertices, which is introduced as the $(k, 2)$ -subgraph problem in [14]. We give an $O(\log n)$ -approximation algorithm for this problem which improves upon the $O(\log^2 n)$ -approximation of [14].

1 Introduction

We consider some network design problems where in each one we are given an undirected graph $G = (V, E)$ with a terminal set $T \subseteq V$ (including a node $r \in T$ called root) and some cost functions defined on the edges, plus an integer $k \geq 1$. The goal is to find a subgraph satisfying certain properties with minimum cost

* Supported by Alberta Innovates.

** Supported by NSERC and an Alberta Innovates New Faculty award.

which contains at least k terminals. Below, we describe each of these problems in details.

Bounded Diameter or Shallow-Light Steiner Tree and k -Steiner Tree: Suppose we are given an undirected graph $G = (V, E)$, a cost function $c : E \rightarrow \mathbb{R}^+$, a length function $\ell : E \rightarrow \mathbb{R}^+$, a subset $T \subseteq V$ called terminals which includes a root node r , and a positive bound L . The goal is to find a Steiner tree over terminals T and rooted at r such that the cost of the tree (under c metric) is minimized while the diameter of the tree (under ℓ metric) is at most L . This problem is referred to as Bounded Diameter (BDST) or Shallow-Light Steiner Tree (SLST). In a slightly more general setting, in the input we also have an integer $k \geq 1$ and a feasible solution is an r -rooted Steiner tree containing at least k terminals. We refer to this as Shallow-Light k -Steiner Tree (SLkST).

Another closely related class of network design problems are buy-at-bulk network design problems defined below.

Buy-at-Bulk Steiner Tree (BBST) and k -Steiner Tree (BBkST): Suppose we are given an undirected graph $G = (V, E)$, a set of terminals $T \subseteq V$ including root r , a sub-additive monotone non-decreasing cost function $f_e : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ for each edge e , and positive real demand values $\{\delta_i\}_i$, one for each $t_i \in T$. In the BBST problem the goal is to find an r -rooted Steiner tree to route the demands from terminals to root which minimizes the sum of cost of the edges, where the cost of each edge e is $f_e(\delta(e))$ where $\delta(e)$ is the total demand routed over edge e . This is also referred to as single-sink buy-at-bulk problem. Similar to SLkST, one can generalize the BBST problem by having an extra parameter $k \geq 1$ in the input and a feasible solution is an r -rooted Steiner tree which contains at least k terminals (instead of all of the terminals). This way, we obtain the Buy-at-Bulk k -Steiner Tree (BBkST) problem. It can be shown that the definition of buy-at-bulk problems given above is equivalent (with a small constant factor loss in approximation factor) to the following variation which is also called cost-distance. The input is the same except that instead of function f_e for every edge e , we have two metric functions on the edges: $c : E \rightarrow \mathbb{R}^+$ is called cost and $\ell : E \rightarrow \mathbb{R}^+$ is called length. The cost of a feasible solution H is defined as: $\sum_{e \in H} c(e) + \sum_i \delta_i \cdot L(t_i)$, where $L(t_i)$ is the length (w.r.t ℓ) of the r, t_i -path in H . It is easy to see that this formulation is a special case of buy-at-bulk since a linear function (defined based on c and ℓ) is also sub-additive. It turns out that an α -approximation for the cost-distance version implies a $(2\alpha + \epsilon)$ -approximation algorithm for the buy-at-bulk version too (see [15][16]). For simplicity, we focus on the two cost function (cost+distance) formulation of buy-at-bulk from now on.

Network optimization problems with multiple cost functions, such as buy-at-bulk network design problems, have been studied extensively because of their applications. These problems can model, among others, situations where every edge e (link) can be either purchased at a fixed price $c(e)$ or rented at a price $r(e)$ per amount of flow (or load). The selected edges are required to provide certain bandwidth to satisfy certain demands between nodes of the graph. So if

an edge is rented and there is a flow of $f(e)$ on that edge the cost for that edge will be $r(e) \cdot f(e)$ whereas if the edge is purchased, the cost will be $c(e)$ regardless of the flow. It can be shown that this problem and some other variations can be modeled using buy-at-bulk network design defined above (see [9]). Buy-at-bulk problems and their special cases have been studied through a long line of papers in the operation research and computer science communities after the problem was introduced by Salman et al. [18] (see e.g. [1, 2, 5, 8, 9, 12, 13, 16]).

Another major line of research in network design problems has focused on problems with connectivity requirements where one has another parameter k , and the goal is to find a subgraph satisfying the connectivity requirements with a lower bound k on the total number of vertices. The most well-studied problem in this class is the minimum k -spanning tree problem, a.k.a. k -MST. The approximation factor for this problem was improved from \sqrt{k} [17] to 2 [7] in a series of papers. A very natural common generalization of both the k -MST problem and the minimum cost λ -edge-connected spanning subgraph problem is the (k, λ) -subgraph problem introduced in [14]. In this paper we focus on the case of $\lambda = 2$:

$(k, 2)$ -Subgraph Problem: In the (k, λ) -subgraph problem, we are given a (multi)graph $G = (V, E)$ with a cost function $c : E \rightarrow \mathbb{R}^+$, and a positive integer k . The goal is to find a minimum cost λ -edge-connected subgraph containing at least k vertices.

We should point out that the cost function c is arbitrary (i.e. does not necessarily satisfy the triangle inequality). Furthermore, we are not allowed to take more copies of an edge that present in the graph. In particular, if G is a simple graph the solution must be simple too. The (k, λ) -subgraph problem contains some classical problems as special cases. For example, $(k, 1)$ -subgraph problem is the k -Minimum Spanning Tree problem (k -MST) and $(|V|, \lambda)$ -subgraph is simply asking for a minimum cost λ -edge-connected spanning subgraph. It was proved in [14] that the minimum densest k -subgraph problem has a poly-logarithmic reduction to the (k, λ) -subgraph problem. Since the densest k -subgraph problem has proved to be an extremely difficult problem (the best approximation algorithm for it has ratio $O(n^{\frac{1}{4}})$ [4]), this shows that for general λ , the (k, λ) -subgraph problem is a very hard problem too.

Our results: Our first result is an improved bicriteria approximation for SLkST.

Theorem 1. *There is a polynomial time ($O(\log^2 n), O(\log n)$)-approximation for SLkST. More specifically, the algorithm finds a k -Steiner tree of diameter at most $O(L \cdot \log n)$ whose cost is at most $O(\text{OPT}^* \cdot \log^2 n)$ where OPT^* is the cost of an LP relaxation of the problem.*

To prove this theorem we combine ideas from all of [3, 5, 6, 12]. We first show that the algorithm of Marathe et al. [15] for SLST actually finds a solution with diameter at most $O(L \cdot \log |T|)$ whose cost is at most $O(\text{OPT}^* \cdot \log |T|)$, where OPT^* is the cost of a natural LP-relaxation, so we give a stronger bound (based on an LP relaxation) for the cost of their algorithm. This is based on ideas of [6] which gives a deterministic version of the algorithm of [16] for BBST.

Then we use an idea in [12] to write an LP for SL_kST and use a trick in [3] for rounding this LP (the problem considered in [3] is completely unrelated to SL_kST, namely k -ATSP tour problem). The only previous result for SL_kST was [9] which had ratio $(O(\log^4 n), O(\log^2 n))$. This was obtained by applying the following theorem iteratively:

Theorem 2. [9] *There is a polynomial time algorithm that given an instance of the SL_kST problem with diameter bound L returns a $\frac{k}{8}$ -Steiner tree with diameter at most $O(\log n \cdot L)$ and cost at most $O(\log^3 n \cdot \text{OPT})$, where OPT is the cost of an optimum shallow-light k -Steiner tree with diameter bound L .*

Then a set-cover type analysis yields an $(O(\log^4 n), O(\log^2 n))$ -approximation for SL_kST. In [9], the following lemma was also proved:

Lemma 1. [9] *Suppose we are given an approximation algorithm for the SL_kST problem which returns a solution with at least $\frac{k}{8}$ terminals and has diameter at most $\alpha \cdot L$ and cost at most $\beta \cdot \text{OPT}$. Then we can obtain an approximation algorithm for the BB_kST problem such that given an instance of BB_kST in which all demands $\delta_i = 1$ and a parameter $M \geq \text{OPT}$ (where OPT is the optimum cost of the BB_kST instance) returns a solution of cost at most $O((\alpha + \beta) \log k \cdot M)$.*

The corollary of this lemma and Theorem 2 is an $O(\log^4 n)$ -approximation for the BB_kST for unit demand instances; this can also be extended to an $O(\log^3 n \cdot \log D)$ -approximation for general demands where $D = \sum_t \delta_t$. Using Theorem 1 and Lemma 1 we obtain:

Corollary 1. *There is an $O(\log^2 n \cdot \log D)$ -approximation for BB_kST, where D is the sum of demands.*

This improves the result of [9] for BB_kST by a $\log n$ factor. Finally, we improve the result of [14] for the $(k, 2)$ -subgraph problem:

Theorem 3. *There is an $O(\log n)$ -approximation for the $(k, 2)$ -subgraph problem.*

This is based on rounding an LP relaxation of the problem similar to the one presented in [14].

2 Shallow-Light Steiner Trees

In this section we prove Theorem 1. In order to prove this we first show that the algorithm of [15] in fact bounds the integrality gap of the SLST problem too. Recall that the instance of SLST consists of a graph $G = (V, E)$ with costs $c(e)$, lengths $\ell(e)$, terminal set $T \subseteq V$ including a node r . The goal is to find a Steiner tree H over T with minimum $\sum_{e \in H} c(e)$ such that the diameter w.r.t. ℓ function is at most L . First, let us briefly explain the algorithm of [15] for SLST. Denote the given instance of SLST by \mathcal{I} and define graph F over terminals as below.

For every pair of terminals $u, v \in T$, let $b(u, v)$ be the (approximate) lowest c -cost path between them whose length (under ℓ) is no more than L (there is an FPTAS for computing the value of $b(u, v)$ [10]); let the weight of the edge between (u, v) in F be cost of $b(u, v)$. It is a simple exercise to show that in the optimum solution of \mathcal{I} , we can pair the terminals (except possibly one if the number of them is odd) in such a way that the unique paths connecting the pairs in the optimum are all edge-disjoint. Therefore, the total cost of these paths is at most the value of optimum solution, denoted by OPT , and the length of each of them is at most L . So, if we consider a minimum cost maximum matching in F , the cost of this matching is at most $(1 + \epsilon)\text{OPT}$. We find a minimum cost maximum matching in F and let say terminals $\{u_i, v_i\}_i$ are paired. We pick one of the two (arbitrarily), say u_i and remove v_i from the terminal set; let this new instance be \mathcal{I}' . Clearly the cost of optimum solution on \mathcal{I}' , denoted by OPT' , is at most OPT (as the original solution is still feasible). Also, for any solution of \mathcal{I}' , we can add the paths defined by $b(u_i, v_i)$ to connect v_i to u_i . This gives a solution to instance \mathcal{I} of cost at most $\text{OPT}' + (1 + \epsilon)\text{OPT}$ and the diameter increases by at most L . We can do this repeatedly for $O(\log |T|)$ iterations until $|T| = 1$, since each time the number of terminals drops by a constant factor.

We use the same approach as in [6] to bound the integrality gap of SLST. This LP is a flow-based LP (like those used in [5, 6]). We use the idea of [2] which only considers bounded lengths flow paths. For each terminal $t \in T$ let \mathcal{P}_t be the set of all paths of length at most L from t to r in G . We assume that the terminals are at distinct nodes (we can enforce this by attaching some dummy nodes with edge cost and length equal to zero to the original nodes). Therefore, \mathcal{P}_t and $\mathcal{P}_{t'}$ are disjoint. For every edge e we have an indicator variable x_e which indicates whether edge e belongs to the tree H or not. For each path $p \in \bigcup_t \mathcal{P}_t$, $f(p)$ indicates whether path p is used to connect a terminal to the root.

$$\begin{aligned} \textbf{LP-SLST} \quad & \min \quad \sum_e c(e) \cdot x_e \\ \text{s.t.} \quad & \sum_{p \in \mathcal{P}_t | e \in p} f(p) \leq x_e \quad \forall e \in E, \quad t \in T \quad (1) \\ & \sum_{p \in \mathcal{P}_t} f(p) \geq 1 \quad t \in T \quad (2) \\ & x_e, f(p) \geq 0 \quad \forall e \in E, \quad p \in \bigcup_t \mathcal{P}_t \quad (3) \end{aligned}$$

Define graph F over terminals T as above, i.e. the weight of edge $e = (u, v) \in F$ for two terminals $u, v \in T$ will be the cost of $(1 + \epsilon)$ -approximate minimum c -cost u, v -path of length at most L computed using algorithm of [10]. Let (x^*, f^*) be an optimal solution to LP-SLST with cost OPT^* . We show that the cost of algorithm of [5] is at most $O(\text{OPT}^* \cdot \log |T|)$ while the diameter is at most $O(L \cdot \log |T|)$. The proof of the following lemma is similar to that of Lemma 2.1 in [6] and is omitted here.

Lemma 2. *The graph F contains a matching M of size at least $|T|/3$ whose cost is at most $(1 + \epsilon)\text{OPT}^*$.*

Suppose we have a matching M as above with cost C_M . For every pair of terminals u_i, v_i matched by M pick one of the two as the hub for connecting both of them to r and remove the other one from T . Let OPT' be the LP cost of the

new instance. The current solution (x^*, f^*) is still feasible for the new instance; therefore $\text{OPT}' \leq \text{OPT}^*$. Also, the cost of routing all terminals that were deleted to their hubs is at most $C_M \leq (1+\epsilon)\text{OPT}^*$. Doing this iteratively, an easy inductive argument (using the fact that the number of terminals drops by a constant factor at each iteration) shows that we obtain a solution whose cost is at most $O(\log |T| \cdot \text{OPT}^*)$ and the diameter of the solution is at most $O(L \cdot \log |T|)$.

Now we prove Theorem 1. Our algorithm is based on rounding a natural LP relaxation of the problem. Before presenting the LP we explain how we preprocess the input. We first guess a value OPT' such that $\text{OPT} \leq \text{OPT}' \leq 2\text{OPT}$. We do a binary search between zero and the largest possible value of OPT (e.g. $\sum_{e \in E} c(e)$). The solution returned by the algorithm satisfies the bounds if $\text{OPT}' \geq \text{OPT}$. If the algorithm fails we adjust our guess. We define $V' \subseteq V$ to be the set of vertices that have a path p to r with $c(p) \leq \text{OPT}'$ and length at most L . Let G be the graph obtained from deleting all the vertices of $V \setminus V'$. The following LP is similar to LP-SLST, except that we have an indicator variable y_t for every terminal.

$$\begin{aligned} \textbf{LP-SLkST} \quad & \min \quad \sum_e c(e) \cdot x_e \\ \text{s.t.} \quad & \sum_{p \in P_t | e \in p} f(p) \leq x_e \quad \forall e \in E, \quad t \in T \quad (4) \\ & \sum_{p \in \mathcal{P}_t} f(p) \geq y_t \quad t \in T \quad (5) \\ & \sum_{t \in T} y_t \geq k \quad (6) \\ & y_t \leq 1 \quad t \in T \quad (7) \\ & x_e, f(p) \geq 0 \quad \forall e \in E, \quad p \in \cup_t \mathcal{P}_t \end{aligned}$$

Our rounding algorithm is similar to those in [5, 3] for two completely different problems (density version of Buy-at-Bulk Steiner tree in [5] and k -ATSP tour in [3]). It is easy to verify that a shortest-path algorithm gives a separation oracle for the dual LP; so we can solve this LP (even-though it has exponentially many variables). Suppose that (x^*, y^*, f^*) is an optimum feasible solution to LP-SLkST with value OPT^* . Our first step is to convert (x^*, y^*, f^*) to an approximate solution in which y_t values are of the form 2^{-i} , $0 \leq i \leq \lceil 3 \log n \rceil$. Lemma 3 is analogous of Lemma 9 in [3].

Lemma 3. *There is a feasible solution (x', y', f') to LP-SLkST of cost at most 4OPT^* such that each y'_t is equal to 2^{-i} for some $0 \leq i \leq \lceil 3 \log n \rceil$.*

Let T_i be the set of terminals with $y'_t = 2^{-i}$ and $k_i = |T_i|$, for $0 \leq i \leq \lceil 3 \log n \rceil$. Note that $\sum_{i=0}^{\lceil 3 \log n \rceil} 2^{-i} \cdot k_i \geq k$. Consider the instance of SLST defined over $T_i \cup \{r\}$. First observe that we can obtain a feasible solution (x'', f'') to LP-SLST over this instance of SLST of cost at most $2^{i+2} \cdot \text{OPT}^*$ in the following way: define $x''_e = 2^i \cdot x'_e$ for each edge $e \in E$ and $f''(p) = 2^i \cdot f'(p)$ for each $t \in T_i$ and path $p \in \mathcal{P}_t$. The cost of this solution is $O(2^{i+2} \cdot \text{OPT}^*)$ since $x''_e = 2^{i+2} \cdot x_e^*$. Now since we proved the integrality gap of LP-SLST is $O(\log n)$, we obtain the following:

Lemma 4. *For each T_i , we can find a Steiner tree over $T_i \cup \{r\}$, rooted at r of total cost $O(2^{i+2} \cdot \text{OPT}^* \cdot \log n)$ and diameter $O(L \cdot \log n)$.*

Next we prove the following lemma which is similar to Theorem 10 in [3].

Lemma 5. *Given a Steiner tree H_i over T_i with total cost $O(2^{i+2} \cdot \text{OPT}^* \log n)$ and diameter $O(L \cdot \log n)$, for every $0 \leq i \leq \lceil 3 \log n \rceil$ we can find a Steiner tree H'_i rooted at some $r_i \in T_i$ containing at least $\lceil k_i/2^i \rceil$ terminals of T_i of cost at most $O(\text{OPT}^* \cdot \log n)$ and diameter at most $O(L \cdot \log n)$.*

For now, let us assume this lemma and see how to complete the proof. Suppose that H'_i is the Steiner tree promised by Lemma 5 which contains $\lceil k_i/2^i \rceil$ terminals of T_i and is rooted at a node r'_i . Let p_i be the minimum cost path from r'_i to r with length at most L (note that because of the pre-processing we did, such path p_i exists). Let $H''_i = H'_i \cup p_i$ and let $H = \bigcup_i H''_i$. Observe that H contains at least $\sum_{i=0}^{\lceil 3 \log n \rceil} 2^{-i} \cdot k_i \geq k$ terminals. Also, the total cost of H is at most $\sum_{i=0}^{\lceil 3 \log n \rceil} c(H''_i) \leq O(\text{OPT}^* \cdot \log^2 n)$. Since the diameter of each H''_i is at most $O(L \cdot \log n)$ (because diameter of H'_i is at most $O(L \cdot \log n)$ and we added a path p_i of length at most L to H'_i) and since all of H''_i 's share the root r , the diameter of H is at most $O(L \cdot \log n)$ as well. This completes the proof of Theorem 11.

So it only remains to prove Lemma 5. If we are given the Steiner tree H_i over T_i we use the following lemma with $\beta = \lceil k_i/2^i \rceil$ to edge-decompose H_i into F_1, \dots, F_d such that the number of terminals of each F_i is in $[\beta, 3\beta]$. It follows that $d = \theta(2^i)$ and so by an averaging argument, at least one of F_i 's has cost $O(\text{OPT}^* \cdot \log n)$. The proof of the following lemma is simple (see [11]).

Lemma 6. *Given a rooted tree F containing a set of k terminals and given an integer $1 \leq \beta \leq k$ we can edge-decompose F into trees F_1, \dots, F_d with the number of terminals of each F_i in $[\beta, 3\beta]$, $1 \leq i \leq d$.*

3 An $O(\log n)$ -Approximation for the $(k, 2)$ -Subgraph Problem

In this section we prove Theorem 3. In fact (similar to the algorithm in [4]) our algorithm works for a slightly more general case in which along with the weighted graph $G = (V, E)$ and integer k we are also given a set of terminals $T \subseteq V$ and the goal is to find a minimum cost 2-edge-connected subgraph that contains at least k terminals.

Our algorithm will round an LP relaxation directly instead of iteratively finding good density partial solutions as done in [4]. This is similar to the overall structure of the algorithm we presented for the SLkST. We work with the rooted version of the problem, in which we are given an extra parameter $r \in V$ in the input and the solution must contain root r . Note that, it is sufficient to find a solution in which every terminal has two edge-disjoint paths to r . First we preprocess the graph by deleting the vertices that cannot be part of any optimum solution. For that for every vertex v we find two edge-disjoint paths between v and r of minimum total cost, let us denote it by $d_2(v, r)$. For this we can use a minimum cost flow algorithm between v and r [19]. Suppose we know have guessed a value OPT' such that $\text{OPT} \leq \text{OPT}' \leq 2\text{OPT}$, where OPT is the value of

optimum solution. Clearly every vertex v with $d_2(v, r) > \text{OPT}'$ cannot be part of any optimum solution and can be safely deleted. We work with this pruned version of graph G . Our algorithm is guided by the solution of an LP relaxation of the problem. Consider the following LP relaxation which is similar to what proposed by Lau *et al.* [14].

$$\begin{aligned} \text{LP-k2EC} \quad \min \quad & \sum_e c(e) \cdot x_e \\ \text{s.t.} \quad & x(\delta(U)) \geq 2y_v \quad U \subseteq V - \{r\}, v \in U \quad (8) \\ & x(\delta(U)) - x_{e'} \geq y_v \quad U \subseteq V - \{r\}, v \in U, e' \in \delta(U) \quad (9) \\ & \sum_{v \in T} y_v \geq k \quad (10) \\ & y_r = 1 \quad (11) \\ & y_t \leq 1 \quad t \in T \quad (12) \\ & x_e, y_t \geq 0 \quad \forall e \in E, t \in T \end{aligned}$$

There are two types of indicator variables, x_e for each $e \in E$ and y_v for each $v \in T$; for every subset $U \subseteq V$, $\delta(U)$ is the set of edges across the cut $(U, V - U)$. Constraints (8) and (9) guarantee 2-edge-connectivity to the root. Our algorithm solves this LP and then uses the solution to find an integral solution of cost at most $O(\log(n))$ apart from the optimal value, in order to do that we merge ideas from [3] and [14]. As argued in [14] this LP is a relaxation of the $(k, 2)$ -subgraph problem and we can find an optimum solution of this LP. We run the following algorithm whose detailed steps are explained below.

$(k, 2)$ -Subgraph Algorithm (k2EC)

- Input:** Graph $G = (V, E)$, terminal set $T \subseteq V$ with root r , and integer $k \geq 1$
Output: a 2-edge-connected subgraph containing at least k terminals including r
1. Guess a value of OPT' for optimum solution and run the following algorithm.
 2. $U \leftarrow r$
 3. Start from original graph G and remove all the vertices with $d_2(v, r) > \text{OPT}'$
 4. Solve LP-K2EC and let its solution be (x^*, y^*)
 5. Obtain (x', y') from (x^*, y^*) according to Lemma 7
 6. Let T_i be the set of terminals v with $y'_v = 2^{-i}$ plus the root, for $0 \leq i \leq \lceil 3 \log(n) \rceil$
 7. Find a 2-edge-connected subgraph H_i over $T_i \cup \{r\}$ with cost $O(2^i \cdot \text{OPT}^*)$
 8. From H_i , find a 2-edge-connected subgraph H'_i containing r and at least $\lceil |T_i|/2^i \rceil$ and at most $2\lceil |T_i|/2^i \rceil$ vertices of T_i of cost at most $O(\text{OPT}^*)$ and add it to U ; if failed for any i then double the guess for OPT' and start from Step 2.
 9. Return U .

Fig. 1. Algorithm K2EC

In the rest of this section we prove Theorem 3 using Algorithm K2EC. First we provide the details of the steps of the algorithm. Suppose L is the k th smallest $d_2(v, r)$ value. Clearly $L \leq \text{OPT} \leq kL$. We can start with L as our guess for OPT' and if the algorithm fails to return a feasible solution of cost at most $O(\text{OPT}' \cdot \log n)$ then we double our guess OPT' and run the algorithm again.

Let (x^*, y^*) be an optimum feasible solution to LP-k2EC with value OPT^* . For Step 5 of K2EC we round y values of the LP following the schema in [3]. The proof of following lemma is very similar to Lemma 3.

Lemma 7. *There is a feasible solution (x', y') to LP-K2EC of cost at most 4OPT^* such that all nonzero entries of y' belong to $\{2^{-i} | 0 \leq i \leq \lceil 3 \log(n) \rceil\}$.*

Let T_i be the set of terminals with $y'_i = 2^{-i}$ and $k_i = |T_i|$, for $0 \leq i \leq \lceil 3 \log n \rceil$. Note that $\sum_{i=0}^{\lceil 3 \log n \rceil} 2^{-i} \cdot k_i \geq k$. Consider an instance of classical survivable network design problem over terminals in $T_i \cup \{r\}$ with connectivity requirement 2 from every node in T_i to root. In the following lemma we show that we can compute a 2-edge-connected subgraph H_i over $T_i \cup \{r\}$ of cost at most $O(2^i \cdot \text{OPT}^*)$. This describes how to perform Step 7. The proof of this lemma is similar to Lemma 5.2 in [14].

Lemma 8. *In Step 7, For each $0 \leq i \leq \lceil 3 \log n \rceil$, we can find a 2-edge-connected subgraph H_i of cost at most $2^{i+3} \cdot \text{OPT}^*$ containing terminals $T_i \cup \{r\}$.*

In the following we show how to find subgraph H'_i in Step 8, which is 2-edge-connected, has root r , and cost $O(\text{OPT}')$, assuming that $\text{OPT}' \geq \text{OPT}$. Note that union of all H_i 's ($0 \leq i \leq \lceil 3 \log n \rceil$) will be 2-edge-connected (since r is common in H_i 's), has at least k terminals, and has cost $O(\text{OPT}' \cdot \log n)$, which completes the proof of approximation ratio.

To show how to find a subgraph H'_i we use the same trick as in Section 5.1 of [14]. A nowhere-zero 6-flow in a directed graph $D = (V, A)$, is a function $f : A \rightarrow \mathbb{Z}_6$ such that we have flow conservation at every node (i.e. $f(\delta^{in}(v)) = f(\delta^{out}(v))$) and no edge gets f value of zero. If there is an orientation of an undirected graph H in which a nowhere-zero 6-flow can be defined we say H has a nowhere-zero 6-flow. Seymour [20] proved that every 2-edge-connected graph has a nowhere-zero 6-flow which can also be found in polynomial time. We obtain a multigraph $D = (H_i, A)$ from H_i by placing $f(e)$ copies of e with the direction defined by the flow. From Lemma 8 and the fact that we have at most 6 copies of each edge, the cost of D can be at most $6 \times 2^{i+3} \cdot \text{OPT}^*$. Note that D does not have directed cycle of length 2, therefore has an Eulerian Walk. Start from r and build an Eulerian walk and partition the walk into the segments P_1, P_2, \dots, P_ℓ each of which includes $\lceil |T_i|/2^i \rceil$ terminals of H_i except possibly P_ℓ which can have between $\lceil |T_i|/2^i \rceil$ and $2\lceil |T_i|/2^i \rceil$ terminals. Thus, $\ell \geq \max(1, 2^{i-1})$ and so there is an index $1 \leq q \leq \ell$ such that the cost of path P_q is at most $6 \times 2^{i+2} \cdot \text{OPT}^*/2^{i-1} = 48\text{OPT}^*$. Let u, w be the endpoints of P_q and let Q_u^1 and Q_u^2 be the two edge-disjoint paths of $d_2(u, r)$ (in G) and Q_w^1 and Q_w^2 be the two edge-disjoint paths of $d_2(w, r)$ (again in G) of minimum total cost. The sum of costs of Q_u^1, Q_u^2, Q_w^1 , and Q_w^2 is at most $2\text{OPT}'$. Let F_q be the simple graph in G defined by the edges of P_q and let $H'_i = F_q \cup Q_u^1 \cup Q_u^2 \cup Q_w^1 \cup Q_w^2$. It follows that H'_i has cost at most $48\text{OPT}^* + 2\text{OPT}' \leq 50\text{OPT}'$. It is easy to verify that that H'_i is 2-edge-connected.

References

1. Andrews, M., Zhang, L.: Approximation algorithms for access network design. *Algorithmica* 32(2), 197–215 (2002); Preliminary version in Proc. of IEEE FOCS (1998)
2. Awerbuch, B., Azar, Y.: Buy-at-bulk network design. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS 1997 (1997)
3. Bateni, M., Chuzhoy, J.: Approximation Algorithms for the Directed k-Tour and k-stroll Problems. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX 2010, LNCS, vol. 6302. Springer, Heidelberg (2010)
4. Bhaskara, A., Charikar, M., Chlamtac, E., Feige, U., Vijayaraghavan, A.: Detecting High Log-Densities – an $O(n^{1/4})$ -Approximation for Densest k -Subgraph. In: Proceedings of Symposium on the Theory of Computing, STOC (2010)
5. Chekuri, C., Hajiaghayi, M., Kortsarz, G., Salavatipour, M.: Approximation Algorithms for Non-Uniform Buy-at-Bulk Network Design. *SIAM J. on Computing* 39(5), 1772–1798 (2009)
6. Chekuri, C., Khanna, S., Naor, J.: A Deterministic Approximation Algorithm for the Cost-Distance Problem Short paper in. In: Proc. of ACM-SIAM SODA, pp. 232–233 (2001)
7. Garg, N.: Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In: Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing (STOC), pp. 396–402 (2005)
8. Gupta, A., Kumar, A., Pal, M., Roughgarden, T.: Approximation via cost-sharing: a simple approximation algorithm for the multicommodity rent-or-buy problem. In: Proceedings of the 44rd Symposium on Foundations of Computer Science, FOCS 2003 (2003)
9. Hajiaghayi, M.T., Kortsarz, G., Salavatipour, M.R.: Approximating buy-at-bulk and shallow-light k -steiner tree. *Algorithmica* 53(1), 89–103 (2009)
10. Hassin, R.: Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research* 17(1), 36–42 (1992)
11. Khani, M.R., Salavatipour, M.R.: Approximation Algorithms for Min-max Tree Cover and Bounded Tree Cover Problems. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) RANDOM 2011 and APPROX 2011. LNCS, vol. 6845, pp. 302–314. Springer, Heidelberg (2011)
12. Kortsarz, G., Nutov, Z.: Approximating Some Network Design Problems with Node Costs. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) APPROX 2009. LNCS, vol. 5687, pp. 231–243. Springer, Heidelberg (2009)
13. Kumar, A., Gupta, A., Roughgarden, T.: A Constant-Factor Approximation Algorithm for the Multicommodity Rent-or-Buy Problem. In: Proceedings of FOCS 2002 (2002)
14. Lau, L., Naor, S., Salavatipour, M.R., Singh, M.: Survivable network design with degree or order constraints. *SIAM J. on Computing* 39(3), 1062–1087 (2009)
15. Marathe, M., Ravi, R., Sundaram, R., Ravi, S.S., Rosenkrantz, D., Hunt, H.B.: Bicriteria network design. *Journal of Algorithms* 28(1), 142–171 (1998)
16. Meyerson, A., Munagala, K., Plotkin, S.: Cost-Distance: Two Metric Network Design. *SIAM J. on Computing*, 2648–1659 (2008)
17. Ravi, R., Sundaram, R., Marathe, M.V., Rosenkrants, D.J., Ravi, S.S.: Spanning trees short or small. *SIAM Journal on Discrete Mathematics* 9(2), 178–200 (1996)
18. Salman, F.S., Cherian, J., Ravi, R., Subramanian, S.: Approximating the Single-Sink Link-Installation Problem in Network Design. *SIAM J. on Optimization* 11(3), 595–610 (2000)
19. Schrijver, A.: Combinatorial optimization: Polyhedra and Efficiency. Springer, Berlin (2003)
20. Seymour, P.D.: In Graph Theory and related topics. In: Proc. Waterloo, pp. 341–355. Academic Press (1977/1979)

Improved Approximation Algorithms for Routing Shop Scheduling

Wei Yu and Guochuan Zhang

College of Computer Science, Zhejiang University, Hangzhou, 310027, China
{yuwei2006831,zgc}@zju.edu.cn

Abstract. We investigate a generalization of classical shop scheduling where n jobs are located at the vertices of a general undirected graph and m machines must travel between the vertices to process the jobs. The aim is to minimize the makespan. For the open shop problem, we develop an $O(\log m \log \log m)$ -approximation algorithm that significantly improves upon the best known $O(\sqrt{m})$ -approximation algorithm. For the flow shop problem, we present an $O(m^{2/3})$ -approximation algorithm that improves upon the best known $\max\{\frac{m+1}{2}, \rho\}$ -approximation algorithm, where ρ is the approximation factor for metric TSP.

Keywords: Routing Scheduling, Open Shop, Flow Shop, Approximation Algorithm.

1 Introduction

In the classical scheduling problem, the machines and the jobs are supposed to be situated at the same location, and hence there is no time lags for the machines to process two successive jobs or operations. However, in a generalization of the classical scheduling problem, called routing-scheduling, the jobs are distributed at the vertices of an undirected network and the machines travel between the vertices to process the jobs. The routing-scheduling problem can model a lot of problems in real-world applications. Examples include route planning of traveling repairmen who serve customers located at different places [1], processing of big or heavy parts distributed at various locations, scheduling of robots that provide daily maintenance operations on immovable machines scattered over a workshop [6], and so on. Note that the routing-scheduling problem can also be seen as the corresponding scheduling problem with sequence-dependent setup times between the jobs [2,3], although it is more natural to consider the problem from a point view of network.

There is a large amount of research on the routing-scheduling problem. The majority of these research involves the single-stage routing-scheduling, see [1, 4, 9, 13, 15, 16, 17, 18, 22, 23]. In this work we focus on the multi-stage routing-scheduling problem, i.e., routing shop scheduling, which is described below. We are given an undirected edge-weighted graph $G = (V, E)$, where $V = \{0, 1, 2, \dots, n\}$ is the vertex set and E is the edge set. There are n jobs, where job j is placed at vertex j for $j = 1, 2, \dots, n$. We have m machines

M_1, M_2, \dots, M_m which originally stay at vertex 0, Job j consists of m operations $O_{1,j}, O_{2,j}, \dots, O_{m,j}$, where $O_{i,j}$ should be processed by machine M_i for $p_{i,j}$ time units without any interruption. To process these jobs the machines travel between the vertices at the same speed. And the travel time between vertices j and k , denoted by $t_{j,k}$, equals the length of the shortest path between them. Clearly, $t_{j,k}$'s satisfy the triangle inequality.

A feasible schedule is to plan a routing for each machine and schedule the operations such that at any time each job is processed by at most one machine (the machine must reach the job) and each machine processes at most one job. In such a schedule S , denote $s_{i,j}$ to be the starting time of operation $O_{i,j}$ and let C_j be the completion time of job j (by which all its operations are done). The makespan of a schedule S is defined as $C_{\max}(S) = \max_{1 \leq j \leq n} \{C_j(S) + t_{j,0}\}$, by which all machines have returned to their home, i.e., vertex 0, after the completion of all jobs.

We consider two models. For open shop the operations of a job may be processed in an arbitrary order, while for flow shop the operations $O_{1,j}, O_{2,j}, \dots, O_{m,j}$ of job j must be processed in this order. We aim to minimize the makespan. By putting “R” to the standard three-field notation for the corresponding scheduling problem ($O||C_{\max}$ and $F||C_{\max}$, see Lawler et al. [19]), the two problems, i.e., routing open shop and routing flow shop, are denoted as $RO||C_{\max}$ and $RF||C_{\max}$ (or $ROm||C_{\max}$ and $Rfm||C_{\max}$ for a fixed number of machines), respectively. For the problem defined on a line or tree graph, we denote it by adding “line” or “tree” before the corresponding notations.

Obviously apart from the open shop and flow shop problems, the well known metric traveling salesman problem (metric TSP for short) is also special cases of our models. For metric TSP, Christofides [11] gave the best known $3/2$ -approximation algorithm. Throughout the paper, we denote the approximation factor for metric TSP by ρ whenever an algorithm for the problem is required.

1.1 Previous Work

For routing flow shop, Averbakh and Berman [5] considered $tree-RF2||C_{\max}$ and presented a polynomial time exact algorithm for the problem with the restriction that each machine has to travel along a shortest tour. Yu et al. [24] proved that $tree-RF2||C_{\max}$ is NP-hard and devised a $10/7$ -approximation algorithm. For $RF||C_{\max}$, Averbakh and Berman [6] presented a simple $\max\{\frac{m+1}{2}, \rho\}$ -approximation algorithm, given a ρ -approximate tour on the underlying graph.

As for routing open shop, Averbakh et al. [7] gave a $6/5$ -approximation algorithm for a special case of $line-RO2||C_{\max}$, which was proven NP-hard by Averbakh et al. [8]. The authors also dealt with $RO||C_{\max}$, they proposed an $(\frac{m+1}{2} + \rho)$ -approximation algorithm by using the above algorithm for $RF||C_{\max}$ and developed an improved $(1 + \frac{\rho}{2})$ -approximation algorithm for the two-machine case for each $\rho \leq 2$. Chernykh et al. [10] gave a better $13/8$ -approximation algorithm for $RO2||C_{\max}$. Moreover, for $RO||C_{\max}$ they devised a best known $O(\sqrt{m})$ -approximation algorithm by utilizing the idea of job-aggregation and the greedy algorithm for the classical open shop problem.

1.2 Our Results and Techniques

In this paper we concentrate on the routing shop scheduling with an arbitrary number of machines. By introducing new approaches we significantly improve the previous results. Our main results are summarized below.

Theorem 1. *For any $\epsilon > 0$, there is an $O(\log m(\log \log m)^{1+\epsilon})$ -approximation algorithm for $RO||C_{\max}$.*

Theorem 2. *There exists an $O(m^{2/3})$ -approximation algorithm for $RF||C_{\max}$.*

In both algorithms we always start with a ρ -approximate tour T_ρ on the underlying graph. Then by properly partitioning the tour into disjoint paths and associating each path with an aggregated job we transfer the original instance into an instance of $O||C_{\max}$ or $F||C_{\max}$ with the number of jobs polynomially bounded by the number of machines. Moreover, the new instance is relatively easily approximated and there is not much loss by aggregating jobs.

For $RO||C_{\max}$ after transferring the original instance I into an instance I' of $O||C_{\max}$, we exchange the roles of the machines and the jobs, and apply the algorithm in [12] for a generalization of flow shop problem, called an acyclic job shop problem, to generate a schedule S' with the makespan bounded by polylogarithmic times of the optimal value of I . Moreover, in S' the processing order of the jobs on each machine is the same as some prescribed permutation. By properly choosing the permutation, we may transfer S' into a feasible schedule S of I whose makespan increases by a constant-factor of the optimal value of I compared to S' . Therefore, the makespan of S is bounded by a polylogarithmic times the optimal value of I .

For $RF||C_{\max}$, we aggregate the jobs to transfer the original instance I into an instance I' of $F||C_{\max}$ with the same machine set and $n' \leq 2m^{2/3} + 1$ jobs. Then using the algorithm proposed by Nagarajan and Sviridenko [20] for permutation flow shop we obtain a permutation schedule S' . The makespan of S' is at most $O(\sqrt{\min\{m, n'\}} \cdot (L' + p'_{\max}))$, where L' and p'_{\max} are the maximum machine load and the maximum processing time of I' , respectively. By the construction of I' , we can bound $L' + p'_{\max}$ within an $O(m^{1/3})$ factor of the optimal value of I . Finally, the property that S' is a permutation schedule, which can not be guaranteed by the algorithm of Czumaj and Scheideler [12], allows us to transfer S' into a permutation schedule S of I with the makespan bounded by the claimed ratio.

The rest of the paper is organized as follows. Section 2 gives some preliminaries and notations. In Section 3 we study the approximation algorithm for $RO||C_{\max}$. In Section 4 we discuss the problem $RF||C_{\max}$.

2 Preliminaries

For $RO||C_{\max}$ or $RF||C_{\max}$, one may assume that each machine always travels towards the next job to be processed by it. Therefore, m permutations

$\alpha_1, \alpha_2, \dots, \alpha_m$ on the job set $\{1, 2, \dots, n\}$ are sufficient to specify a schedule for $RF||C_{\max}$, where α_i denotes the routing of M_i . After completing of the $(j-1)$ st job of α_i , M_i moves to the j th job following the permutation α_i . Upon arrival of the j th job of α_i , M_i processes it immediately once M_{i-1} has finished this job; else, waits there.

However, just the machine routings are not sufficient to specify a schedule for $RO||C_{\max}$ since the processing order of the operations of each job is not prescribed. Therefore, we specify a schedule for $RO||C_{\max}$ with the starting times of all operations.

Given a schedule S for $RF||C_{\max}$ or $RO||C_{\max}$, it is called a permutation schedule if S is compatible with some permutation α , i.e., each machine processes the jobs according to $\alpha_i = \alpha$.

An important property of permutation schedules is given below.

Lemma 1. *Given an instance I of $RO||C_{\max}$ or $RF||C_{\max}$ and a permutation schedule compatible with $\pi = (j_1, j_2, \dots, j_n)$ for the instance I' , which is obtained by ignoring the travel times in I , then a permutation schedule compatible with π for I with the makespan increasing by the length of the tour induced by π , i.e., $\sum_{k=0}^n t_{j_k, j_{k+1}}$ (set $j_0 = j_{n+1} = 0$), can be constructed in polynomial time.*

Proof. Let S' be a permutation schedule compatible with π for I' . We obtain a permutation schedule S compatible with π for I by inserting travel times between the jobs. When inserting travel time between adjacent jobs j and j' on some machine, the processing of all jobs behind j on the machine are delayed by $t_{j,j'}$ time units. Since S' is a permutation schedule, for $k = 1, 2, \dots, n$ the above procedure results in the same delay for the operations of job j_k , i.e., $\sum_{h=0}^{k-1} t_{j_h, j_{h+1}}$, and never produces overlap of any two operations of each job. So S is feasible for I and it can be observed that the makespan increases by $\sum_{k=0}^n t_{j_k, j_{k+1}}$ compared to S' . \square

Next we introduce some notations to be used throughout the paper. Let T (resp. T_ρ) denote the shortest (resp. ρ -approximate) tour on V as well as its length. Obviously, a tour can be specified by a permutation of the jobs.

Given a permutation $\pi = (j_1, j_2, \dots, j_n)$ on $\{1, 2, \dots, n\}$, a consecutive subsequence $(j_h, j_{h+1}, \dots, j_l)$ of π is called maximal increasing if $j_h < j_{h+1} < \dots < j_l$ holds and $j_l > j_{l+1}$ unless $l = n$. A maximal decreasing consecutive subsequence is defined similarly. With these definitions we obtain a unique partition of π into consecutive subsequences $\pi^1, \pi^2, \dots, \pi^k$ such that π^1, π^2, \dots are maximal increasing and π^2, π^3, \dots are maximal decreasing, and this partition is referred to as the ID-partition of π .

Proposition 1. *If $T_\rho = (1, 2, \dots, n)$, then one machine processing the jobs according to π with ID-partition $\pi^1, \pi^2, \dots, \pi^k$ travels at most kT_ρ time units.*

For convenience, we denote an instance of $RO||C_{\max}$ or $RF||C_{\max}$ with $I = (G, m, n, p)$, where G , m , n , p represent the underlying graph, the number of machines, the number of jobs, and the processing time matrix (the entry $p_{i,j}$ indicates the processing time of $O_{i,j}$), respectively. Given $I = (G, m, n, p)$, let

$L_i = \sum_{j=1}^n p_{i,j}$ be the load of machine M_i and $p_j = \sum_{i=1}^m p_{i,j}$ be the processing time of job j . The maximum load is denoted by $L = \max_{1 \leq i \leq m} L_i$ and $p_{\max} = \max_{1 \leq j \leq n} p_j$ indicates the maximum processing time. Let C_{\max}^O (resp. C_{\max}^F) be the optimal makespan for routing open shop (resp. routing flow shop). Clearly, we have

Lemma 2. (i) $C_{\max}^O \geq \max\{L + T, p_{\max}\}$; (ii) $C_{\max}^F \geq \max\{L + T, p_{\max}\}$.

Similarly, we denote an instance of $O||C_{\max}$ or $F||C_{\max}$ by $I = (m, n, p)$ in which the maximum load and the maximum processing time L and p_{\max} , respectively.

3 Routing Open Shop

For $RO||C_{\max}$, Chernykh et al. [10] used the idea of job-aggregation and a greedy algorithm for $O||C_{\max}$ to devise an $O(\sqrt{m})$ -approximation algorithm. In this section we will obtain an improved algorithm with performance ratio $O(\log m \log \log m)$. This algorithm first aggregates the jobs and then invokes an algorithm for the acyclic job shop problem in [12] after exchanging the roles of the machines and the jobs.

We describe the algorithm as follows. Given an instance I of $RO||C_{\max}$, we first construct an instance I' of $O||C_{\max}$ in which the number m of machines is the same as I and the number n' of jobs is at most $2m + 1$. Then we obtain a schedule S' of I' by exchanging the roles of the machines and the jobs and using the algorithm proposed by Czumaj and Scheideler [12] for a generalization of flow shop problem, called an acyclic job shop problem, in which each job has at most one operation on each machine and the operations of the same job is processed in a prescribed order, not necessarily identical for different jobs. The makespan of S' is at most $O(\log m \log \log m)$ -factor of the sum of two lower bounds of the optimal value of I' . And each of these two lower bounds can not exceed a constant-factor of the optimal value of I . Eventually, we transform S' into a schedule S of I whose makespan increases by a constant-factor of the optimal value of I compared to S' .

The construction of the instance I' of $O||C_{\max}$ proceeds as follows. First we find a ρ -approximate tour T_ρ on the underlying graph by invoking some algorithm for metric TSP, then we renumber the jobs such that $T_\rho = (1, 2, \dots, n)$. Next we partition the tour into disjoint paths $P_0 = \{0\}$ and $P_l = \{j_{l-1}+1, \dots, j_l\}$ for $l = 1, 2, \dots, n'$ (the value of n' will be clear later), where j_l is the minimum index such that at least one of the following conditions is satisfied (set $j_0 = 0$ and $t_{n,n+1} = t_{n,0}$ for convenience).

- (i) $\sum_{j=j_{l-1}+1}^{j_l} p_j \geq L$
- (ii) $\sum_{j=j_{l-1}+1}^{j_l} t_{j,j+1} \geq T_\rho/m$
- (iii) $j_l = n$

Since the total processing time of all jobs is at most mL , the number of paths satisfying condition (i) can not exceed m . Since the length of the tour is T_ρ ,

the number of paths satisfying condition (ii) is at most m . Moreover, there is exactly one path satisfying condition (iii). So the total number n' of paths is at most $2m + 1$.

To obtain the instance I' , we associate each path P_l for $l = 1, 2, \dots, n'$ with a job l whose processing time on machine M_i ($i = 1, 2, \dots, m$) is

$$p'_{i,l} = \sum_{j=j_{l-1}+1}^{j_l} p_{i,j} + \sum_{j=j_{l-1}+1}^{j_l-1} t_{j,j+1}.$$

Hence the load of machine M_i is

$$\begin{aligned} L'_i &= \sum_{l=1}^{n'} p'_{i,l} = \sum_{l=1}^{n'} \sum_{j=j_{l-1}+1}^{j_l} p_{i,j} + \sum_{l=1}^{n'} \sum_{j=j_{l-1}+1}^{j_l-1} t_{j,j+1} \\ &= L_i + T_\rho - \sum_{l=0}^{n'} t_{j_l, j_{l+1}} \leq L_i + T_\rho. \end{aligned}$$

Moreover, the processing time of job l is

$$p'_l = \sum_{i=1}^m p'_{i,l} = \sum_{j=j_{l-1}+1}^{j_l-1} p_j + p_{j_l} + m \sum_{j=j_{l-1}+1}^{j_l-1} t_{j,j+1} \leq L + p_{\max} + T_\rho,$$

where the inequality follows because neither condition (i) nor condition (ii) holds for $j_l - 1$ by the definition of j_l . Thus the the maximum load L' and the maximum processing time p'_{\max} of instance I' can be bounded as follows:

$$L' \leq L + T_\rho, \quad p'_{\max} \leq L + p_{\max} + T_\rho.$$

Next we show how to convert a permutation schedule S' of I' compatible with π , which has ID-partition $\pi^1, \pi^2, \dots, \pi^k$, into a permutation schedule S of I with the makespan increasing by at most kT_ρ . Firstly, in S' we put a dummy job 0 with the processing time of each operation zero at time 0 on each M_i . Secondly, we associate job l of I' with path P_l of I for $l = 0, 1, \dots, n'$, and replace the processing of each job l ($l = 1, 2, \dots, n'$) on M_i in I' by consecutive processing of jobs in P_l of I on M_i with travel time $t_{j,j+1}$ between the processing of job j and $j + 1$ for $j = j_{l-1} + 1, \dots, j_l - 1$. Moreover, the jobs in P_l are processed in increasing order of their numbering if l belongs to some subsequence in $\{\pi^1, \pi^3, \dots\}$ and the jobs in P_l are processed in decreasing order of their numbering if l belongs to some subsequence in $\{\pi^2, \pi^4, \dots\}$. Lastly, for each $l = 0, 1, \dots, n'$ we insert travel time between the last processed job in P_l of I and the first processed job in $P_{l'}$ of I if job l of I' is processed immediately before l' in S' . Then we have obtained a permutation schedule S of I . Note that the first two steps can not increase the makespan since the travel times between the jobs of I in P_l are included in the processing time of job l of I' . Combining Proposition 11 and Lemma 11, the last step increases the makespan by at most kT_ρ , since π has ID-partition $\pi^1, \pi^2, \dots, \pi^k$.

Since I' and S can be constructed in polynomial time, we have

Lemma 3. *There exists a polynomial time algorithm that transforms any instance $I = (G, m, n, p)$ of $RO||C_{\max}$ into an instance $I' = (m', n', p')$ of $O||C_{\max}$ such that: (i) $m' = m$, $n' \leq 2m+1$; (ii) $p'_{\max} \leq L + p_{\max} + T_p$; (iii) $L' \leq L + T_p$; (iv) Given a permutation schedule S' of I' compatible with π , which has ID-partition $\pi^1, \pi^2, \dots, \pi^k$, a permutation schedule S of I with the makespan increasing by at most kT_p can be obtained in polynomial time.*

By similar scaling and rounding arguments to the proof of Lemma 3.1 in Shmoys et al. [21] we obtain

Lemma 4. *There exists a polynomial time algorithm that transforms any instance $I = (m, n, p)$ of $F||C_{\max}$ into an instance $I' = (m', n', p')$ of $F||C_{\max}$ such that: (i) $m' = m$, $n' = n$; (ii) $L' = O(mn^2)$, $p'_{\max} = O(m^2n)$; (iii) $\frac{\Delta}{mn}(L' + p'_{\max}) \leq L + p_{\max}$, where $\Delta = \max_{i,j}\{p_{i,j}\}$; (iv) Given a schedule S' of I' with makespan M , a schedule S of I with makespan at most $\frac{\Delta}{mn}M + \Delta$ can be obtained in polynomial time.*

Czumaj and Scheideler [12] proved the following result.

Lemma 5. *For any constant $\epsilon > 0$, there exists a polynomial time algorithm that delivers a schedule for any instance $I = (m, n, p)$ of $F||C_{\max}$ of makespan at most $O(lb \log lb (\log \log lb)^{1+\epsilon})$, where $lb = \max\{L, p_{\max}\}$.*

Combining Lemmas 4 and 5 we obtain

Lemma 6. *For any constant $\epsilon > 0$, there exists a polynomial time algorithm that delivers a schedule for any instance $I = (m, n, p)$ of $F||C_{\max}$ of makespan at most $O(\log(mn)(\log \log(mn))^{1+\epsilon} \cdot (L + p_{\max}))$.*

Using the above lemma we can prove

Lemma 7. *For any constant $\epsilon > 0$, given an instance $I' = (m', n', p')$ of $O||C_{\max}$ and a permutation π on the jobs, then a permutation schedule S compatible with π of makespan at most $O(\log(m'n')(\log \log(m'n'))^{1+\epsilon} \cdot (L' + p'_{\max}))$ can be found in polynomial time.*

Proof. Without loss of generality, we assume that $\pi = (1, 2, \dots, n')$. From instance $I' = (m', n', p')$ of $O||C_{\max}$, we construct an instance $I'' = (n', m', p')$ of $F||C_{\max}$ by exchanging the roles of the machines and the jobs. We associate each operation $O_{i,j}$ of I' with an operation $O_{j,i}$ of I'' with the same processing time. Moreover, the order of the operations of job i is prescribed according to π , i.e. $O_{1,i}, O_{2,i}, \dots, O_{n',i}$, for $i = 1, 2, \dots, m'$. Then we have $L'' = p'_{\max}$, $p''_{\max} = L'$, where L'' and p''_{\max} denote the maximum load and maximum processing time in I'' . Running the algorithm in Lemma 6 on I'' will produce a schedule S'' of makespan at most

$$\begin{aligned} & O(\log(m'n')(\log \log(m'n'))^{1+\epsilon} \cdot (L'' + p''_{\max})) \\ &= O(\log(m'n')(\log \log(m'n'))^{1+\epsilon} \cdot (L' + p'_{\max})). \end{aligned}$$

Moreover, the operations $O_{1,i}, O_{2,i}, \dots, O_{n',i}$ are processed in this order. Again by reversing the roles of the machines and the jobs in S'' we obtain a schedule S' of I' compatible with $\pi = (1, 2, \dots, n')$ whose makepan equals to that of S'' . \square

Combining Lemmas 3|7|2(i) we may prove Theorem 1.

Proof of Theorem 1. Given an instance $I = (G, m, n, p)$ of $RO||C_{\max}$, we construct an instance $I' = (m', n', p')$ of $O||C_{\max}$ by Lemma 3. Using Lemma 7 on I' and $\pi = (1, 2, \dots, n')$ we obtain a permutation schedule of I' with makespan at most $O(\log(m'n')(\log \log(m'n'))^{1+\epsilon} \cdot (L' + p'_{\max}))$. Then by Lemma 3(iv), for any constant $\epsilon > 0$ we can find a permutation schedule in polynomial time whose makespan is at most

$$O(\log(m'n')(\log \log(m'n'))^{1+\epsilon} \cdot (L' + p'_{\max})) + T_\rho,$$

which is actually $O(\log m(\log \log m)^{1+\epsilon} \cdot (L + p_{\max} + T))$ by Lemma 3(i)(ii)(iii) and $T_\rho \leq \rho T$. By Lemma 2(i), the theorem follows. \square

4 Routing Flow Shop

The idea of exchanging the roles of the machines and the jobs used in the routing open shop can not be applied to the routing flow shop, since the schedule delivered may violate the constraints for flow shop model, i.e. the operations of the same job should be processed successively by M_1, M_2, \dots, M_m . To improve the previous bound for this problem we have to find some new approach. In this section we first aggregate the jobs and then use an algorithm for permutation flow shop to derive an $O(m^{2/3})$ -approximation algorithm for $RF||C_{\max}$.

Nagarajan and Sviridenko [20] showed

Lemma 8. *There exists a polynomial time algorithm that delivers a permutation schedule for any instance $I = (m, n, p)$ of $F||C_{\max}$ of makespan at most $O(\sqrt{\min\{m, n\}} \cdot (L + p_{\max}))$.*

Now we state the algorithm as follows. Given an instance I of $RF||C_{\max}$, the algorithm first constructs an instance I' of $F||C_{\max}$ in which the number m of machines is the same as I and the number n' of jobs is at most $2m^{2/3} + 1$. Using the algorithm proposed by Nagarajan and Sviridenko [20] for permutation flow shop we obtain a permutation schedule S' of I' compatible with some permutation π , which has ID-partition $\pi^1, \pi^2, \dots, \pi^k$. The makespan of S' is at most $O(\sqrt{\min\{m, n'\}})$ times the sum of two lower bounds of the optimal value of I' . And each of the two lower bounds can not exceed an $O(m^{1/3})$ -factor of the optimal value of I . Lastly, we transform S' into a permutation schedule S of I with the makespan increasing by at most $kT_\rho \leq n'T_\rho$, which is at most $\rho n'$ times the optimal value of I by Lemma 2(ii).

Given an instance $I = (G, m, n, p)$ of $RF||C_{\max}$, we construct similarly an instance $I' = (m', n', p')$ of $F||C_{\max}$ as in the last section after modifying conditions (i)(ii) as below.

- (i') $\sum_{j=j_{l-1}+1}^{j_l} p_j \geq m^{1/3} L$
- (ii') $\sum_{j=j_{l-1}+1}^{j_l} t_{j,j+1} \geq T_\rho / m^{2/3}$

Moreover, each job in I' should be processed successively by M_1, M_2, \dots, M_m , the same as in I . Then we obtain

Lemma 9. *There exists a polynomial time algorithm that transforms any instance $I = (G, m, n, p)$ of $RF||C_{\max}$ into an instance $I' = (m', n', p')$ of $F||C_{\max}$ such that: (i) $m' = m$, $n' \leq 2m^{2/3} + 1$; (ii) $p'_{\max} \leq m^{1/3}L + p_{\max} + m^{1/3}T_p$; (iii) $L' \leq L + T_p$; (iv) Given a permutation schedule S' of I' compatible with π , which has ID-partition $\pi^1, \pi^2, \dots, \pi^k$, a permutation schedule S of I with the makespan increasing by at most kT_p can be obtained in polynomial time.*

Combining Lemmas 8, 9, 2(ii) we may prove Theorem 2

Proof of Theorem 2. Given an instance $I = (G, m, n, p)$ of $RF||C_{\max}$, we construct an instance $I' = (m', n', p')$ of $F||C_{\max}$ by Lemma 9. Using Lemma 9 on I' we obtain a permutation schedule compatible with some permutation π , which has ID-partition $\pi^1, \pi^2, \dots, \pi^k$, with makespan at most $O(\sqrt{\min\{m', n'\}} \cdot (L' + p'_{\max}))$. Then by Lemma 9(iv) we can find in polynomial time a permutation schedule of I with makespan at most $O(\sqrt{\min\{m', n'\}} \cdot (L' + p'_{\max})) + kT_p$, which is actually $O(m^{2/3}(L + p_{\max} + T))$ by Lemma 9(i)(ii)(iii) and $k \leq n'$. Then by Lemma 2(ii) the theorem follows. \square

Remark. If we define the makespan as the largest completion time of the jobs, the results in Theorems 1 and 2 still hold. To obtain algorithms for these variants, we simply replace T_p by a path H_σ , which is generated by a σ -approximation algorithm for the metric shortest Hamiltonian path problem (metric SHPP for short), in the above algorithms. Metric SHPP seeks to find a shortest path that starts from vertex 0 and goes through all vertices of the underlying graph. Hoogeveen [14] gave a $3/2$ -approximation algorithm for metric SHPP. Then a similar analysis leads to Theorems 1 and 2 by using two lower bounds $L + H$ and p_{\max} , where H is the optimal value of metric SHPP.

References

1. Afrati, F., Cosmadakis, S., Papadimitriou, C.H., Papageorgiou, G., Papakostantinou, N.: The complexity of the traveling repairman problem. Informatique Théorique et Applications 20(1), 79–87 (1986)
2. Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187, 985–1032 (2008)
3. Allahverdi, A., Soroudi, H.M.: The significance of reducing setup times/setup costs. European Journal of Operational Research 187, 978–984 (2008)
4. Augustine, J.E., Seiden, S.: Linear time approximation schemes for vehicle scheduling problems. Theoretical Computer Science 324, 147–160 (2004)
5. Averbakh, I., Berman, O.: Routing two-machine flowshop problems on networks with special structure. Transportation Science 30, 303–314 (1996)
6. Averbakh, I., Berman, O.: A simple heuristic for m-machine flow-shop and its applications in routing-scheduling problems. Operations Research 47, 165–170 (1999)
7. Averbakh, I., Berman, O., Chernykh, I.D.: A 6/5-approximation algorithm for the two-machine routing open-shop problem on a 2-node network. European Journal of Operational Research 166, 3–24 (2005)

8. Averbakh, I., Berman, O., Chernykh, I.D.: The routing open-shop problem on a network: complexity and approximation. *European Journal of Operational Research* 173, 531–539 (2006)
9. Bhattacharya, B., Carmi, P., Hu, Y., Shi, Q.: Single Vehicle Scheduling Problems on Path/Tree/Cycle Networks with Release and Handling Times. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 800–811. Springer, Heidelberg (2008)
10. Chernykh, I., Dryuck, N., Kononov, A., Sevastyanov, S.: The Routing Open Shop Problem: New Approximation Algorithms. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 75–85. Springer, Heidelberg (2010)
11. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA (1976)
12. Czumaj, A., Scheideler, C.: A new algorithmic approach to the general Lovász local lemma with applications to scheduling and satisfiability problems. In: The Proceeding of the 32nd Symposium on Theory of Computing, pp. 38–47 (2000)
13. Gaur, D.R., Gupta, A., Krishnamurti, R.: A $\frac{5}{3}$ -approximation algorithm for scheduling vehicles on a path with release and handling times. *Information Processing Letters* 86, 87–91 (2003)
14. Hoogeveen, J.A.: Analysis of Christofide's heuristic: some paths are more difficult than cycles. *Operations Research Letters* 10, 291–295 (1991)
15. Karuno, Y., Nagamochi, H.: 2-Approximation algorithms for the multi-vehicle scheduling problem on a path with release and handling times. *Discrete Applied Mathematics* 129, 433–447 (2003)
16. Karuno, Y., Nagamochi, H.: An approximability result of the multi-vehicle scheduling problem on a path with release and handling times. *Theoretical Computer Science* 312, 267–280 (2004)
17. Karuno, Y., Nagamochi, H., Ibaraki, T.: Vehicle scheduling on a tree with release and handling time. *Annals of Operations Research* 69, 193–207 (1997)
18. Karuno, Y., Nagamochi, H., Ibaraki, T.: Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks. *Networks* 39, 203–209 (2002)
19. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: algorithms and complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P. (eds.) *Handbooks in Operations Research and Management Science. Logistics of Production and Inventory*, vol. 4, pp. 445–522. North-Holland, Amsterdam (1993)
20. Nagarajan, V., Sviridenko, M.: Tight Bounds for Permutation Flow Shop Scheduling. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 154–168. Springer, Heidelberg (2008)
21. Shmoys, D., Stein, C., Wein, J.: Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing* 23(3), 617–632 (1994)
22. Tsitsiklis, J.N.: Special cases of traveling salesman and repairman problems with time windows. *Networks* 22, 263–282 (1992)
23. Yu, W., Liu, Z.: Single-vehicle scheduling problems with release and service times on a line. *Networks* 57, 128–134 (2011)
24. Yu, W., Liu, Z., Wang, L., Fan, T.: Routing open shop and flow shop scheduling problems. *European Journal of Operational Research* 213, 24–36 (2011)

Contraction-Based Steiner Tree Approximations in Practice

Markus Chimani* and Matthias Woste

Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany
`{markus.chimani, matthias.woste}@uni-jena.de`

Abstract. In this experimental study we consider contraction-based Steiner tree approximations. This class contains the only approximation algorithms that guarantee a constant approximation ratio below 2 and still may be applicable in practice. Despite their vivid evolution in theory, these algorithms have, to our knowledge, never been thoroughly investigated in practice before, which is particularly interesting as most of these algorithms' approximation guarantees only hold when some (constant) parameter k tends to infinity, while the running time is exponentially dependent on this very k .

We investigate different implementation aspects and parameter choices which finally allow us to construct algorithms feasible for practical use. Then we compare these algorithms against each other and against state-of-the-art approaches.

1 Introduction

The *Steiner Tree Problem (STP)* is defined as follows: Let $G = (V, E)$ be an undirected graph, $c : E \rightarrow \mathbb{R}^+$ an edge cost function, and $R \subseteq V$ a subset of required nodes, called the *terminals*. The objective is to find a minimum cost tree in G that contains all terminals (but not necessarily all non-terminal nodes—*Steiner nodes*— $V \setminus R$).

The STP is NP-hard [9] and MAXSNP-hard [4] and is generally considered a fundamental combinatorial optimization problem. It has numerous direct applications ranging from VLSI over network design to computational biology, and serves as a basis for a vast number of extended or generalized problems, as price-collecting Steiner trees, Steiner networks, stochastic Steiner trees, and many more. Its pervasive applicability gave rise to a lot of research virtually from all algorithmic points of view, including heuristics, meta-heuristics, approximation algorithms, as well as exact algorithms based on fixed parameter tractability, integer linear programs (ILPs), and other branching or enumeration strategies. In nearly all of these fields, there exist experimental studies of the developed algorithms, and there is a well-established benchmark set—SteinLib [12]—for cross-comparisons. A notable exception, interestingly, is the field of strong approximation algorithms, on which we concentrate in this paper.

There exists a surprising variety of 2-approximations (in fact, $2 - 2/|R|$) for the STP, ranging from constructive combinatorial algorithms based on shortest paths, over local-search algorithms, to primal-dual schemes; see, e.g., [29, 30], [28], and [20, 31] for surveys, respectively. Such algorithms have been investigated practically before, and especially the latter two approaches are known to give very good bounds in practice.

* Markus Chimani was funded by a Carl-Zeiss-Foundation juniorprofessorship.

Nearly all approximations with strictly stronger approximation ratio are greedy algorithms of pure combinatorial nature and use a common concept, first introduced by Zelikovsky [32]: Start with a heuristic solution and iteratively pick a terminal subset $R' \subset R$ of maximal size k (k is a fixed constant); replace the edge connections between the nodes R' with a (probably optimal) Steiner tree w.r.t. R' , if beneficial. We summarize these algorithms as *contraction-based* approximation algorithms.

To our knowledge, none of the algorithms with an approximation ratio below 2 (or $2 - 2/|R|$) has ever been thoroughly investigated in practice. The single exception seems to be [1], where the oldest 11/6-approximation is considered in a special VLSI setting on some generated 20x20 grid graphs with 5 and 8 terminals. Yet, they do not discuss running times. Even multiple of the original, theory-focused publications state that a practical evaluation would be worthwhile [23][24].

Contribution. We consider the class of contraction-based approximations: We propose certain practical improvements and investigate, among others, the obtained solution quality, their practical running time, and their dependency on (as well as the feasible values for) k . The latter is particularly interesting, as the algorithms' approximation guarantees are only achieved for choices of constant k when $k \rightarrow \infty$, where the running time is exponentially dependent on k .

In Section 2 we give an overview over the historic development of STP approximation algorithms with a performance ratio strictly below 2. In Section 3 we present the general greedy contraction framework and its resulting algorithms. We propose algorithmic improvements and discuss their implementations in Section 4. Finally, a thorough investigation of the practical behavior of the considered algorithms and improvements is given in Section 5 alongside with comparing their performance to the state-of-the-art.

2 History

The first algorithm achieving a ratio below 2 was given by Zelikovsky [32], with an improved running time described in [33] later in the same year. We denote this algorithm, which guarantees an approximation ratio of $11/6 \approx 1.833$, by CONTRABS. In the view of contraction-based algorithms, as coarsely sketched above, this algorithm only considers $k = 3$. Generalizing this approach to consider arbitrary (fixed) k (and a slightly different contraction procedure), this ratio was improved to 1.746 in [3]. Thereby, as well as in all the subsequent algorithms, the obtained approximation ratio is given for the case that k tends to infinity—a concept that is of course not directly useful in practice. Furthermore, the algorithms' running times are not merely exponential in k with a constant basis (as, e.g., 2^k), but the bases are dependent on the instance size.

The subsequently presented algorithm CONTRREL [34] is a variation of CONTRABS (for general k), which achieves a ratio of 1.693 by modifying the evaluation function used to select subtrees. Using a slight variant of this algorithm as a preprocessing (using an independent $k' \rightarrow \infty$) before CONTRREL, a ratio of 1.644 was shown in [10]. Running this algorithm ℓ times, gives a ratio of 1.588 when $k, k', \ell \rightarrow \infty$ [7]. Finally, the algorithm CONTRLOSS (originally called *loss-contracting*) achieves an approximation ratio of 1.55 [23]. It goes back to the simpler and more practical scheme of

CONTRABS, proposing a new evaluation function based on evaluating the *loss* (originally introduced in [10]) caused by contractions. We note that for the practically more relevant choices $k = 3, 4$ (see below) all the above k -dependent algorithms tend to achieve ratios roughly around 1.9–1.8. Only two algorithms with ratio below 2 are not of the above contraction type: Prömel and Steger [22] presented an (originally parallel) randomized $O(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n^{11+\alpha} \log n)$ algorithm guaranteeing a $5/3 + \varepsilon$ approximation ratio using $\alpha < 2.376$. Its running time seems prohibitive for practical use. The currently best approximation ratio of ≈ 1.39 was achieved by Byrka et al. [5] via iterative randomized rounding of LP-relaxations. As the contraction-based algorithms, it considers k -restricted components and achieves the ratio only for $k \rightarrow \infty$. The fact that the algorithms needs to solve exponentially many ($O(k \cdot n^k)$) linear programs—instead of “only” enumerating that many subtrees as the above algorithms do—makes it inapplicable in practice.

3 Contraction Framework

Now, we will briefly summarize the considered approximation algorithms, by describing them within a common, general framework, along the lines of Zelikovsky’s *greedy contraction framework* [34]. Given the edge-weighted network $G = (V, E, c)$, let $G' = (V, E', d)$ denote its metric closure. Clearly, we have a 1-to-1 correspondence between subgraphs in G and G' by substituting edges in G' via the corresponding shortest paths in G . Hence, we will only use G' within the algorithms. Let $F \subset E'$ ($K \subset G'$), then $d(F)$ ($d(K)$) denotes the sum of the edge weights of the edge set F (component K , resp.). Furthermore, let G'_R denote the subgraph of G' induced by the nodes R .

The general idea of the considered algorithm is the following: We start with a minimum spanning tree $\text{MST}(G'_R)$ of G'_R . As shown by Kou et al. [13], transforming this tree into G' , and subsequently into G , gives (after removing cycles) a 2-approximation for the STP. The contraction framework now (greedily) investigates the best possibility to introduce Steiner nodes with degree > 2 into $\text{MST}(G'_R)$, always strictly improving the solution. To do so, we evaluate the improvement achievable by a *k -restricted full component* $K \subset G'$, i.e., a Steiner tree over a subset $R' \subset R$ with $3 \leq |R'| \leq k$ in which all terminals are leaves. Introducing such a component into our intermediate solution is done via the concept of *contraction*: to contract an edge e , we simply set its weight to 0 and denote the resulting graph by G/e . We may contract a component K , denoted by G'/K , by contracting the edges between its terminals.

Let $\text{save}(H, K) := d(\text{MST}(H)) - d(\text{MST}(H/K))$ denote the cost difference between the minimum spanning trees in the non-contracted graph H and the contracted graph H/K . We can define multiple different *win*-functions to decide on the best component to include: The function $\text{win}_{\text{abs}}(H, K) := \text{save}(H, K) - d(K)$ was introduced for CONTRABS, with k fixed to 3, and gives an approximation ratio of 11/6. The function $\text{win}_{\text{rel}}(H, K) := \frac{\text{save}(H, K)}{d(K)}$ was used for CONTRREL and leads to the ratio 1.693 for $k \rightarrow \infty$ which is worse than 11/6 for $k = 3, 4$. Finally, $\text{win}_{\text{loss}}(H, K) := \frac{\text{win}_{\text{abs}}(H, K)}{\text{loss}(K)}$ was introduced for CONTRLOSS giving an approximation ratio of 1.55 for

$k \rightarrow \infty$, but is again weaker than $11/6$ for $k = 3, 4$. This function uses the concept of $\text{loss}(K) := d(K')$ of a full component K , where $K' \subset K$ is a minimum cost subgraph of K connecting all Steiner nodes to terminals. The *loss contraction* then contracts all Steiner nodes of K into its terminals along the paths of K' .

4 Algorithm Engineering

In this section, we investigate different algorithmic variants in order to deal with the enormous number of evaluated full components. Only the last of these variants will waive its approximation ratio in favor of a faster computation.

Component Generation (`generate=[all, voronoi, ondemand]`). The traditional method to obtain a correct algorithm (and the only one known to be applicable for any k and win -function) is to enumerate all possible $O(|R|^k)$ components. We denote this approach by `generate=all`.

For $k = 3$ (components are stars) and win_{abs} , the use of Voronoi regions (around the terminals) was proposed in [32] to speed-up the identification of potential Steiner nodes as the center of the star: we only have to consider Steiner nodes lying in one of the terminal triple's Voronoi regions. We denote this variant by `generate=voronoi`.

In the subsequent paper [33]—again for $k = 3$ and win_{abs} —it was proposed to completely omit the explicit generation phase. Instead, in each pass, a star with maximum win_{abs} is constructed directly. We denote this option by `generate=ondemand`.

Component Reduction (`reduce=[on, off]`). The following conceptually very simple strategy has not been considered before, as it does not improve the asymptotic running time. As our experiments will reveal, however, it constitutes a strong building block to obtain practically reasonable running times. Considering the algorithmic steps, it is simple to observe the following property, which is also used in the proofs of [32].

Proposition 1. *Let H be any complete metric network, then for any two components K_1, K_2 , we have: $\text{save}(H, K_2) \geq \text{save}(H/K_1, K_2)$.*

Based thereon, we can observe that the win -function of a component K never increases during the multiple passes of the evaluation phase. Hence we can discard components with non-positive win -function from \mathcal{K} immediately and permanently as soon as we discover them. However, only win_{abs} and win_{loss} can in fact become negative. For win_{rel} , we can only remove a component as soon as its win -value becomes 0, which only happens when all the component's terminal nodes are already tree-wise connected via edges of cost 0. We denote this strategy by `reduce=on`.

Save Calculation (`save=[static, dynamic, enum]`). In each pass of the evaluation phase we have to evaluate the win -function for each component in \mathcal{K} . As this list is large, it is crucial to perform this operation as fast as possible. Observe that only the values $\text{save}(H, K) := d(\text{MST}(H)) - d(\text{MST}(H/K))$ change, dependent on the pass' current, already partially contracted, network H . As an invariant we will know $T := \text{MST}(H)$ at the beginning of each pass. We obtain a minimum spanning tree T' of H/K from T in two steps: first, we add a 0-cost tree ($\subset F$) connecting the terminals R' of K , giving an intermediate graph T'' . As T'' now contains cycles, we break them

by greedily removing expensive edges, called *save-edges*, whose cost sum constitutes $\text{save}(H, K)$. To quickly identify these edges, we can build a (binary) *weight tree* $W(T)$ of T , where inner vertices represent edges in T , and leaves represent the terminals R' . When T contains only a single terminal r , its weight tree is the node r . Otherwise, we define $W(T)$ recursively: The root represents the heaviest edge e in T separating two terminals; when removing e , the tree decomposes into two subtrees, whose respective weight trees form the left and right child of the root.

Zelikovsky already observed in [33] that finding the save-edges w.r.t. some contraction can be done by finding the lowest common ancestors (LCAs) of the terminals R' in $W(T)$. While it is long known that LCA computation is possible in $\langle O(n), O(1) \rangle$ time [6]—i.e., linear preprocessing time (resulting in some additional data structure D) and constant query time, where $n := |R|$ —these algorithms were usually considered too complex and cumbersome to implement in practice. In 2000, a conceptually simple and practical scheme was devised [2]. As a full implementation of the latter algorithm would induce too much overhead (see also Section 5), we implemented the $\langle O(n \log n), O(1) \rangle$ version of the latter publication, and denote this by `save=static`.

A downside of this approach is the need to rebuild $W(T)$ and D whenever a component gets contracted, i.e., after each pass. To overcome this, we generate the tree once at the beginning of the evaluation phase and show that we can efficiently update it dynamically from then on. We denote this algorithm variant by `save=dynamic`: Glossing over the details, it turns out that it suffices to traverse from the terminal leaves R' upwards, ending at the highest LCA. Thereby, we only re-attach the full subtrees that are not contained in these traversals in constant time per traversed node. Similarly, only a subrange of D would have to be reconstructed but the practical overhead of such an operation outweighs the (practically very fast) full construction of D' based on $W(T')$.

Our experiments reveal that this dynamic variant does not decrease the algorithms' overall running times significantly. Hence, instead of trying more and more complex variations, we simplified our algorithmic approach, denoted by `save=enum`: We pre-compute a triangular $|R| \times |R|$ matrix at the start of each pass, explicitly storing the most expensive edge separating two terminals. This can be done by recursion in $O(n^2)$. Interestingly, the first original paper [32] already suggested such an approach, which was eventually discarded in favor of the theoretically beneficial LCA approach in [33]. **Evaluation Passes** (`pass=[multi, one]`). The original algorithm (`pass=multi`) may require up to $O(|R|)$ many passes in the evaluation phase; in each phase, each component of the full list \mathcal{K} (if not pre-reduced) has to be evaluated. In contrast to this, we propose a faster scheme, denoted by `pass=one`, inspired by ideas of the algorithm of Berman and Ramaiyer [3] but without any approximation guarantee: After all components are generated, \mathcal{K} is sorted in decreasing order by the initial *win*-value of each component. Then, we make a single pass over \mathcal{K} in this order: we reassess the component's *win* and directly perform the contraction if this value is positive.

5 Experiments

In the following practical investigation, we use an Intel Xeon E5520, 2.27 GHz, 8 GB RAM, running Debian 6. The binaries are compiled in 32bit with `g++ 4.4.5` and the `-O1`

optimization flag. All algorithms are implemented as part of the free C++ Open Graph Drawing Framework (OGDF) [15] (which, due to its by now extended focus, will be renamed into *Open Graphalgorithms and Datastructures Framework* in its next release). We implemented the practically most promising CONTRABS, CONTRREL, and CONTRLOSS algorithm variants. We will first study the effect of our previously described implementation parameters. Using the best choices, we then compare the contraction-based approximations against each other, as well as against alternative approaches.

In the following, we consider the Steiner tree instances of the well-known and extensive *SteinLib* benchmark set [12]—we omit the sets ES1000FST and ES10000FST as they are too large for our algorithms. We evaluate our algorithms’ solution quality by computing a *gap* as $(a - u)/u$ (usually given in %, i.e., multiplied by 1000), where a is the algorithm’s solution value and u the optimal (or best known) solution value as given by SteinLib. Note that there was never the case that $a < u$. We apply a time limit of 1 hour per instance and algorithm; all running times are given in seconds.

Additionally to the classification of SteinLib, we also group the instances by other means: *Difficult* consists of all instances that can, according to SteinLib, not be solved to proven optimality within one hour; *Large* consists of all instances with > 16000 edges or > 8000 nodes, i.e., roughly the largest 10% of the instances; *NonOpt* consists of all instances which, according to SteinLib, are not solved to proven optimality yet.

Evaluation of Performance Improvements. We first investigate the influence of our implementation parameters, which do not influence the resulting solution values. While we conducted the following experiments for the full array of different parameters, we will only report on the main findings. They are usually consistent when changing the other parameters (e.g. the *win*-function), if not stated otherwise.

Consider the original CONTRABS algorithm ($k = 3$, win_{abs} , `generate=all`): With `reduce=off`, the size of \mathcal{K} would only decrease by 1 per evaluation pass. Using the reduction strategy, \mathcal{K} shrinks to 0.65% (779686 to 5065, in absolute numbers) on average, after the first pass. Therefore `reduce=on` offers large benefits while introducing no overhead and will hence always be set in the following. The statistics show further interesting details: Overall, only 0.0027% (21 in absolute numbers) of all the initially evaluated triples are actually contracted during an average algorithm run.

Figure 1(left) shows the running times for different choices of `generate`: clearly, the exhaustive scheme is the clear loser, while Voronoi regions give a huge benefit. Generating the next component on demand is often close to the Voronoi scheme (and on two sets even better). This finding remains consistent when considering our alternative instance groupings. Hence, for $k = 3$ with win_{abs} , we will always use Voronoi regions in the following; for all other settings we have to resort to `generate=all`.

The final option to improve the running time is the calculation of *save*, see Figure 1(right). While the dynamic update of the weight tree slightly improves the overall running time, its effect is not too strong. Surprisingly, the simple brute-force idea of computing a full matrix performs almost always best in practice—the overhead of the LCA preprocessing simply does not pay off. Again, this also holds for our alternative instance groupings. For our further experiments we hence always use `save=enum`.

Practical choices for k . Now, we have to investigate the possible choices for k . Herein, we do so for CONTRLOSS (`reduce=on`, `save=enum`, `generate=all`), but the

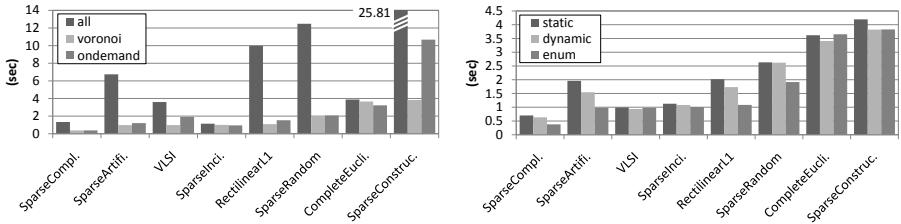


Fig. 1. Running time of CONTRABS (`reduce=on, pass=multi`), averaged over the instances solved by all considered configurations. (left) different generation schemes (`save=enum`). (right) different save calculation methods (`generate=voronoi`). We omit CompleteRandom, SparseEuclidian, and CrossGrid as their running times are almost 0.

results are analogous for CONTRREL. We say that the algorithms *fails*, if it does not finish within 1 hour or if it is terminated due to insufficient memory (32bit binaries). Large values of k will lead to more failures, as the number of considered components (all full components with up to k terminals) monotonously and drastically increases.

For $k = 3$, about 17% of all instances fail. This ratio increases to 47% for $k = 4$. Already for $k = 5$, we observe a failure rate of over 78%, i.e., only the very smallest instances can still be tackled. Overall, we have to conclude that $k \geq 5$ is not feasible.

Comparison of Approximation Algorithms. Having settled on the most effective parameters, we can study the practical performance of the considered contraction-based algorithms by taking their running time as well as their solution quality (relative gap to the optimal or best known solution) into account. This also allows us to study the practical influence of our heuristic `pass=one` strategy. For context, we compare traditional heuristics by Takahashi and Matsuyama (TM80) [26] and Mehlhorn (MEH88) [14]. Practical comparisons of further 2-approximations can be found, e.g., in [28].

Table II summarizes the results w.r.t. different CONTRABS and CONTRREL variants. Results lying on the Pareto front (i.e., variants where no faster algorithm gives better results) are marked in bold. It turns out that the CONTRREL configurations are almost always dominated. Consistent with findings in previous publications, MEH88 is very fast but results in high gaps, while TM80 offers a very good balance between those two measures. The 11/6-approximation CONTRABS often gives even better solution values, while still having reasonable running times. Furthermore, we observe that the single pass variant is usually not worth it: While `pass=one` lacks the quality of `multi`, it is not substantially faster. The first pass vastly dominates the running time of the subsequent passes, due to the strong performance of `reduce=on`.

The most recent CONTRLOSS algorithm performs worst among all variants (Table 2)—despite its best approximation factor of 1.55 for $k \rightarrow \infty$. While it is much slower, it also virtually always also gives worse solutions than CONTRABS. This is consistent with the fact that the ratio is only 1.935 (1.883) for $k = 3$ (4, respectively).

¹ Our implementation uses ideas of [16][18] but not all algorithmic finesse concerning the running time. Furthermore, it runs the basic algorithm $|R|$ times, once for each possible start node, to achieve qualitywise better results.

Table 1. Average running time and gap of different algorithm variants. For each class, we consider all the instances (“#” denotes their number) solved by all algorithms considered in this table. Bold entries indicate that they belong to the Pareto front.

pass=generate=	#	MEH88	TM80	CONTRABS, $k = 3$						CONTRREL, $k = 3$					
				one voronoi			multi voronoi			ondemand			one voronoi		
				time	gap	time	gap	time	gap	time	gap	time	gap	time	gap
SparseEuclidian	15	0.00	40	0.00	3	0.00	1	0.00	0	0.00	0	0.01	1	0.01	4
CompleteRand.	10	0.00	40	0.02	4	0.03	1	0.02	1	0.03	1	0.02	4	0.02	2
CrossGrid	44	0.00	89	0.03	22	0.05	13	0.05	11	0.07	10	0.06	22	0.06	22
SparseComplete	6	0.00	91	0.13	20	0.38	49	0.38	41	0.38	31	0.56	27	0.52	34
SparseArtificial	6	0.00	81	0.08	11	1.00	20	0.98	19	1.20	23	1.40	11	1.25	10
VLSI	150	0.00	59	0.48	14	0.97	13	0.98	4	1.91	4	0.99	14	1.01	12
SparseIncidence	400	0.00	279	0.84	146	0.97	88	1.01	75	0.95	76	0.99	82	1.04	81
RectilinearL1	221	0.00	41	0.11	11	1.11	13	1.08	6	1.53	6	2.02	12	1.53	11
SparseRandom	82	0.00	65	0.6	13	1.90	13	1.92	10	2.07	10	2.98	17	2.67	14
CompleteEuclid.	14	0.00	11	2.57	11	3.69	1	3.65	0	3.22	0	3.67	4	3.69	8
SparseConstruct.	46	0.00	353	0.81	64	3.74	124	3.83	106	10.68	108	5.11	89	5.83	110
Difficult	45	0.01	340	1.69	283	3.14	116	3.20	99	4.19	99	3.51	107	3.36	104
Large	101	0.02	299	4.75	242	7.18	105	7.35	90	10.63	91	7.89	105	8.23	96
NonOpt	54	0.00	396	4.78	145	7.79	137	8.17	121	13.73	121	9.03	112	9.97	123
(all)	994	0.00	158	0.56	69	1.18	48	1.19	39	1.73	39	1.54	44	1.47	44

Table 2. Average running time and gap of CONTRLOSS for $k = 3, 4$ compared to CONTRABS (generate=voronoi, reduce=on, pass=multi, save=enum). Bold values indicate elements of the Pareto front. For each graph class we average over the instances (“#” gives their number) solved by the respective CONTRLOSS (which are always also solved by CONTRABS).

	VLSI		Difficult		Large		NonOpt		<all>			
	#	time	#	time	gap	#	time	gap	#	time	gap	
CONTRABS $k = 3$	94	0.02	3	30	0.23	102	43	1.34	109	27	2.99	141
CONTRLOSS $k = 3$	73.42	6	480.45	107	529.44	110	573.67	142	861	120.50	42	
CONTRABS $k = 3$	50	0.00	2	2	0.02	88	7	0.19	77	5	8.53	192
CONTRLOSS $k = 4$	1105.58	4	2840.35	136	1167.71	108	1336.2	172	545	471.30	35	

Comparison to Exact Algorithms. As noted before, there are many practical evaluations of alternative heuristics and 2-approximations, see, e.g., [8, 30, 29]. Furthermore, the last decade has seen the light of practically strong exact algorithms [17, 11, 21, 27], despite the STP’s NP-hardness. An extensive comparison of our algorithms to all those would be beyond the scope of this paper, and, as it turns out, even superfluous: We consider the currently fastest exact algorithm by Polzin and Daneshmand PD03 [21], which is based on a branching scheme over a strong dual-ascent primal-dual heuristic (2-approximation). Unfortunately, the code is not freely available, but we can use the detailed data of Polzin’s PhD-thesis [19]. Their runs were performed as a single thread on a Sunfire 15000 (900 MHz SPARC III+, SunOS 5.9, g++ 2.95.3)², and their running times are hence clearly not directly comparable. Table 3 shows the average running times of PD03 and our fastest approximation CONTRABS, over the SteinLib’s VLSI instances that were solved within 1 hour by the latter algorithm. We observe that, while the approximation does never find the optimal solution, it is also only about 4 times faster: considering the SPEC benchmark [25], we see that our hardware is about 6x–10x

² It is noted in [19] that this machine gives roughly half the speed of an Athlon XP 1800+ (1.53 GHz) from the year 2003.

Table 3. Comparing CONTRABS (win_{abs} , $generate=voronoi$, $reduce=on$, $save=enum$, $pass=multi$) to the best known exact approach PD03. The hardware for the latter algorithm was substantially slower than our machine, cf. text.

		VLSI	ALUE	ALUT	DIW	DMXA	GAP	MSM	TAQ
#		113	13	8	21	14	13	30	14
CONTRABS	time gap (%)	0.18 4	0.43	0.48	0.20	0.02	0.22	0.05	0.18
PD03	time	(0.70)	(1.66)	(1.13)	(1.19)	(0.15)	(0.44)	(0.36)	(0.38)
	“speed-up”	3.9x	3.9x	2.4x	6.0x	7.5x	2.0x	7.2x	2.1x

faster. Hence we have to conclude that exact algorithms are not only already faster than current contraction-based approximations, but even applicable to larger graphs.

6 Conclusions and Thoughts

We considered the class of contraction-based approximation algorithms, implemented and tested some of its main and most promising representatives, and tuned them to the point that they can be somehow reasonably applied to practical problems. Most interestingly, the oldest and supposedly weakest contraction-based $11/6$ -approximation outperforms the newer algorithms, also mainly because the latter suffer from the fact that only small values for k are feasible in practice.

Generally, the sophisticated and theoretically advanced approximation algorithms seem to suffer from exactly the main ingredients in practice that made them possible in theory: the sheer number of considered full components (in particular when considering larger k) simply cannot be compensated for by clever algorithmic choices. It would be necessary to devise an approximation scheme—perhaps building on ideas of $generate=voronoi$ and strengthening them—which can filter the required components before actually enumerating them.

References

1. Alexander, M.J., Robins, G.: New performance-driven FPGA routing algorithms. *IEEE Trans. Computer-aided Design of Integrated Circuits and Systems* 15(2), 1505–1517 (1996)
2. 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)
3. Berman, P., Ramaiyer, V.: Improved approximations for the steiner tree problem. *Journal of Algorithms* 17(3), 381–408 (1994)
4. Bern, M., Plassmann, P.: The steiner problem with edge lengths 1 and 2. *Information Processing Letters* 32(4), 171–176 (1989)
5. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved LP-based approximation for steiner tree. In: *STOC 2010*, pp. 583–592. ACM (2010)
6. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing* 13(2), 338–355 (1984)
7. Hougardy, S., Prömel, H.J.: A 1.598 approximation algorithm for the steiner problem in graphs. In: *SODA 1999*, pp. 448–453. SIAM (1999)
8. Hwang, F.K., Richards, D.S.: Steiner tree problems. *Networks* 22(1), 55–89 (1992)
9. Karp, R.M.: Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, pp. 85–103. Plenum Press, New York (1972)

10. Karpinski, M., Zelikovsky, A.: New approximation algorithms for the steiner tree problems. *Journal of Combinatorial Optimization* 1, 47–65 (1997)
11. Koch, T., Martin, A.: Solving steiner tree problems in graphs to optimality. *Networks* 32, 207–232 (1998)
12. Koch, T., Martin, A., Voß, S.: SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37 (2000), <http://elib.zib.de/steinlib>
13. Kou, L., Markowsky, G., Berman, L.: A fast algorithm for steiner trees. *Acta Informatica* 15(2), 141–145 (1981)
14. Mehlhorn, K.: A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters* 27(3), 125–128 (1988)
15. Open Graph Drawing Framework (OGDF) v.2010.10, <http://www.ogdf.net>
16. Poggi de Aragão, M., Ribeiro, C.C., Uchoa, E., Werneck, R.F.: Hybrid local search for the Steiner problem in graphs. In: MIC 2001, pp. 429–433 (2001)
17. Poggi de Aragão, M., Uchoa, E., Werneck, R.F.: Dual heuristics on the exact solution of large steiner problems. *Elect. Notes in Discr. Math.* 7, 150–153 (2001)
18. Poggi de Aragão, M., Werneck, R.F.F.: On the Implementation of MST-Based Heuristics for the Steiner Problem in Graphs. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 259–272. Springer, Heidelberg (2002)
19. Polzin, T.: Algorithms for the Steiner Problem in networks. PhD thesis, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I (2003)
20. Polzin, T., Daneshmand, S.V.: Primal-dual Approaches to the Steiner Problem. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 214–225. Springer, Heidelberg (2000)
21. Polzin, T., Vahdati Daneshmand, S.: Improved algorithms for the steiner problem in networks. *Discrete Applied Mathematics* 112(1-3), 263–300 (2001)
22. Prömel, H., Steger, A.: RNC-Approximation Algorithms for the Steiner Problem. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 559–570. Springer, Heidelberg (1997)
23. Robins, G., Zelikovsky, A.: Improved steiner tree approximation in graphs. In: SODA 2000, pp. 770–779. SIAM (2000)
24. Robins, G., Zelikovsky, A.: Minimum steiner tree construction. In: The Handbook of Algorithms for VLSI Phys. Design Automation, pp. 487–508. CRC Press (2009)
25. Standard Performance Evaluation Corporation (SPEC), <http://www.spec.org/index.html> (last accessed April 15, 2011)
26. Takahashi, H., Matsuyama, A.: An approximate solution for the steiner problem in graphs. *Math. Japonica* 24(6), 573–577 (1980)
27. Uchoa, E., Poggi de Aragão, M., Ribeiro, C.C.: Preprocessing steiner problems from VLSI layout. *Networks* 40(1), 38–50 (2002)
28. Uchoa, E., Werneck, R.F.: Fast local search for steiner trees in graphs. In: ALENEX 2010, pp. 1–10. SIAM (2010)
29. Voß, S.: Steiner's problem in graphs: heuristic methods. *Discrete Applied Mathematics* 40(1), 45–72 (1992)
30. Winter, P., MacGregor Smith, J.: Path-distance heuristics for the steiner problem in undirected networks. *Algorithmica* 7, 309–327 (1992)
31. Wong, R.: A dual ascent approach for steiner tree problems on a directed graph. *Mathematical Programming* 28, 271–287 (1984)
32. Zelikovsky, A.: An 11/6-approximation algorithm for the network steiner problem. *Algorithmica* 9(5), 463–470 (1993)
33. Zelikovsky, A.: A faster approximation algorithm for the steiner tree problem in graphs. *Information Processing Letters* 46(2), 79–83 (1993)
34. Zelikovsky, A.: Better approximation bounds for the network and euclidean steiner tree problems. Technical report (1996)

Covering and Piercing Disks with Two Centers

Hee-Kap Ahn¹, Sang-Sub Kim¹, Christian Knauer²,
Lena Schlipf³, Chan-Su Shin⁴, and Antoine Vigneron⁵

¹ Department of Computer Science and Engineering, POSTECH, Pohang, Korea
`{heekap,helmet1981}@postech.ac.kr`

² Institute of Computer Science, Universität Bayreuth, 95440 Bayreuth, Germany
`christian.knauer@uni-bayreuth.de`

³ Institute of Computer Science, Freie Universität Berlin, Germany
`schlipf@mi.fu-berlin.de`

⁴ Department of Digital and Information Engineering,
Hankuk University of Foreign Studies, Yongin, Korea
`cssin@hufs.ac.kr`

⁵ Geometric Modeling and Scientific Visualization Center,
KAUST, Thuwal, Saudi Arabia
`antoine.vigneron@kaust.edu.sa`

Abstract. We consider new versions of the two-center problem where the input consists of a set \mathcal{D} of disks in the plane. We first study the problem of finding two smallest congruent disks such that each disk in \mathcal{D} intersects one of these two disks. Then we study the problem of covering the set \mathcal{D} by two smallest congruent disks. We give exact and approximation algorithms for these versions.

1 Introduction

The standard *two-center problem* is a well known and extensively studied problem: Given a set P of n points in the plane, find two smallest congruent disks that cover all points in P . The best known deterministic algorithm runs in $O(n \log^2 n \log^2 \log n)$ [3] and there is a randomized algorithm with expected running time $O(n \log^2 n)$ [8]. There has also been a fair amount of work on several variations of the two-center problem: for instance, the two-center problem for weighted points [7], and for a convex polygon [14].

In this paper we consider new versions of the problem where the input consists of a set \mathcal{D} of n disks (instead of points): In the *intersection problem* we want to compute two smallest congruent disks C_1 and C_2 such that each disk in \mathcal{D} intersects C_1 or C_2 , while in the *covering problem*, all disks in \mathcal{D} have to be contained in the union of C_1 and C_2 . To the best of our knowledge these problems have not been considered so far. However, linear-time algorithms are known for both the covering and the intersection problem with only one disk [9,11,12].

Our results. In order to solve the intersection problem, we first consider the two-piercing problem: Given a set of disks, decide whether there exist two points

such that each disk contains at least one of the points. We show that this problem can be solved in $O(n^2 \log^2 n)$ expected time and $O(n^2 \log^2 \log \log n)$ deterministic time. Using these algorithms we can solve the intersection problem in $O(n^2 \log^3 n)$ expected time and $O(n^2 \log^4 n \log \log n)$ deterministic time.

For the covering problem we consider two cases: In the *restricted case* each $D \in \mathcal{D}$ has to be fully covered by one of the disks C_1 or C_2 . In the *general case* a disk $D \in \mathcal{D}$ can be covered by the union of C_1 and C_2 . We show how the algorithms for the intersection problem can be used to solve the restricted covering case and present an exact algorithm for the general case. We complement these results by giving efficient approximation algorithms for both cases. All the missing proofs and details will be found in the full paper version.

All the results presented in this paper are summarized in the following table.

	Exact algorithm	$(1 + \epsilon)$ -approximation
Intersection problem	$O(n^2 \log^4 n \log \log n)$ $O(n^2 \log^3 n)$ expected time	—
General covering problem	$O(n^3 \log^4 n)$	$O(n + 1/\epsilon^3)$
Restricted covering problem	$O(n^2 \log^4 n \log \log n)$ $O(n^2 \log^3 n)$ expected time	$O(n + (1/\epsilon^3) \log 1/\epsilon)$

Notation. The radius of a disk D is denoted by $r(D)$ and its center by $c(D)$. The circle that forms the boundary of D is denoted by ∂D .

2 Intersecting Disks with Two Centers

In this section we consider the following intersection problem: Given a set of disks $\mathcal{D} = \{D_1, \dots, D_n\}$, we want to find two smallest congruent disks C_1 and C_2 such that each disk $D \in \mathcal{D}$ has a nonempty intersection with C_1 or C_2 .

Based on the observation below, there is an $O(n^3)$ algorithm for this problem.

Observation 1. Let C_1, C_2 be an optimal solution. Let ℓ be the bisector of the segment connecting the centers of C_1 and C_2 . Then, $C_i \cap D \neq \emptyset$ for every $D \in \mathcal{D}$ whose center lies on the same side of ℓ as the center of C_i , for $i = \{1, 2\}$.

A simple approach would be, for every bipartition of the centers of the disks in \mathcal{D} by a line ℓ , to compute the smallest disk intersecting the disks on each side of ℓ , and return the best result over all bipartitions. Since there are $O(n^2)$ such partitions, and the smallest disk intersecting a set of disks can be found in linear time [11], this algorithm runs in $O(n^3)$ time.

We will present faster algorithms for the intersection problem. We first introduce a related problem. For a real number δ and a disk D , the δ -inflated disk $D(\delta)$ is a disk concentric to D and whose radius is $r(D) + \delta$. Consider the following decision problem:

Given a value δ , are there two points p_1 and p_2 such that $D(\delta) \cap \{p_1, p_2\} \neq \emptyset$ for every $D \in \mathcal{D}$?

Let δ^* be the minimum value for which the answer to the decision problem is “yes”.

To ensure that the radii of δ -inflated disks are nonnegative, we require that $\delta \geq -r_{\min}$, where r_{\min} is the smallest radius of the disks in \mathcal{D} . Without loss of generality, we assume that no disk in \mathcal{D} contains another disk in \mathcal{D} .

2.1 Decision Algorithm

Given a value δ , we construct the arrangement of the δ -inflated disks $D_i(\delta)$, $i = 1 \dots n$ in the plane. This arrangement consists of $O(n^2)$ cells, each cell being a 0, 1, or 2-face. We traverse all the cells in the arrangement in a depth-first manner and do the followings: We place one center point, say p_1 , in a cell. The algorithm returns “yes” if all the disks that do not contain p_1 have a nonempty common intersection. Otherwise, we move p_1 to a neighboring cell, and repeat the test until we visit every cell. This naive approach leads to a running time $O(n^3)$: we traverse $O(n^2)$ cells, and each cell can be handled in linear time.

The following approach allows us to improve this running time by almost a linear factor. We consider a traversal of the arrangement of the δ -inflated disks by a path γ that crosses only $O(n^2)$ cells, that is, some cells may be crossed several times, but on average each cell is crossed $O(1)$ times. It can be achieved by choosing the path γ to be the Eulerian tour of the depth-first search tree from the naive approach.

While we move the center p_1 along γ and traverse the arrangement, we want to know whether the set of disks \mathcal{D}' that do not contain p_1 have a non-empty intersection. To do this efficiently, we use a segment tree [6]. Each disk of \mathcal{D} may appear or disappear several times during the traversal of γ : each time we cross the boundary of a cell, one disk is inserted or deleted from \mathcal{D}' . So each disk appears in \mathcal{D}' along one or several segments of γ . We store these segments in a segment tree. As there are only $O(n^2)$ crossings with cell boundaries along γ , this segment tree is built over a total of $O(n^2)$ endpoints and thus has total size $O(n^2 \log n)$: Each segment of γ along which a given disk of \mathcal{D} is in \mathcal{D}' is inserted in $O(\log n)$ nodes of the segment tree. Each node v of the segment tree stores a set $\mathcal{D}_v \subseteq \mathcal{D}$ of input disks; from the discussion above, they represent disks that do not contain p_1 during the whole segment of γ that is represented by v . In addition, we store at node v the intersection $I_v = \bigcap \mathcal{D}_v$ of the disks stored at v . Each such intersection I_v is a convex set bounded by $O(n)$ circle arcs, so we store them as an array of circular arcs sorted along the boundary of I_v . In total it takes $O(n^2 \log^2 n)$ time to compute the intersections I_v for all nodes v in the segment tree, since each disk is stored at $O(n \log n)$ nodes on average and the intersection of k disks can be computed in $O(k \log k)$ time.

We now need to decide whether at some point, when p_1 moves along γ , the intersection of the disks in \mathcal{D}' (that is, disks that do not contain p_1) is nonempty. To do this, we consider each leaf of the segment tree separately. At each leaf, we test whether the intersection of the disks stored at this leaf and all its ancestors is non-empty. So it reduces to emptiness testing for a collection of $O(\log n)$ circular polygons with $O(n)$ circle arcs each. We can solve this in $O(\log^2 n)$

expected time by randomized convex programming [5][13], using $O(\log n)$ of the following primitive operations:

1. Given I_i, I_j and vector $a \in \mathbb{R}^2$, find the extreme point $v \in I_i \cap I_j$ that minimizes $a \cdot v$.
2. Given I_i and a point p , decide whether $p \in I_i$.

We can also solve this problem in $O(\log^2 n \log \log n)$ time using deterministic convex programming [4]. So we obtain the following result:

Lemma 1. *Given a value δ , we can decide in $O(n^2 \log^2 n)$ expected time or in $O(n^2 \log^2 n \log \log n)$ worst-case time whether there exist two points such that every δ -inflated disk intersects at least one of them.*

2.2 Optimization Algorithm

The following lemma shows that the optimum δ^* can be found in a set of $O(n^3)$ possible values.

Lemma 2. *When $\delta = \delta^*$, either p_1 or p_2 is a common boundary point of three δ^* -inflated disks, a tangent point of two δ^* -inflated disks or a δ^* -inflated disk with radius zero.*

We can use an implicit binary search approach to find δ^* , which leads to an expected running time of $O(n^2 \log^3 n)$. To get a deterministic algorithm, we use the parametric search technique, with the deterministic decision algorithm of Lemma 1. As the generic algorithm, we use an algorithm that computes in $O(\log n)$ time the arrangement of the inflated disks using $O(n^2)$ processors [2], so we need to run the decision algorithm $O(\log^2 n)$ times, and the total running time becomes $O(n^2 \log^4 n \log \log n)$.

Theorem 1. *Given a set \mathcal{D} of n disks in the plane, we can compute two smallest congruent disks whose union intersects every disk in \mathcal{D} in $O(n^2 \log^3 n)$ expected time, and in $O(n^2 \log^4 n \log \log n)$ deterministic time.*

3 Covering Disks with Two Centers

In this section we consider the following covering problem: Given a set of disks $\mathcal{D} = \{D_1, \dots, D_n\}$, compute two smallest congruent disks C_1 and C_2 such that each disk $D \in \mathcal{D}$ is covered by C_1 or C_2 . In the *general case*, a disk $D \in \mathcal{D}$ must be covered by $C_1 \cup C_2$. In the *restricted case*, each disk $D \in \mathcal{D}$ has to be fully covered by C_1 or by C_2 .

3.1 The General Case

We first give a characterization of the optimal covering. The optimal covering of a set \mathcal{D}' of disks by one disk is determined by at most three disks of \mathcal{D}' touching

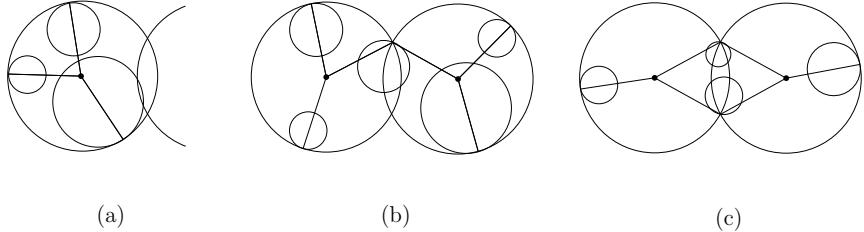


Fig. 1. The three configurations for the optimal 2-center covering of disks

the optimal covering disk such that the convex hull of the contact points contains the center of the covering disk. (See Figure 1(a).)

When covering by two disks, a similar argument applies, and thus the optimal covering disks (C_1^*, C_2^*) are determined by at most five input disks.

Lemma 3. *The optimal covering by two disks C_1^*, C_2^* satisfies one of the following conditions.*

1. *For some $i \in \{1, 2\}$, the disk C_i^* is the optimal one-disk covering of the disks contained in C_i^* , as in Figure 1(a).*
2. *There is an input disk that is neither fully contained in C_1^* nor in C_2^* , but contains one point of $\partial C_1^* \cap \partial C_2^*$ as in Figure 1(b).*
3. *There are two input disks D_i, D_j , possible $i = j$, each of them being neither fully covered by C_1^* nor C_2^* , such that D_i contains one of the points of $\partial C_1^* \cap \partial C_2^*$ and D_j contains the other point of $\partial C_1^* \cap \partial C_2^*$, as in Figure 1(c).*

In all cases, each covering disk C is determined by at most three touching disks, and the contact points contain the center $c(C)$ in their convex hull.

Based on this characterization, we obtained an $O(n^5)$ -time algorithm. Using a decision algorithm and the parametric search technique, the running time can be improved substantially: Let r^* be the radius of an optimal solution for the general case of covering by two disks. We describe a decision algorithm based on the following lemma that, for a given $r > 0$, returns “yes” if $r \geq r^*$, and “no” otherwise.

Lemma 4. *Assume that $r \geq r^*$. There exists a pair of congruent disks C_1, C_2 of radius r such that their union contains the input disks, an input disk D touches C_1 from inside, and one of the following property holds.*

- (a) *C_1 covers only D .*
- (b) *There is another input disk touching C_1 from inside.*
- (c) *There is another input disk D_i such that D_i is not contained in C_2 , but it touches a common intersection t of ∂C_1 and ∂C_2 that is at distance $2r$ from the touching point of D . If this is the case, we say that D and t are aligned with respect to C_1 .*
- (d) *There are two disks D_i and D_j , possibly $i = j$, such that D_i touches a common intersection of ∂C_1 and ∂C_2 , and D_j touches the other common intersection of ∂C_1 and ∂C_2 .*

Decision Algorithm. The cases are enumerated as in Lemma 4.

Case (a). Choose an input disk D . C_1 has radius r and covers only D . Then C_2 is the smallest disk containing $\mathcal{D} \setminus D$. If the radius of C_2 is $\leq r$, we return “yes”.

Case (b). We simply choose a pair of input disks D and D' . There are two candidates for C_1 , as C_1 has radius r and touches D and D' . So we consider separately each of the two candidate for C_1 . Then C_2 is chosen to be the smallest disk containing the input disks, or the portions of input disks (crescents) that are not covered by C_1 ; we can compute it in $O(n)$ time. If for one of the two choices of C_1 , the corresponding disk C_2 has radius $\leq r$, we return “yes”.

Case (c). For each input disk D , we do the following.

1. For the circle A with center $c(D)$ and radius $2r - r(D)$, compute $A \cap D'$ for every other disk D' . Let t be such an intersection point.
2. For each t ,
 - (a) remove (part of) the input disks covered by the covering disk determined by D and t , and compute the smallest disk covering the remaining input.
 - (b) If this algorithm returns a covering disk with radius $\leq r$, return “yes”.

Case (d). For each input disk D that touches C_1 from inside, we do the following. Let i be the index of the first input disk that the circular arc of C_1 from the touching point hits in clockwise orientation. Let j be the index of the last input disk that the circular arc leaves. We claim that the number of pairs of type (i, j) is $O(n)$.

This claim can be easily proved by observing that, while we rotate a covering disk C around an input disk D in clockwise orientation, C sweeps the plane and the input disks in such a manner that the first input disk intersected by the arc of C from the tangent point in clockwise orientation changes only $O(n)$ times; To see this, consider the union of the input disks. The union has $O(n)$ complexity. The last input disk intersected also changes $O(n)$ times. So the pairing along the rotation can be done by scanning two lists (first and last) of disks. For each pair (i, j) , we still have some freedom of rotating C_1 around D within some interval (C_2 changes accordingly.) During the rotation, an input disk not covered by the union of C_1 and C_2 may become fully covered by the union, or vice versa. We call such an event an *I/O event*. Note that an I/O event occurs only when an input disk touches C_2 from inside. Again, we claim that the number of I/O events for each pair (i, j) is $O(n)$.

For this claim, consider a pair (i, j) . During the rotation, the first intersection point moves along the boundary of disk D_i and the last intersection point moves along the boundary of disk D_j . Therefore, the movement of C_2 is determined by these two intersection points. Clearly C_1 has at most n I/O events. For C_2 , the trajectory of its center is a function graph which “behaves well” – Since it is a function on the radii of disk D_i and disk D_j , and their center locations, it is not in a complicated form (and its degree is low enough) that there are only $O(n)$ events.

We compute all I/O events and sort them. At the beginning of the rotation of C_1 around D , we compute the number of input disks that are not fully covered,

and we set the variable *counter* to this number. Then we handle I/O events one by one and update the counter. If the counter becomes 0, we return “yes”.

Lemma 5. *Given a value $r > 0$, we can decide in $O(n^3 \log n)$ time whether there exists two disks with radius r that cover a set of given disks in the plane.*

For the optimization algorithm we use parametric search.

Theorem 2. *Given a set of n disks in the plane, we can find a pair of congruent disks with smallest radius whose union covers all of them in $O(n^3 \log^4 n)$ time.*

Constant Factor Approximation. We apply the well known greedy k -center approximation algorithm by Gonzalez [10] to our general covering case. It works as follows: First pick an arbitrary point c_1 in the union $\bigcup \mathcal{D}$ of our input disks. For instance, we could choose c_1 to be the center of D_1 . Then compute a point $c_2 \in \bigcup \mathcal{D}$ that is farthest from c_1 . It can be done in linear time by brute force. These two points are the centers of our two covering disks, and we choose their radius to be as small as possible, that is, the radius of the two covering disks is the maximum distance from any point in $\bigcup \mathcal{D}$ to its closest point in $\{c_1, c_2\}$. This algorithm is a 2-approximation algorithm, so we obtain the following result:

Theorem 3. *We can compute in $O(n)$ time a 2-approximation for the general covering problem for a set \mathcal{D} of n disks.*

($1 + \epsilon$)-Approximation. Our $(1 + \epsilon)$ -approximation algorithm is an adaptation of an algorithm by Agarwal and Procopiuc [1]. We start by computing a 2-approximation for the general covering case in $O(n)$ time using our algorithm from Theorem 3. Let C_1, C_2 be the disks computed by this approximation algorithm and let r be their radius. We consider a grid of size $\delta = \lambda \epsilon r$ over the plane, where λ is a small enough constant. That is, we consider the points with coordinates $(i\delta, j\delta)$ for some integers i, j . Observe that there are only $O(1/\epsilon^2)$ grid points in $C_1 \cup C_2$. The center of each disk D is moved to a nearby grid point. That is, a center (x, y) is replaced by $(\delta \lceil x/\delta \rceil, \delta \lceil y/\delta \rceil)$. If two or more centers are moved to the same grid point, we only keep the disk with the largest radius. All the centers are now grid points inside $C_1 \cup C_2$, or at distance at most $\sqrt{2}\delta$ from the boundary of this union, so we are left with a set of $O(1/\epsilon^2)$ disks. We now replace this new set of disks by grid points: each disk is replaced by the grid points which are closest to the boundary of this disk and lie inside this disk. In order to compute these points we consider each column of the grid separately: The intersection of each disk with this column is an interval, and we replace the interval by the lowest and the highest grid point lying inside this interval. Since the set of disks has size $O(1/\epsilon^2)$ and the number of columns is $O(1/\epsilon)$, it takes in total $O(1/\epsilon^3)$ time. The set of grid points we obtain is denoted by P_g and its size is $O(1/\epsilon^2)$. We compute two smallest disks E_1, E_2 that cover P_g in $O(\frac{1}{\epsilon^2} \log^2 \frac{1}{\epsilon} \log^2 \log \frac{1}{\epsilon})$ time using the algorithm from Chan [3]. Choosing the constant λ small enough and increasing the radii of E_1, E_2 by $2\sqrt{2}\delta$, these disks are a $(1 + \epsilon)$ -approximation of the solution to our general disk cover problem.

Theorem 4. *Given a set \mathcal{D} of n disks in the plane, a $(1 + \epsilon)$ -approximation for \mathcal{D} in the general covering case can be computed in $O(n + 1/\epsilon^3)$ time.*

3.2 The Restricted Case

Observation 1 can be adapted to the restricted covering case.

Observation 2. *Let ℓ be the bisector of an optimal solution C_1 and C_2 . Then, $D \subset C_i$ for every $D \in \mathcal{D}$ whose center lies in the same side of ℓ as the center of C_i , for $i = \{1, 2\}$.*

Hence, the restricted covering problem can be solved in $O(n^3)$ time, since for a set of n disks \mathcal{D} the smallest disk covering all $D \in \mathcal{D}$ can be computed in $O(n)$ time [12] and there are $O(n^2)$ different bipartitions of the centers of the disks.

The algorithm from Section 2 can also be adapted to solve the restricted covering problem. We consider the decision problem, which can be formulated as follows: Given a set of n disks \mathcal{D} and a value δ , we want to decide whether there exists two disks C_1, C_2 with radius δ , such that each disk $D_i \in \mathcal{D}$ is covered by either C_1 or C_2 . This implies that for each disk $D_j \in \mathcal{D}$ covered by C_i holds: $d(c(D_j), c(C_i)) + r(D_j) \leq \delta$, for $i = \{1, 2\}$. Let r_{\max} be the maximum of radii of all disks in \mathcal{D} . It holds that $\delta \geq r_{\max}$, since if $\delta < r_{\max}$ there clearly exists no two disks with radius δ which cover \mathcal{D} . We can formulate the problem in a different way.

Given a value δ , do there exist two points, p_1 and p_2 , in the plane such that $D^*(\delta) \cap \{p_1, p_2\} \neq \emptyset$ for every $D \in \mathcal{D}$, where $D^*(\delta)$ is a disk concentric to D and whose radius is $\delta - r(D) \geq 0$.

Recall the definition of δ -inflated disks from Section 2. Every disk $D \in \mathcal{D}$ was replaced by a disk concentric to D and whose radius was $r(D) + \delta$. Here we actually need to replace each disk D by a disk that is concentric to D and has a radius $\delta - r(D)$. Since we know that $\delta \geq r_{\max}$, we add an initialization step, in which every disk D is replaced by a disk concentric to D and whose radius is $r_{\max} - r(D)$. Then we can use exactly the same algorithm than in Section 2 in order to compute a solution for the restricted covering problem. Let δ^* be the solution value computed by this algorithm. Clearly the solution for the covering problem is then $\delta^* + r_{\max}$. We summarize this result in the following theorem.

Theorem 5. *Given a set \mathcal{D} of n disks in the plane, we can compute two smallest congruent disks such that each disk in \mathcal{D} is covered by one of the disks in $O(n^2 \log^3 n)$ expected time or in $O(n^2 \log^4 n \log \log n)$ worst-case time.*

Constant Factor Approximation. Let C_1, C_2 denote an optimal solution to the general case, and let r_g be their radius. Then any solution to the restricted case is also a solution to the general case, so we have r_g is at most the radius of the optimal solution to the restricted case. On the other hand, the inflated disks $C_1(2r_g), C_2(2r_g)$ form a solution to the restricted case, because any disk

contained in $C_1 \cup C_2$ should be contained in either $C_1(2r_g)$ or $C_2(2r_g)$. So we obtain a 6-approximation algorithm for the restricted case by first applying our 2-approximation algorithm for the general case (Theorem 3) and then multiplying by 3 the radius of the two output disks:

Theorem 6. *Given a set \mathcal{D} of n disks in the plane, we can compute in $O(n)$ time a 6-approximation to the restricted covering problem.*

As in the general case, we will see below how to improve it to a linear time algorithm for any constant approximation factor larger than 1.

($1 + \epsilon$)-Approximation. Recall Observation 2. Let ℓ be the bisector of an optimal solution. Then each disk $D \in \mathcal{D}$ is covered by the disk C_i whose center lies in the same side of the center of C_i , $i \in \{1, 2\}$. Hence, if we know the bisector, we know the bipartition of the disks. First, we show how to compute an optimal solution in $O(n \log n)$ time if the direction of the bisector is known. Later on we explain how this algorithm is used in order to obtain a $(1 + \epsilon)$ approximation.

Fixed Orientation. W.l.o.g, assume that the bisector is vertical. After sorting the centers of all $D \in \mathcal{D}$ by their x -values, we sweep a vertical line ℓ from left to right, and maintain two sets \mathcal{D}_1 and \mathcal{D}_2 : \mathcal{D}_1 contains all disks whose centers lie to the left of ℓ and $\mathcal{D}_2 = \mathcal{D} \setminus \mathcal{D}_1$. Let C_1 be the smallest disk covering \mathcal{D}_1 and C_2 the smallest disk covering \mathcal{D}_2 . While sweeping ℓ from left to right, the radius of C_1 is nondecreasing and the radius of C_2 nonincreasing and we want to compute $\min \max(r(C_1), r(C_2))$. Hence, we can perform a binary search on the list of the centers of the disks in \mathcal{D} . Each step takes $O(n)$ time, thus we achieve a total running time of $O(n \log n)$.

Sampling. We use $2\pi/\epsilon$ sample orientations chosen regularly over 2π , and compute for each orientation the solution in $O(n \log n)$ time. The approximation factor can be proven by showing that there is a sample orientation that makes angle at most ϵ with the optimal bisector. The solution for this line is at most $(1 + \epsilon)$ times the optimal solution.

Theorem 7. *For a given a set \mathcal{D} of n disks in the plane, a $(1 + \epsilon)$ approximation for the restricted covering problem for \mathcal{D} can be computed in $O((n/\epsilon) \log n)$ time.*

The running time can be improved to $O(n + 1/\epsilon^3 \log 1/\epsilon)$ in the following way. We start with computing a 6-approximation in $O(n)$ time, using Theorem 6. Let C'_1 and C'_2 be the resulting disks, and let r' be their radius. As in the proof of Theorem 4, we round the centers of all input disks $D \in \mathcal{D}$ to grid points inside $C'_1 \cup C'_2$, with a grid size $\delta' = \lambda' \epsilon r'$, for some small enough constant λ' . Then we apply our FPTAS from Theorem 2 to this set of rounded disks and inflate the resulting disks by a factor of $\sqrt{2}\delta$. These disks are a $(1 + \epsilon)$ -approximation for the optimal solution. As there are only $O(1/\epsilon^2)$ rounded disks, it is done in $O((1/\epsilon^3) \log 1/\epsilon)$ time.

Theorem 8. *For a given a set \mathcal{D} of n disks in the plane, a $(1 + \epsilon)$ approximation for the restricted covering problem for \mathcal{D} can be computed in $O(n + (1/\epsilon^3) \log 1/\epsilon)$ time.*

Acknowledgments. Work by Ahn was supported by the National Research Foundation of Korea Grant funded by the Korean Government (MEST) (NRF-2010-0009857). Work by Schlipf was supported by the German Science Foundation (DFG) within the research training group 'Methods for Discrete Structures' (GRK 1408). Work by Shin was supported by the National Research Foundation of Korea Grant funded by the Korean Government (MEST) (NRF-2011-0002827).

References

1. Agarwal, P., Procopiuc, C.: Exact and approximation algorithms for clustering. *Algorithmica* 33(2), 201–226 (2002)
2. Agarwal, P.K., Sharir, M., Toledo, S.: Applications of parametric searching in geometric optimization. *J. Algorithms* 17, 292–318 (1994)
3. Chan, T.M.: More planar two-center algorithms. *Comput. Geom. Theory Appl.* 13, 189–198 (1997)
4. Chan, T.M.: Deterministic algorithms for 2-d convex programming and 3-d online linear programming. *J. Algorithms* 27(1), 147–166 (1998)
5. Clarkson, K.L.: Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM* 42, 488–499 (1995)
6. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008)
7. Drezner, Z.: The planar two-center and two-median problems. *Transportation Science* 18, 351–361 (1984)
8. Eppstein, D.: Faster construction of planar two-centers. In: Proc. of SODA 1997, pp. 131–138 (1997)
9. Fischer, K., Gartner, B.: The smallest enclosing ball of balls: combinatorial structure and algorithms. In: Proc. of SoCG 2003, pp. 292–301 (2003)
10. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.* 38, 293–306 (1985)
11. Löffler, M., van Kreveld, M.: Largest bounding box, smallest diameter, and related problems on imprecise points. *Comput. Geom. Theory Appl.* 43, 419–433 (2010)
12. Megiddo, N.: On the ball spanned by balls. *Discr. Comput. Geom.* 4, 605–610 (1989)
13. Sharir, M., Welzl, E.: A Combinatorial Bound for Linear Programming and Related Problems. In: Finkel, A., Jantzen, M. (eds.) STACS 1992. LNCS, vol. 577, pp. 567–579. Springer, Heidelberg (1992)
14. Shin, C.-S., Kim, J.-H., Kim, S.K., Chwa, K.-Y.: Two-center Problems for a Convex Polygon. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 199–210. Springer, Heidelberg (1998)

Generating Realistic Roofs over a Rectilinear Polygon*

Hee-Kap Ahn¹, Sang Won Bae², Christian Knauer³, Mira Lee⁴,
Chan-Su Shin⁵, and Antoine Vigneron⁶

¹ Department of Computer Science and Engineering, POSTECH, Pohang, Korea
heekap@postech.ac.kr

² Department of Computer Science, Kyonggi University, Suwon, Korea
swbae@kgu.ac.kr

³ Institute of Computer Science, Universität Bayreuth, 95440 Bayreuth, Germany
christian.knauer@uni-bayreuth.de

⁴ Department of Computer Science, KAIST, Daejeon, Korea
mira@kaist.ac.kr

⁵ Department of Digital and Information Engineering,
Hankuk University of Foreign Studies, Yongin, Korea
cssin@hufs.ac.kr

⁶ Geometric Modeling and Scientific Visualization Center,
KAUST, Thuwal, Saudi Arabia
antoine.vigneron@kaust.edu.sa

Abstract. Given a simple rectilinear polygon P in the xy -plane, a *roof* over P is a terrain over P whose faces are supported by planes through edges of P that make a dihedral angle $\pi/4$ with the xy -plane. In this paper, we introduce *realistic roofs* by imposing a few additional constraints. We investigate the geometric and combinatorial properties of realistic roofs, and show a connection with the straight skeleton of P . We show that the maximum possible number of distinct realistic roofs over P is $\binom{(n-4)/2}{\lfloor(n-4)/4\rfloor}$ when P has n vertices. We present an algorithm that enumerates a combinatorial representation of each such roof in $O(1)$ time per roof without repetition, after $O(n^4)$ preprocessing time. We also present an $O(n^5)$ -time algorithm for computing a realistic roof with minimum height or volume.

1 Introduction

The *straight skeleton* $SK(P)$ of a polygon P is a straight line graph embedded in P , defined as the trace of the vertices of P when P is shrunk [12], each edge moving inwards parallel to itself at the same speed. (See Fig. 1b.) It has

* Work by Ahn was supported by the National Research Foundation of Korea Grant funded by the Korean Government (MEST) (NRF-2010-0009857). Work by Bae and Lee was supported by National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0005512). Work by Shin was Supported by research grant 2011 funded by Hankuk U. of Foreign Studies.

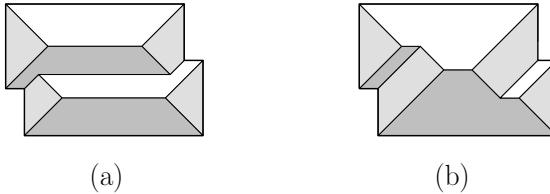


Fig. 1. A polygon P that admits two roofs (a) and (b). The projection of the edges of roof (b) onto the xy -plane is the straight skeleton $SK(P)$

received considerable attention in the past few years, because of its various applications [3, 7, 8, 10, 15, 17], and because the best known algorithms for computing the straight skeleton [6, 9, 11] are not known to be optimal.

We assume that P initially lies in the xy -plane and is shrunk at unit speed while moving upward at unit speed. Then P traces a three-dimensional polyhedral terrain, and $SK(P)$ can be obtained as the projection of the edges of this terrain onto the xy -plane. It follows directly from this definition that each face of this terrain lies in a plane that makes a dihedral angle of $\pi/4$ with the xy -plane. Furthermore, each face is incident to at least one edge of P . Aichholzer et al. [1] noticed that, for a given polygon P , several different terrains may have these two properties, and called such terrains *roofs* over the polygon P . For instance, Fig. 1 shows a rectilinear polygon (that is, a polygon whose edges are parallel to the x -axis or the y -axis) that admits two different roofs.

The problem of automatically reconstructing the 3D model of a building when only its groundplan is available, or even the reconstructing a complete city model from satellite data, has also been studied extensively [4, 5, 14, 12, 16, 13]. In particular, Brenner [5] designed an algorithm that computes all the possible roofs over a rectilinear polygon. Several theoretical results on this problem, mostly based on the straight skeleton, have been presented [1, 2, 6, 9, 11]. The roof structure associated with the straight skeleton is not necessarily the right reconstruction; for instance, it seems that the roof in Fig. 1a is more realistic than the roof given by the straight skeleton (b). Therefore, it is important to extend straight skeleton algorithms, so as to be able to find all the possible roofs.

In this paper, we consider *realistic* roofs over a rectilinear groundplan P . Restricting P to be rectilinear is reasonable in practice: it was reported in 2003 that around 45 percent of building groundplans are close to being rectilinear [14]. We define realistic roofs by adding a few constraints to the roof definition by Aichholzer et al. [1]. One important difference is that realistic roofs do not have vertices that are local minima. This is often required for roofs in the real-world, as rainwater cannot be well drained from a local minimum along the roof surface.

Our main results are threefold:

1. We show that the maximum possible number of realistic roofs over a rectilinear groundplan P is exactly $\binom{(n-4)/2}{\lfloor(n-4)/4\rfloor}$, which asymptotically grows as $2^n/\sqrt{n}$.
2. We present an algorithm that generates all combinatorial representations of realistic roofs over P in $O(1)$ time per roof, after $O(n^4)$ preprocessing time.

3. We give polynomial-time algorithms for computing optimal roofs. In particular, we can compute a roof with minimum height, or minimum volume, in $O(n^5)$ time.

Our representation of a realistic roof is based on several new observations, including a connection to the straight skeleton. As a result, the roof corresponding to a given representation can be built in linear time. It is also worth noting that there are rectilinear polygons P over which there is only a single roof, regardless of the number n of vertices. Thus, the number of realistic roofs can be any integer between 1 and $\binom{(n-4)/2}{\lfloor(n-4)/4\rfloor}$. This is why it is important to develop output sensitive algorithms such as ours, with a running time polynomial on the input size n , and linear in the number of realistic roofs for the given input polygon. By contrast, no polynomial bound on the running time of Brenner's algorithm [5] is known. Due to space limitation, this extended abstract does not include all the proofs of our results.

2 Preliminaries

Throughout this paper, we denote by P a rectilinear simple polygon in the xy -plane. Hence, the number n of vertices of P is even, and the number of reflex (non-convex) vertices is $(n - 4)/2$. The boundary of P , that is, the union of its edges, is denoted by ∂P . A polygonal surface R is a *roof* if and only if it satisfies the following conditions **R1–R4**.

- **R1:** R is z -monotone and every face of R lies on or above the xy -plane.
- **R2:** The projection of R onto the xy -plane coincides with P , and $\partial P \subset R$.
- **R3:** Each face of R makes a dihedral angle $\pi/4$ with the xy -plane.
- **R4:** Each face of R is incident to an edge of P .

Aichholzer et al. [1] defined a roof over a simple polygon (not necessarily rectilinear) as a terrain that satisfies conditions **R1–R3**. We add condition **R4** to their definition of roofs because a roof having faces disconnected from its boundary may be difficult to build. We call the edges of P incident to a face f of R the *ground edges* of f , and we denote by $\text{edge}(f)$ any of the ground edges of f . (See Fig. 2a.) Note that all the ground edges lie in a common line since the ground edges of f belong to the intersection between the xy -plane and the plane supporting f .

In the discussion below, we consider the edges in the interior of a roof: We do not consider the edges of P as being edges of a roof over P . Hence, each edge of a roof is shared by a pair of faces. An edge e of a roof R is *convex* if R is locally convex along the relative interior of e ; *reflex* otherwise. We classify the edges of a roof into four types: convex edges parallel to the xy -plane are *ridges* (**rd** for short), reflex edges parallel to the xy -plane are *valleys* (**v1**). Convex edges which are not ridges are simply *convex edges* (**cv**), and reflex edges which are not valleys are *reflex edges* (**rf**). Edges not parallel to the xy -plane are oriented from the

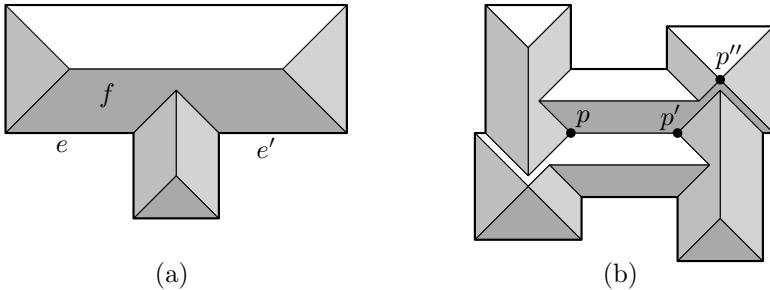


Fig. 2. Two examples of roofs. (a) The face f has two ground edges e and e' . (b) Two vertices p, p' are local minima. Vertex p'' is a degenerate vertex which is not adjacent to any ridge or valley.

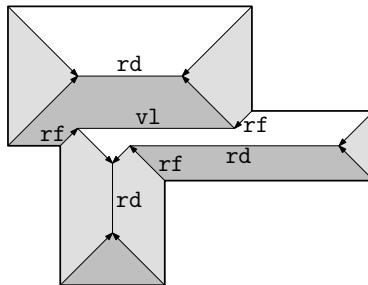


Fig. 3. Labeling the edges of a roof. The edges without label are all convex edges (**cv**).

endpoint with smaller z -coordinate to the other one with larger z -coordinate; we call the endpoint with smaller z -coordinate the *origin* of the edge. (See Fig. 3.)

A valley or a ridge of a roof is an edge shared by two faces whose ground edges are parallel. A convex or reflex edge of a roof is an edge shared by two faces whose ground edges are orthogonal. A vertex of a roof is an endpoint of a ridge or a valley e unless e degenerates to a point; In this case, the degenerate edge is a vertex. (See Fig. 2b.)

A roof may have a vertex that is a local minimum, as shown by Aichholzer et al. [1]. (See Fig. 2b.) The roof of a building, however, is often required to be able to drain rainwater down to the ground. This requirement would not prevent a roof from having a valley, but it does prevent vertices from being local minima [2]. Thus, we will say that a roof R is *realistic* if, in addition to **R1–4**, the following condition **R5** holds:

- **R5:** No vertex of R is a local minimum.

Throughout this paper, we use the following notation. A *maximal rectangle* contained in the rectilinear polygon P is an axis-aligned rectangle Q contained in P , such that no other axis-aligned rectangle contained in P contains Q . For a point p on or above the xy -plane we let $z(p)$ be the z -coordinate of p , we let \bar{p} be

its orthogonal projection onto the xy -plane, and we let $D(p)$ be the axis-parallel square on the xy -plane centered at \bar{p} with side length $2z(p)$. For a line segment s parallel to the x -axis or the y -axis, we denote by $D(s)$ the axis-parallel rectangle $\bigcup_{p \in s} D(p)$. For any two points p, q , we denote by $d_\infty(p, q)$ the L_∞ distance between p and q , and we denote $d_\infty(p, A) := \inf_{a \in A} d_\infty(p, a)$ for any set A .

Our definition of a roof implies the following:

Observation 1. *Let R be a roof over a rectilinear simple polygon and let H be a plane parallel to the z -axis.*

- (i) *If H is parallel to the x -axis or the y -axis, then each segment of $R \cap H$ has slope in $\{-1, 0, 1\}$.*
- (ii) *If H makes an angle $\pi/4$ with the x -axis, then each segment of $R \cap H$ has slope $\pm 1/\sqrt{2}$.*

The lemma below follows from this observation.

Lemma 1. *Let R be a roof over a rectilinear simple polygon P . The following hold.*

- (a) *For any point $p \in R$, we have $z(p) \leq d_\infty(\bar{p}, \partial P)$ and $D(p) \subseteq P$.*
- (b) *For each edge e of P , there exists a unique face f of R incident to e .*
- (c) *Every face f of R is monotone with respect to the line supporting edge(f).*

Recall the definition of the straight skeleton in Section 1. If P moves upward at the same speed as it shrinks, it draws a polygonal surface, which we denote by $R^*(P)$. (See Fig. 1b.) The edges of this polygonal surface project vertically onto the straight skeleton $SK(P)$. Aichholzer et al. [1] studied $R^*(P)$ for simple polygons, and the following theorem follows easily from their work:

Theorem 1. *Let P be a simple rectilinear polygon with n vertices. Then, the roof $R^*(P)$ with no valley over P is the point-wise highest roof over P . It can be constructed in $O(n)$ time.*

3 Properties of Realistic Roofs

Throughout this section, we direct reflex or convex edges from the lower endpoint to the higher, so that an edge is *outgoing* from the lower endpoint and is *incoming* to the higher.

3.1 Local Properties of Valleys and Ridges

We will first list all possible configurations for roof vertices. It is derived from a careful case analysis, which will appear in the full version of this paper. In this section, we only sketch the proof of this vertex classification.

Consider a valley v of a roof R over P . We classify each endpoint of valley v according to the local configuration. (See Fig. 4) More specifically, for each endpoint p of the valley v , we consider the two faces f and f' incident to v and

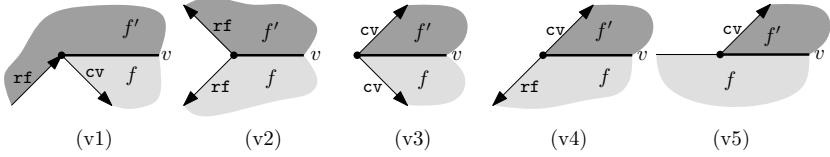


Fig. 4. Five configurations near a vertex incident to a valley v . Vertices of type (v1) can be found in Fig. II(a) or III

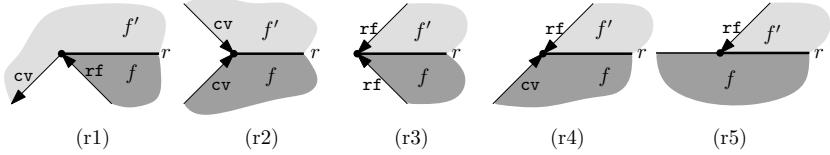


Fig. 5. Five possible configurations near a vertex incident to a ridge r . Vertices of type (r1) appear in Fig. III, type (r2) and (r3) appear in Fig. II(b), and type (r4) and (r5) are degenerate cases.

the edges e and e' incident to p , and f and f' , respectively. Recall that v is parallel either to the x -axis or the y -axis.

A careful study of the local configuration around valley endpoints reveals that there are only five possible combinations for e and e' , as shown in Fig. 4: (v1)(v4) one is a convex edge and the other a reflex edge; (v2) both are reflex edges; (v3) both are convex edges; or (v5) one is parallel to v . Note that for type (v5) the parallel incident edge must be another valley, because of face f .

Lemma 2. *Let R be any realistic roof over P and v be any valley of R . Then, both endpoints of v are of type (v1).*

By similar arguments, we can also classify the endpoints of the ridges into five types. (See Fig. 5) Endpoints of ridges of each type can also have additional incident edges.

Lemma 3. *Let R be a realistic roof and v be any valley of R . Then, v is adjacent neither to a valley nor to a ridge. Therefore, both endpoints of v have degree exactly three.*

Not all vertices are incident to a valley or a ridge. Such vertices are shown to be degenerate cases of a valley or a ridge.

Lemma 4. *Let R be a roof and p be any vertex not incident to a valley or a ridge. Then, p is of degree exactly four and falls into one of the three types:*

- (d1) a local maximum (a degenerate ridge with endpoints of type (r2));
- (d2) a saddle point (a degenerate valley with endpoints of type (v1));
- (d3) a local minimum (a degenerate valley with endpoints of type (v2));

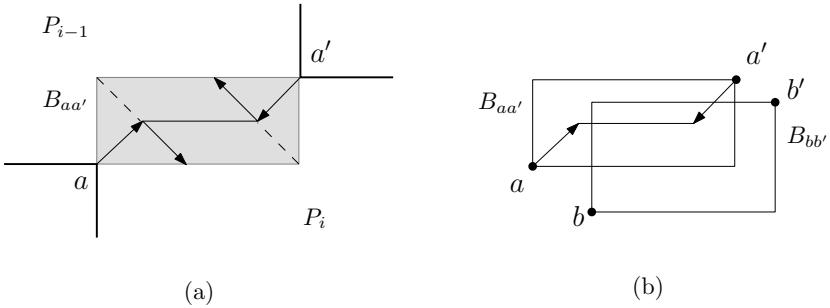


Fig. 6. (a) A candidate pair (a, a') defining a valley in a realistic roof. (b) For two candidate pairs (a, a') and (b, b') , if aa' and bb' do not intersect, either $B_{aa'}$ and $B_{bb'}$ are disjoint, or $B_{aa'}$ contains exactly one corner of $B_{bb'}$ and $B_{bb'}$ contains exactly one corner of $B_{aa'}$.

In addition, if R is realistic, then it has no vertex of type (d3).

From the discussion above, we obtain the list of all possible vertex types in realistic roofs:

Corollary 1. *Each vertex of a realistic roof has one of the following types: (v1), (r1)–(r5), (d1), and (d2).*

Thus, we obtain the following characterization of reflex edges.

Lemma 5. *Each reflex edge of a realistic roof R over P is outgoing from a reflex vertex of P .*

From now on, unless stated otherwise, we will consider vertices of type (d2) as valleys, as they can be seen as degenerate valleys. Similarly, a vertex of type (d1) will be considered as a ridge.

From the results above, any valley v of a realistic roof R over a rectilinear polygon P is associated with two reflex vertices a and a' of P and two adjacent reflex edges e and e' incident to a and a' , respectively. (See Fig. 6) We denote by $B_{aa'}$ the smallest axis-aligned rectangle containing a and a' , and we define the polyline $C_{aa'} := e \cup v \cup e'$. We then have the following configuration for valleys of realistic roofs.

Corollary 2. *Let v be a valley of a realistic roof R over a rectilinear polygon P associated with two reflex vertices a and a' of P . Then, v is fully determined by the pair (a, a') of reflex vertices as follows:*

1. a and a' are opposite corners of $B_{aa'}$.
2. $C_{aa'}$ is contained in $B_{aa'}$ and centrally symmetric around the center of $B_{aa'}$.
3. The valley v coincides with the ridge of $R^*(B_{aa'})$.

Corollary 2 tells us that a valley in a realistic roof is determined uniquely by a pair of reflex vertices a and a' of P . Thus, the number of valleys in a realistic roof R over P is not more than $(n - 4)/4$, since P has $(n - 4)/2$ reflex vertices.

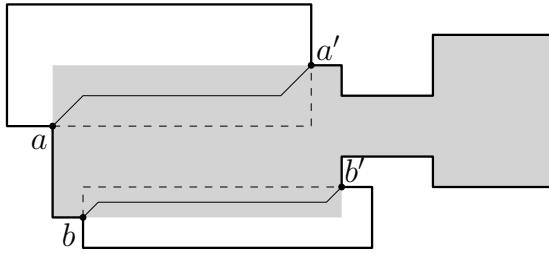


Fig. 7. The polygons P_i corresponding to the compatible set $\{(a, a'), (b, b')\}$. One of the polygons is shaded, the other two are bounded by the dashed polyelines.

3.2 Global Structure of Realistic Roofs

We say that two reflex vertices a, a' of P are *opposite vertices* if they are facing opposite directions, as in Fig. 6a.

Lemma 6. *Let a and a' be reflex vertices of P . If the pair (a, a') defines a valley in a realistic roof, then it satisfies the following conditions:*

1. *The pair (a, a') is opposite.*
2. *$B_{aa'} \setminus \{a, a'\}$ is contained in the interior of P .*

We call such a pair (a, a') , satisfying the conditions of Lemma 6, a *candidate pair*. For a realistic roof R , each valley is associated with a candidate pair and thus R induces a set V of candidate pairs. On the other hand, not all sets V of candidate pairs realize a realistic roof. We call a set V of candidate pairs *compatible* if there is a realistic roof R that induces V . Also, we denote by $R_P(V)$ a roof over P realized by a compatible set V . In the full version of this paper, we will prove the following:

Lemma 7. *A set V of candidate pairs is compatible for P if and only if $C_{aa'} \cap C_{bb'} = \emptyset$ holds for any two pairs $(a, a'), (b, b') \in V$.*

Theorem 2. *Let P be a rectilinear polygon with n vertices and V be a compatible set of k candidate pairs with respect to P . Then there exists a unique roof $R_P(V)$ that induces V . In addition, there exist $k + 1$ rectilinear polygons P_1, \dots, P_{k+1} contained in P such that*

- $\bigcup_i P_i = P$,
- the total number of vertices of the P_i 's is at most n , and
- $R_P(V)$ coincides with the upper envelope of $R^*(P_i)$ for all $i = 1, \dots, k + 1$.

The polygons P_i from Theorem 2 can be easily obtained by cutting P along the polyline $C_{aa'}$ for each candidate pair $(a, a') \in V$, and then adding rectangle $B_{aa'}$ to each of the two parts bounded by $C_{aa'}$. (See Fig. 7)

Based on the results above, we are also able to provide sharp bounds on the number of possible realistic roofs.

Theorem 3. *For any even integer $n \geq 4$, the maximum number of distinct realistic roofs over a rectilinear, simple polygon with n vertices is $\binom{(n-4)/2}{\lfloor (n-4)/4 \rfloor}$.*

4 Enumerating Realistic Roofs and Computing Optimal Roofs

By Theorem 2, a compatible set V of candidate pairs for P determines a unique realistic roof $R_P(V)$ over P . We thus regard a compatible set V as a representation of a realistic roof $R = R_P(V)$ over P , and give an algorithm that generates all compatible sets for P . In the full version of this paper, we will show how to build the roof $R_P(V)$ corresponding to a given compatible set V in linear time:

Lemma 8. *Let P be a rectilinear n -gon and V be any compatible set for P . Then, the roof $R_P(V)$ over P determined by V can be constructed in $O(n)$ time.*

We let $1, \dots, m$ denote the $m = (n-4)/2$ reflex vertices of P in counter-clockwise order. We only consider pairs (a, a') such that $a < a'$. Also, we define a dummy candidate pair $(0, m+1)$ for the simplicity of the description.

Lemma 9. *For two distinct candidate pairs (i, j) and (s, t) of P , $\{(i, j), (s, t)\}$ is compatible if and only if $s, t \in \{i+1, i+2, \dots, j-2, j-1\}$ or $s, t \in \{j+1, j+2, \dots, m, 1, \dots, i-2, i-1\}$.*

Lemma 9 enables us to check the compatibility of a set of candidate pairs just by considering the order of candidate pairs, instead of the geometric structure of the pairs. We preprocess in $O(n^4)$ time the set of candidate pairs so as to efficiently enumerate the set of candidate pairs. Then we start with a current compatible set $\mathcal{V} = \emptyset$ and generate recursively the next compatible sets by adding candidate pairs to \mathcal{V} one by one in sorted order. Details will be given in the full version.

Theorem 4. *Given a rectilinear polygon P with n vertices, m of which are reflex, after $O(n^4)$ -time preprocessing, the compatible sets of P can be enumerated in $O(1)$ time per representation. We can construct a roof for each compatible set in $O(n)$ time.*

We also obtain polynomial-time algorithms for computing optimal roofs.

Theorem 5. *Given a simple rectilinear polygon P with n vertices, we can compute in $O(n^5)$ time a realistic roof over P with a minimum volume. Within the same time bound, we can also compute a roof whose maximum height is minimized.*

Acknowledgments. Part of this research was done at Dagstuhl. We would like to thank Peter Brass, Otfried Cheong, Xavier Goaoc, Hyeyon-Suk Na, and Alexander Wolff for helpful discussions on this problem.

References

1. Aichholzer, O., Albertsa, D., Aurenhammer, F., Gärtner, B.: A novel type of skeleton for polygons. *J. Universal Comput. Sci.* 1, 752–761 (1995)

2. Aichholzer, O., Aurenhammer, F.: Straight Skeletons for General Polygonal Figures in the Plane. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 117–226. Springer, Heidelberg (1996)
3. Barequet, G., Goodrich, M.T., Levi-Steiner, A., Steiner, D.: Straight-skeleton based contour interpolation. In: Proc. 14th ACM-SIAM Symp. Discrete Alg. (SODA), pp. 119–127 (2003)
4. Brenner, C.: Interactive modelling tools for 3d building reconstruction. In: Fritsch, D., Spiller, R. (eds.) Photogrammetric Week 1999, pp. 23–34 (1999)
5. Brenner, C.: Towards fully automatic generation of city models. Int. Archives of Photogrammetry and Remote Sensing XXXIII(Part B3), 85–92 (2000)
6. Cheng, S.-W., Vigneron, A.: Motorcycle graphs and straight skeletons. Algorithmica 47, 159–182 (2007)
7. Cloppet, F., Stamon, G., Olivia, J.-M.: Angular bisector network, a simplified generalized Voronoi diagram: Application to processing complex intersections in biomedical images. IEEE Trans. Pattern Anal. Mach. Intell. 22, 120–128 (2000)
8. Demaine, E.D., Demaine, M.L., Lindy, J.F., Souvaine, D.L.: Hinged Dissection of Polytopes. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 205–217. Springer, Heidelberg (2005)
9. Eppstein, D., Erickson, J.: Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. Discrete Comput. Geom. 22, 569–592 (1999)
10. Huber, S., Held, M.: Theoretical and practical results on straight skeletons of planar straight-line graphs. In: Symposium on Computational Geometry, pp. 171–178 (2011)
11. Huber, S., Held, M.: Theoretical and practical results on straight skeletons of planar straight-line graphs. In: Proc. 27th Annu. ACM Sympos. Comput. Geom., pp. 171–178 (2011)
12. Khoshelham, K., Li, Z.L.: A split-and-merge technique for automated reconstruction of roof planes. Photogrammetric Engineering and Remote Sensing 71(7), 855–863 (2005)
13. Krauß, T., Lehner, M., Reinartz, P.: Generation of coarse 3D models of urban areas from high resolution stereo satellite images. Int. Archives of Photogrammetry and Remote Sensing XXXVII, 1091–1098 (2008)
14. Laycock, R.G., Day, A.M.: Automatically generating large urban environments based on the footprint data of buildings. In: Proc. 8th ACM Sympos. Solid Model. Appl., pp. 346–351 (2003)
15. Oliva, J.-M., Perrin, M., Coquillart, S.: 3D reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram. Comput. Graph. Forum 15(3), 397–408 (1996)
16. Sohn, G., Huang, X.F., Tao, V.: Using a binary space partitioning tree for reconstructing polyhedral building models from airborne lidar data. Photogrammetric Engineering and Remote Sensing 74(11), 1425–1440 (2008)
17. Tănase, M., Veltkamp, R.C.: Polygon decomposition based on the straight line skeleton. In: Proc. 19th ACM Sympos. Comput. Geom., pp. 58–67 (2003)

Computing the Visibility Polygon Using Few Variables

Luis Barba¹, Matias Korman², Stefan Langerman^{2,*},
and Rodrigo I. Silveira^{3, **}

¹ Universidad Nacional Autónoma de México (UNAM), Mexico D.F., Mexico
`1.barba@uxmcc2.iimas.unam.mx`

² Université Libre de Bruxelles (ULB), Brussels, Belgium
`{mkormanc, slanger}@ulb.ac.be`

³ Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
`rodrigo.silveira@upc.edu`

Abstract. We present several algorithms for computing the visibility polygon of a simple polygon \mathcal{P} from a viewpoint inside the polygon, when the polygon resides in read-only memory and only few working variables can be used. The first algorithm uses a constant number of variables, and outputs the vertices of the visibility polygon in $O(n\bar{r})$ time, where \bar{r} denotes the number of reflex vertices of \mathcal{P} that are part of the output. The next two algorithms use $O(\log r)$ variables, and output the visibility polygon in $O(n \log r)$ randomized expected time or $O(n \log^2 r)$ deterministic time, where r is the number of reflex vertices of \mathcal{P} .

1 Introduction

The *visibility polygon* of a simple polygon \mathcal{P} from a viewpoint q is the set of all points of \mathcal{P} that can be seen from q , where two points p and q can see each other whenever the segment pq is contained in \mathcal{P} . The visibility polygon is a fundamental concept in computational geometry and one of the first problems studied in planar visibility. The first correct and optimal algorithm for computing the visibility polygon from a point was found by Joe and Simpson [11]. It computes the visibility polygon from a point in linear time and space. We refer the reader to the survey of O'Rourke [14] and the book of Gosh [9] for an extensive survey of such problems.

In this paper we look for an algorithm that computes the visibility polygon of a given point and uses few variables. This kind of algorithms not only provides an interesting trade-off between running time and memory needed, but also is very useful in portable devices where important hardware constraints are present (such as the ones found in digital cameras or portable phones).

A significant amount of research has focused on the design of algorithms that use few variables, some of them even dating from the 80s [12]. Although

* Maître de Recherches du FRS-FNRS.

** Funded by the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235.

many models exist, most of the research considers that the input is in some kind of read-only data structure. In addition to the input values we are allowed to use few additional variables (typically a logarithmic amount). One of the most studied problems in this setting is that of selection. For any constant $\epsilon \in (0, 1)$, Munro and Raman [13] give an algorithm that runs in $O(n^{1+\epsilon})$ time and uses $O(1/\epsilon)$ variables. Frederickson [8] improved this result with an algorithm whose running time is $O(n \log^* s + n \log n / \log s)$ when s working variables are available (and $s \in \Omega(\log n) \cap O(2^{\log n / \log^* n})$). Whenever only $O(\log n)$ variables are available, Raman and Ramnath [15] gave an algorithm whose running time is $O(n \log^2 n)$. More recently Chan [5] provided several lower bounds for performing selection with few variables and modified the $O(n \log \log_s n)$ expected time algorithm of Munro and Raman so that it works for any input array.

Recently, there has been an interest in finding algorithms for geometric problems that use a constant number of variables: Given a set of n points, the well-known gift-wrapping algorithm (also known as Jarvis march [16]) can be used to report the points on the convex hull in $O(n\bar{h})$ time using a constant number of variables, where \bar{h} is the number of vertices on the convex hull. Asano and Rote [3] and afterwards Asano *et al.* [2] gave efficient methods for computing well-known geometric structures, such as the Delaunay triangulation, Voronoi diagram and minimum spanning tree using a constant number of variables (in $O(n^2)$, $O(n^2)$ and $O(n^3)$ time, respectively). Observe that, since these structures have linear size, they are not stored but reported.

To the best of our knowledge, there is no known method that computes the visibility polygon of a given point and uses few variables. The question of finding a constant workspace algorithm to compute the visibility polygon in a polygon of size n was explicitly posed as an open problem by Asano *et al.* [1]. A natural approach to the problem would be to transform the linear time-algorithm of Joe and Simpson [11] to a constant workspace algorithm. However, their algorithm cannot be directly adapted to our setting, since their method uses a stack which could contain up to $\Omega(n)$ vertices.

Results. In this paper, we consider a completely different approach. It is easy to realize that the differences between \mathcal{P} and the visibility polygon of a given point depends on the number of reflex vertices. For example, if \mathcal{P} is a convex polygon, the two polygons will be equal, but we cannot expect this to hold when the number of reflex vertices grows. Therefore whenever possible we will express the running time of our algorithms not only in terms of n , the complexity of \mathcal{P} , but also in terms of r and \bar{r} (the number of reflex vertices in the input and the number of them also present in the output, respectively). This approach continues a line of research relating the combinatorial and computational properties of polygons to the number of their reflex vertices. We refer the reader to [4] and references found therein for a deep review of existing similar results.

In Section 3 we give an output-sensitive algorithm that reports the vertices of the visibility polygon in $O(n\bar{r})$ time using $O(1)$ variables. In Section 4 we show that if we are allowed to use $O(\log r)$ variables, the problem can be solved in $O(n \log r)$ randomized expected time or $O(n \log^2 r)$ deterministic time.

2 Preliminaries

Model definition and considerations on input/output precision. We use a slight variation of the constant workspace model (sometimes also referred as *log space*), introduced by Asano [3]. In this model, an algorithm can use a constant number of variables and assume that each variable or pointer contains a data word of $O(\log n)$ bits. Implicit storage consumption required by recursive calls is also considered a part of the workspace.

The input of the problem is a polygon \mathcal{P} in a read-only data structure. In the usual constant workspace model, we are allowed to perform random access to any of the values of the input in constant time. However, in this paper we consider a weaker model in which the only allowed operation to the input is obtaining coordinates of the next counterclockwise vertex of a given vertex of \mathcal{P} . This is the case in which, for example, the vertices of \mathcal{P} are given in a list in counterclockwise order.

Many other similar models exist in the literature. We note that in some of them (like the *streaming* [10] or the *multi-pass* model [6]) the values of the input can only be read once or a fixed number of times. We follow the model of constant workspace and allow scanning the input as many times as necessary.

The algorithm given in Section 3 uses a weaker model than the constant workspace model (since random access to the vertices of the input is not used). The methods given in Section 4 use a logarithmic number of variables, providing another compromise between the space and time complexity of this problem.

Let n be the number of vertices of \mathcal{P} . We do not make any assumptions on whether the input coordinates are rational or real numbers (in some implicit form). The only operations that we perform on the input are determining whether a given point is above, below or on a given line and determining the intersection point between two lines. In both cases, the line is defined as passing through two points of the input, hence both operations can be expressed as finding the root of linear equations whose coefficients are values of the input. We assume that these two operations can be done in constant time. Observe that if the coordinates of the input are algebraic values, we can express the coordinates of the output as “the intersection point of the line passing through points v_i and v_j and the line passing through v_k and v_l ” (where v_i, v_j, v_k and v_l are vertices of the input).

Other definitions. The boundary of \mathcal{P} is denoted by $\partial\mathcal{P}$. We regard \mathcal{P} as a closed subset of the plane. We are also given a point q inside \mathcal{P} , from where the visibility polygon needs to be computed. The segment connecting points p and q is denoted by pq . We say that a point $p \in \mathcal{P}$ is visible (with respect to q) if and only if $pq \subset \mathcal{P}$ (otherwise we say that p is *not visible*). The set of points that are visible from q is denoted by $\text{Vis}_{\mathcal{P}}(q)$ and is called the *visibility polygon* of q . It is easy to see that the visibility polygon of q is a closed polygon whose vertices are either vertices of \mathcal{P} or the intersection point between a segment of \mathcal{P} and a ray emanating from q (see Figure 1).

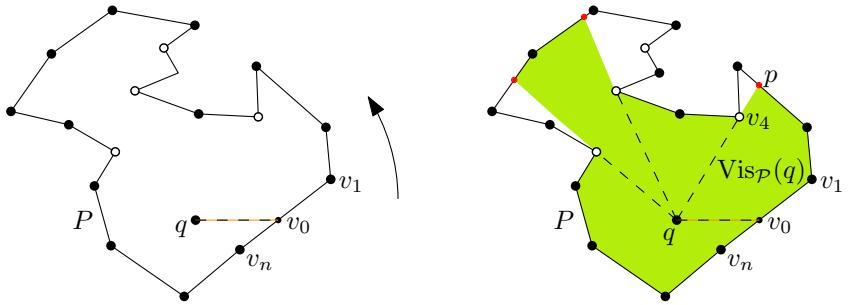


Fig. 1. Left: general setting, vertices that are reflex with respect to q are shown with a white point (black otherwise). Right: the visibility polygon $\text{Vis}_P(q)$ of q ; point p is the *shadow* of vertex v_4 .

From now on, for simplicity in the explanation, we assume that no line passes through q and two vertices of P . We note, however, that our algorithms can be adapted to work in the general case with only minor changes. We also need to define what a *reflex* vertex is in our context. Given any vertex v_k , the line ℓ_k passing through v_k and q splits $P \setminus \ell_k$ into disjoint components. We say that vertex v_k is *reflex with respect to q* if the angle at the vertex interior to P is more than π and the vertices v_{k-1} and v_{k+1} lie on the same connected component of $\mathbb{R}^2 \setminus \ell_k$ (see Figure II). Observe that any vertex that is reflex with respect to q is a reflex vertex (in the usual sense), but the converse is not true.

Intuitively speaking, reflex vertices with respect to q are the vertices where topological changes might occur in the visibility polygon. That is, the positions where the polygon boundary can change between visible or invisible. Since the point q is fixed, from now on we omit the “with respect to q ” term and simply refer to these points as reflex. Let r be the number of reflex vertices of P . We also define \bar{r} as the number of reflex vertices of P that are present in $\text{Vis}_P(q)$. Observe that we always have $\bar{r} \leq r < n$.

Given any two points $p_1, p_2 \in \partial P$, there is a unique path that travels counterclockwise along the boundary of P from p_1 to p_2 . Let $\text{Chain}(p_1, p_2)$ be the set of points traversed in this path, including both p_1 and p_2 (this set is called the *chain* between p_1 and p_2). We say that a chain is *visible* if all the points of the chain are visible from q . A visible chain $C = \text{Chain}(p_1, p_2)$ is *CCW-maximal* if it is visible and no other visible chain starting at p_1 strictly contains C . We will use the term $\text{Chain}(p, p)$ to denote the whole polygon boundary.

Given a point $p \in P$, let $\theta(p)$ be the angle that the ray emanating from q and passing through p makes with the positive x -axis, $0 \leq \theta(p) < 2\pi$. Let v_0 be the closest point on ∂P to q , such that $\theta(v_0) = 0$. It is easy to see that v_0 is visible and can be computed in linear time. Without loss of generality, we will treat v_0 as a vertex (even though it does not need to be one). Moreover, we will assume that the vertices are numbered such that v_0 is on edge v_nv_1 .

A point p on ∂P is the *shadow* of a reflex vertex v if p is collinear with q and v , and p is visible from q . Due to the general position assumption, v must be

unique and p must be an interior point of an edge. As a result, each reflex vertex is uniquely associated to a shadow point (and *vice versa*).

In this paper, we will often use a ray shooting-like operation, which we call $\text{RAYSHOOTING}(p)$. This is a basic operation that given a point $p \in \partial\mathcal{P}$, considers the ray $\rho(q, p)$ and reports the first point of \mathcal{P} hit by the ray. This operation can be done in linear time, by scanning the edges of \mathcal{P} one by one and reporting the point closest to q that intersects the ray. Another similar operation that our algorithms will use is computing the shadow of a given reflex vertex v . For that case, we will use the $\text{FINDSHADOW}(p)$ operation, which gives the shadow of p if p is a reflex vertex (p otherwise). Finally we note that, due to space constraints, some of the proofs of this paper have been omitted.

3 An $O(n\bar{r})$ Algorithm Using $O(1)$ Variables

In this section we present an output-sensitive algorithm that spends linear time for each reflex vertex in the visibility polygon. The idea of the algorithm is to compute maximal visible chains as they appear on the boundary of \mathcal{P} . Each iteration of the algorithm starts from a point that is known to be visible, and finds the last point of the CCW-maximal chain that starts at that point. This is repeated until the initial vertex is found again.

The following lemma characterizes the endpoints of CCW-maximal chains.

Lemma 1. *Let $p \in \partial\mathcal{P}$ be a point visible from q , and let $\mathcal{C} = \text{Chain}(p, p')$ be a CCW-maximal chain. Then p' is either (i) equal to p , (ii) a reflex vertex of \mathcal{P} , or (iii) the shadow of some reflex vertex of \mathcal{P} .*

Based on the previous lemma, the algorithm will start from v_0 and walk on $\partial\mathcal{P}$ in counterclockwise direction, identifying the endpoint of the current CCW-maximal chain.

Let p be a visible point and let v_r be the first reflex vertex found on $\partial\mathcal{P}$ when going counterclockwise from p . $\text{Chain}(p, v_r)$ and the segments qp and qv_r define a region that we will denote by $\mathcal{R}(p, v_r)$. The following observation is crucial for the algorithm.

Lemma 2. *Let p be a visible point and let v_r be the first reflex vertex encountered on $\partial\mathcal{P}$ when going from p in counterclockwise direction. Then v_r is visible if and only if $\mathcal{R}(p, v_r)$ contains no vertex from \mathcal{P} , or equivalently, if and only if no edge of \mathcal{P} crosses the segment qv_r .*

Let $\text{FINDNEXTREFLEXVERTEX}(p)$ be a routine that returns the first reflex vertex v_r found on $\partial\mathcal{P}$ counterclockwise from p in $O(n)$ time. If no such vertex exists the remaining portion of \mathcal{P} is convex (and thus all vertices until v_0 can be reported as visible). Otherwise, vertex v_r is identified and we need to find the end of the chain p' , which starts at p . For that we will determine if $\mathcal{R}(p, v_r)$ is empty, or equivalently, if $\text{RAYSHOOTING}(v_r) = v_r$.

If so, we have $p' = v_r$ and we can report all vertices in the chain $\text{Chain}(p, v_r)$. Then the next visible chain must start at the shadow of v_r , which can be found with the $\text{FINDSHADOW}(v_r)$ operation. Otherwise, from Lemma 11 we know that v_r is not visible and p' must be the shadow of some reflex vertex in $\text{Chain}(v_r, v_0)$. To find that reflex vertex we need one more observation.

Observation 1. *Let p be a visible point and let v_r be the first reflex vertex encountered on $\partial\mathcal{P}$ when going from p in counterclockwise direction. If v_r is not visible, then the CCW-maximal chain starting at p ends at the shadow of the reflex vertex in $\mathcal{R}(p, v_r)$ with smallest CCW-angle with respect to the segment qp .*

Therefore the algorithm only needs to find the reflex vertex with smallest angle that lies inside $\mathcal{R}(p, v_r)$, which can be done in $O(n)$ time by simply walking on $\partial\mathcal{P}$ and keeping track of the intersections with the line segment qv_r . Figure 2 illustrates a step of the algorithm.

Observe that in all cases, the total time spent in finding the end of the current chain and the beginning of the next one is $O(n)$. Hence, it follows that the total time of the algorithm is now $O(\bar{r}n)$, where \bar{r} is the number of vertices reflex w.r.t. q in the visibility polygon of q . Moreover, this algorithm can be implemented such that only one pass through the polygon vertices is needed for each reflex vertex that appears in the output (plus one initial pass for finding v_0). Algorithm 11 sketches how the algorithm would look. Note also that such implementation requires only 3 variables and 1 bit of additional space.

Theorem 1. *The visibility polygon of a point q can be computed in $O(n\bar{r})$ time using constant workspace, where \bar{r} is the number of reflex vertices that are part of the visibility polygon. More precisely, it can be computed in $(\bar{r} + 1)$ passes through the input and using 3 variables and 1 bit.*

4 An $O(n \log r)$ Algorithm Using $O(\log r)$ Variables

In this section we take a completely different approach to solve the problem. We consider the visibility problem in polygonal chains (instead of the whole polygon) and use a divide and conquer approach to split the problem into two subproblems. For this purpose, we must adjust the visibility definitions to make them consistent for chains.

Let $\mathcal{C} = \{p_0, \dots, p_i, \dots, p_m\}$ be a polygonal subchain of the boundary of \mathcal{P} , such that p_0, p_m are both visible points on the boundary of \mathcal{P} and p_i is a vertex of \mathcal{P} , $1 \leq i \leq m - 1$. Assume without loss of generality that $\alpha = \theta(p_0) < \theta(p_m) = \beta$, and

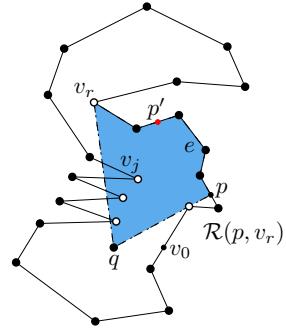


Fig. 2. When the next reflex vertex, v_r in the figure, is not visible, the end of the current visibility chain (p') can be found by walking on $\partial\mathcal{P}$. The shaded region is $\mathcal{R}(p, v_r)$.

Algorithm 1. Output-sensitive algorithm for visibility polygon

```

1:  $p \leftarrow v_0, r \leftarrow +\infty$ 
2: repeat
3:   Walk starting from  $p$  until next reflex vertex before  $v_0$ 
4:    $r \leftarrow$  index of reflex vertex found (or  $+\infty$  if none found)
5:   if  $r < +\infty$  then
6:     Walk from  $v_r$  until  $p$ , keep track of vertex with smallest angle inside  $\mathcal{R}(p, v_r)$ 
7:      $j \leftarrow$  vertex with smallest angle in  $\mathcal{R}(p, v_r)$  ( $j=+\infty$  if none found)
8:     if  $j = +\infty$  then
9:       (*  $v_r$  is visible *)
10:      Walk and report all points from  $p$  until finding  $v_r$ , including  $v_r$  (but not  $p$ )
11:      Let  $v_s$  be the shadow of  $v_r$ 
12:       $p \leftarrow v_s$ 
13:    else
14:      (*  $v_j$  found,  $v_r$  is not visible *)
15:      Let  $p'$  be the shadow of  $v_j$ 
16:      Walk and report all points from  $p$  until finding  $p'$ , including  $p'$ 
17:       $p \leftarrow v_j$ 
18:    end if
19:  end if
20: until  $r = +\infty$ 
21: Report all vertices between  $p$  and  $v_0$ 

```

let $\mathcal{P}_C = \{q, p_0, p_1, \dots, p_m, q\}$ be the polygon contained in \mathcal{P} obtained by joining q with both endpoints of C ; see Figure 3. We say that a point x on a polygonal chain C is C -visible (from q), if the segment qx is completely contained in the polygon \mathcal{P}_C . Let $\text{Vis}_C(q) = \{x \in C : x \text{ is } C\text{-visible}\}$ be the set of all C -visible points of C .

Lemma 3. Let $C = \{p_0, \dots, p_m\}$ be a polygonal chain such that p_0, p_m are both visible points on the boundary of \mathcal{P} , and let x be a visible point inside C lying on the edge $p_j p_{j+1}$. If $C_1 = \{p_0, \dots, p_j, x\}, C_2 = \{x, p_{j+1}, \dots, p_m\}$, then $\text{Vis}_C(q) = \text{Vis}_{C_1}(q) \cup \text{Vis}_{C_2}(q)$ and $\text{Vis}_{C_1}(q) \cap \text{Vis}_{C_2}(q) = x$.

Using a similar argument it is easy to see that if x is C -visible, and both p_0, p_m are visible from q , then x is a visible point of \mathcal{P} . Thus Lemma 3 allows us to divide the problem of finding $\text{Vis}_{\mathcal{P}}(q)$ into a series of subproblems that can be solved independently without compromising the output.

Let $\Delta(C)$ be the interior of the cone with apex q defined by the rays going from q and passing through both endpoints of C (and crossing the interior of the chain). Our divide and conquer

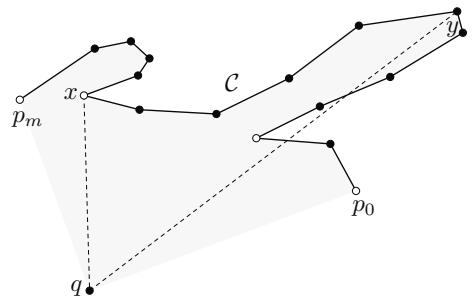


Fig. 3. Polygonal chain C and its associated polygon \mathcal{P}_C . Point x is C -visible, y is not.

algorithm is as follows. On each step of the algorithm we choose a random reflex vertex z inside the cone $\Delta(\mathcal{C})$, we then perform a ray shooting query to find the first point x on \mathcal{C} in the direction of z . We split the polygonal chain \mathcal{C} at x , thereby obtaining two subchains $\mathcal{C}_1, \mathcal{C}_2$; see Figure 4. We repeat the process recursively first on \mathcal{C}_1 and then on \mathcal{C}_2 , until \mathcal{C} is split into a series of subchains, each containing no reflex vertex inside their associated cone. Since the changes in visibility occur only at reflex vertices, the visible vertices inside each split subchain can be reported independently in the order in which they are found; see Figure 4.

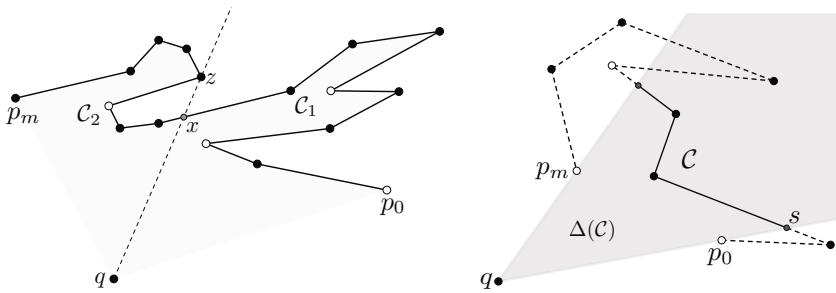


Fig. 4. Left: Split of \mathcal{C} into two subchains $\mathcal{C}_1, \mathcal{C}_2$ using a visible point x in the direction of a random reflex vertex z . Right: A polygonal chain \mathcal{C} with no reflex vertices inside the cone $\Delta(\mathcal{C})$, note that only one subchain of \mathcal{C} is visible.

The main algorithm presented in this section is summarized in Algorithm 2. The subroutine REPORTVISIBLECHAIN(x) takes a visible point x on \mathcal{C} , walks from x towards p_m and reports every vertex until finding an edge e of \mathcal{C} intersecting the ray $\rho(q, p_m)$, once found that intersection is reported and the subroutine ends.

Lemma 4. *Algorithm 2 reports every visible edge of $\text{Vis}_{\mathcal{C}}(q)$ in counterclockwise order.*

Lemma 5. *The expected running time of Algorithm 2 is $O(n \log r)$.*

Proof. (Sketch) The running time of Algorithm 2 can be analyzed in a similar way to the one of quicksort. On each step, a random pivot is chosen and it is used to split the chain \mathcal{C} , such that each subchain obtained contains in expectation a fraction of the reflex vertices. Thus if we look at the recursion tree, it is easy to see that on each level $O(n)$ operations are needed and the expected depth of the tree is $O(\log r)$, resulting in an expected running time of $O(n \log r)$. \square

Theorem 2. *The visibility polygon of point q can be computed in $O(n \log r)$ expected time using $O(\log r)$ variables.*

Algorithm 2. Given a polygonal chain $\mathcal{C} = \{p_0, \dots, p_m\}$ such that p_0, p_m are both visible points of \mathcal{P} , algorithm to compute $\text{Visc}(q)$

```

1:  $k \leftarrow$  number of reflex vertices of  $\mathcal{C}$  inside the cone  $\Delta(\mathcal{C})$ 
2: if  $k = 0$  then
3:    $s \leftarrow \text{FINDSHADOW}(p_0)$ 
4:   if  $s \neq p_0$  then
5:     Report  $p_0$ 
6:   end if
7:   REPORTVISIBLECHAIN( $s$ )
8: else
9:    $r \leftarrow$  random number in  $\{1, \dots, k\}$ 
10:   $z \leftarrow$  the  $r$ -th reflex vertex of  $\mathcal{C}$  inside  $\Delta(\mathcal{C})$ 
11:   $x \leftarrow \text{RAYSHOOTING}(z)$ 
12:  Call Algorithm 2 on  $\mathcal{C}_1 = \{p_0, \dots, x\}$ 
13:  Call Algorithm 2 on  $\mathcal{C}_2 = \{x, \dots, p_m\}$ 
14: end if

```

Proof. (Sketch) In each step of the recursion a constant number of variables are needed. Hence, the number of words used by the algorithm is proportional to the depth of the recursion. In order to avoid using an excessive amount of memory, we use a standard trick of restarting the algorithm whenever the recursion tree becomes too deep. \square

We now describe a deterministic variant of Algorithm 2 that runs on $O(n \log^2 r)$ time using $O(\log r)$ words. Let $\mathcal{R} = \{\theta(v_i) : v_i \text{ is a reflex vertex inside } \Delta(\mathcal{C})\}$ and let n, k be the number of vertices of \mathcal{C} and the cardinality of \mathcal{R} respectively. Recall that in steps 9 and 10 of Algorithm 2 we are choosing a random reflex vertex v inside $\Delta(\mathcal{C})$ and using it to split the chain \mathcal{C} into two subchains $\mathcal{C}_1, \mathcal{C}_2$, such that the number of reflex vertices inside $\Delta(\mathcal{C}_1)$ and $\Delta(\mathcal{C}_2)$ is balanced in expectation. The next variant of Algorithm 2 replaces the random selection with a deterministic selection algorithm that guarantees a balanced split, albeit at a slight increase in the running time. The algorithm is based in the approximate median pair algorithm proposed by Raman and Ramnath [15], in which an approximation of the median of a set of m elements can be computed using $O(\log m)$ variables in $O(\log m)$ passes.

Theorem 3. *The visibility polygon of point q can be computed in $O(n \log^2 r)$ time using $O(\log r)$ variables.*

5 Closing Remarks

One expects that the running time of a constant workspace algorithm increases when compared to other models in which memory is not an issue. In most geometric problems, the increase in the running time is almost equal to the reduction in memory space [23]. The algorithm given in Section 3 follows a similar pattern, since in the worst case the time-space product matches the one of Joe and

Simpson [11] (from $O(n^2)$ to $O(n\bar{r})$). However, notice that the improvement of the algorithm given in Section 4 is much better, since the time-space product becomes $O(n \log n) \times O(\log r) = O(n \log n \log r)$.

If we compare the two methods given in this paper, we obtain a linear reduction in the running time by increasing the memory space by a logarithmic factor. This is due to the fact that this space allowed us to use more powerful techniques, such as divide and conquer. It would be interesting to study if the same result holds for other geometric problems (such as computing the Voronoi diagram, Delaunay triangulation or convex hull).

References

1. Asano, T., Mulzer, W., Rote, G., Wang, Y.: Constant-work-space algorithm for geometric problems. Submitted to Journal of Computational Geometry (2010)
2. Asano, T., Mulzer, W., Wang, Y.: Constant-work-space Algorithm for a Shortest Path in a Simple Polygon. In: Rahman, M. S., Fujita, S. (eds.) WALCOM 2010. LNCS, vol. 5942, pp. 9–20. Springer, Heidelberg (2010)
3. Asano, T., Rote, G.: Constant-working-space algorithms for geometric problems. In: CCCG, pp. 87–90 (2009)
4. Bose, P., Carmi, P., Hurtado, F., Morin, P.: A generalized Winternitz theorem. Journal of Geometry (in press)
5. Chan, T.M.: Comparison-based time-space lower bounds for selection. ACM Trans. Algorithms 6, 26:1–26:16 (2010)
6. Chan, T.M., Chen, E.Y.: Multi-pass geometric algorithms. Discrete & Computational Geometry 37(1), 79–102 (2007)
7. Devroye, L.: A note on the height of binary search trees. J. ACM 33, 489–498 (1986)
8. Frederickson, G.N.: Upper bounds for time-space trade-offs in sorting and selection. J. Comput. Syst. Sci. 34(1), 19–26 (1987)
9. Ghosh, S.: Visibility Algorithms in the Plane. Cambridge University Press, New York (2007)
10. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: SIGMOD, pp. 58–66 (2001)
11. Joe, B., Simpson, R.B.: Corrections to Lee’s visibility polygon algorithm. BIT Numerical Mathematics 27, 458–473 (1987), doi:10.1007/BF01937271
12. Munro, J.I., Paterson, M.: Selection and sorting with limited storage. Theor. Comput. Sci. 12, 315–323 (1980)
13. Munro, J.I., Raman, V.: Selection from read-only memory and sorting with minimum data movement. Theor. Comput. Sci. 165, 311–323 (1996)
14. O’Rourke, J.: Visibility. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 28, pp. 643–664. CRC Press, Inc. (2004)
15. Raman, V., Ramnath, S.: Improved Upper Bounds for Time-space Tradeoffs for Selection with Limited Storage. In: Arnborg, S. (ed.) SWAT 1998. LNCS, vol. 1432, pp. 131–142. Springer, Heidelberg (1998)
16. Seidel, R.: Convex hull computations. In: Handbook of Discrete and Computational Geometry, 2nd edn., ch. 22, pp. 495–512. CRC Press, Inc. (2004)

Minimizing Interference in Ad-Hoc Networks with Bounded Communication Radius

Matias Korman

Université Libre de Bruxelles (ULB), Brussels, Belgium
`mkormanc@ulb.ac.be`

Abstract. We consider a topology control problem in which we are given a set of n sensors in \mathbb{R}^d and we would like to assign a communication radius to each of them. The radii assignment must generate a strongly connected network and have low receiver-based interference (defined as the largest in-degree of the network). We give an algorithm that generates a network with $O(\log \Delta)$ interference, where Δ is the interference of a uniform-radius network. Since the radius of each sensor only depends on its neighbors, it can be computed in a distributed fashion. Moreover, this construction will never assign communication radius larger than R_{\min} to a sensor, where R_{\min} is the minimum value needed to obtain strong connectivity. We also show that $\Omega(\log n)$ interference is needed for some instances, making our algorithms asymptotically optimal.

1 Introduction

Ad-hoc networks are commonly used whenever a number of electronic devices are spread across a geographical area and no central communication hub exists [2][10]. In order to send messages between two sensors located far from each other, the message is repeated by other devices located between them. Due to technical constraints, the devices normally have very limited power sources (such as a small battery or solar cells). Since energy is the limiting factor for the operability of these networks, various methods have been proposed to reduce energy consumption [3][7][12].

In most cases the transmission radius is the major source of power dissipation in wireless networks. Another issue that strongly affects energy consumption is interference. Intuitively, the interference of a network is defined as the largest number of sensors that can directly communicate with a single point in the plane. Indeed, lowering the interference reduces the number of package collisions and saves considerable amounts of energy which would otherwise be spent in retransmission.

In this paper we look for an algorithm that assigns a transmission radius to a given list of sensors in a way that the network is connected and has low interference. This problem is considered one of the most relevant open optimization problems in the field [3]. Additionally we consider the case in which no sensor can be assigned a large radius; although theoretically one could assign an arbitrarily large radius to a sensor, in many cases this is not possible (due to

hardware constraints, environmental noise, quick battery drainage, etc.). Hence, the constraint in which no sensor can have a radius larger than a given constant comes as a natural extension of the problem.

2 Definitions and Results

We model each device as a point in \mathbb{R}^d (typically $d = 2$) and its transmission radius as a positive real value. Given a set S of sensors, we look for a radii assignment $r : S \rightarrow \mathbb{R}$. Any such assignment defines a graph $G_r = (S, E)$ with S as the ground set in which there is a directed edge $st \in E$ if and only if $r(s) \geq d(s, t)$, where $d(\cdot, \cdot)$ denotes the Euclidean distance. We note that in many cases symmetric communications have been considered essential to reduce protocol complexity. Since we consider that energy saving is the main focus of the problem, we will study the asymmetric case. Clearly, a requirement for any assignment r is that the associated graph is strongly connected (i.e., for any $u, v \in S$ there is a directed path from u to v in G_r). Whenever this happens, we say that r is *valid*.

In this paper, we use the *received based interference* model [12]. For a fixed radii assignment r , the *interference* $I(p)$ of any point $p \in \mathbb{R}^d$ is defined as the number of sensors that can directly communicate with p (that is, $I(p) = |\{s \in S \mid r(s) \geq d(s, p)\}|$). The *interference* of the network (or the radii assignment) is defined as the point in \mathbb{R}^d with largest interference.¹

A more geometric interpretation of the interference is as follows. For each sensor $s \in S$ place a disk of radius $r(s)$ centered at s . The interference of a point is equal to the depth of that point in the arrangement of disks (analogously, the interference of the network is equal to the depth of the deepest point). This model has been widely accepted, since it has been empirically observed that most of the package collisions happen at the receiver. The interested reader can check [10] or [12] to see other models of the interference.

Computing the radii assignment that minimizes the interference of a given point set is NP-hard. More specifically, Buchin [4] showed that it is NP-hard to obtain a valid radii assignment that minimizes the network's interference, or even approximate it with a factor of $4/3$ or less.² As a result, most of the previous research focuses in constructing valid networks with bounded interference, regardless of what the optimal assignment is for the given instance.

A simple approach to solve the problem is the uniform-radius network. In this network all sensors are given an equal radius R_{\min} , defined as the smallest possible radius needed in order to have a strongly connected network. Unfortunately it is easy to see that the interference of this approach, commonly denoted by Δ , can be as high as n (for example, a single point located far from a large cluster of points).

¹ In fact, the definition of [12] only measures interference at the sensors. The extension to measuring the interference to \mathbb{R}^d was done in [7].

² We note that Buchin's result only claims hardness for the undirected case. However, the exactly same construction also shows NP-hardness for the directed case as well.

For the undirected 1-dimensional case (or *highway model*), von Rickenbach *et al.* [12] gave an algorithm that constructs a network with $O(\sqrt{\Delta})$ interference (and showed that this algorithm approximates the minimum possible interference by a factor of $O(\sqrt[4]{\Delta})$). Afterwards Halldórsson and Tokuyama [7] generalized the construction to higher dimensions (although the approximation factor does not hold anymore). We note that the latter construction also has bounded radius. Specifically, the radii assignment r given in [7] satisfies $r(s) \leq \sqrt{d}R_{\min}$ for all sensors $s \in S$.

Less work has been done for the undirected case. To our knowledge, the only known result is the one of [12] in which a simpler problem called *all-to-one* was studied. In this variation, one would like to assign radii in a way that all sensors can communicate to a specific sensor $s \in S$ (observe that this is a weaker condition since strong connectivity implies this fact, but the reciprocal is not true).

In this work, we combine the above algorithms and give method to solve the d -dimensional directed interference problem. First, we show how to construct a strongly connected network with at most $O(\log n)$ interference. We then show that our algorithm is asymptotically optimal, since $\Omega(\log n)$ interference might be needed in some cases. Finally, in Section 4 we show that any radii assignment can be transformed to another one with (asymptotically speaking) the same interference and the additional property that no sensor is assigned a communication radius larger than R_{\min} . In our construction, the radius of a sensor is only affected by the sensors located in its neighborhood. As a result, this network can be constructed using only local information, even if the original assignment didn't have this property.

3 Layered Nearest Neighbor Algorithm

As a warm-up, we give a method to construct a network of low interference disregarding radii constraints. As done in [12], our algorithm repeatedly uses the Nearest Neighbor Graph $\text{NNG}(S)$ of a set S of points. It is well known that the $\text{NNG}(S)$ graph is not in general connected. Hence, we first study how these components behave.

Lemma 1. *Each weakly connected component of $\text{NNG}(S)$ is composed of two disjoint trees oriented towards their roots. Moreover, the nearest neighbor of each root is the opposite root.*

Proof. Recall that a weakly connected component of a graph G is a maximal subgraph of G such that for every pair of vertices u, v in the subgraph, there is an undirected path from u to v . For simplicity in the proof, we assume that the nearest neighbor of every sensor of S is unique. Whenever this situation does not hold, it suffices to set an arbitrary lexicographic order to the sensors of S and breaking ties by choosing the lexicographically smallest.

Let S' induce a weakly connected component of S and let p, q be the sensors that attain the smallest distance of S' (i.e., $d(p, q) = \min_{u, v \in S'} d(u, v)$). For any

point $p \in S$, let $n(p)$ be its nearest neighbor. Observe that $n(p) = q$ and $n(q) = p$ (i.e., they are the nearest neighbor of each other). These two sensors will be the two roots of S' . Since the out-degree of each vertex is exactly one, any other sensor of S' must connect to either p, q or another sensor already connected to either p or q . \square

Corollary 1. *A weakly connected component of $\text{NNG}(S)$ has size at least two.*

The above result gives the basic principle behind our algorithm. We start by computing $\text{NNG}(S)$ and assign the minimum possible radii to all points of S . In each component we discard all but one root and repeat the process on the non-discarded roots. This way we obtain different layers of nearest neighbor graphs, each time with a smaller point set. If at some point only one sensor s_0 remains in S , the algorithm assigns very large radius to s_0 and finishes.

Specifically, our algorithm proceeds as follows:

1. Assign $i := 0$ and $S_0 := S$.
2. Compute $\text{NNG}(S_i)$.
3. For each connected component of $\text{NNG}(S_i)$ select a root (arbitrarily). Let S_{i+1} be the set of selected roots.
4. For each $p \in S_i \setminus S_{i+1}$ assign $r(p) := d(p, n(p))$.
5. If $|S_{i+1}| = 1$, assign infinite radius to the point in it. Otherwise increase i by one and go to step 2.

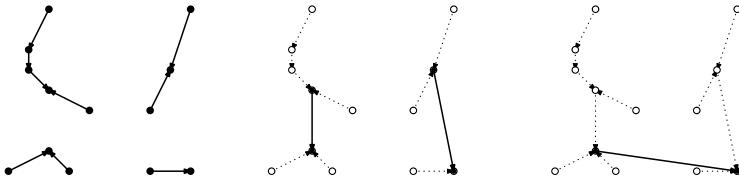


Fig. 1. Execution of the algorithm that constructs the LNN network (from left to right). In each step, all points (except the selected roots) are marked to their nearest neighbor and their radii are assigned. Points whose radii have been assigned are depicted as white points (and their edges are dashed).

We call the obtained network the *layered nearest network* of S (and denote it by $\text{LLN}(S)$ or simply LLN , see an example in Figure 1).

Lemma 2. *The LLN algorithm stops after at most $\log_2 n$ iterations. Moreover, the total running time is bounded by $O(n \log n)$.*

Proof. By Corollary 1, the size of S_i is at least halved in each iteration, hence at most $\log_2 n$ steps will be sufficient. The computationally most expensive part of the algorithm of each step is to compute the graph $\text{NNG}(S_i)$, which can be

done in $O(|S_i| \log |S_i|)$ time for any fixed dimension [5,11]. By summing the cost of each step, we can bound the total running time by

$$\sum_i O(|S_i| \log |S_i|) \leq \sum_{i=0}^{\log n} O\left(\frac{n}{2^i} \log\left(\frac{n}{2^i}\right)\right) < O(n \log n) \sum_{i=0}^{\infty} \frac{1}{2^i} = O(n \log n)$$

□

Given a sensor $s \in S$, we define its *level* as the largest index i such that $t \in S_i$. We now show that the network is valid and has low interference.

Lemma 3. *The LNN network is strongly connected.*

Proof. Let $k \leq \log_2 n$ be the total number of iterations of the algorithm. Recall that the set S_k consists of a single sensor s_k and that $r(s_k) = \infty$. In particular, there is a directed edge $s_k t$ for any $t \in S$. Thus, it suffices to show that for any $t \in S$, there exists a directed path to s_k .

Let $t \in S$ be any sensor, and let i be its index. If $i = k$ the claim is trivial, thus we focus on the $i < k$ case. By construction of the algorithm and Lemma 1, there exists a directed path from t to a sensor t' whose level is $i + 1$. By using this property $k - i$ times we obtain a directed path from t to s_k . □

Lemma 4. *For any fixed dimension, $p \in \mathbb{R}^d$ and $i \leq \log_2 n$, there exist at most $O(1)$ sensors v of level i such that $r(v) \geq d(p, v)$.*

Proof. This result follows from the fact that the in-degree I_d of any vertex of the NNG graph in \mathbb{R}^d is bounded by a constant that only depends on the dimension. The exact bound of the in-degree of the NNG is only known for small values of d , but some general upper bounds are known [9]. In particular, for $d = 2$ it is easy to see that the largest in-degree of a point is 5 (if the points are in general position), 6 otherwise.

Let p be any point in \mathbb{R}^d . We will show that for any $i \leq \log_2 n$, the sensors of level i can only contribute to at most I_d interference to p . Consider the graph $\text{NNG}(S_i \cup \{p\})$: by definition any sensor $s \in S_i$ contributes to the interference of p if and only if $d(s, p) \leq d(s, n(s))$. Observe that this is equivalent to the fact that the edge sp is in $\text{NNG}(S_i \cup \{p\})$. The Lemma follows from the fact that this can only happen at most I_d times. □

Corollary 2. *The interference of the LNN network is at most $O(\log n)$.*

By combining all the above results we obtain the following theorem.

Theorem 1. *For any set S of n points in \mathbb{R}^d , we can find a valid radii assignment whose interference is at most $O(\log n)$. This assignment can be computed in $O(n \log n)$ time using $O(n)$ space.*

The LNN gives the asymptotically smallest possible interference, since $\Theta(\log n)$ interference might be necessary in some cases.

Lemma 5. *For any $n > 0$ there exists a set of n points in which any valid radii assignment will have at least $\lfloor \log_2 n \rfloor - 1$ interference, even when $d = 1$.*

A very similar result for the *all-to-one* problem was given in [6]. Due to lack of space, the details needed to adapt the proof to our environment can be found in the extended version of this paper [8].

4 Bounded Radius Network

In this section we consider the case in which sensors have a hard cap on the maximum radius. If we are only interested in connectivity, the simplest approach is to consider the uniform-radius network. Recall that in this network all sensors are assigned the same radius R_{\min} defined as the minimum possible value so that the associated network is strongly connected. It is easy to see that R_{\min} is equal to the length of the longest edge of the minimum spanning tree of S . Let G_u be the network associated to the uniform radius network and let Δ be its interference (note that Δ can be as high as n).

The LNN network has low interference but may assign large radius to some sensors. Likewise, the uniform radius network assigns small radius to all sensors but might have high interference. In the following we will combine both ideas to create a network that has the nice properties of both. For the purpose we use a clustering technique similar to the approach used in [7]. As added benefits, we will be able to construct the network in a distributed fashion and reduce the interference to $O(\log \Delta)$.

In the following, we state the main result of this section.

Theorem 2. *Given a valid radii assignment algorithm \mathcal{A} that certifies that the interference of any set of n points is at most $i(n)$, there exists a radii assignment algorithm $\bar{\mathcal{A}}$ whose interference is at most $O(i(O(\Delta)) + 1)$. Moreover, no sensor of S is assigned a communication radius larger than R_{\min} in $\bar{\mathcal{A}}$.*

We first give an intuitive idea of our construction. The algorithm classifies the elements of S into clusters. For each cluster we select a constant number of sensors (which we call the *leaders* of the cluster) and assign them communication radius R_{\min} . The main property of this set is the fact that they are capable of sending messages to any other sensor of the cluster by only hopping through other leaders. By construction, any other sensor whose communication radius is R_{\min} will be able to communicate to a leader ℓ (and thus all other sensors of the cluster by hopping through ℓ). As a result, there is no need for any sensor to have communication radius strictly larger than R_{\min} . Finally, we will connect clusters in a way that interference will not grow except by a constant value.

More formally, the algorithm is as follows. Virtually partition the plane into d -dimensional cubes of side length R_{\min} (each of these cubes will be referred as a *bucket*). For each bucket B , let S_B be the sensors inside B (i.e., $S_B = S \cap B$). Without loss of generality we can assume that no point lies in two buckets (this can be obtained by doing a symbolic perturbation of the point set). We say that

two sensors $s, t \in S$ belong to the same cluster if and only if they belong to the same bucket B and there is a path connecting them in the subgraph $G_u[S_B]$, where $G[S']$ denotes the subgraph of a graph $G = (S, E)$ induced by a subset of vertices $S' \subseteq S$.

Lemma 6. *For any fixed dimension d , there can be at most $O(1)$ clusters inside any bucket. Moreover, in each cluster there are at most $O(\Delta)$ sensors.*

Proof. Virtually partition B into d^d cubes of side length R_{\min}/d . Since all sub-buckets have side length R_{\min}/d , the largest distance between any two sensors in the sub-bucket is $\sqrt{d}R_{\min}/d = R_{\min}/\sqrt{d} \leq R_{\min}$. If there exists a bucket B with more than d^d clusters, we use the pigeonhole principle and obtain that there must exist at least one sub-bucket with sensors of two different clusters. In particular, there will be an edge in $G_u[S_B]$ that connects these sensors. This contradicts with the definition of cluster, since they belong to the same bucket. Proof for the second claim is identical (this time we would find a sensor whose interference in G_u is larger than Δ). \square

The set of leaders of a given cluster c inside a bucket B is constructed as follows: virtually partition B into d^d squares of side length R_{\min}/d . If all the points belong to the same sub-bucket, we pick any point as the leader. Otherwise, for any two different sub-buckets we pick any edge e in $G_u[c]$ connecting the two sub-buckets (if any exists) and add the vertices of e to the set of leaders. We repeat this process for all pairs of sub-buckets and obtain the set of leaders.

Lemma 7. *For any cluster c , its associated set L_c of leaders has constant size. Moreover, for any sensor $s \in c$, there exists a leader $\ell_s \in L_c$ such that $d(s, \ell_s) \leq R_{\min}$.*

Proof. For every pair of sub-buckets occupied by sensors of c , two sensors are added into L_c . Since a bucket is partitioned into d^d sub-buckets, at most $2 \times \binom{d^d}{2}$ sensors will be present in L_c (a constant for any fixed dimension). In order to show the second claim, we will show that for any sensor s there exists a leader ℓ_s that belongs to the same sub-bucket. Combining this result with the fact that the largest distance between any two points inside the same sub-bucket is at most R_{\min}/\sqrt{d} will give the lemma. If all sensors are located in a single sub-bucket, the claim is trivially true. Otherwise, for any given sensor $s \in c$, let $t \in c$ be any sensor located in a different sub-bucket, $\pi = (s = v_0, \dots, v_k = t)$ any path connecting them in $G_u[c]$, and let $i > 0$ be the smallest index such that v_i does not belong to the same sub-bucket as s . By definition of leaders, there must be a sensor of the sub-bucket in which v_{i-1} belongs to in L_c (since the edge $v_{i-1}v_i$ is present in $G_u[c]$ and the two sensors belong to different sub-buckets). In particular, this leader will be within R_{\min} communication distance to v_{i-1} (and s , since they belong to the same sub-bucket). \square

We denote by $r_{\mathcal{A}(S)}$ the radii assignment that algorithm \mathcal{A} gives to S (that is, $r_{\mathcal{A}(S)} : S \rightarrow \mathbb{R}$). Also, let $G_{r_{\mathcal{A}(S)}}$ be the network that \mathcal{A} constructs for a set S

of sensors. We assign radius to all sensors s of cluster c as follows. If $s \in L_c$, we assign radius R_{\min} . Otherwise, we assign radius equal to $\min\{r_{\mathcal{A}(c)}(s), R_{\min}\}$. Finally, we add a small modification to certify connectivity between clusters. For any two clusters c, c' , we pick any edge $uv \in G_u$ such that $u \in c, v \in c'$ and $d(u, v) \leq R_{\min}$ (if any exists) and increase the radius of both u and v to R_{\min} (if the assigned radius was smaller). Let $r_{\bar{\mathcal{A}}(S)}$ be the obtained radii assignment and $G_{r_{\bar{\mathcal{A}}(S)}}$ its associated network.

Lemma 8. $G_{r_{\bar{\mathcal{A}}(S)}}$ is strongly connected.

Proof. We start by showing that any leader ℓ of a cluster c can send messages to any other sensor $s \in c$, even if we only allow hopping through leaders. Let $\pi = (\ell = v_0, \dots, v_k = s)$ be the shortest path connecting them in $G_u[c]$. For any $i \leq k$, let b_i be the sub-bucket to which sensor v_i belongs to. By construction of the set of leaders, each time we have $b_i \neq b_{i+1}$, there exist two leaders ℓ'_i and ℓ'_{i+1} that are in sub-buckets b_i and b_{i+1} , respectively. Moreover, the leaders also satisfy $d(\ell'_i, \ell'_{i+1}) \leq R_{\min}$. Whenever $b_i = b_{i+1}$, we simply set $\ell'_i = \ell_i$ and $\ell'_{i+1} = \ell'_i$ (and $\ell'_0 = \ell$). Observe that for any $i > 0$, the sensors ℓ_i and ℓ'_i are leaders. In particular, there exist a direct edge connecting ℓ_i and ℓ'_i as well as ℓ'_i and ℓ_{i+1} . Hence, the path $\pi' = (\ell = \ell'_0, \ell_1, \ell'_1, \dots, \ell'_{k-1}, \ell_k, v_k = s)$ will be feasible in $G_{r_{\bar{\mathcal{A}}(S)}}$.

We now show that $G_{r_{\bar{\mathcal{A}}(S)}}$ is strongly connected. For any two sensors $s, t \in S$, let $\pi = (s = v_0, \dots, v_k = t)$ be the shortest path connecting them in $G_{r_{\bar{\mathcal{A}}(S)}}$. We will show that, for any $i < k$, sensor v_i can send messages to v_{i+1} . By construction of the network, the only situation in which the edge $v_i v_{i+1}$ is not present in $G_{\bar{\mathcal{A}}(S)}$ is when the radius of v_i was larger than R_{\min} and we reduced it to R_{\min} . By Lemma 7, sensor v_i will be able to send messages to some leader $\ell \in L_c$.

We now distinguish two cases according to whether or not v_i and v_{i+1} belong to the same cluster. In the first case, we use the above reasoning and obtain a path connecting ℓ and v_{i+1} , which combined with the link $v_i \ell$ gives a directed path from v_i to v_{i+1} . In the latter case by construction of $G_{r_{\bar{\mathcal{A}}(S)}}$, there will be two sensors u, w that belong to the clusters of v_i and v_{i+1} and satisfy $d(u, w) \leq R_{\min}$. By concatenating the paths from v_i to ℓ , ℓ to u and u to w , sensor v_i is capable of reaching w . Observe now that $r_{\bar{\mathcal{A}}(S)}(w) = R_{\min}$ and w is in the same cluster as v_{i+1} , thus by repeating the above argument will be able to communicate with v_{i+1} .

□

Lemma 9. $G_{\bar{\mathcal{A}}(S)}$ has $O(i(O(\Delta)) + 1)$ interference.

Proof. Consider first a single cluster c and recall that, by Lemma 6, it has at most $O(\Delta)$ sensors. Hence, if we use the radii assignment $r_{\mathcal{A}(c)}$, we would obtain that c can contribute at most $i(O(\Delta))$ interference to any point p in \mathbb{R}^d . We now look at the sensors s whose radii were increased in the execution of our algorithm. This happens in only if (i) s is a leader of c or (ii) the radius of s was increased to have connectivity between clusters. By Lemma 7, the set L_c

has constant size thus case (i) can only happen a constant number of times. Case (ii) can only happen when we connect two clusters that share an edge in G_u . Observe that this can only happen to clusters that are in the same or a neighboring bucket of the one containing c . There are exactly 3^d such buckets (a constant for fixed dimension). Since, by Lemma 6, each bucket can have a constant number of clusters, the total number of times that the case (ii) can happen is also bounded by a constant that only depends on the dimension.

Since the two cases in which the radius of a sensor is increased only happen a constant number of times, any cluster can only contribute $i(O(\Delta)) + O(1)$ to the total interference of a point. Since no sensor is assigned radius larger than R_{\min} , only sensors that are located in the bucket containing p or any of the neighboring buckets can affect to the interference of p . Using again Lemma 6, we obtain that $p \in \mathbb{R}^d$ can only be reached by a constant number of clusters. \square

This completes the proof of Theorem 2. Combining this result with the *NN* network we obtain a network with low interference and bounded radii.

Theorem 3. *For any set S of n points in \mathbb{R}^d , there exists a valid radii assignment r of $O(\log \Delta)$ interference that satisfies $r(s) \leq R_{\min}$ for all $s \in S$. Moreover, such assignment can be computed in $O(T(n))$ time, where $T(n)$ is the time needed to compute the minimum spanning tree of a set of n points in \mathbb{R}^d .*

Due to space constraints, proof of this Theorem is deferred to the extended version of this paper [8]. The key observation is the fact that we can avoid explicitly constructing G_u and use the minimum spanning tree instead.

Remark. The currently best known algorithms that compute the minimum spanning tree run in $O(n \log n)$ time (for $d = 2$), $O((n \log n)^{4/3})$ (for $d = 3$) or $O(n^{2 - \frac{2}{(d/2+1)\epsilon}})$ (for higher dimensions) [1].

Also, observe that the radius assigned to a sensor s in this algorithm only depends on the sensors the bucket containing s and its neighboring buckets. As a result, each sensor can compute its communication radius using of only local information, provided that the value of R_{\min} is known in advance. Unfortunately, the value of R_{\min} is a global property that cannot be easily computed in a distributed environment. Whenever this value is unknown, it can be replaced by any larger value (like for example the largest possible communication radius). By doing so we retain the local construction property and only increase the interference from $O(\log \Delta)$ to $O(\log n)$.

5 Conclusion

In this paper we have given a method to construct a strongly connected network. In addition to having low interference, it has bounded maximum radius and can be constructed in a distributed fashion. Matching lower bounds for both the interference and maximum radius are also given, hence the algorithm is asymptotically optimal in both criteria. We also note that, although the LNN

construction of Section 3 does not hold in undirected networks, the clustering technique of Section 4 can be used in undirected networks. It remains open to find an algorithm that approximates the interference of the optimal network (similar to the one found in [12] for the one-dimensional directed case).

The key property of our clustering approach is the fact that we can construct a set of constant size that forms a connected dominating set of $G_u[c]$. Although it is possible to reduce the size of such a set (indeed for $d = 2$ any cluster can be dominated with at most 5 sensors), the exponential dependency on d cannot be avoided. For example, consider the case in which $R_{\min} = 1$ and there exists a cluster with many sensors covering a unit hypercube. Any dominating set of that cluster must cover the whole cube. Since the ratio between the unit cube and the unit ball is $\frac{(d/2)!}{\pi^{d/2}} \approx \frac{\sqrt{\pi} d^{d/2}}{(2e\pi)^{d/2}}$, at least such many sensors will be needed to dominate the cluster.

References

1. Agarwal, P., Edelsbrunner, H., Schwarzkopf, O., Welzl, E.: Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.* 6, 407–422 (1991)
2. Benkert, M., Gudmundsson, J., Haverkort, H., Wolff, A.: Constructing minimum-interference networks. *Comput. Geom. Theory Appl.* 40(3), 179–194 (2008)
3. Bilò, D., Proietti, G.: On the complexity of minimizing interference in ad-hoc and sensor networks. *Theor. Comput. Sci.* 402(1), 43–55 (2008)
4. Buchin, K.: Minimizing the maximum interference is hard. *CoRR*, abs/0802.2134 (2008)
5. Clarkson, K.L.: Fast algorithms for the all nearest neighbors problem. In: Proceedings of the 24th Annual Symposium on Foundations of Computer Science, pp. 226–232. IEEE Computer Society, Washington, DC (1983)
6. Fussen, M., Wattenhofer, R., Zollinger, A.: Interference arises at the receiver. In: In Proceedings of Int. Conference on Wireless Networks, Communications, and Mobile Computing, WIRELESSCOM (2005)
7. Halldórsson, M., Tokuyama, T.: Minimizing interference of a wireless ad-hoc network in a plane. *Theor. Comput. Sci.* 402(1), 29–42 (2008)
8. Korman, M.: Minimizing interference in ad-hoc networks with bounded communication radius. *CoRR*, abs/1102.2785 (2011)
9. Paterson, M., Yao, F.F.: On Nearest-Neighbor Graphs. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 416–426. Springer, Heidelberg (1992)
10. Santi, P.: Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.* 37(2), 164–194 (2005)
11. Vaidya, P.M.: An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.* 4, 101–115 (1989)
12. von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Algorithmic models of interference in wireless ad hoc and sensor networks. *IEEE/ACM Trans. Netw.* 17(1), 172–185 (2009)

Hamiltonian Paths in the Square of a Tree

Jakub Radoszewski¹ and Wojciech Rytter^{1,2,*}

¹ Department of Mathematics, Computer Science and Mechanics,
University of Warsaw, Warsaw, Poland

{jrad,rytter}@mimuw.edu.pl

² Faculty of Mathematics and Informatics,
Copernicus University, Toruń, Poland

Abstract. We introduce a new family of graphs for which the Hamiltonian path problem is non-trivial and yet has a linear time solution. The square of a graph $G = (V, E)$, denoted as G^2 , is a graph with the set of vertices V , in which two vertices are connected by an edge if there exists a path of length at most 2 connecting them in G . Harary & Schwenk (1971) proved that the square of a tree T contains a Hamiltonian cycle if and only if T is a caterpillar, i.e., it is a single path with several leafs connected to it. Our first main result is a simple graph-theoretic characterization of trees T for which T^2 contains a Hamiltonian path: T^2 has a Hamiltonian path if and only if T is a horsetail (the name is due to the characteristic shape of these trees, see Figure 1). Our next results are two efficient algorithms: linear time testing if T^2 contains a Hamiltonian path and finding such a path (if there is any), and linear time preprocessing after which we can check for any pair (u, v) of nodes of T in constant time if there is a Hamiltonian path from u to v in T^2 .

Keywords: tree, square of a graph, Hamiltonian path.

1 Introduction

Classification of graphs admitting Hamiltonian properties is one of the fundamental problems in graph theory, no good characterization of hamiltonicity is known. There are multiple results dealing with algorithms for finding Hamiltonian cycles in particular families of graphs, see e.g. [2].

The k -th power of a graph $G = (V, E)$, denoted as G^k , is a graph over the set of vertices V , in which the vertices $u, v \in V$ are connected by an edge if there exists a path from u to v in G of length at most k . The graph G^2 is called the square of G , while G^3 is called the cube of G .

Hamiltonian cycles in powers of graphs have received considerable attention in the literature. The classical result in this area, by Fleischner [3], is that the square of every 2-connected graph has a Hamiltonian cycle, see also the alternative proofs [24][12]. Afterwards Hendry & Vogler [7] proved that every connected $S(K_{1,3})$ -free graph has a Hamiltonian square (where $S(K_{1,3})$ is the subdivision graph of $K_{1,3}$), and later Abderrezak et al. [1] proved the same result for graphs

* The author is supported by grant no. N206 566740 of the National Science Centre.

satisfying a weaker condition. As for the cubes of graphs, Karaganis [8] and Sekanina [10] proved that G^3 is Hamiltonian if and only if G is connected. Afterwards, Lin & Skiena [9] gave a linear time algorithm finding a Hamiltonian cycle in the cube of a graph. The analysis of Hamiltonian properties of squares of graphs was also extended to powers of infinite graphs (locally finite graphs), see [5,10,11].

In this paper we consider Hamiltonian properties of powers of trees. Harary & Schwenk [6] proved that the square of a tree T has a Hamiltonian cycle if and only if T is a caterpillar, i.e., it is a single path with several leafs connected to it. On the other hand, the k -th power of every tree is Hamiltonian for any $k \geq 3$ [9,10]. However, there was no previous characterization of trees with squares containing Hamiltonian paths. We introduce a class of trees T , called here (u, v) -horsetails, such that T^2 contains a Hamiltonian path connecting a given pair of nodes u, v if and only if T is a (u, v) -horsetail. We provide linear time algorithms finding Hamiltonian paths in squares of trees.

2 Caterpillars and Horsetails

We say that P is a k -path in G if P is a path in G^k , similarly we define a k -cycle. Additionally define a kH -path and a kH -cycle as a Hamiltonian path and a Hamiltonian cycle in G^k respectively. If G^k contains a kH -cycle (a kH -path respectively) we call G k -Hamiltonian (k -traceable respectively).

A tree T is called a *caterpillar* if the subtree of T obtained by removing all the leafs of T is a path B (possibly an empty path). A caterpillar is called non-trivial if it contains at least one edge. We call the path $B = (v_1, \dots, v_l)$ the *spine* of the caterpillar T . By $\text{Leafs}(v_i)$ we denote the set of leafs connected to the node v_i . A tree has a $2H$ -cycle if and only if it is a caterpillar [9], see Fig. 5.

We proceed to the classification of 2-traceable trees. Let T be a tree and let $u, v \in V$ be two of its nodes, $u \neq v$. Let

$$P = (u = u_1, u_2, \dots, u_{k-1}, u_k = v)$$

be the path in T connecting u and v . We call the nodes in P the *main* nodes, and the neighbors of the nodes in P from the set $V \setminus P$ are called *secondary* nodes. By $\text{Layer}(u_i)$, $i = 1, \dots, k$, we denote the connected component containing u_i in $T \setminus (P \setminus \{u_i\})$.

Each main node u_i for which $\text{Layer}(u_i)$ is a caterpillar can be classified as:

- type A:** at most one component of $\text{Layer}(u_i) \setminus \{u_i\}$ is a non-trivial caterpillar
- type B:** exactly two components of $\text{Layer}(u_i) \setminus \{u_i\}$ are non-trivial caterpillars
- free node:** $\text{Layer}(u_i) \setminus \{u_i\}$ is empty.

In other words, u_i is a type A node if u_i is an endpoint of the spine of the caterpillar $\text{Layer}(u_i)$ or a leaf adjacent to the endpoint of the spine. The node u_i is a type B node if u_i is an inner node of the spine of the caterpillar $\text{Layer}(u_i)$. Also note that every free node is a type A node.

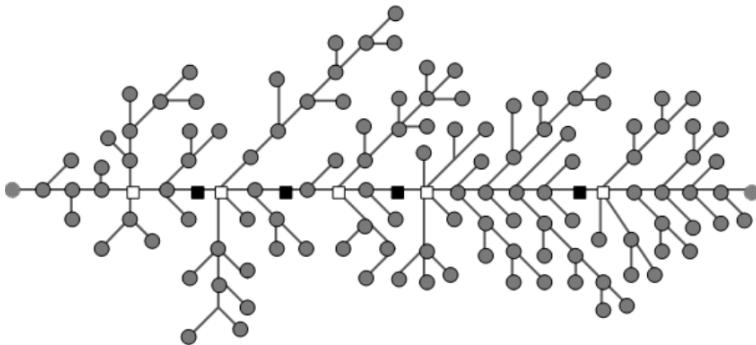


Fig. 1. An example (u, v) -horsetail. White squares represent type B nodes, black squares represent free nodes. The nodes u and v are the leftmost and rightmost nodes on the horizontal path (the main path), they are also free nodes (in this example, in general case they can be non-free).

By $u_{i_1}, u_{i_2}, \dots, u_{i_m}$, $1 \leq i_1 < i_2 < \dots < i_m \leq k$, we denote all the type B nodes in T , additionally define $i_0 = 0$ and $i_{m+1} = k + 1$. The tree T is called a (u, v) -horsetail if

- (\star) every main node u_i is a type A or a type B node, and
- ($\star\star$) for every $j = 0, \dots, m$, at least one of the nodes $u_{i_j+1}, u_{i_j+2}, \dots, u_{i_{j+1}-1}$ is a free node.

A tree is a *horsetail* if it is a (u, v) -horsetail for some u, v .

The condition ($\star\star$) can also be formulated as follows:

- between any two type B nodes in P there is at least one free node, and
- there is at least one free node located before any type B node in P , and
- there is at least one free node located after any type B node in P , and
- there is at least one free node in P .

In particular, the nodes u_1 and u_k must be type A nodes. See Fig. 1 and 2 for examples of horsetails and trees which are not horsetails.

In the next section we show that T contains a 2H-path connecting the nodes u and v if and only if T is a (u, v) -horsetail, see also Fig. 3. We present a linear time algorithm finding a 2H-path from u to v in a (u, v) -horsetail. Afterwards, in Section 4 we provide a linear preprocessing time algorithm which enables constant time queries for existence of a 2H-path from u to v . We also obtain a linear time test if T is 2-traceable.

3 2H-Paths in (u, v) -Horsetails

In this section we describe a linear time algorithm finding a 2H-path from u to v in a (u, v) -horsetail. We then show that a tree contains a 2H-path from u to v if and only if it is a (u, v) -horsetail.

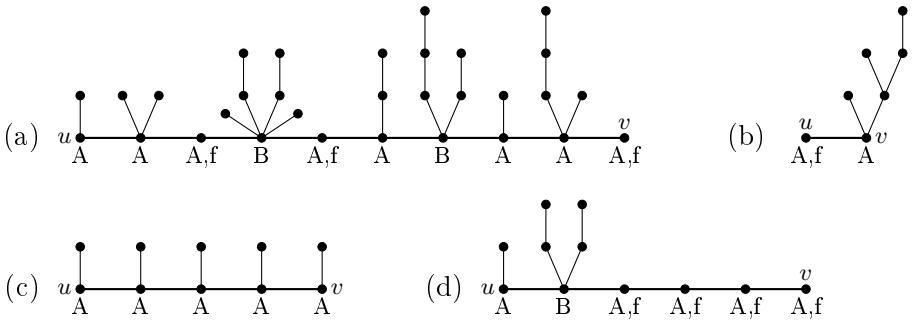


Fig. 2. (a) and (b): The main paths (P) of example horsetails; A, B and f denote a type A node, type B node and a free node respectively. (c) and (d): The main paths of example trees which are not (u, v) -horsetails.

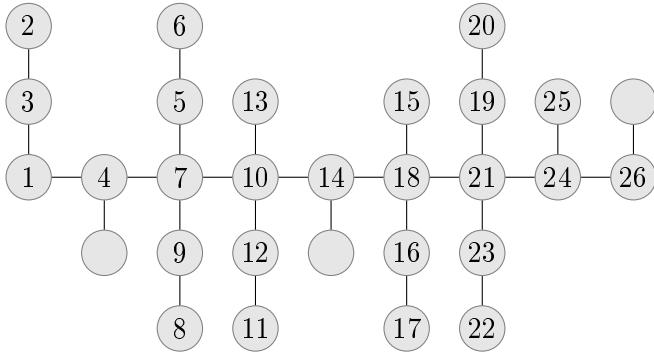


Fig. 3. The tree without the unlabeled nodes is a $(1, 26)$ -horsetail having a 2H-path $1, 2, 3, \dots, 26$. If any of the unlabeled nodes is present in the tree, the tree is not a $(1, 26)$ -horsetail and does not contain any 2H-path from 1 to 26.

For a 2-path S , by $\text{first}(S)$ and $\text{last}(S)$ we denote the first and the last node in S . Let S be a 2H-path from u to v in a (u, v) -horsetail T . We say that S is a *layered* 2H-path if the nodes of T are visited in S in the order of the subtrees $\text{Layer}(u_i)$, i.e., first all the nodes of $\text{Layer}(u_1)$ are visited in some order, then all the nodes of $\text{Layer}(u_2)$ etc. Then we can divide S into parts (S_1, S_2, \dots, S_k) , such that every S_i contains only nodes from $\text{Layer}(u_i)$. Note that, for any i , each of the nodes $\text{first}(S_i), \text{last}(S_i)$ is either a main node or a secondary node.

The following algorithm *2H-path-horsetail* finds a layered 2H-path in a (u, v) -horsetail T , see Fig. 4. Informally, the algorithm works as follows.

In each layer $\text{Layer}(u_i)$ a 2H-path S_i is found, using an auxiliary routine *2H-path-caterpillar*($\text{Layer}(u_i), \text{first}, \text{last}$), and appended to the constructed 2H-path S . The algorithm constructs the 2H-path in a greedy manner: for each i , $\text{last}(S_i) = u_i$, if only this is possible to achieve. If so, we say that after the i -th layer the algorithm is in the *main* phase, otherwise it is in the *secondary* phase.

In the algorithm we use auxiliary functions:

- *any-secondary-node*($\text{Layer}(w)$), which returns an arbitrary secondary node from the layer $\text{Layer}(w)$;
- *a-secondary-node-preferably-leaf*($\text{Layer}(w)$), which finds a secondary node being a leaf in $\text{Layer}(w)$, if such a secondary node exists, and any secondary node from this layer otherwise;
- *another-secondary-node*($\text{Layer}(w)$), which returns a secondary node from $\text{Layer}(w)$ different from the secondary node returned by the previous *a-secondary-node-preferably-leaf*($\text{Layer}(w)$) call.

Algorithm. 2H-path-horsetail(T, u, v)

```

1: Compute the main nodes  $u_1, \dots, u_k$  and the layers  $\text{Layer}(u_1), \dots, \text{Layer}(u_k)$ ;
2: first :=  $u_1$ ;
3: if free( $u_1$ ) then last :=  $u_1$ 
4: else last := any-secondary-node( $\text{Layer}(u_1)$ );
5:  $S := 2\text{H-path-caterpillar}(\text{Layer}(u_1), \text{first}, \text{last})$ ;
6: for  $i := 2$  to  $k$  do
7:    $w := u_i$ ;
8:   if  $w$  is a free node then
9:      $\text{first} := \text{last} := w$ ;
10:    else if  $last$  is a secondary node then
11:       $\text{first} := w$ ;
12:       $last := a\text{-secondary-node-preferably-leaf}(\text{Layer}(w))$ ;
13:    else
14:       $\text{first} := a\text{-secondary-node-preferably-leaf}(\text{Layer}(w))$ ;
15:       $last := w$ ;
16:    if 2H-path-caterpillar( $\text{Layer}(w), \text{first}, \text{last}$ ) = NONE then
17:       $last := another\text{-secondary-node}(\text{Layer}(w))$ ;
18:     $S := S + 2\text{H-path-caterpillar}(\text{Layer}(w), \text{first}, \text{last})$ ;
19: return  $S$ ;
```

Fig. 4. Finding a 2H-path in a horsetail in linear time

Let us investigate how the algorithm works for respective types of nodes u_i . We have $\text{first}(S_1) = u_1$; the 2-path S_1 ends in a secondary node in $\text{Layer}(u_1)$ if only u_1 is not free. Otherwise, obviously, $\text{last}(S_1) = u_1$. Hence, after $\text{Layer}(u_1)$ the algorithm is in the main phase only if u_1 is free.

Afterwards, whenever u_i is a *free* node, we simply visit it, with $\text{first}(S_i) = \text{last}(S_i) = u_i$. Hence, after any free node the algorithm switches to the main phase.

Otherwise, if u_i is a *type A* node, we have two cases. If the algorithm was in the main phase then we can choose any secondary node for $\text{first}(S_i)$ and finish the 2H-path in $\text{Layer}(u_i)$ in $\text{last}(S_i) = u_i$ — this is the desired configuration. On the

other hand, if $\text{last}(S_{i-1})$ is a secondary node, then we must have $\text{first}(S_i) = u_i$ and we finish in a secondary node. Thus, visiting a non-free type A node does not alter the phase of the algorithm.

Now let u_i be a *type B* node. Due to the (**) condition of the definition of a horsetail, before entering $\text{Layer}(u_i)$ the algorithm is in the main phase. Thus we can choose $\text{first}(S_i)$ as a secondary node. There is no 2H-path S_i ending in u_i in this case (the proof follows), therefore $\text{last}(S_i)$ is another secondary node in $\text{Layer}(u_i)$. Hence, a type B node changes the main phase into the secondary phase.

Finally, if u_k is a free node then $\text{first}(S_k) = \text{last}(S_k) = u_k$ and this ends the path S . Otherwise, we must have $\text{last}(S_k) = u_k$, hence $\text{first}(S_k)$ must be a secondary node and after the $(k-1)$ -th layer the algorithm must be in the main phase. This is, however, guaranteed by the (**) condition of the definition of a horsetail (see also the alternative formulation of the condition).

Thus we obtain the correctness of the algorithm provided that the requested 2H-paths in the caterpillars $\text{Layer}(u_i)$ exist. We conclude the analysis with the following Theorem I. In its proof we utilize the following auxiliary lemma, we omit its proof in this version of the paper.

Lemma 1. *Every 2H-cycle in a caterpillar with the spine $B = (v_1, \dots, v_l)$, $l \geq 1$, and the leafs $\text{Leafs}(v_1), \dots, \text{Leafs}(v_l)$, has the form*

$$\begin{aligned} v_1 & P_2 v_3 \dots v_{l-1} P_l v_l P_{l-1} \dots P_3 v_2 P_1 \quad \text{for } 2 \mid l \\ v_1 & P_2 v_3 \dots P_{l-1} v_l P_l v_{l-1} \dots P_3 v_2 P_1 \quad \text{for } 2 \nmid l \end{aligned}$$

where P_i , for $i = 1, \dots, l$, is an arbitrary permutation of the set $\text{Leafs}(v_i)$.



Fig. 5. To the left: a 2H-cycle $1, 2, \dots, 8$ in a caterpillar with the spine $1-3-7-5$ of even length. To the right: a 2H-cycle $1, 2, \dots, 10$ in a caterpillar with an odd spine $1-3-9-5-7$.

Theorem 1. *Assume that T is a (u, v) -horsetail. Then the algorithm 2H-path-horsetail(T, u, v) finds a layered 2H-path from u to v in linear time.*

Proof. It suffices to prove that the 2H-path-caterpillar procedure returns the 2H-paths as requested, depending on the type of the node u_i .

If u_i is a type A node (in particular, if $i = 1$ or $i = k$) then the arguments of the 2H-path-caterpillar call are u_i and a secondary node in $\text{Layer}(u_i)$. In this

case, u_i is either an endpoint of the spine of the caterpillar $\text{Layer}(u_i)$ or a leaf of $\text{Layer}(u_i)$ connected to an endpoint of the spine. Let v_1, \dots, v_l be the spine nodes of $\text{Layer}(u_i)$; we assume that $l > 0$, since otherwise the conclusion is trivial. Thus, we are looking for a 2H-path in $\text{Layer}(u_i)$ connecting the nodes v_1 and w for some $w \in \text{Leafs}(v_1)$, or the nodes v_l and w for some $w \in \text{Leafs}(v_l)$. Note that we could also have the pair of nodes v_1 and v_2 or v_{l-1} and v_l here, however these cases are eliminated in the algorithm by choosing *a-secondary-node-preferably-leaf*. Equivalently, we are looking for a 2H-cycle in $\text{Layer}(u_i)$ containing an edge between the nodes v_1 and w or v_l and w . By Lemma 11 for any $w \in \text{Leafs}(v_1)$ ($w \in \text{Leafs}(v_l)$ respectively) there exists a 2H-cycle in $\text{Layer}(u_i)$ containing the edge v_1w (v_lw respectively), and this 2H-cycle can be found in linear time w.r.t. the size of $\text{Layer}(u_i)$. This concludes that the algorithm works correctly in this case.

Now consider any type B node u_i . We will also use the denotation v_1, \dots, v_l for the spine nodes of $\text{Layer}(u_i)$, we have $l \geq 3$. This time u_i is an inner spine node of the caterpillar $\text{Layer}(u_i)$, $u_i = v_j$ for some $1 < j < l$. In the algorithm we first try to find a 2H-path in $\text{Layer}(u_i)$ connecting this node with a secondary node w , i.e., one of the neighbours of v_j in $\text{Layer}(u_i)$, $w \in \{v_{j-1}, v_{j+1}\} \cup \text{Leafs}(v_j)$. Equivalently, we search for a 2H-cycle in $\text{Layer}(u_i)$ containing an edge v_jw . By Lemma 11 such a 2H-cycle does not exist. Thus, the condition in line 16 of the algorithm is false and we are now looking for a 2H-path in $\text{Layer}(u_i)$ connecting two secondary nodes, which are distance-2 nodes in $\text{Layer}(u_i)$. This is equivalent to finding a 2H-cycle containing the corresponding edge of $\text{Layer}(u_i)^2$. We need to consider a few cases: the node v_{j-1} and a leaf from $\text{Leafs}(v_j)$, a symmetric case: v_{j+1} and a leaf from $\text{Leafs}(v_j)$, finally v_{j-1} and v_{j+1} . Note that the last case (v_{j-1} and v_{j+1}) is valid only if $\text{Leafs}(v_j) = \emptyset$, this is due to the choice of *a-secondary-node-preferably-leaf* in line 14. Now it suffices to note that in all the cases the corresponding 2H-cycle exists and can be found efficiently. Indeed, the cycle from Lemma 11 is of the form $\dots v_{j+1}P_jv_{j-1} \dots$ where P_j is any permutation of the set $\text{Leafs}(v_j)$. This concludes that also type B nodes are processed correctly. \square

Our next goal is to show that T has a 2H-path from u to v if and only if T is a (u, v) -horsetail. First, we note that it is sufficient to consider layered 2H-paths in trees. This is stated formally as the following lemma, we omit the proof in this version of the paper.

Lemma 2. *If T has a 2H-path from u to v then T has a layered 2H-path from u to v .*

This fact already implies that each of the components $\text{Layer}(u_i)$ is a caterpillar.

Lemma 3. *If T has a layered 2H-path from u to v then each of the components $\text{Layer}(u_i)$ is 2-Hamiltonian.*

Proof. Let S be a layered 2H-path from u to v in T . Then the 2-path S_i is a 2H-path in $\text{Layer}(u_i)$. Recall that $\text{first}(S_i)$, $\text{last}(S_i)$ are either main or secondary nodes in $\text{Layer}(u_i)$. Every pair of such nodes are adjacent in the square

of $\text{Layer}(u_i)$, hence using an additional edge S_i can be transformed into a 2H-cycle in $\text{Layer}(u_i)$, so $\text{Layer}(u_i)$ is 2-Hamiltonian. \square

In the following two lemmas we show that every tree having a 2H-path satisfies the conditions (\star) and $(\star\star)$ of a horsetail.

Lemma 4. *If a tree T contains a 2H-path connecting the nodes u and v then T satisfies the condition (\star) of a (u, v) -horsetail.*

Proof. Assume to the contrary that a node u_i is neither a type A nor a type B node. By Lemma 3, the layer $\text{Layer}(u_i)$ is a caterpillar. Let v_1, \dots, v_l be the spine nodes of $\text{Layer}(u_i)$. Then u_i corresponds to some node $w \in \text{Leafs}(v_j)$, for $1 < j < l$. Hence, v_j is the only secondary node in $\text{Layer}(u_i)$. By Lemma 2, S is a layered 2H-path, $S = (S_1, \dots, S_k)$. The 2-path S_i is a 2H-path in $\text{Layer}(u_i)$ connecting the nodes v_j and w , since both $\text{first}(S_i)$ and $\text{last}(S_i)$ must be main or secondary nodes of $\text{Layer}(u_i)$. Thus S_i is also a 2H-cycle in $\text{Layer}(u_i)$. However, by Lemma 1, no such 2H-cycle exists in a caterpillar, a contradiction. \square

One can prove the necessity of the $(\star\star)$ condition by showing that after visiting a free node u_i any layered 2H-path in T is in the main phase, that otherwise a type A node keeps the phase of the 2H-path unaltered and that a type B node always changes the main phase into the secondary phase, see Fig. 6.

Lemma 5. *If a tree T contains a 2H-path connecting the nodes u and v then T satisfies the condition $(\star\star)$ of a (u, v) -horsetail.*

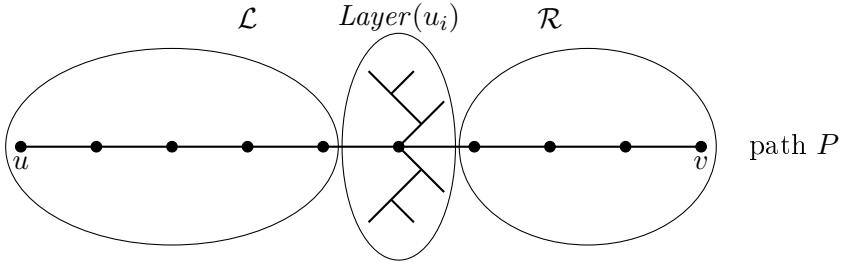


Fig. 6. Let S_L , S_i and S_R be the parts of a layered 2H-path S corresponding to \mathcal{L} , $\text{Layer}(u_i)$ and \mathcal{R} . If u_i is a type B node then $\text{last}(S_L) \in P$, $\text{first}(S_i), \text{last}(S_i) \notin P$, $\text{first}(S_R) \in P$.

As a corollary of Lemmas 4 and 5, we obtain the aforementioned result.

Theorem 2. *If a tree T contains a 2H-path connecting the nodes u and v then T is a (u, v) -horsetail.*

4 Efficient Algorithm for Horstail Queries

For a tree $T = (V, E)$, we introduce a *horsetail query* of the form: “*is T a (u, v) -horsetail for given nodes $u, v \in V$?*”. In this section we present a linear preprocessing time algorithm which answers horsetail queries for any tree in constant time. We also show how to check if T is 2-traceable (and provide a pair of nodes u, v such that T is a (u, v) -horsetail) in linear time.

We start the analysis with the special case of T being a caterpillar. Answering horsetail queries in this case is easy, we omit the proof of the following simple lemma.

Lemma 6. *Horsetail queries in a caterpillar can be answered in constant time with linear preprocessing time.*

Now we present the algorithm for an arbitrary tree $T = (V, E)$.

1. Let $U = \{w \in V : T \setminus \{w\} \text{ contains } \geq 3 \text{ non-trivial connected components}\}$. Here, again, we call a tree non-trivial if it contains at least one edge.
 - (a) If $U = \emptyset$ then T is a caterpillar. Thus T is 2-traceable, and by Lemma 6, horsetail queries in T can be answered in constant time.
From now on we assume that $U \neq \emptyset$.
2. Let T' be the subtree of T induced by U , i.e., the minimal subtree of T containing all the nodes from the set U .
 - (a) If T' is not a path then T is not 2-traceable. Indeed, if T is a (u, v) -horsetail then all the elements of U must lay on the path from u to v . From now on we assume that T' is a path $P' = (u'_1, \dots, u'_m)$, possibly $m = 1$.
3. By L_i , $i = 1, \dots, m$, we denote the connected component containing u'_i in $T \setminus (P' \setminus \{u'_i\})$. If T is a (u, v) -horsetail then $u \in L_1$ and $v \in L_m$ (or symmetrically).
 - (a) Check if every component L_2, \dots, L_{m-1} satisfies part (\star) of the definition of a horsetail, in particular, if it is a caterpillar (a linear time test). If not then T is not 2-traceable.
4. Let D_1, \dots, D_p and F_1, \dots, F_q be the connected components of $L_1 \setminus \{u'_1\}$ and $L_m \setminus \{u'_m\}$ respectively, ordered by the number of nodes (non-increasingly). Note that, by the definition of the set U , each D_i and each F_i is a caterpillar.
 - (a) If $m > 1$ and at least one of the subtrees D_4, F_4 exists and is non-trivial then T is not 2-traceable. Indeed, regardless of the choice of u and v , the layer containing the node u'_1 (u'_m respectively) would not be a caterpillar. Similarly, if $m = 1$, D_5 exists and is non-trivial then T is not 2-traceable.
5. Denote by X_1 the set of all the nodes w of non-trivial components D_i such that the path from w to u'_1 contains at least one free node (excluding u'_1). By X_2 we denote the set of all the remaining nodes of non-trivial components D_i . Finally, by X_3 we denote the set of all the nodes of all the trivial components D_i . Analogically, we define the sets Y_1, Y_2 and Y_3 , replacing u'_1 with u'_m and D_i with F_i .

- (a) Assume that $m > 1$. For every pair of indices $1 \leq a, b \leq 3$, pick any nodes $u \in X_a$ and $v \in Y_b$ (provided that $X_a \neq \emptyset$ and $Y_b \neq \emptyset$) and check if T is a (u, v) -horsetail — it suffices to check here if u'_1 and u'_m satisfy the condition (\star) and if the path from u to v satisfies the condition $(\star\star)$. If so then for every pair of nodes $u \in X_a$ and $v \in Y_b$ the tree T is a (u, v) -horsetail, otherwise T is not a (u, v) -horsetail for any such pair.
- (b) Assume that $m = 1$. Similarly as in the step 5(a), we consider any pair of nodes u, v from the sets X_a and X_b . This time, however, we pick any elements from *different* components (i.e., $u \in D_i, v \in D_j$ for $i \neq j$). Again we see that the condition (\star) for u'_1 and the condition $(\star\star)$ hold for the chosen nodes u and v if and only if these conditions hold for any pair of nodes from different components from X_a and X_b .

In both cases we obtain a constant number of pairs of sets A_i, B_i such that T is a (u, v) -horsetail if and only if $u \in A_i$ and $v \in B_i$ for some index i , and u, v are in different components D_j in the case $m = 1$.

Clearly, the above algorithm can be implemented in linear time, it suffices to employ depth-first search of selected subtrees of T . We obtain the following result.

Theorem 3. *Horsetail queries in any tree can be answered in constant time with linear preprocessing time. Moreover, testing if a tree is 2-traceable can be done in linear time.*

References

1. Abderrezak, M.E.K., Flandrin, E., Ryjáček, Z.: Induced $S(K_1, 3)$ and Hamiltonian cycles in the square of a graph. *Discrete Mathematics* 207(1-3), 263–269 (1999)
2. Diestel, R.: *Graph Theory*, 4th edn. Springer, Heidelberg (2010)
3. Fleischner, H.: The square of every two-connected graph is Hamiltonian. *J. Combin. Theory (Series B)* 16, 29–34 (1974)
4. Georgakopoulos, A.: A short proof of Fleischner’s theorem. *Discrete Mathematics* 309(23-24), 6632–6634 (2009)
5. Georgakopoulos, A.: Infinite Hamilton cycles in squares of locally finite graphs (2006) (preprint)
6. Harary, F., Schwenk, A.: Trees with Hamiltonian square. *Mathematika* 18, 138–140 (1971)
7. Hendry, G., Vogler, W.: The square of a $S(K_1, 3)$ -free graph is vertex pancyclic. *Journal of Graph Theory* 9, 535–537 (1985)
8. Karaganis, J.J.: On the cube of a graph. *Canad. Math. Bull.* 11, 295–296 (1968)
9. Lin, Y.-L., Skiena, S.: Algorithms for square roots of graphs. *SIAM J. Discrete Math.* 8(1), 99–118 (1995)
10. Sekanina, M.: On an ordering of the set of vertices of a connected graph. Technical Report 412, Publ. Fac. Sci. Univ. Brno. (1960)
11. Thomassen, C.: Hamiltonian paths in squares of infinite locally finite blocks. *Annals of Discr. Math.* 3, 269–277 (1978)
12. Říha, S.: A new proof of the theorem by Fleischner. *J. Comb. Theory Ser. B* 52, 117–123 (1991)

Dominating Induced Matchings for P_7 -free Graphs in Linear Time

Andreas Brandstädt¹ and Raffaele Mosca²

¹ Institut für Informatik, Universität Rostock, D-18051 Rostock, Germany
ab@informatik.uni-rostock.de

² Dipartimento di Scienze, Università degli Studi “G. D’Annunzio”
Pescara 65121, Italy
r.mosca@unich.it

Abstract. Let G be a finite undirected graph with edge set E . An edge set $E' \subseteq E$ is an *induced matching* in G if the pairwise distance of the edges of E' in G is at least two; E' is *dominating* in G if every edge $e \in E \setminus E'$ intersects some edge in E' . The *Dominating Induced Matching Problem* (DIM, for short) asks for the existence of an induced matching E' which is also dominating in G ; this problem is also known as the *Efficient Edge Domination Problem*.

The DIM problem is related to parallel resource allocation problems, encoding theory and network routing. It is NP-complete even for very restricted graph classes such as planar bipartite graphs with maximum degree three. However, its complexity was open for P_k -free graphs for any $k \geq 5$; P_k denotes a chordless path with k vertices and $k - 1$ edges. We show in this paper that the weighted DIM problem is solvable in linear time for P_7 -free graphs in a robust way.

Keywords: dominating induced matching, efficient edge domination, P_7 -free graphs, linear time algorithm, robust algorithm.

1 Introduction and Basic Notions

Packing and covering problems in graphs and their relationships belong to the most fundamental topics in combinatorics and graph algorithms and have a wide spectrum of applications in computer science, operations research and many other fields. Packing problems ask for a maximum collection of objects which are not in conflict, and the Maximum Matching problem for graphs is a good example for this, while covering problems ask for a minimum collection of objects which cover some or all others, and the Problems Minimum Vertex Cover ([GT1] of [13]) and Minimum Dominating Set ([GT2] of [13]) are good examples of such problems. In this paper, we study a problem which combines both requirements.

Let G be a simple undirected graph with vertex set V and edge set E . A subset M of E is an *induced matching* in G if the G -distance of every pair of edges $e, e' \in M$, $e \neq e'$, is at least two, i.e., $e \cap e' = \emptyset$ and there is no edge $xy \in E$ with $x \in e$ and $y \in e'$. A subset $M \subseteq E$ is a *dominating edge set* if every

edge $e \in E \setminus M$ shares an endpoint with some edge $e' \in M$, i.e., if $e \cap e' \neq \emptyset$. A *dominating induced matching* (*d.i.m.* for short) is an induced matching which is also a dominating edge set. Note that M is a d.i.m. in G if and only if it is a dominating vertex set in the line graph $L(G)$ and an independent vertex set in the square $L(G)^2$ of the line graph. Thus, the DIM problem is simultaneously a packing and a covering problem. Let us say that *an edge $e \in E$ is matched by M* if $e \in M$ or there is some $e' \in M$ with $e \cap e' \neq \emptyset$. Thus, M is a d.i.m. of G if and only if every edge of G is matched by M but no edge is matched twice.

The *Dominating Induced Matching Problem* (*DIM*, for short) asks whether a given graph has a dominating induced matching. This can also be seen as a special 3-colorability problem, namely the partition into three independent vertex sets A, B, C such that $G[B \cup C]$ is an induced matching. Dominating induced matchings are also called *edge packings* in some papers, and DIM is known as the *Efficient Edge Domination Problem* (*EED* for short). A brief history of EED as well as some applications in the fields of resource allocation, encoding theory and network routing are presented in [14] and [17]. Grinstead et al. [14] show that EED is NP-complete in general. It remains hard for bipartite graphs [19]. In particular, [18] shows the intractability of EED for planar bipartite graphs and [10] for very restricted bipartite graphs with maximum degree three (the restrictions are some forbidden subgraphs). In [4], it is shown that the problem remains NP-complete for planar bipartite graphs with maximum degree three but is solvable in polynomial time for hole-free graphs (which was an open problem in [18] and is still mentioned as an open problem in [9]; actually, [9, 18] mention that the complexity of DIM is an open problem for weakly chordal graphs which are a subclass of hole-free graphs). In [9], as another open problem, it is mentioned that for any $k \geq 5$, the complexity of DIM is unknown for the class of P_k -free graphs. Note that the complexity of the related problems Maximum Independent Set and Maximum Induced Matching is unknown for P_5 -free graphs, and a lot of work has been done on subclasses of P_5 -free graphs.

In this paper, we show that for P_7 -free graphs, DIM is solvable in linear time. Actually, we consider the edge-weighted optimization version of DIM, namely the *Minimum Dominating Induced Matching Problem* (*MDIM*), which asks for a dominating induced matching M in $G = (V, E)$ of minimum weight with respect to some given weight function $\omega : E \rightarrow \mathbb{R}$ (if existent). Our algorithm for P_7 -free graphs is based on a structural analysis of such graphs having a d.i.m. It is robust in the sense of [21] since it is not required that the input graph is P_7 -free; our algorithm either determines an optimal d.i.m. correctly or finds out that G has no d.i.m. or is not P_7 -free. Due to space limitation, we omit all proofs and some procedures; see [7] for a full version of this paper.

Let G be a finite undirected graph without loops and multiple edges. Let V denote its vertex set and E its edge set; let $|V| = n$ and $|E| = m$. Open and closed neighborhood of a vertex, independent vertex sets, true twins, complement graph, modules, homogeneous sets, distances between edges, connected and 2-connected components are all defined as usual; see [7] for details.

It is well known that in a connected graph G with connected complement \overline{G} , the maximal homogeneous sets are pairwise disjoint and can be determined in linear time (see, e.g., [20]).

The 2-connected components are also called *blocks*. It is well known that the blocks of a graph can be determined in linear time [15] (see also [1]).

A *chordless path* P_k (*chordless cycle* C_k , respectively) has k vertices, say v_1, \dots, v_k , and edges $v_i v_{i+1}$, $1 \leq i \leq k-1$ (and $v_k v_1$, respectively). We say that such a path (cycle, respectively) has length k . Let K_i denote the clique with i vertices. Let $K_4 - e$ or *diamond* be the graph with four vertices and five edges, say vertices a, b, c, d and edges ab, ac, bc, bd, cd ; its *mid-edge* is the edge bc . Let W_4 denote the graph with five vertices consisting of a C_4 and a universal vertex (see Figure II). Let $K_{1,k}$ denote the star with one universal vertex and k independent vertices. A star is *nontrivial* if it contains a P_3 or an edge, otherwise it is *trivial*.

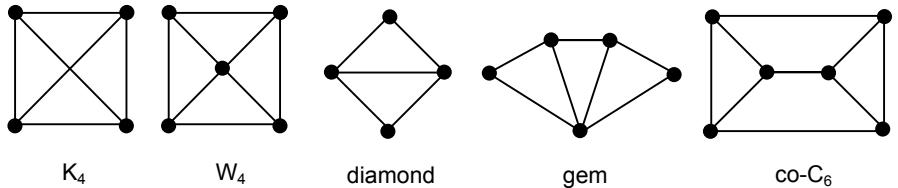


Fig. 1. K_4 , W_4 , diamond, gem, and $\text{co-}C_6$

For a set \mathcal{F} of graphs, a graph G is called \mathcal{F} -*free* if G contains no induced subgraph from \mathcal{F} .

If M is a d.i.m., an edge is *matched by* M if it is either in M or shares a vertex with some edge in M . Likewise, a vertex is *matched* if it is in $V(M)$.

2 Simple Properties of Graphs with Dominating Induced Matching

The following observations are helpful (some of them are mentioned e.g. in [4]):

Observation 1. *Let M be a d.i.m. in G .*

- (i) *M contains at least one edge of every odd cycle C_{2k+1} in G , $k \geq 1$, and exactly one edge of every odd cycle C_3, C_5, C_7 of G .*
- (ii) *No edge of any C_4 can be in M .*
- (iii) *If C is a C_6 then either exactly two or none of the C -edges are in M .*

If an edge $e \in E$ is contained in any d.i.m. of G , we call it *mandatory* (or *forced*) in G . Mandatory edges are useful for some kinds of reductions.

Observation 2. *The mid-edge of any diamond in G is mandatory.*

If an edge xy is mandatory, we can reduce the graph as follows: Delete x and y and all edges incident to x and y , and give all edges in distance one to xy the weight ∞ . This means that these edges are not in any d.i.m. of finite weight in G . For a set M of mandatory edges, let $Reduced(G, M)$ denote the reduced graph as defined above. Obviously, this graph is an induced subgraph of G and can be determined in linear time for given G and M . Moreover:

Observation 3. *Let M' be an induced matching which is a set of mandatory edges in G . Then G has a d.i.m. M if and only if $Reduced(G, M')$ has a d.i.m. $M \setminus M'$.*

We can also color red all vertices in distance one to a mandatory edge; subsequently, edges ab with a red vertex a cannot be matched in vertex a ; they have to be matched in vertex b . If also b is red then G has no d.i.m.

Subsequently, as a kind of preprocessing, the mid-edges of some of the diamonds will be determined. Since it would be too time-consuming to determine all diamonds in G , we will mainly find such diamonds whose mid-edges are edges between true twins having at least two common neighbors. These are contained in maximal homogeneous sets (which are not stable). Since the edges of any d.i.m. must have pairwise distance at least two, we obtain:

Observation 4. *If G has a d.i.m. then for all vertices v , $G[N(v)]$ is the disjoint union of at most one star with P_3 , and of edges and vertices.*

From the previous observations, it follows (see Figure 1 for K_4 , W_4 , gem, and C_6):

Corollary 1. *If G has a d.i.m. then G is K_4 -free, W_4 -free, gem-free and \overline{C}_k -free for any $k \geq 6$.*

Corollary 2. *Let G have a d.i.m. and let H be a homogeneous set in G which is not stable, i.e., contains an edge.*

- (i) $N(H)$ is stable.
- (ii) If $|N(H)| \geq 2$ then H is the disjoint union of edges.
- (iii) Vertices x and y are true twins with at least two common neighbors in G if and only if they appear as an edge in a homogeneous set H with $|N(H)| \geq 2$.

The following procedure deals with homogeneous sets H with $|N(H)| = 1$; then all connected components of H together with z are leaf blocks in G .

Procedure Hom-1-DIM:

Given: A non-stable homogeneous set H in G with $N(H) = \{z\}$.

Task: Determine some mandatory edges or find out that G has no d.i.m.

- (a) If H contains a cycle or P_4 then STOP - G has no d.i.m.
- (b) (Now H is a P_4 -free forest.) If H contains at least two stars with P_3 then STOP - G has no d.i.m.

- (c) (*Now H is a P_4 -free forest which contains at most one star with P_3 .*) If H contains exactly one star with P_3 , say abc then $M := M \cup \{bz\}$. If another connected component of H contains an edge then STOP - G has no d.i.m.
- (d) (*Now H is a P_3 -free forest, i.e., a disjoint union of edges $E'(H)$ and vertices $V'(H)$.*) If $E'(H)$ contains at least two edges then $M := M \cup E'(H)$. If $V'(H) \neq \emptyset$ then STOP - G has no d.i.m.
- (e) (*Now H is a disjoint union of at most one edge and vertices $V'(H)$.*) If there is an edge ab in H and $V'(H) \neq \emptyset$ then $M := M \cup \{az\}$ or $M := M \cup \{bz\}$ (depending on the better weight).

We postpone the discussion of the two final cases $E'(H) = \{ab\}$ and $V'(H) = \emptyset$ or $E'(H) = \emptyset$ and $V'(H) \neq \emptyset$. Obviously, the following holds:

Lemma 1. *Procedure Hom-1-DIM is correct and can be carried out in linear time.*

In the final case of a homogeneous set H with only one neighbor z where H consists of just one edge ab , abz forms a leaf block. For graph G , let G^* denote the graph obtained from G by omitting all such triangle leaf blocks. Obviously, G^* can be constructed in linear time. We will need this construction in our algorithm P_7 -Free-DIM for DIM in section 4. There, we also need the following transformation: For every triangle leaf block abc with cut-vertex c and corresponding edge weights $w(ab)$, $w(ac)$, $w(bc)$, let $Tr(G, abc)$ be the graph with the same cut-vertex c where the triangle abc is replaced by a P_3 $a'b'c$ with weights $w(a'b') := w(ab)$ and $w(b'c) := \min(w(ac), w(bc))$. Let $Tr(G)$ be the result of applying $Tr(G, abc)$ to all triangle leaf blocks abc of G . Obviously, G has a d.i.m. if and only if $Tr(G, abc)$ has a d.i.m., and the optimal weights of d.i.m.'s in G and $Tr(G, abc)$ are the same. The only problem is the fact that the new graph is not necessarily P_7 -free when G is P_7 -free. We will apply this construction only in one case, namely when the internal blocks of G form a distance-hereditary bipartite graph; then $Tr(G)$ is also distance hereditary bipartite.

Finally we need the following:

Proposition 1. *For a given set E' of edges, it can be tested in linear time whether E' is a d.i.m., and likewise, whether E' is an induced matching.*

3 Structure of P_7 -free Graphs with Dominating Induced Matching

Throughout this section, let $G = (V, E)$ be a connected P_7 -free graph having a d.i.m.; then the vertex set V has the partition $V = I \cup V(M)$ with independent vertex set I . We suppose that $xy \in M$ is an edge in a P_3 and consider the distance levels $N_i = N_i(xy)$, $i \geq 1$, with respect to the edge xy . Note that every edge of an odd hole C_5 , C_7 , respectively, is in a P_3 . For triangles abc , this is not fulfilled if a and b are true twins. However, true twins with at least two common neighbors will lead to mandatory edges, and true twins a, b with only one common neighbor c form a leaf block abc which will be temporarily omitted by constructing G^* and looking for an odd cycle in G^* .

3.1 Distance Levels with Respect to an M -Edge in a P_3

Since we assume that $xy \in M$, clearly, $N_1 \subseteq I$ and thus:

$$N_1 \text{ is a stable set.} \quad (1)$$

Moreover, no edge between N_1 and N_2 is in M . Since $N_1 \subseteq I$ and all neighbors of vertices in I are in $V(M)$, we have:

$$N_2 \text{ is a disjoint union of some edges and vertices in } M. \quad (2)$$

Let M_2 denote the set of edges in N_2 and let S_2 denote the set of isolated vertices in N_2 ; $N_2 = V(M_2) \cup S_2$. Obviously:

$$M_2 \subseteq M \text{ and } S_2 \subseteq V(M). \quad (3)$$

Let M_3 denote the set of M -edges with one endpoint in S_2 (and the other endpoint in N_3). Since xy is contained in a P_3 , i.e., there is a vertex r such that y, x, r induce a P_3 , we obtain some further properties:

$$N_5 = \emptyset. \quad (4)$$

This kind of argument will be used later again - we will say that the subgraph induced by $x, y, N_1, v_2, v_3, v_4, v_5$ contains an induced P_7 . Obviously, by (3) and the distance condition, the following holds:

$$\text{No edge in } N_3 \text{ and no edge between } N_3 \text{ and } N_4 \text{ is in } M. \quad (5)$$

Furthermore the following statement holds.

$$N_4 \text{ is a disjoint union of edges and vertices.} \quad (6)$$

Let M_4 denote the set of edges in N_4 and let S_4 denote the set of isolated vertices in N_4 ; $N_4 = V(M_4) \cup S_4$. Note that by (4) and (5), $S_4 \subseteq I$. Since every edge ab in N_4 together with a predecessor c in N_3 forms a triangle, and $ac, bc \notin M$, by (5) necessarily:

$$M_4 \subseteq M. \quad (7)$$

By Observation ④ (i), in every odd cycle C_3, C_5 and C_7 of G , exactly one edge must be in M . Thus, (5) implies:

$$N_3 \cup S_4 \text{ is bipartite.} \quad (8)$$

Note that in general, N_3 is not a stable set. Let $T_{one} := \{t \in N_3 : |N(t) \cap S_2| = 1\}$, and $T_{two} := \{t \in N_3 : |N(t) \cap S_2| \geq 2\}$. Note that if uv is an edge with $u \in T_{two}$ then $uv \notin M$ and uv must be matched by an M -edge at v since it cannot be matched at u because of the distance condition; in particular, $T_{two} \subseteq I$. In general, (5) will lead to some forcing conditions since the edges in N_3 and between N_3 and N_4 have to be matched. If an edge $uv \in E$ cannot be matched at u then it has to be matched at v - in this case, as described later, we color the

vertex v green if it has to be matched by an M_3 edge. (For an algorithm checking the existence of a d.i.m., it is useful to observe that if vertices in distance one get color green then no d.i.m. exists.)

Let $S_3 := (N(M_2) \cap N_3) \cup (N(M_4) \cap N_3) \cup T_{two}$. Then $S_3 \subseteq N_3$ and $S_3 \subseteq I$. Furthermore, since $S_4 \subseteq I$, one obtains:

$$S_3 \cup S_4 \text{ is a stable set.} \quad (9)$$

Let $T_{one}^* := T_{one} \setminus S_3$. Then $N_3 = S_3 \cup T_{one}^*$ is a partition of N_3 . In particular, T_{one}^* contains the M -mates of the vertices of S_2 . Recall that M_3 denotes the set of M -edges with one endpoint in S_2 (and the other endpoint in T_{one}^*).

3.2 Edges in and between T_i and T_j , $i \neq j$

Let $S_2 = \{u_1, u_2, \dots, u_k\}$, and let $T_i := T_{one}^* \cap N(u_i)$, $i = 1, \dots, k$. Then $T_{one}^* = T_1 \cup \dots \cup T_k$ is a partition of T_{one}^* . The following condition is necessary for the existence of M_3 :

For all $i = 1, \dots, k$, $T_i \neq \emptyset$, and exactly one vertex of T_i is in $V(M_3)$. (10)

Recall that by Observation 4, $G[T_i]$ is the disjoint union of at most one star with P_3 , and of edges and vertices. Furthermore, $G[T_i]$ cannot contain two edges, i.e., for all $i = 1, \dots, k$:

$G[T_i]$ is a disjoint union of vertices and at most one star with an edge. (11)

Assume that T_i contains the star Y_i with an edge.

For all $i, j = 1, \dots, k$, $i \neq j$, Y_i sees no vertex of T_j . (12)

Claim 1. For all $i = 1, \dots, k$, there is at most one $j \neq i$ such that a vertex in T_i sees a vertex in T_j .

Let us say that T_i sees T_j if there are vertices in T_i and T_j which see each other. Now by Claim 1, for every $i = 1, \dots, k$, T_i either sees no T_j , $j \neq i$ (T_i is isolated), or sees exactly one T_j , $j \neq i$ (T_i and T_j are paired).

Claim 2. If T_i and T_j are paired then $G[T_i \cup T_j]$ contains at most two components among the four following ones: Y_i (defined above), Y_j (defined above), Y'_i which is a star with center in T_i and the other vertices in T_j , Y'_j which is a star with center in T_j and the other vertices in T_i ; in particular, at most one from $\{Y_i, Y_j\}$ does exist.

The above claims are useful tools to detect M_3 . Then let us observe that:

- (i) if a vertex $t_i \in T_i$ sees a vertex of $S_3 \cup S_4$, then $u_i t_i \in M_3$;
- (ii) if a vertex $t_i \in T_i$ is the center of the star Y_i or Y'_i (in case of paired sets), with a P_3 then $u_i t_i \in M_3$.

Let us say that a vertex $t_i \in T_i$ is *green* if it enjoys one of the above two conditions (i), (ii). Then the following statement holds for all $i = 1, \dots, k$:

$$G[T_i] \text{ contains at most one green vertex, say } t_i^* \quad (13)$$

and

$$G[T_i \setminus N(t_i^*)] \text{ is edgeless.} \quad (14)$$

All this leads to a corresponding procedure Check(xy) (described in [7]) which will be carried out a fixed number of times in our algorithm P_7 -Free-DIM in section 4.

4 The Algorithm for the General P_7 -free Case

In the previous chapters we have analyzed the structure of P_7 -free graphs having a d.i.m. Now we are going to use these properties for an efficient algorithm for solving the DIM problem on these graphs.

Algorithm P_7 -Free-DIM.

Given: A connected graph $G = (V, E)$ with edge weights.

Task: Determine a d.i.m. in G of finite minimum weight (if existent) or find out that G has no d.i.m. or is not P_7 -free.

- (a) If G is bipartite then carry out procedure P_7 -Free-Bipartite-DIM (described in [7] and based on a certified linear-time recognition of distance-hereditary bipartite graphs [2]).
- (b) (*Now G is not bipartite.*) If G is a cograph (which can be determined in linear time [8][11]) then apply procedure Cograph-DIM (described in [7]). If G is not a cograph but \overline{G} is not connected then STOP - G has no d.i.m.
- (c) (*Now G is not bipartite and \overline{G} is connected.*) Let $M := \emptyset$. Determine the maximal homogeneous sets H_1, \dots, H_k of G (which can be done in linear time [20]). For all $i \in \{1, \dots, k\}$ do the following steps (c.1), (c.2):
 - (c.1) If $|N(H_i)| = 1$ then carry out procedure Hom-1-DIM.
 - (c.2) In the case when $|N(H_i)| \geq 2$ and H_i is not a stable set, check whether $N(H_i)$ is stable and H_i is a disjoint union of edges; if not then STOP - G has no d.i.m., otherwise, for all edges xy in H_i , let $M := M \cup \{xy\}$.
- (d) Construct $G' = \text{Reduced}(G, M)$.
- (e) For every connected component C of G' , do: If C is bipartite then carry out procedure P_7 -Free-Bipartite-DIM for C . Otherwise construct C^* (where the triangle leaf blocks are temporarily omitted) and carry out Find-Odd-Cycle-Or- P_7 (described in [7]) for C^* , and if an odd cycle is found, carry out Check(ab) in the graph C for all (at most seven) edges of the odd cycle. Add the resulting edge set to the mandatory edges from steps (c.1), (c.2), respectively. If however, C^* is bipartite then with procedure P_7 -Free-Bipartite-DIM for C^* , find out if the procedure unsuccessfully stops or if there is a C_6 in C^* ; in the last case, do Check(ab) for all edges of the C_6 .

Finally, if C^* is distance hereditary bipartite, construct $Tr(C)$ (the omitted triangle leaf blocks are attached as P_3 's and the resulting graph is distance hereditary bipartite) and solve DIM for this graph using the clique-width argument (or using the linear time algorithm for DIM on chordal bipartite graphs given in [4]).

- (f) Finally check once more whether M is a d.i.m. of G . If not then G has no d.i.m., otherwise return M .

Theorem 1. *Algorithm P_7 -Free-DIM is correct and runs in linear time.*

5 Conclusion

In this paper we have solved the DIM problem in linear time for P_7 -free graphs which solves an open problem from [9]. Actually, we solve the problem in a robust way in the sense of [21]: The algorithm either solves the problem correctly or finds out that the input graph has no d.i.m. or is not P_7 -free. This avoids to recognize whether the input graph is P_7 -free; the known recognition time bound is much worse than linear time.

For P_5 -free graphs, it was known that DIM is solvable in time $\mathcal{O}(n^2)$ as a consequence of the fact that the clique-width of (P_5, gem) -free graphs is bounded [5,6] and a clique-width expression can be constructed in time $\mathcal{O}(n^2)$ [3]. In [9], it is mentioned that DIM is expressible in a certain kind of Monadic Second Order Logic, and in [12], it was shown that such problems can be solved in linear time on any class of bounded clique-width assuming that the clique-width expressions are given or can be determined in the same time bound.

It is a challenging open question whether for some k , the DIM problem is NP-complete for P_k -free graphs.

Acknowledgement. The first author gratefully acknowledges a research stay at the LIMOS institute, University of Clermont-Ferrand, and the inspiring discussions with Anne Berry on dominating induced matchings.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: *The Design and Analysis of Computer Algorithms*. Addison-Wesley (1974)
2. Bandelt, H.-J., Mulder, H.M.: Distance-hereditary graphs. *J. Combin. Th (B)* 41, 182–208 (1986)
3. Bodlaender, H.L., Brandstädt, A., Kratsch, D., Rao, M., Spinrad, J.: On algorithms for (P_5, gem) -free graphs. *Theor. Comput. Sci.* 349, 2–21 (2005)
4. Brandstädt, A., Hundt, C., Nevries, R.: Efficient Edge Domination on Hole-Free Graphs in Polynomial Time. In: López-Ortiz, A. (ed.) *LATIN 2010. LNCS*, vol. 6034, pp. 650–661. Springer, Heidelberg (2010)
5. Brandstädt, A., Kratsch, D.: On the structure of (P_5, gem) -free graphs. *Discrete Applied Mathematics* 145, 155–166 (2005)

6. Brandstädt, A., Le, H.-O., Mosca, R.: Chordal co-gem-free and (P_5, gem) -free graphs have bounded clique-width. *Discrete Applied Mathematics* 145, 232–241 (2005)
7. Brandstädt, A., Mosca, R.: Dominating induced matchings for P_7 -free graphs in linear time, Technical report CoRR, arXiv:1106.2772v1 [cs.DM] (2011)
8. Bretscher, A., Corneil, D.G., Habib, M., Paul, C.: A Simple Linear Time LexBFS Cograph Recognition Algorithm. *SIAM J. Discrete Math.* 22(4), 1277–1296 (2008)
9. Cardozo, D.M., Korpelainen, N., Lozin, V.V.: On the complexity of the dominating induced matching problem in hereditary classes of graphs. Electronically Available in *Discrete Applied Math.* (2011)
10. Cardoso, D.M., Lozin, V.V.: Dominating Induced Matchings. In: Lipshteyn, M., Levit, V.E., McConnell, R.M. (eds.) *Graph Theory, Computational Intelligence and Thought*. LNCS, vol. 5420, pp. 77–86. Springer, Heidelberg (2009)
11. Corneil, D.G., Perl, Y., Stewart, L.K.: A linear recognition algorithm for cographs. *SIAM J. Computing* 14, 926–934 (1985)
12. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique width. *Theory of Computing Systems* 33, 125–150 (2000)
13. Garey, M.R., Johnson, D.S.: *Computers and Intractability – A Guide to the Theory of NP-completeness*. Freeman, San Francisco (1979)
14. Grinstead, D.L., Slater, P.L., Sherwani, N.A., Holmes, N.D.: Efficient edge domination problems in graphs. *Information Processing Letters* 48, 221–228 (1993)
15. Hopcroft, J.E., Tarjan, R.E.: Efficient Algorithms for Graph Manipulation [H]. *Communications of the ACM* 16(6), 372–378 (1973)
16. Liang, Y.D., Lu, C.L., Tang, C.Y.: Efficient Domination on Permutation Graphs and Trapezoid Graphs. In: Jiang, T., Lee, D.T. (eds.) *COCOON 1997*. LNCS, vol. 1276, pp. 232–241. Springer, Heidelberg (1997)
17. Livingston, M., Stout, Q.: Distributing resources in hypercube computers. In: Proceedings 3rd Conf. on Hypercube Concurrent Computers and Applications, pp. 222–231 (1988)
18. Lu, C.L., Ko, M.-T., Tang, C.Y.: Perfect edge domination and efficient edge domination in graphs. *Discrete Applied Math.* 119(3), 227–250 (2002)
19. Lu, C.L., Tang, C.Y.: Efficient domination in bipartite graphs (1997) (manuscript)
20. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. *Discrete Math.* 201, 189–241 (1999)
21. Spinrad, J.P.: *Efficient Graph Representations*, Fields Institute Monographs. American Math. Society (2003)

Finding Contractions and Induced Minors in Chordal Graphs via Disjoint Paths^{*}

Rémy Belmonte¹, Petr A. Golovach², Pinar Heggernes¹, Pim van ’t Hof¹,
Marcin Kamiński³, and Daniël Paulusma²

¹ Department of Informatics, University of Bergen, Norway

{remy.belmonte,pinar.heggernes,pim.vanthof}@ii.uib.no

² School of Engineering and Computing Sciences, Durham University, UK
{petr.golovach,daniel.paulusma}@durham.ac.uk

³ Département d’Informatique, Université Libre de Bruxelles, Belgium
marcin.kaminski@ulb.ac.be

Abstract. The k -DISJOINT PATHS problem, which takes as input a graph G and k pairs of specified vertices (s_i, t_i) , asks whether G contains k mutually vertex-disjoint paths P_i such that P_i connects s_i and t_i , for $i = 1, \dots, k$. We study a natural variant of this problem, where the vertices of P_i must belong to a specified vertex subset U_i for $i = 1, \dots, k$. In contrast to the original problem, which is polynomial-time solvable for any fixed integer k , we show that this variant is NP-complete even for $k = 2$. On the positive side, we prove that the problem becomes polynomial-time solvable for any fixed integer k if the input graph is chordal. We use this result to show that, for any fixed graph H , the problems H -CONTRACTIBILITY and H -INDUCED MINOR can be solved in polynomial time on chordal graphs. These problems are to decide whether an input graph G contains H as a contraction or as an induced minor, respectively.

1 Introduction

We study algorithmic problems that involve deciding whether the structure of a graph H appears as a pattern within the structure of another graph G . Table II shows seven graph containment relations that can be obtained by combining vertex deletions (VD), edge deletions (ED), and edge contractions (EC). For example, a graph H is an *induced minor* of a graph G if H can be obtained from G by a sequence of graph operations, including vertex deletions and edge contractions, but not including edge deletions. The corresponding decision problem, in which G and H form the ordered input pair (G, H) , is called INDUCED MINOR. The other rows in Table II are to be interpreted similarly.

With the exception of GRAPH ISOMORPHISM, all problems in Table II are known to be NP-complete when G and H are both input (cf. [15]). In search of

* This work is supported by EPSRC (EP/G043434/1) and Royal Society (JP100692), and by the Research Council of Norway (197548/F20).

Table 1. Seven known containment relations obtained by graph operations. The missing combination “no yes yes” corresponds to the minor relation if we allow an extra operation that removes isolated vertices.

Containment Relation	VD	ED	EC	Decision Problem
minor	yes	yes	yes	MINOR
induced minor	yes	no	yes	INDUCED MINOR
contraction	no	no	yes	CONTRACTIBILITY
subgraph	yes	yes	no	SUBGRAPH ISOMORPHISM
induced subgraph	yes	no	no	INDUCED SUBGRAPH ISOMORPHISM
spanning subgraph	no	yes	no	SPANNING SUBGRAPH ISOMORPHISM
isomorphism	no	no	no	GRAPH ISOMORPHISM

tractability, it is common to fix the graph H as a part of the problem definition, and to consider only the graph G as input. We indicate this by adding “ H -” in front of the names of the decision problems. A celebrated result by Robertson and Seymour [16] states that H -MINOR can be solved in cubic time for any fixed graph H . The problems H -SUBGRAPH ISOMORPHISM, H -INDUCED SUBGRAPH ISOMORPHISM, H -SPANNING SUBGRAPH ISOMORPHISM and H -GRAPH ISOMORPHISM can readily be solved in polynomial time for any fixed graph H . In contrast, there exist graphs H such that H -INDUCED MINOR and H -CONTRACTIBILITY are NP-complete. Although a complete complexity classification of these two problems is still not known, there are many partial results.

Fellows, Kratochvíl, Middendorf, and Pfeiffer [5] gave both polynomial-time solvable and NP-complete cases for the H -INDUCED MINOR problem on general input graphs. The smallest known NP-complete case is a graph H on 68 vertices [5]. A number of polynomial-time solvable and NP-complete cases for the H -CONTRACTIBILITY problem on general input graphs can be found in a series of papers started by Brouwer and Veldman [3], followed by Levin, Paulusma and Woeginger [13][14], and van ’t Hof et al. [9]. The smallest NP-complete cases are when H is a path or a cycle on 4 vertices [3]. When it comes to input graphs with a particular structure, both H -INDUCED MINOR [5] and H -CONTRACTIBILITY [11] can be solved in polynomial time on planar graphs for any fixed graph H . The same is true when the input graphs are split graphs, as shown by Belmonte, Heggernes, and van ’t Hof [1] and Golovach et al. [8]. Finally, Golovach, Kamiński and Paulusma [7] showed that H -CONTRACTIBILITY and H -INDUCED MINOR are polynomial-time solvable on chordal graphs whenever H is a fixed tree or a fixed split graph. For any fixed H that is neither a tree nor a split graph, the computational complexity of these two problems on chordal graphs was left open.

In this paper, we show that H -CONTRACTIBILITY and H -INDUCED MINOR can be solved in polynomial time on chordal graphs, for any fixed graph H . In fact, our result implies algorithms with running time $|V_G|^{O(|V_H|^2)}$ for CONTRACTIBILITY and INDUCED MINOR on input pairs (G, H) where G is chordal. Our result is best possible in the sense that both these problems are NP-complete

already on input pairs (G, H) , where G is a split graph and H is a threshold¹ graph [1]. Moreover, both problems are $W[1]$ -hard parameterized by $|V_H|$ when G and H are both split graphs [8]. This means that it is unlikely that these two problems can be solved in time $f(|V_H|)|V_G|^{O(1)}$ on input pairs (G, H) that are split graphs, such that the function f is independent of $|V_G|$.

Our approach differs from previous approaches [1, 7, 8], as it is based on an application of the following problem. Here, a *terminal pair* in a graph $G = (V, E)$ is a specified pair of vertices s_i and t_i called *terminals*, and the *domain* of a terminal pair (s_i, t_i) is a specified subset $U_i \subseteq V$ containing both s_i and t_i . We say that an (s_i, t_i) -path and an (s_j, t_j) -path are *vertex-disjoint* if they have no common vertices except the vertices in $\{s_i, t_i\} \cap \{s_j, t_j\}$.

SET-RESTRICTED k -DISJOINT PATHS

Instance: A graph G , terminal pairs $(s_1, t_1), \dots, (s_k, t_k)$, and domains U_1, \dots, U_k .

Question: Does G contain k mutually vertex-disjoint paths P_1, \dots, P_k such that

P_i is a path from s_i to t_i using only vertices from U_i for $i = 1, \dots, k$?

Observe that the domains U_1, \dots, U_k can have common vertices. Furthermore, if we let every domain contain all vertices of G , we obtain the well-known problem k -DISJOINT PATHS. This problem can be solved in cubic time for any fixed k , as was shown by Robertson and Seymour [16]. In contrast to this result, we show that SET-RESTRICTED k -DISJOINT PATHS is NP-complete even for $k = 2$. However, we show that on chordal graphs SET-RESTRICTED k -DISJOINT PATHS can be solved in polynomial time for any fixed integer k . If k is part of the input, then this problem is NP-complete for chordal graphs, as Kammer and Tholey [10] showed that its special case DISJOINT PATHS remains NP-complete when restricted to this graph class.

2 Preliminaries

Let $G = (V, E)$ be a graph. If the vertex set and edge set of G are not specified, then we use V_G and E_G to denote these sets, respectively. The number of vertices in an input graph is denoted by n . A subset $U \subseteq V$ is a *clique* if there is an edge in G between any two vertices of U . A vertex is called *simplicial* if its neighbors form a clique. We write $G[U]$ to denote the subgraph of G induced by $U \subseteq V$, i.e., the graph on vertex set U and an edge between any two vertices whenever there is an edge between them in G . Two sets $U, U' \subseteq V$ are called *adjacent* if there exist vertices $u \in U$ and $u' \in U'$ such that $uu' \in E$.

The *edge contraction* of an edge uv in G deletes the vertices u and v from G , and replaces them by a new vertex adjacent to precisely those vertices to which u or v were adjacent. A graph H is a *contraction* of a graph G if H can be obtained by performing a series of edge contractions in G . An H -*witness structure* \mathcal{W} is a partition of V_G into $|V_H|$ nonempty sets $W(x)$, one set for each $x \in V_H$, called *H-witness sets*, such that (i) each $W(x)$ induces a connected subgraph of G ,

¹ Threshold graphs form a subset of split graphs, which form a subset of chordal graphs.

and (ii) for all $x, y \in V_H$ with $x \neq y$, sets $W(x)$ and $W(y)$ are adjacent in G if and only if x and y are adjacent in H . Clearly, H is a contraction of G if and only if G has an H -witness structure satisfying conditions (i) and (ii); H can be obtained by contracting each witness set into a single vertex.

A *tree decomposition* of a graph G is a pair $(\mathcal{T}, \mathcal{X})$, where \mathcal{X} is a collection of subsets of V_G , called *bags*, and \mathcal{T} is a tree whose vertices, called *nodes*, are the sets of \mathcal{X} , such that the following three properties are satisfied. First, $\bigcup_{X \in \mathcal{X}} X = V_G$. Second, for each edge $uv \in E_G$, there is a bag $X \in \mathcal{X}$ with $u, v \in X$. Third, for each $x \in V_G$, the set of nodes containing x forms a subtree of \mathcal{T} .

A graph is *chordal* if it does not contain a chordless cycle on at least four vertices as an induced subgraph. It is easy to see that the class of chordal graphs is closed under edge contractions. Chordal graphs can be recognized in linear time [17]. Every chordal graph contains at most n maximal cliques, and, if it is not a complete graph, at least two non-adjacent simplicial vertices [4]. A graph G is chordal if and only if it has a tree decomposition whose set of bags is exactly the set of maximal cliques of G [6]. Such a tree decomposition is called a *clique tree* and can be constructed in linear time [2].

3 Set-Restricted Disjoint Paths

In this extended abstract some proofs, in particular the proof of the following theorem, are omitted due to limited space.

Theorem 1. *The SET-RESTRICTED 2-DISJOINT PATHS problem is NP-complete.*

We now apply dynamic programming to prove that SET-RESTRICTED k -DISJOINT PATHS can be solved in polynomial time on chordal graphs for any fixed integer k . The first key observation is that the existence of k disjoint paths is equivalent to the existence of k disjoint *induced*, i.e. chordless, paths. The second key observation is that any induced path contains at most two vertices of any clique. Our algorithm can easily be modified to produce the required paths (if they exist).

Kloks [12] showed that every tree decomposition of a graph can be converted in linear time to a *nice tree decomposition*, such that the size of the largest bag does not increase, and the total size of the tree is linear in the size of the original tree. A tree decomposition $(\mathcal{T}, \mathcal{X})$ is *nice* if \mathcal{T} is a binary tree with root X_r such that the nodes of \mathcal{T} are of four types: (i) a *leaf node* X is a leaf of \mathcal{T} and has size $|X| = 1$; (ii) an *introduce node* X has one child X' with $X = X' \cup \{v\}$ for some vertex $v \in V_G$; (iii) a *forget node* X has one child X' with $X = X' \setminus \{v\}$ for some vertex $v \in V_G$; (iv) a *join node* X has two children X' and X'' with $X = X' = X''$. Since each chordal graph G has a clique tree, it also has a nice tree decomposition with the additional property that each bag is a (not necessary maximal) clique in G .

Now we are ready to describe our algorithm for SET-RESTRICTED k -DISJOINT PATHS. Let k be a positive integer, and let G be a chordal graph with k pairs of *terminal* vertices $(s_1, t_1), \dots, (s_k, t_k)$ that have domains U_1, \dots, U_k , respectively. If G is disconnected, we check for each pair of terminals (s_i, t_i) whether s_i and t_i

belong to the same connected component. If not, then we return **No**. Otherwise, we consider each connected component and its set of terminals separately. Hence, we may assume without loss of generality that G is connected. We then proceed as follows.

First, we construct in linear time a nice tree decomposition $(\mathcal{T}, \mathcal{X})$ with root X_r for G such that each bag is a clique in G . Next, we apply a dynamic programming algorithm over $(\mathcal{T}, \mathcal{X})$. We omit the details here and only describe what we store in tables corresponding to the nodes of \mathcal{T} . For any node $X_i \in V_{\mathcal{T}}$, we denote by \mathcal{T}_i the subtree of \mathcal{T} with root X_i that is induced by the descendants of X_i . We also define the subgraph $G_i = G[\bigcup_{j \in V_{\mathcal{T}_i}} X_j]$. For a node X_i , the table stores a collection of the records $\mathcal{R} = ((State_1, R_1), \dots, (State_k, R_k))$, where each $State_j$ can have one of the four values:

Not initialized, Started from s , Started from t , Completed,

and $R_1, \dots, R_k \subseteq X_i$ are ordered sets without common vertices except (possibly) terminals, $R_j \subseteq U_j$, and $0 \leq |R_j| \leq 2$ for $j \in \{1, \dots, k\}$. These records correspond to the partial solution of SET-RESTRICTED k -DISJOINT PATHS for G_i with the following properties.

- If $State_j = \text{Not initialized}$, then $s_j, t_j \notin V_{G_i}$. If $R_j = \emptyset$, then (s_j, t_j) -paths have no vertices in G_i in the partial solution. If $R_j = \langle z \rangle$, then z is the unique vertex of a (s_j, t_j) -path in G_i in the partial solution. If $R_j = \langle z_1, z_2 \rangle$, then z_1, z_2 are vertices in a (s_j, t_j) -path, z_1 is the predecessor of z_2 in the path, and this path has no other vertices in G_i .
- If $State_j = \text{Started from } s$, then $s_j \in V_{G_i}$, $t_j \notin V_{G_i}$ and R_j contains either one or two vertices. If $R_j = \langle z \rangle$, then the partial solution contains an (s_j, z) -path with the unique vertex $z \in X_i$. If $R_j = \langle z_1, z_2 \rangle$, then the partial solution contains an (s_j, z_2) -path such that z_1 is the predecessor of z_2 with exactly two vertices $z_1, z_2 \in X_i$.
- If $State_j = \text{Started from } t$, then $s_j \notin V_{G_i}$, $t_j \in V_{G_i}$ and R_j contains either one or two vertices. If $R_j = \langle z \rangle$, then the partial solution contains a (z, t_j) -path with the unique vertex $z \in X_i$. If $R_j = \langle z_1, z_2 \rangle$, then the partial solution contains an (z_1, t_j) -path such that z_2 is the successor of z_1 with exactly two vertices $z_1, z_2 \in X_i$.
- If $State_j = \text{Completed}$, then $s_j \in V_{G_i}$, $t_j \in V_{G_i}$. The partial solution in this case contains an (s_j, t_j) -path, and R_j is the set of vertices of this path in X_i . If $R_j = \langle z_1, z_2 \rangle$, then z_1 is the predecessor of z_2 in the path.

Observe that since we are solving the decision problem, we do not keep (s_j, t_j) -paths or their subpaths themselves. The properties of the algorithm are summarized in the following theorem.

Theorem 2. *The SET-RESTRICTED k -DISJOINT PATHS problem can be solved on chordal graphs in time $n^{O(k)}$.*

For our purposes, we need to generalize Theorem 2 from paths to trees. Instead of terminal pairs (s_i, t_i) we speak of *terminal sets* S_i , each contained in a subset U_i called the *domain* of S_i . Moreover, we say that a tree T_i containing S_i is an S_i -tree. Then an S_i -tree and an S_j -tree are *vertex-disjoint* if they have no common vertices except the vertices in $S_i \cap S_j$. The SET-RESTRICTED k -DISJOINT TREES problem takes as input a graph G , terminal sets S_1, \dots, S_k , and domains U_1, \dots, U_k , and asks whether G contains mutually vertex-disjoint trees T_1, \dots, T_k such that T_i is an S_i -tree containing only vertices from U_i for $i = 1, \dots, k$.

Corollary 1. *The SET-RESTRICTED k -DISJOINT TREES problem can be solved on chordal graphs in $n^{O(p)}$ time, where $p = \sum_{i=1}^k |S_i|$ is the total size of the terminal sets.*

4 Contractions and Induced Minors in Chordal Graphs

First we give a structural characterization of chordal graphs that contain a fixed graph H as a contraction. Then we present our polynomial-time algorithm for solving H -CONTRACTIBILITY on chordal graphs for any fixed graph H , and show how it can be used to solve H -INDUCED MINOR as well.

For the statements of the structural results below, let G be a connected chordal graph, let \mathcal{T}_G be a clique tree of G , and let H be a graph with $V_H = \{x_1, \dots, x_k\}$. For a set of vertices $A \subseteq V_G$, we let $G(A)$ denote the induced subgraph of G obtained by recursively deleting simplicial vertices that are not in A . Since every leaf in any clique tree contains at least one simplicial vertex, we immediately obtain Lemma 1 below. This lemma, in combination with Lemma 2, is crucial for the running time of our algorithm.

Lemma 1. *For any set $A \subseteq V_G$, every clique tree of $G(A)$ has at most $|A|$ leaves.*

Lemma 2. *The graph H is a contraction of G if and only if there is a set $A \subseteq V_G$ such that $|A| = k$ and H is a contraction of $G(A)$.*

Proof. First suppose that H is a contraction of G . Let \mathcal{W} be an H -witness structure \mathcal{W} of G . For each $i \in \{1, \dots, k\}$, we choose an arbitrary vertex $a_i \in W(x_i)$, and let $A = \{a_1, \dots, a_k\}$. Suppose that G has a simplicial vertex $v \notin A$, and assume without loss of generality that $v \in W(x_1)$. Because $v \neq a_1$ and $a_1 \in W(x_1)$, we find that $|W(x_1)| \geq 2$. Hence, $W(x_1)$ contains a vertex u adjacent to v . The graph G' , obtained from G by deleting v , is isomorphic to the graph obtained from G by contracting uv , since v is simplicial. Because u and v belong to the same witness set, namely $W(x_1)$, this implies that H is a contraction of G' . Using these arguments inductively, we find that H is a contraction of $G(A)$.

Now suppose that A is a subset of V_G with $|A| = k$, and that H is a contraction of $G(A)$. Deleting a simplicial vertex v in a graph is equivalent to contracting an

edge incident with v . This means that $G(A)$ is a contraction of G . Because H is a contraction of $G(A)$ and contractibility is a transitive relation, we conclude that H is a contraction of G as well. \square

For a subtree T of \mathcal{T}_G , we say that a vertex $v \in V_G$ is an *inner* vertex for T if v only appears in the maximal cliques of G that are nodes of T . By $I(T) \subseteq V_G$ we denote the set of all inner vertices for T . For a subset $S \subseteq V_G$, let \mathcal{T}_S be the unique minimal subtree of \mathcal{T}_G that contains all maximal cliques of G that have at least one vertex of S ; we say that a vertex v is an *inner* vertex for S if $v \in I(\mathcal{T}_S)$, and we set $I(S) = I(\mathcal{T}_S)$. Lemma 4 below provides an alternative and useful structural description of G if it contains H as a contraction. We need the following lemma to prove Lemma 4.

Lemma 3. *Let $S \subseteq V_G$ and let T be a subgraph of G that is a tree such that $S \subseteq V_T \subseteq I(S)$. Then $K \cap V_T \neq \emptyset$ for each node K of \mathcal{T}_S .*

Proof. Let K be a node of \mathcal{T}_S . If $K \cap S \neq \emptyset$, then clearly $K \cap V_T \neq \emptyset$. Suppose that $K \cap S = \emptyset$. Because \mathcal{T}_S is the unique minimal subtree of \mathcal{T}_G that contains all maximal cliques of G that have at least one vertex of S , we find that K separates two nodes K_1 and K_2 in \mathcal{T}_S for which $K_1 \cap S \neq \emptyset$ and $K_2 \cap S \neq \emptyset$. This means that K separates two vertices $u \in K_1 \cap S$ and $v \in K_2 \cap S$ in G . Since T is a tree and $u, v \in V_T$, at least one vertex of T must be in K . \square

Let l denote the number of leaves in \mathcal{T}_G ; if \mathcal{T}_G consists of one node, then we say that this node is a leaf of \mathcal{T}_G .

Lemma 4. *The graph H is a contraction of G if and only if there are mutually disjoint nonempty sets of vertices $S_1, \dots, S_k \subseteq V_G$, each of size at most l , such that*

1. $V_G \subseteq I(S_1) \cup \dots \cup I(S_k)$;
2. $V_{\mathcal{T}_{S_i}} \cap V_{\mathcal{T}_{S_j}} \neq \emptyset$ if and only if $x_i x_j \in E_H$ for $1 \leq i < j \leq k$;
3. G has mutually vertex-disjoint trees T_1, \dots, T_k with $S_i \subseteq V_{T_i} \subseteq I(S_i)$ for $1 \leq i \leq k$.

Proof. First suppose that H is a contraction of G . Consider a corresponding H -witness structure \mathcal{W} of G . For $i = 1, \dots, k$, let \mathcal{T}_i be the subgraph of \mathcal{T}_G induced by the maximal cliques of G that contain one or more vertices of $W(x_i)$. Because each $W(x_i)$ induces a connected subgraph of G , each \mathcal{T}_i is connected. This means that \mathcal{T}_i is a subtree of \mathcal{T}_G , i.e., $\mathcal{T}_i = \mathcal{T}_{W(x_i)}$. We construct S_i as follows. For each leaf K of \mathcal{T}_i , we choose a vertex of $W(x_i) \cap K$ and include it in the set S_i . Because \mathcal{T}_G has l leaves, each \mathcal{T}_i has at most l leaves. Hence, $|S_i| \leq l$ for $i = 1, \dots, k$. We now check conditions 1–3 of the lemma.

1. By construction, we have $\mathcal{T}_i = \mathcal{T}_{S_i}$. All vertices of $W(x_i)$ are inner vertices for \mathcal{T}_i , so $W(x_i) \subseteq I(\mathcal{T}_i) = I(\mathcal{T}_{S_i}) = I(S_i)$. Hence, $V_G = \bigcup_{i=1}^k W(x_i) \subseteq \bigcup_{i=1}^k I(S_i)$.

2. Any two vertices $u, v \in V_G$ are adjacent if and only if there is a maximal clique K in G containing u and v . Hence, two witness sets $W(x_i)$ and $W(x_j)$ are adjacent if and only if there is a maximal clique K in G such that $K \cap W(x_i) \neq \emptyset$ and $K \cap W(x_j) \neq \emptyset$. This means that $W(x_i)$ and $W(x_j)$ are adjacent if and only if $V_{T_i} \cap V_{T_j} \neq \emptyset$. It remains to recall that $T_i = T_{S_i}$ and $T_j = T_{S_j}$, and that two witness sets $W(x_i)$ and $W(x_j)$ are adjacent if and only if $x_i x_j \in E_H$.

3. Every $G[W(x_i)]$ is a connected graph. Hence, every $G[W(x_i)]$ contains a spanning tree T_i . Because the sets $W(x_1), \dots, W(x_k)$ are mutually disjoint, the trees T_1, \dots, T_k are mutually vertex-disjoint. Moreover, as we already deduced, $S_i \subseteq V_{T_i} = W(x_i) \subseteq I(S_i)$ for $i = 1, \dots, k$.

Now suppose that there are mutually disjoint nonempty sets of vertices $S_1, \dots, S_k \subseteq V_G$, each of size at most l , that satisfy conditions 1–3 of the lemma. By condition 3, there exist mutually vertex-disjoint trees T_1, \dots, T_k with $S_i \subseteq V_{T_i} \subseteq I(S_i)$ for $i = 1, \dots, k$. By condition 1, we have $V_G \subseteq I(S_1) \cup \dots \cup I(S_k)$. This means that there is a partition X_1, \dots, X_k of $V_G \setminus \bigcup_{i=1}^k V_{T_i}$, where some of the sets X_i can be empty, such that $X_i \subseteq I(S_i)$ for $i = 1, \dots, k$. Let $W(x_i) = V_{T_i} \cup X_i$ for $i = 1, \dots, k$. We claim that the sets $W(x_1), \dots, W(x_k)$ form an H -witness structure of G . By definition, $W(x_1), \dots, W(x_k)$ are mutually disjoint, nonempty, and $W(x_1) \cup \dots \cup W(x_k) = V_G$, i.e., they form a partition of V_G . It remains to show that these sets satisfy conditions (i) and (ii) of the definition of an H -witness structure.

(i) By definition, each tree T_i is connected. By Lemma 3, each node K of T_{S_i} contains a vertex of T_i . By definition, $X_i \subseteq I(S_i)$, which implies that for each $v \in X_i$, there is a node K in T_{S_i} such that $v \in K$. Because K is a clique in G , we then find that v is adjacent to at least one vertex of T_i . Therefore, each $W(x_i)$ induces a connected subgraph of G .

(ii) For the forward direction, suppose that $W(x_i)$ and $W(x_j)$ are two adjacent witness sets. Then there exist two vertices $u \in W(x_i)$ and $v \in W(x_j)$ such that $uv \in E_G$. Let K be a maximal clique that contains both u and v . Because $u \in W(x_i)$ and $v \in W(x_j)$, we find that K is a node of T_{S_i} and of T_{S_j} , respectively. Hence, $V_{T_{S_i}} \cap V_{T_{S_j}} \neq \emptyset$, which means that $x_i x_j \in E_H$ by 2.

For the reverse direction, let x_i and x_j be two adjacent vertices in H . By condition 2, we find that $V_{T_{S_i}} \cap V_{T_{S_j}} \neq \emptyset$. Hence, there is a node $K \in V_{T_{S_i}} \cap V_{T_{S_j}}$. By Lemma 3, we deduce that K contains a vertex $u \in V_{T_i}$ and a vertex $v \in V_{T_j}$. Because K is a clique in G , this means that u and v are adjacent. Because $V_{T_i} \subseteq W(x_i)$ and $V_{T_j} \subseteq W(x_j)$, we obtain $u \in W(x_i)$ and $v \in W(x_j)$, respectively. Hence, $W(x_i)$ and $W(x_j)$ are adjacent. \square

We are now ready to describe our algorithm for H -CONTRACTIBILITY on chordal graphs, for any fixed graph H . Although the presented algorithm solves the decision problem, it can be modified to produce an H -witness structure if one exists.

Theorem 3. *For any fixed graph H , the H -CONTRACTIBILITY problem can be solved in polynomial time on chordal graphs.*

Proof. Let G be a chordal graph on n vertices and let H be a graph on k vertices with $V_H = \{x_1, \dots, x_k\}$. If $k > n$ or the number of connected components of G and H are different, then return No. Suppose that G and H have $r > 1$ connected components G_1, \dots, G_r and H_1, \dots, H_r , respectively. For each permutation $\langle i_1, \dots, i_r \rangle$ of the ordered set $\langle 1, \dots, r \rangle$, check whether H_{i_j} is a contraction of G_j for every $j \in \{1, \dots, r\}$. Return Yes if this is the case for some permutation, and No otherwise. Hence, we may assume that G and H are connected.

Construct a clique tree T_G of G . If T_G has at least $k + 1$ leaves, then consider each set $A \subseteq V_G$ with $|A| = k$, and continue with $G(A)$ instead of G . This is allowed due to Lemma 2. Note that a clique tree of $G(A)$ has at most $|A| = k$ leaves due to Lemma 1. Hence, we may assume that T_G has at most k leaves.

Consider each collection of mutually disjoint sets S_1, \dots, S_k , where each S_i is a subset of V_G with $1 \leq |S_i| \leq k$. For each collection S_1, \dots, S_k , construct the subtrees T_{S_1}, \dots, T_{S_k} of T_G and the sets $I(S_1), \dots, I(S_k)$, and test whether conditions 1–3 of Lemma 4 are satisfied. If so, the algorithm returns Yes; otherwise, it returns No. Correctness of this algorithm is an immediate consequence of Lemma 4.

We now analyze the running time. The number of permutations of the ordered set $\langle 1, \dots, r \rangle$ is at most $r! \leq k!$. Constructing T_G takes linear time. The number of k -element subsets A of V_G is $n^{O(k)}$. The number of collections of sets S_1, \dots, S_k with $|S_i| \leq k$ for $i = 1, \dots, k$ is $n^{O(k^2)}$. Constructing subtrees T_{S_1}, \dots, T_{S_k} of T_G and sets $I(S_1), \dots, I(S_k)$, and testing if conditions 1–2 of Lemma 4 are satisfied, takes $n^{O(1)}$ time. As a result of Corollary 1, testing whether condition 3 of Lemma 4 is satisfied takes $n^{O(k^2)}$ time. Hence, the total running time is $n^{O(k^2)}$. Consequently, we can test in this running time whether a given chordal graph G contains a given graph H as a contraction. When the graph H , and hence k , is fixed, the running time is polynomial in n . \square

The algorithm for H -CONTRACTIBILITY can be modified for H -INDUCED MINOR, but it is easier to use the following observation. Let $P_1 \bowtie G$ denote the graph obtained from a graph G by adding a new vertex and making it adjacent to every vertex of G .

Lemma 5 ([9]). *Let G and H be two arbitrary graphs. Then G contains H as an induced minor if and only if $(P_1 \bowtie G)$ contains $(P_1 \bowtie H)$ as a contraction.*

Since $P_1 \bowtie G$ is chordal whenever G is chordal, we can combine Lemma 5 with Theorem 3 to obtain the following result, with the same running time as in the proof of Theorem 3.

Corollary 2. *For any fixed graph H , the H -INDUCED MINOR problem can be solved in polynomial time on chordal graphs.*

5 Concluding Remarks

Kammer and Tholey [10] improved the running time of DISJOINT PATHS from cubic for general graphs [16] to linear for chordal graphs. Is SET-RESTRICTED

DISJOINT PATHS fixed-parameter tractable on chordal graphs, when parameterized by k ?

Recall that the problems CONTRACTIBILITY and INDUCED MINOR are $W[1]$ -hard for pairs (G, H) that are chordal graphs, when parameterized by $|V_H|$ [8]. Is either of these problems fixed-parameter tractable on pairs (G, H) that are interval graphs, which constitute an important subclass of chordal graphs?

References

1. Belmonte, R., Heggernes, P., van ’t Hof, P.: Edge Contractions in Subclasses of Chordal Graphs. In: Ogiwara, M., Tarui, J. (eds.) TAMC 2011. LNCS, vol. 6648, pp. 528–539. Springer, Heidelberg (2011)
2. Blair, J.R.S., Peyton, B.: An introduction to chordal graphs and clique trees. *Graph Theory and Sparse Matrix Computation* 56, 1–29 (1993)
3. Brouwer, A.E., Veldman, H.J.: Contractibility and NP-completeness. *Journal of Graph Theory* 11, 71–79 (1987)
4. Dirac, G.A.: On rigid circuit graphs. *Anh. Math. Sem. Univ. Hamburg* 25, 71–76 (1961)
5. Fellows, M.R., Kratochvíl, J., Middendorf, M., Pfeiffer, F.: The complexity of induced minors and related problems. *Algorithmica* 13, 266–282 (1995)
6. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B* 16, 47–56 (1974)
7. Golovach, P.A., Kamiński, M., Paulusma, D.: Contracting a Chordal Graph to a Split Graph or a Tree. In: Murlak, F., Sankowski, P. (eds.) MFCS 2011. LNCS, vol. 6907, pp. 339–350. Springer, Heidelberg (2011)
8. Golovach, P.A., Kamiński, M., Paulusma, D., Thilikos, D.M.: Containment relations in split graphs (manuscript)
9. van ’t Hof, P., Kamiński, M., Paulusma, D., Szeider, S., Thilikos, D.M.: On graph contractions and induced minors. *Discrete Applied Mathematics* (to appear)
10. Kammer, F., Tholey, T.: The k -Disjoint Paths Problem on Chordal Graphs. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 190–201. Springer, Heidelberg (2010)
11. Kamiński, M., Paulusma, D., Thilikos, D.M.: Contractions of Planar Graphs in Polynomial Time. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 122–133. Springer, Heidelberg (2010)
12. Kloks, T.: Treewidth, Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
13. Levin, A., Paulusma, D., Woeginger, G.J.: The computational complexity of graph contractions I: polynomially solvable and NP-complete cases. *Networks* 51, 178–189 (2008)
14. Levin, A., Paulusma, D., Woeginger, G.J.: The computational complexity of graph contractions II: two tough polynomially solvable cases. *Networks* 52, 32–56 (2008)
15. Matoušek, J., Thomas, R.: On the complexity of finding iso- and other morphisms for partial k -trees. *Discrete Mathematics* 108, 343–364 (1992)
16. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B* 63, 65–110 (1995)
17. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing* 13, 66–579 (1984)

Recognizing Polar Planar Graphs Using New Results for Monopolarity

Van Bang Le and Ragnar Nevries

University of Rostock, Germany

le@informatik.uni-rostock.de, ragnar.nevries@uni-rostock.de

Abstract. Polar and monopolar graphs are natural generalizations of bipartite or split graphs. A graph $G = (V, E)$ is polar if its vertex set admits a partition $V = A \cup B$ such that A induces a complete multipartite and B the complement of a complete multipartite graph. If A is even a stable set then G is called monopolar.

Recognizing general polar or monopolar graphs is NP-complete and, as yet, efficient recognition is available only for very few graph classes.

This paper considers monopolar and polar graphs that are also planar. On the one hand, we show that recognizing these graphs remains NP-complete, on the other hand we identify subclasses of planar graphs on which polarity and monopolarity can be checked efficiently. The new NP-completeness results cover very restricted graph classes and are sharper than all previous known cases. On the way to the positive results, we develop new techniques for efficient recognition of subclasses of monopolar graphs. These new results extend nearly all known results for efficient monopolar recognition.

1 Introduction

Polar graphs, introduced and studied by Tyshkevich and Chernyak [20,21] in 1985, are a natural generalization of bipartite and split graphs. A graph $G = (V, E)$ is *polar* if its vertex set admits a *polar partition*. A partition $V = A \cup B$ is *polar* if the subgraph $G[A]$ induced by A is complete multipartite graph, that is a \overline{P}_3 -free graph, and $G[B]$ is the complement of a complete multipartite graph, that is a P_3 -free graph.

Chernyak et al. [4] show that *polarity*, the problem to recognize if a given graph is polar, is NP-complete in general. The best known hardness result is given by Churchley and Huang [6] who show that polarity remains NP-complete for triangle-free graphs. On the positive side, polynomial time algorithms for polarity have been found for cographs [13], chordal graphs [11], line graphs [5,12,16] and permutation graphs [10]. Moreover, as polarity can be expressed in monadic second order logic without edge set quantification, efficient algorithms are directly available for graph classes with bounded tree-width [17] or clique-width [8] and, hence, for distance-hereditary graphs, outer planar graphs and series parallel graphs.

When thinking about polar recognition it is often useful to consider two subclasses of polar graphs, the so-called *monopolar* graphs and *unipolar* graphs.

A graph $G = (V, E)$ is monopolar (unipolar) if it has a polar partition $V = A \cup B$ where A is a stable set (a clique). These two subclasses are interesting because most of the known efficient recognition algorithms for polar graphs of a specific graph class first check unipolarity and monopolarity and then, if both are not true, test if there is a polar partition that is neither uni- nor monopolar. For complexity considerations the monopolar graphs are interesting, because it is known from [21] that unipolar graphs can be efficiently recognized in general.

Monopolarity, the problem that asks if a graph is monopolar, remains NP-complete [15], even for triangle-free graphs [6]. This may give the impression that checking monopolarity is the hard core when checking polarity, but this is confuted by Churchley and Huang in [6] who show that on claw-free graphs monopolarity is efficiently solvable while deciding polarity remains NP-complete. Nevertheless, we are not aware of a graph class on which polarity is easy but monopolarity is hard.

Our contributions. First we give two NP-completeness results that have lots of interesting implications. That includes NP-completeness for monopolarity and polarity on planar graphs and triangle-free graphs of maximum degree 3 as well as NP-completeness for polarity on claw-free co-planar graphs and $(2K_2, C_5)$ -free graphs, a generalization of split graphs, that implies NP-completeness for the well known classes of hole-free and P_5 -free graphs.

We develop a new technique for recognizing monopolar graphs based on a simple 2-SAT reduction. The reduction works only on a special graph class, but we can extend this class in two different ways, gaining efficient monopolarity algorithms for a superclass of chair-free graphs and a superclass of hole-free graphs. Since chair-free graphs generalize line and claw-free graphs and hole-free graphs generalize cographs and chordal graphs, these new results cover all known cases except permutation graphs and some of the graph classes with bounded tree- or clique-width. Actually all new results solve a tightened version of monopolarity that we call monopolar extension. Here, an incomplete, possibly empty, vertex partition $A' \cup B' \subseteq V$ is given and the question is if it can be extended to a monopolar partition $A \cup B = V$ such that $A' \subseteq A$ and $B' \subseteq B$.

Together with the NP-completeness results we generalize the findings in [6] and show for some more graph classes, including hole-free graphs, that monopolarity is tractable but polarity not.

Up to know, this paper is the first that considers planar graphs in the context of polarity. Since we show that monopolarity and polarity are both NP-complete on planar graphs, we inspect subclasses of planar graphs. We show that polarity is efficiently solvable on maximal planar graphs and claw-free planar graphs. The latter one is interesting because the restriction to claw-freeness or planarity alone does not make polarity easier.

Due to space limitations, we concentrate on giving our results and discussing their consequences. Detailed descriptions of the reductions and the proofs will be given in the full version.

Notion and definitions. All graphs $G = (V, E)$ considered are finite, undirected, and simple. Hence, $V = V(G)$ is a finite set of vertices and the edge set

$E = E(G)$ consists of unordered pairs xy with $x, y \in V$ and $x \neq y$. As usual, \overline{G} denotes the complement of G . Given a set of vertices $X \subseteq V$, the subgraph induced by X is written $G[X]$ and $G - X$ stands for $G[V \setminus X]$. For a vertex $v \in V$, we also write $G - v$ for $G[V \setminus \{v\}]$ and for a subset X of vertices we often identify X with $G[X]$. A *block* of a graph G is a maximal induced 2-connected subgraph of G and a vertex v of G is a *cutvertex* if $G - v$ is not connected. We call a block *trivial* if it is a single edge, otherwise it is *non-trivial*. A graph G is said to be F -free, for some other graph F , if G does not contain an induced subgraph isomorphic to F . Moreover, for a set \mathcal{F} of graphs, G is called \mathcal{F} -free if it is F -free for every member $F \in \mathcal{F}$.

The *neighborhood* $N(v)$ of a vertex v in G is the set of all vertices adjacent to v and $\deg(v) = |N(v)|$ is the *degree* of v in G ; set $N[v] = N(v) \cup \{v\}$. A path with k vertices and $k - 1$ edges is denoted by P_k , a cycle of length k denoted by C_k . A C_3 is also called a *triangle* and a C_k , $k \geq 5$, is also called a *hole*. Figure 1 shows the small graphs *claw*, *paw* and *diamond*. We will denote by $P(s, t) = P(t, s)$ a paw in which s is the vertex of degree 3 and t is the vertex of degree 1, and by $D(s, t) = D(t, s)$ a diamond where s and t are the degree 2-vertices.

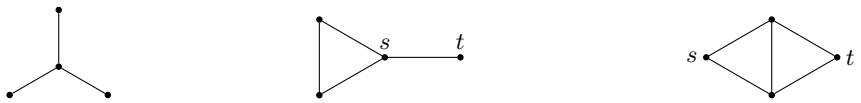


Fig. 1. The small graphs claw (left), paw, diamond (right)

Since a graph is monopolar if and only if its connected components are monopolar and also removing vertices with degree one does not change monopolarity, it is sound to assume that the input of monopolarity algorithms is restricted to connected graphs with minimum degree two.

2 Hardness Results

Using two reductions from MONOTONE PLANAR ONE-IN-THREE 3-SAT (see [17][18][19]), we can show:

Theorem 1. *It is NP-complete to decide if a triangle-free planar graph of maximum degree 3 is monopolar, respectively, polar.*

Theorem 2. *For every fixed $k \geq 4$, it is NP-complete to decide if a (C_4, \dots, C_k) -free planar graph of maximum degree 3 is monopolar, respectively, polar.*

Since polar graphs are closed under taking complements, these theorems imply:

Theorem 3. *Polarity is NP-complete on $3K_1$ -free co-planar n -vertex graphs with minimum degree $n - 4$ and on $(\overline{C}_4, \dots, \overline{C}_k)$ -free co-planar graphs for every fixed $k \geq 4$.*

The following easily provable implications are quite interesting and show that the results are best possible in several means.

Corollary 1. *Monopolarity and polarity are NP-complete on 3-colorable graphs. Polarity is NP-complete on claw-free co-planar graphs and on $(2K_2, C_5)$ -free graphs, and therefore on hole-free graphs and P_5 -free graphs.*

The best known NP-completeness results show that monopolarity is NP-complete on triangle-free graphs and polarity is NP-complete on claw-free graphs [6], which are both included in our results.

In terms of maximum degree, Theorem 1 is best possible, because all graphs with maximum degree 2 are always monopolar. In terms of forbidden induced cycles, Theorem 2 is best possible, because (C_4, \dots) -free graphs are chordal, a class with polynomial time monopolarity and polarity as shown in [1]. The fact that the two problems are hard on 3-colorable graphs is interesting, because since every bipartite graph is monopolar and polar, the problems are trivial on 2-colorable graphs. Theorem 2 and Corollary 1 show that polarity is NP-complete on co-planar graphs, $2K_2$ -free, P_5 -free, chair-free and hole-free graphs. But the next sections show that monopolarity is tractable on the last three classes. It is also tractable on $2K_2$ -free and co-planar graphs, because graphs of these classes have only a polynomial number of inclusion-maximal stable sets (see [3]). Hence, the results go nicely with the results of Churchley and Huang about claw-free graphs, giving more classes that have efficient monopolar recognition while polarity remains hard.

3 A 2-SAT Approach and a Superclass of Chair-Free Graphs

In this section we introduce a graph class on which monopolar extension can be reduced to an instance of 2-SAT. Consequently, monopolar extension can be done efficiently for this graph class, because testing if a 2-CNF formula is satisfiable can be done in linear time (cf. [2, 9, 14]).

Suppose that $G = (V, E)$ admits a monopolar partition $V = A \cup B$. Then for each edge $uv \in E$ it is true $u \notin A$ or $v \notin A$. Further, for each induced paw $P(s, t)$ and for each induced diamond $D(s, t)$ in G it is true $s \in A$ or $t \in A$. Finally, each edge uv of any induced C_4 in G fulfills $u \in A$ or $v \in A$. This facts motivate the following concept:

For a given graph $G = (V, E)$ and two sets $A', B' \subseteq V$ a particular 2-SAT instance $F(G)$ is created as follows:

- The boolean variables are the vertices of G ,
- for each vertex $u \in A'$, u is the corresponding *must-clause*,
- for each vertex $u \in B'$, $\neg u$ is the corresponding *need-not-clause*,
- for each edge uv of G , $(\neg u \vee \neg v)$ is the corresponding *edge-clause*,
- for each induced paw $P(s, t)$ of G , $(s \vee t)$ is the corresponding *paw-clause*,

- for each induced diamond $D(s, t)$ of G , $(s \vee t)$ is the corresponding *diamond-clause*,
- for each induced $C_4 = uvwx$ of G , $(u \vee v)$, $(v \vee w)$, $(w \vee x)$ and $(x \vee u)$ are the corresponding *C_4 -clauses*.

The formula $F(G)$ is the conjunction of all must-clauses, need-not-clauses, edge-clauses, paw-clauses, diamond-clauses, and C_4 -clauses.

In a graph G , an induced P_3 is said to be *good* if at least one of the vertices of the P_3 belongs to a triangle in G or at least one of the edges of the P_3 belongs to an induced C_4 , and *bad* otherwise. We say that a graph G is *good* if each induced P_3 in G is good.

Lemma 1. *Let G be a good graph. Then G has a monopolar partition $A \cup B$ with $A' \subseteq A$ and $B' \subseteq B$ if and only if $F(G)$ is satisfiable.*

Theorem 4. *Monopolar extension is polynomial time solvable on good graphs.*

As examples, 2-connected hole-free graphs and maximal planar graphs are good, where a *maximal planar graph* (or a *planar triangulation*) is a planar graph to which no new edge can be added without violating its planarity:

Lemma 2. *2-connected hole-free graphs and maximal planar graphs are good.*

Since monopolar extension is a generalization of monopolarity, by Theorem 4 and Lemma 2 we obtain:

Corollary 2. *There is a polynomial time algorithm for recognizing monopolar good graphs. In particular, monopolar 2-connected hole-free graphs and monopolar maximal planar graphs can be recognized in polynomial time.*

The following and the next section use the 2-SAT technique to develop efficient algorithms for monopolar extension on greater graph classes than good graphs. Here, we show that monopolar extension is efficiently solvable on a large graph class that contains the chair-free graphs.

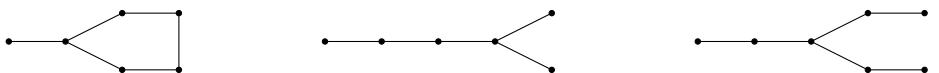


Fig. 2. The graphs F_1 , F_2 and F_3

Consider the graphs F_1 , F_2 and F_3 shown in Figure 2. Let uvw a P_3 of an (F_1, F_2, F_3) -free graph G such that uvw is not good. For monopolarity it turns out that it is sufficient to check monopolarity on $G - v$, and doing this step recursively reduces the problem to monopolarity on good graphs. This approach does not work for monopolar extension, but here with a more sophisticated decomposition of the graph the problem can be reduced to a 2-SAT instance using the findings of the last section:

Theorem 5. *Monopolar extension is polynomial time solvable on (F_1, F_2, F_3) -free graphs.*

By Theorem 5, the following subclasses of (F_1, F_2, F_3) -free graphs allow efficient monopolar extension:

Corollary 3. *There are polynomial time algorithms for monopolarity on chair-free, claw-free, and P_5 -free graphs, line graphs and cographs.*

We remark that [6] gives an $O(n^3)$ algorithm for recognizing claw-free graphs, while we give here an $O(n^4)$ recognition algorithm for the class of chair-free graphs, a class that properly contains all claw-free graphs.

4 A Superclass of Hole-Free Graphs

This section uses the 2-SAT approach of section 3 in a different way. We consider *locally good* graphs, the graph class where the blocks of the graphs are good. We can solve monopolar extension efficiently on these graphs by combining the 2-SAT approach on the blocks with a multicoloring approach on the treelike structure of the blocks of a graph.

The multicoloring approach was presented in [11] for solving monopolar extension on chordal graphs in linear time. At first, we repeat some definitions according to [11]. We can treat a partition $A \cup B$ of the vertices of a graph G as a coloring in which all vertices of A have color **a** (aqua) and all vertices of B have color **b** (blue). A *multicoloring* ℓ of a graph assigns a list $\ell(v)$ of either no color, one or two colors from the color set $\{\mathbf{a}, \mathbf{b}\}$ to every vertex v . We call a vertex v **aqua** if $\ell(v) = \{\mathbf{a}\}$ and **blue** if $\ell(v) = \{\mathbf{b}\}$. A multicoloring ℓ is a *coloring* if every vertex is either **aqua** or **blue**. A coloring or multicoloring ℓ contains a multicoloring ℓ' if ℓ' assigns v a subset of $\ell(v)$ for every vertex v . Then we write $\ell' \preccurlyeq \ell$. We call a multicoloring ℓ *monopolar*, if it contains a coloring ℓ' that implies a monopolar partition of G .

The nodes of the *block-cutvertex tree* T of G are the blocks and cutvertices of G and a block node and a cutvertex node are connected by an edge if the cutvertex belongs to the block. Let T be rooted at some cutvertex node r and so every other node v of T has a *parent*. All nodes with parent p are *children* of p . Analogously, the *grandchildren* of v are the children of the children of v . By T_v we denote the subtree of T rooted at node v and we let $G[T_v]$ be the subgraph of G induced by the vertices of the blocks contained in T_v .

Like the chordal case presented in [11], our algorithm starts with a given graph $G = (V, E)$ and a multicoloring ℓ^0 of G . The algorithm decides if ℓ^0 contains at least one monopolar coloring of G . Therefor, it calculates the block-cutvertex tree T of G and traverses T from bottom to top. In every step, the multicoloring is refined by deleting useless colors from cutvertices. After the root is processed, ℓ^0 contains a monopolar coloring of G if and only if the resulting multicoloring assigns at least one color to every vertex.

Since the blocks of G are good, we can decide efficiently for a block if it admits a monopolar coloring contained in ℓ^0 . The difficulty arises on the cutvertices:

If the cutvertex is colored **blue** and has **blue** neighbors in two blocks, then there is a **blue** colored P_3 in the combined coloring. To handle this problem in the traversal of T , we need to identify cutvertices that already have **blue** neighbors, which is the crucial difference between the chordal case  and the new technique. Whereas in chordal blocks with more than two vertices, every vertex has surely a **blue** neighbor in every monopolar coloring, in good blocks this depends on the structure of the block and the current multicoloring. Therefor, we introduce the following new concept:

Let v be a cutvertex and B a child of v in T_v . We call v *critical* in $G[T_B]$ under a multicoloring ℓ if $\mathbf{b} \in \ell(v)$ and in every monopolar coloring ℓ_{T_B} of $G[T_B]$ with $\ell_{T_B} \preccurlyeq \ell$ and $\ell_{T_B}(v) = \mathbf{b}$ there is a neighbor w of v in $G[T_B]$ with $\ell_{T_B}(w) = \mathbf{b}$. If there is at least one child B of a cutvertex v such that v is critical in T_B under ℓ , we also say that v is critical under ℓ . Notice that every cutvertex v with $\ell(v) = \mathbf{a}$ is not critical under ℓ .

Now we can give the recognition algorithm for polar locally good graphs:

Recognition Algorithm

1. Input: Connected locally good graph G and initial multicoloring ℓ^0 .
2. Construct the block-cutvertex tree T of G , root T at some fixed cutvertex node r and let $\ell := \ell^0$.
3. Traverse T in postorder processing every cutvertex node v :
 - (a) If ℓ contains no monopolar coloring of $G[T_B]$ for a child B of v that colors v **aqua**, then let $\ell(v) := \ell(v) \setminus \{\mathbf{a}\}$.
 - (b) If ℓ contains no monopolar coloring of $G[T_B]$ for a child B of v that colors v **blue**, then let $\ell(v) := \ell(v) \setminus \{\mathbf{b}\}$.
 - (c) Construct the set C_v that contains every child B of v where v is critical in $G[T_B]$ under ℓ .
 - (d) If C_v contains more than one child, then let $\ell(v) := \ell(v) \setminus \{\mathbf{b}\}$.
 - (e) If $\ell(v) = \emptyset$, STOP: “ ℓ^0 contains no monopolar coloring of G .”
4. Output: “ ℓ^0 contains a monopolar coloring of G .”

The input sets A' and B' for monopolar extension can be translated into an initial multicoloring ℓ^0 by setting $\ell^0(v) = \{\mathbf{a}\}$ for every vertex $v \in A'$, $\ell^0(v) = \{\mathbf{b}\}$ for every vertex $v \in B'$ and $\ell^0(v) = \{\mathbf{a}, \mathbf{b}\}$ for all other vertices. We can show that by using the 2-SAT technique we can process every step of the algorithm efficiently, in particular using special monopolar extension instances on the blocks we can identify critical cutvertices efficiently. By showing that the algorithm is correct, we get:

Theorem 6. *Monopolar extension, and hence monopolarity, is polynomial time solvable on locally good graphs.*

Locally maximal planar graphs are the graphs whose blocks are maximal planar. With Lemma , we can immediately follow:

Corollary 4. *Monopolarity is polynomial time solvable on hole-free graphs and locally maximal planar graphs.*

5 Some Efficiently Solvable Cases of Polarity for Planar Graphs

According to Section 2, polarity remains NP-complete for many restrictions of planar graphs and consequently, it is interesting to identify subclasses where polarity is polynomial time decidable. A first example of this kind is easily found, the outerplanar graphs, because they have bounded treewidth. This section provides two other examples, namely maximal planar graphs and claw-free planar graphs.

We begin by analysing possible polar partitions of K_5 -free graphs $G = (V, E)$ that do not contain a $K_{3,3}$ as a not necessarily induced subgraph. Clearly, these graphs contain all planar graphs and thus, our analysis covers planar graphs, too. Now, let $V = A \cup B$ be a polar partition of G , i.e., A consists of stable sets A_1, \dots, A_t such that for all $x, y \in A$ it is true $xy \in E$ if and only if $x \in A_i$ and $y \in A_j$ for some i, j with $i \neq j$. Assume without loss of generality that $1 \leq |A_1| \leq \dots \leq |A_t|$.

Fact 1. *If $|A| \geq 7$ then $t \leq 3$ and $|A| - |A_t| \leq 2$.*

For claw-free planar graphs G , moreover, we have that if $t \geq 2$ then $|A_i| \leq 2$ for all $i \in \{1, \dots, t\}$. Hence, after brute force validating that G has no polar partition with $|A| < 7$ an algorithm may assume that $|A_t| \geq 5$ and thus, $t = 1$. But a polar partition with $t = 1$ is a monopolar partition and if this exists can efficiently be checked on claw-free planar graphs, and hence:

Theorem 7. *Polarity of claw-free planar graphs can be decided in polynomial time.*

For maximal planar graphs G the above fact can be used to reduce polarity to special cases of monopolar extension. In fact, according to Corollary 2 the monopolarity of G is checked in polynomial time, and if this fails the algorithm validates that G has no polar partition with $|A| < 7$. Now, by Fact 1 every polar partition $A \cup B$ of G fulfills that $X := A - A_t$ consists of one or two vertices. To decide if such a polar partition exists, the algorithm can check for every X if $G - X$ has a monopolar partition $A \cup B$ with $A \subseteq \bigcap_{x \in X} N(x)$. This can be done by checking if $G - X$ has a monopolar extension with $A' = \emptyset$ and $B' = V - \bigcap_{x \in X} N(x)$. But, although Corollary 2 states that monopolar extension is efficient on maximal planar graphs, $G - X$ is not necessarily maximal planar. However, if $|X| = 1$ then $G - X$ remains good and by Theorem 4.

Lemma 3. *If $G = (V, E)$ is a maximal planar graph and $x \in V$ then monopolar extension on $G - x$ with $A' = \emptyset$ and $B' = V - N[x]$ can be solved in $O(n^4)$ time.*

If $|X| = 2$, then the problem can be solved even on K_5 -free graph without $K_{3,3}$ as a subgraph by a reduction to 2-SAT:

Lemma 4. *If $G = (V, E)$ is a K_5 -free graph without $K_{3,3}$ as a subgraph and $X = \{x, y\} \subseteq V$ then monopolar extension on $G - X$ and $A' = \emptyset$, $B' = V - N[x] - N[y]$ can be solved in $O(n^3)$ time.*

The previous lemma implies:

Theorem 8. *Polarity of maximal planar graphs can be decided in polynomial time.*

6 Conclusion

In this paper we prove several new NP-completeness results for polarity and monopolarity which are optimal in certain senses. We propose a novel 2-SAT approach and combine it with reductions and a generalization of the multi-coloring approach proposed in [11]. This leads to polynomial time algorithms for monopolar extension of many large graph classes which contain many of the known polynomially solvable cases. The new results for monopolarity on claw-free planar and maximal planar graphs are used to design efficient polarity recognition algorithms for these classes.

Many interesting open questions remain. One of them is: Is there any graph class between line graphs and claw-free graphs that separates NP-complete instances from polynomially solvable ones of the polarity problem on claw-free graphs? One can also ask the same question for polarity of graph classes between chordal graphs and hole-free graphs, for example weakly chordal graphs.

Table 1 summarizes the complexity status of polarity and monopolarity of relevant graph classes.

Table 1. LIN, P, NP-c, and ? means that the complexity status of the corresponding problem is linear, polynomial, NP-complete, unknown, respectively.

Graph class	Monopolarity	Polarity
chordal	LIN ([11])	P ([11])
line graph	P ([5])	P ([5])
line graph of multigraphs	$O(n^3)$ ([6])	?
permutation	P ([10])	P ([10])
weakly chordal	P (this paper)	?
claw-free planar	$O(n^3)$ ([6])	P (this paper)
maximal planar	$O(n^4)$ (this paper)	P (this paper)
claw-free	$O(n^3)$ ([6])	NP-c ([6], this paper)
claw-free co-planar	$O(n^3)$ ([6])	NP-c (this paper)
chair-free	$O(n^4)$ (this paper)	NP-c ([6], this paper)
$(2K_2, C_5)$ -free	$O(n^4)$ (this paper)	NP-c (this paper)
P_5 -free	$O(n^4)$ (this paper)	NP-c (this paper)
hole-free	P (this paper)	NP-c (this paper)
locally good	P (this paper)	NP-c (this paper)
triangle-free planar	NP-c ([6], this paper)	NP-c ([6], this paper)
(C_4, C_5) -free planar	NP-c (this paper)	NP-c (this paper)
3-colorable	NP-c (this paper)	NP-c (this paper)

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. *J. Algorithms* 12, 308–340 (1991)
2. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters* 8, 121–123 (1979); Erratum 14, 195 (1982)
3. Balas, E., Yu, C.S.: On graphs with polynomially solvable maximum-weight clique problem. *Networks* 19, 247–253 (1989)
4. Chernyak, Z.A., Chernyak, A.A.: About recognizing (α, β) classes of polar graphs. *Discrete Math.* 62, 133–138 (1986)
5. Churchley, R., Huang, J.: Line-polar graphs: characterization and recognition (2009) (manuscript); *SIAM Journal on Discrete Mathematics* (accepted)
6. Churchley, R., Huang, J.: The polarity and monopolarity of claw-free graphs (2010) (manuscript) (submitted for publication)
7. Courcelle, B.: The monadic second-order logic of graphs. III. Tree-decompositions, minors and complexity issues. *RAIRO Inform. Theor. Appl.* 26, 257–286 (1992)
8. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* 33, 125–150 (2000)
9. Davis, M., Putnam, H.: A computing procedure for quantification theory. *J. ACM* 7, 201–215 (1960)
10. Ekim, T., Heggernes, P., Meister, D.: Polar Permutation Graphs. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) *IWOCA 2009. LNCS*, vol. 5874, pp. 218–229. Springer, Heidelberg (2009)
11. Ekim, T., Hell, P., Stacho, J., de Werra, D.: Polarity of chordal graphs. *Discrete Math.* 156, 2469–2479 (2008)
12. Ekim, T., Huang, J.: Recognizing line-polar bipartite graphs in time $O(n)$. *Discrete Appl. Math.* (2010)
13. Ekim, T., Mahadev, N.V.R., de Werra, D.: Polar cographs. *Discrete Appl. Math.* 156, 1652–1660 (2008)
14. Even, S., Itai, A., Shamir, A.: On the complexity of timetable and multicommodity flow problems. *SIAM J. Computing* 5, 691–703 (1976)
15. Farrugia, A.: Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *Electron. J. Combin.* 11, #R46 (2004)
16. Huang, J., Xu, B.: A forbidden subgraph characterization of line-polar bipartite graphs. *Discrete Appl. Math.* 158, 666–680 (2007)
17. Laroche, P.: Planar 1-in-3 satisfiability is NP-complete. In: *ASMICS Workshop on Tilings, Deuxième Journées Polyominos et pavages*, Ecole Normale Supérieure de Lyon (1992)
18. Moore, C., Robson, J.M.: Hard Tiling Problems with Simple Tiles. *Discrete & Computational Geometry* 26, 573–590 (2001)
19. Mulzer, W., Rote, G.: Minimun-Weight Triangulation is NP-hard. *J. ACM* 55, Article No. 11 (2008)
20. Tyshkevich, R.I., Chernyak, A.A.: Decompositions of Graphs. *Cybernetics and System Analysis* 21, 231–242 (1985)
21. Tyshkevich, R.I., Chernyak, A.A.: Algorithms for the canonical decomposition of a graph and recognizing polarity. *Izvestia Akad. Nauk BSSR, Ser. Fiz. Mat. Navuk.* 6, 16–23 (1985) (in Russian)

Robustness of Minimum Cost Arborescences

Naoyuki Kamiyama*

Department of Information and System Engineering, Chuo University
kamiyama@ise.chuo-u.ac.jp

Abstract. In this paper, we study the minimum cost arborescence problem in a directed graph from the viewpoint of robustness of the optimal objective value. More precisely, we characterize an input graph in which the optimal objective value does not change even if we remove several arcs. Our characterizations lead to efficient algorithms for checking robustness of an input graph.

1 Introduction

Throughout this paper, we denote by $D = (V, A)$ a directed graph with a vertex set V and an arc set A . Moreover, let r be a special root vertex of V , and define $U := V \setminus \{r\}$. We assume that no arc of A enters r . An r -arborescence in D is a spanning tree in D (when viewed as an undirected graph) directed away from r . We assume that there is at least one r -arborescence in D , i.e., every vertex of V is reachable from r in D . The notion of arborescences is often used for modelling broadcast [1] and evacuation networks [2]. In this paper, we study the *minimum cost arborescence problem* (MCAP for short). In this problem, we are given a directed graph $D = (V, A)$ with a non-negative cost c_a on each arc a of A . The goal is to find a minimum cost r -arborescence $T = (V, B)$ in D , where the cost of T is defined by $\sum_{a \in B} c_a$. We can regard MCAP as a directed graph analogue of the *minimum spanning tree problem* in an undirected graph, and thus MCAP is one of fundamental combinatorial optimization problems in graphs. Polynomial-time algorithms for MCAP were presented by Chu and Liu [3], Edmonds [4] and Bock [5]. Later Fulkerson [6] gave a two-phase algorithm which solves MCAP very elegantly. Gabow, Galil, Spencer and Tarjan [7] gave the current fastest algorithm that runs in $O(m + n \log n)$ time, where n (resp., m) is the number of vertices (resp., arcs) of an input graph.

In this paper, we study MCAP from the viewpoint of *robustness* of its optimal objective value. More precisely, we characterize an input graph D in which the optimal objective value of MCAP does not change even if we remove several arcs of A . Since the notion of arborescences is used for modelling broadcast and evacuation networks, it is practically useful to characterize networks in which the situation does not get worse even if several links are damaged. Furthermore, there is a theoretical motivation. This is the similarity between MCAP and the

* Supported by a Grants-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

shortest path problem. It is known [8] that given specified vertices s, t of V and a length l_a on each arc a of A , deleting any arc will not affect the distance of a shortest path from s to t if and only if there are two arc-disjoint shortest paths from s to t . (In this paper, we indicate a directed one by using terms “path” or “cycle”.) The if-part is trivial, and the only if-part can be proved by using Menger’s theorem [9] in a subgraph consisting of all arcs contained in at least one shortest path from s to t . For the arborescence case, we have the following Edmonds’ theorem [10] corresponding to Menger’s theorem. For a subset X of U , we denote by $\varrho(X)$ the set of arcs a of A such that the head of a is in X , but the tail a is not in X . That is, $\varrho(X)$ is the set of arcs of A entering X .

Theorem 1 (Edmonds [10]). *There are k arc-disjoint r -arborescences in D if and only if $|\varrho(X)| \geq k$ for any non-empty subset X of U .*

Hence, it is theoretically interesting to reveal whether similar statements hold for the arborescence case.

Our goal. As mentioned above, the goal of this paper is to characterize an input graph D in which a trivial necessary or sufficient condition for the following statement is a necessary and sufficient condition.

- (A) The optimal objective value of MCAP does not change by removing any k arcs of A .

Which necessary or sufficient condition for (A) should we choose? From the computational viewpoint, we should choose a condition that can be efficiently checked. As a sufficient condition, we choose the following statement.

- (B) There are $k + 1$ arc-disjoint minimum cost r -arborescences in D .

It is clear that (B) is sufficient for (A) and we can efficiently check (B) by using a matroid intersection algorithm (see Section 3).

Next we consider a necessary condition for (A). Let $D_F^* = (V, A_F^*)$ be some subgraph of D obtained by modifying a graph obtained via Fulkerson’s algorithm [6] for MCAP (for the formal definition, see Section 2). It can be proved that D_F^* contains all arcs of A contained in at least one minimum cost r -arborescence in D . Thus, the following statement is clearly necessary for (A).

- (C) $|\varrho(X) \cap A_F^*| \geq k + 1$ for any non-empty subset X of U .

The relation between these statements is that (B) \Rightarrow (A) \Rightarrow (C). Unfortunately, there statements are not necessary and sufficient conditions in general. In this paper, we characterize an input graph D in which these necessary or sufficient conditions are necessary and sufficient conditions. Furthermore, by using these necessary and sufficient conditions, we give efficient algorithms for checking robustness of an input graph.

Outline. In Section 2, we explain Fulkerson’s algorithm [6] for MCAP. In the proofs of our main results, we will use information obtained from this algorithm. In Section 3, we characterize an input graph D in which (A) \Rightarrow (B). In Section 4, we characterize an input graph D in which (C) \Rightarrow (A).

Terminology. A subset X of V induces an arc a of A if the tail and head of a are in X . A subgraph H of D is *strongly connected* if there is a path from u to v in H for any vertices u, v of H . A non-empty subset X of V is a *strongly component in D* if the subgraph of D induced by X is strongly connected. For any vertex v of V , $\{v\}$ is a strongly component. A vertex v of U is an *r-cut vertex in D* if there is a vertex w of U such that $w \neq v$ and every path from r to w contains v . We say that w is *separated from r by v in D* . Let $n = |V|$ and $m = |A|$.

2 Fulkerson's Algorithm

In this section, we explain Fulkerson's algorithm [6] for MCAP. This algorithm consists of two phases, and we first explain its first phase. Let A_0 be the set of arcs a of A such that $c_a = 0$, and let U_0 be the set of vertices v of U such that $\varrho(\{v\}) \cap A_0 \neq \emptyset$. In this phase, we maintain a subset A_F of A , a collection \mathcal{F} of subsets of U and a non-negative real vector \mathbf{y} on subsets of U . An arc a of A is *tight for \mathbf{y}* if

$$\sum_{\emptyset \neq X \subseteq U : a \in \varrho(X)} y_X = c_a.$$

The first phase of Fulkerson's algorithm can be described as follows. Notice that in (2-1) there must be a subset S satisfying the condition. Otherwise, every vertex of V is reachable from r in a graph (V, A_F) .

Step 1. Set $A_F := A_0$, $\mathcal{F} := \{\{v\} \mid v \in U_0\}$ and $\mathbf{y} := \mathbf{0}$.

Step 2. Repeat the following until every vertex of V is reachable from r in a graph (V, A_F) .

(2-1) Find a subset of S of U such that S is a strongly component in a graph (V, A_F) and $\varrho(S) \cap A_F = \emptyset$.

(2-2) Increase y_S as much as possible until some arc of $A \setminus A_F$ gets tight.

(2-3) Add S to \mathcal{F} and all arcs that became tight in this iteration to A_F .

Step 3. Output A_F , \mathcal{F} and \mathbf{y} .

In the sequel, let A_F , \mathcal{F} and \mathbf{y} be those obtained when the first phase of Fulkerson's algorithm halts. Let D_F be a directed graph (V, A_F) . We call a subset X of U contained in \mathcal{F} a *class*.

Next we explain the second phase of Fulkerson's algorithm. For this, we introduce necessary definitions. It is well-known (for example, see [1]) that \mathcal{F} is laminar, i.e., $X \cap Y = \emptyset$, or $X \subseteq Y$, or $Y \subseteq X$ for any classes X, Y of \mathcal{F} . A class X of \mathcal{F} is a *child* of a class Y of \mathcal{F} if X is a subset of Y and there is no class Z of \mathcal{F} such that $X \subsetneq Z \subsetneq Y$. Since \mathcal{F} contains $\{v\}$ for any vertex v of U , there are children X_1, \dots, X_l of a non-singleton class Y of \mathcal{F} such that $\{X_1, \dots, X_l\}$ is a partition of Y . For a class X of \mathcal{F} , we denote by $D_F[X]$ the subgraph of D_F induced by X , and define $H_X = (V_X, A_X)$ by a graph obtained by contracting each child of X to a single vertex in $D_F[X]$. Obviously, H_X is strongly connected since X is a strongly component in D_F . Moreover, define $H_R = (V_R, A_R)$ by a graph obtained by contracting each maximal class of \mathcal{F} to a single vertex in D_F . We should note that $\{A_X \mid X \in \mathcal{F} \cup \{R\}\}$ is a partition of A_F .

Now we explain the second phase of Fulkerson's algorithm. We first find an arbitrary r -arborescence T_R in H_R . For each maximal class X of \mathcal{F} , T_R contains exactly one arc a_X of $\varrho(X)$. Moreover, a_X is contained in $\varrho(Y)$ for exactly one child Y of X . Then, we find an arborescence T_X in H_X rooted the vertex of H_X that is obtained by contracting Y . Since H_X is strongly connected, there is such an arborescence T_X . We repeat this operation until we reach minimal non-singleton classes of \mathcal{F} . Then, we can construct an r -arborescence T_F in D by combining all arborescences obtained in this operation.

Now we prove that T_F is a minimum cost r -arborescence in D . For this, we need the following LP relaxation of a natural IP formulation of MCAP. Let \mathbf{x} be a non-negative real vector on A .

$$(LP) \quad \begin{cases} \min & \sum_{a \in A} c_a x_a \\ \text{s.t.} & \sum_{a \in \varrho(X)} x_a \geq 1 \quad (\emptyset \neq X \subseteq U) \end{cases}$$

Obviously, the characteristic vector of T_F is feasible for (LP). The dual of (LP) can be described as follows. Let \mathbf{z} be a non-negative real vector on subsets of U .

$$(DP) \quad \begin{cases} \max & \sum_{\emptyset \neq X \subseteq U} z_X \\ \text{s.t.} & \sum_{\emptyset \neq X \subseteq U: a \in \varrho(X)} z_X \leq c_a \quad (a \in A) \end{cases}$$

By the definition of the first phase of Fulkerson's algorithm, \mathbf{y} is feasible for (DP). Now, by using the complementary slackness theorem, we prove that T_F and \mathbf{y} are optimal for (LP) and (DP), respectively. Recall that all subsets X of U such that $y_X > 0$ are contained in \mathcal{F} . Moreover, in the second phase, T_F included exactly one arc of $\varrho(X)$ for a class X of \mathcal{F} . Hence, the constraint of (LP) is satisfied by equality for a subset X of U such that $y_X > 0$. On the other hand, every arc of T_F is contained in A_F . Moreover, any arc of A_F is tight for \mathbf{y} . Thus, the constraint of (DP) is satisfied by equality for any arc of T_F . This complete the proof of the optimality of T_F and \mathbf{y} .

By using the optimality of \mathbf{y} , we can prove the following fact.

Lemma 1. *An r -arborescence $T = (V, B)$ in D is a minimum cost one if and only if B is a subset of A_F and $|\varrho(X) \cap B| = 1$ for any class X of \mathcal{F} .*

Proof. Since the if-part is trivial, we prove the only if-part. By the complementary slackness theorem, an arc of B must be tight for \mathbf{y} . Since we add all tight arcs to A_F during the first phase, B is a subset of A_F . Since a non-singleton subset X of U is contained in \mathcal{F} if and only if $y_X > 0$, the last half holds. \square

3 A Sufficient Condition is Necessary

In this section, we characterize D in which (A) \Rightarrow (B). We first prove that (A) does not imply (B) in general. A graph illustrated in Figure 11 is a counterexample

for the case of $k = 1$. The number attached with each arc represents its cost. The cost of an arc attached with no number is zero. Obviously, (A) is satisfied and the cost of a minimum cost r -arborescence is equal to 1. Since $|V| = 7$ and $|A| = 12$, if there are two arc-disjoint r -arborescences in D , all arcs of A are contained in these arborescences. This implies that there can not be two arc-disjoint minimum cost r -arborescences (i.e., r -arborescences with cost 1) since $\sum_{a \in A} c_a = 3$.

From now on, we try to characterize D in which (A) \Rightarrow (B). Before giving our main result, we consider two extreme cases. The first one is the case where D is acyclic, i.e., there is no cycle in D . In this case, it is not difficult to prove (A) \Rightarrow (B) by the following fact. If D is acyclic, a subgraph $T = (V, B)$ of D is an r -arborescence in D if and only if exactly one arc of B enters each vertex of U . The other one is the case where $c_a = 0$ for any arc a of A . In this case, we can prove (A) \Rightarrow (B) by Theorem 1. The following theorem contains these two extreme cases. Let A_+ be the set of arcs a of A such that $c_a > 0$.

Theorem 2. *If $c_a = 0$ for any arc a of A contained in at least one cycle and $|\varrho(S) \cap A_+| \leq k + 1$ for any non-singleton strongly component S in D , then the optimal objective value of MCAP does not change by removing any k arcs of A if and only if there are $k + 1$ arc-disjoint minimum cost r -arborescences in D .*

Proof. Since the if-part is trivial, we prove the other direction. Let \mathcal{S} be a collection of maximal strongly components in D that is not equal to $\{r\}$. Obviously, strongly components in \mathcal{S} are disjoint and \mathcal{S} is a partition of U . Since the cost of an arc contained in at least one cycle is equal to 0, the cost of an arc induced by a strongly component of \mathcal{S} is equal to 0. For a strongly component S of \mathcal{S} , let $c_S = \min_{a \in \varrho(S)} c_a$. Let \mathcal{S}_+ be the set of strongly components S of \mathcal{S} such that $c_S > 0$, and let M be the sum of c_S for all strongly components S of \mathcal{S}_+ . Since at least one arc enters each strongly component S of \mathcal{S}_+ , the cost of an r -arborescence in D is no less than M . For a strongly component S of \mathcal{S} , let A_S be the set of arcs a of $\varrho(S)$ such that $c_a = c_S$, and let $A_{\mathcal{S}}$ be the set of arcs of A that is induced by some strongly component of \mathcal{S} or is contained in A_S for some strongly component S of \mathcal{S} . It can be easily checked that in this case $A_{\mathcal{S}} = A_F$.

Since the optimal objective value of MCAP does not change by removing any k arcs of A , by Lemma 1 at least $k + 1$ arcs of A_S enters each non-empty subset of X of U . Thus, by Theorem 1, there are $k + 1$ arc-disjoint r -arborescences T_1, \dots, T_{k+1} in a graph (V, A_S) . What remains is to prove that T_1, \dots, T_{k+1} are minimum cost r -arborescences in D . For this, it suffices to prove that $|\varrho(S) \cap B_i| = 1$ for any strongly component S of \mathcal{S}_+ , where B_i is the arc set of T_i . This is because if this is true, the cost of each T_i is equal to M . If S is a singleton, this clearly follows from the definition of an arborescence. If S is not a singleton, since we assume that at most $k + 1$ arcs of A_+ enters any non-singleton strongly component S in D , each arborescence T_i contains exactly one arc of $\varrho(S)$. \square

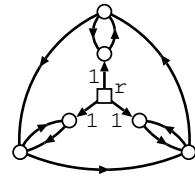


Fig. 1. A counterexample

In the example of Figure 2, U is a strongly component and $3 (= k + 2)$ arcs of A_+ enters this component. Hence, Theorem 2 is a complete characterization in this sense.

Time complexity. It is known [9] that we can find a minimum cost union of $k + 1$ arc-disjoint r -arborescences in $O(m^7)$ time by using Edmonds' matroid characterization [12] for arc-disjoint r -arborescences and a weighted matroid intersection algorithm (for example, see [13]). Thus, from Theorem 2, if $c_a = 0$ for any arc a of A contained in at least one cycle and $|\varrho(S) \cap A_+| \leq k + 1$ for any non-singleton strongly component S in D , then we can check whether or not the optimal objective value of MCAP changes by removing some k arcs of A in $O(m^7)$ time.

4 A Necessary Condition is Sufficient

In this section, we characterize D in which $(C) \Rightarrow (A)$. Recall that in Section 1, we said that A_F^* is obtained by modifying A_F . Of course, we can consider the statement (C) in which A_F^* is replaced by A_F . We call this statement $(C)'$. Here we show that $(C)'$ does not imply (A) even for the case of $k = 1$. A graph of Figure 2 is a counterexample. In this example, it can be easily checked that A_F contains all arcs of A and $(C)'$ is satisfied. However, the cost of a minimum cost r -arborescence in D is equal to 5, and if we remove the broken arc, the cost changes to 6.

Here we give the formal definition of a graph $D_F^* = (V, A_F^*)$. Recall the definitions used in the second phase of Fulkerson's algorithm in Section 2. Assume that there is an r -cut vertex v in H_R and an arc a entering v whose tail is separated from r by v in H_R . We call such an arc a *redundant*. It is not difficult to see from Lemma 1 that a redundant arc a can not be contained in a minimum cost r -arborescence in D since other arc of A_R having the same head of a must be contained. Let A_{red} be the set of all redundant arcs in H_R . Let X and Y be a maximal class of \mathcal{F} and a child of X , respectively. Let v_Y be the vertex of H_X that is obtained by contracting Y . A vertex v_Y is a *source* of H_X if $\varrho(Y) \cap (A_F \setminus A_{\text{red}})$ contains an arc that is not induced by X . Let \overline{H}_X be a graph obtained from H_X by adding a new vertex r_X and arcs from r_X to all sources of H_X . Assume that there is an r_X -cut vertex v in \overline{H}_X . As in the case of H_R , an arc entering v whose tail is separated from r_X by v can not be in a minimum cost r -arborescence in D . Thus, we add all redundant arcs in H_X to A_{red} . We repeat this operation until we reach minimal non-singleton classes of \mathcal{F} , and define $A_F^* = A_F \setminus A_{\text{red}}$. Notice that A_F^* contains all arcs of A contained in at least one minimum cost r -arborescence in D . For each $X \in \mathcal{F} \cup \{R\}$, we use the notation $H_X^* = (V_X, A_X^*)$ for representing the graph H_X after removing redundant arcs. Namely, H_X^* is obtained from D_F^* by contracting all children of

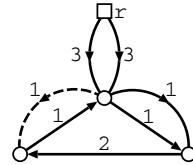


Fig. 2. A counterexample

X to a single vertex in a subgraph of D_F^* induced by X . Define sources of H_X^* as in H_X (r is a source of H_R^*). We can see that there is no redundant arc in A_F^* .

Unfortunately, even if we remove redundant arcs, (C) does not imply (A) in general. A graph of Figure 3 is a counterexample for the case $k = 2$. In this graph, A_F^* contains all arcs of A and (C) is satisfied. However, if we remove the broken arcs, the optimal objective value changes from 1 to 2. On the other hand, we can prove that $(C) \Rightarrow (A)$ in the case of $k = 1$. This is a main result of this section.

Before proving a main theorem, we give several facts that will be used in its proof. Recall that H_X is strongly connected. Hence, it is not difficult to see that every vertex of H_X^* is reachable from every source in H_X^* . (If a path P contain a redundant arc a , since the tail of a (say, t) is separated by the head of a (say, h), P first passes through h , and then t , and h again. That is, we can take a “shortcut”.) The following lemma plays an important role in the proof of a main theorem.

Lemma 2. *For $X \in \mathcal{F} \cup \{R\}$ and an arc a of H_X^* , there is an arborescence in H_X^* rooted at some source of H_X^* and containing a .*

Proof. Let h and t be the head and the tail of a , respectively. Recall that there is a path from every source of H_X^* to t . Since a is not redundant, i.e., t is not separated by h (from r_X in \bar{H}_X), there is a path P from some source s of H_X^* to t that does not contain h . Let P_a be a path obtained by adding a to P . If we can prove that there is an s -arborescence in H_X^* containing P_a , the proof is done. We can construct such an s -arborescence in the following “greedy” manner. Initially, we set $T = P_a$. We repeat the following until T becomes an s -arborescence in H_X^* . Pick a vertex v of H_X^* that is not contained in T . Since every vertex of H_X^* is reachable from s , there is a path Q from s to v in H_X^* . Let Q' be a subpath of Q after the last common vertex of T and Q (along Q). Then, even if we add Q' to T , the result graph is still a partial s -arborescence in H_X^* . By repeating this, we can obtain an s -arborescence in H_X^* containing P_a . \square

Now we are ready to give a main result of this section.

Theorem 3. *The optimal objective value of MCAP does not change by removing any arc of A if and only if $|\varrho(X) \cap A_F^*| \geq 2$ for any non-empty subset X of U .*

Proof. Since the only if-part is trivial, we prove the other direction. Let a be an arc of A . We will prove that even if we remove a from A , we can construct an r -arborescence $T = (V, B)$ in D_F^* such that exactly one arc of B enters any class of \mathcal{F} . By Lemma 1, such an r -arborescence T is a minimum cost one, and the proof is done. If a is not contained in A_F^* , this clearly holds. Thus, we assume that a is an arc of A_F^* .

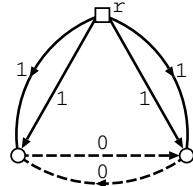


Fig. 3. A counterexample

We first assume that a is contained in H_R^* . Since at least two arcs of A_F^* enters any non-empty subset of U , at least two arcs of A_R^* enters any non-empty subset of $V_R \setminus \{r\}$. Thus, even if we remove a from A_R^* , there is an r -arborescence in H_R^* . Hence, by applying the second phase of Fulkerson's algorithm to all classes of \mathcal{F} and combining these arborescences, we can construct a desired r -arborescence in D .

Next we assume that a is contained in H_X^* for a class X of \mathcal{F} . In this case, if we can prove that even if we remove a from A_X^* , there is an arborescence T_X^* in H_X^* rooted at some source s of H_X^* , we can construct a desired r -arborescence in D as follows. Assume that X is a child of a class Z of \mathcal{F} . Let S be a subset of U such that s is obtained by contracting S . Since s is a source of H_X^* , there is an arc a of $\varrho(S) \cap A_F^*$ that is not induced by X . Let v_X be a vertex of H_Z^* that is obtained by contracting X . If a is also an arc of $\varrho(Z) \cap A_F^*$, v_X is a source of H_Z^* and we can construct a v_X -arborescence in H_Z^* . (Recall that every vertex is reachable from every source in H_Z^* .) If a is induced by Z , by Lemma 2 there is an arborescence in H_Z^* rooted at some source of H_Z^* containing a . We repeat this operation until we reach H_R^* . For other classes of \mathcal{F} , we can construct an arborescence like the second phase of Fulkerson's algorithm. By combining these arborescences, we can construct a desired r -arborescence in D .

What remains is to prove that there is an arborescence T_X^* in H_X^* rooted at some source of H_X^* even if we remove a from A_X^* . For this, we need to observe the structure of H_X^* . Let C be the set of r_X -cut vertices of \overline{H}_X . For a vertex v of C , we denote by I_v the set of vertices of V_X that are separated from r_X by v in \overline{H}_X . We first show that a collection $\mathcal{I} = \{I_v \mid v \in C\}$ is laminar. Assume that there are vertices v, w of C such that $I_v \cap I_w$, $I_v \setminus I_w$ and $I_w \setminus I_v$ are non-empty. Let x be a vertex of $I_v \cap I_w$. Without loss of generality, we can assume that there is a path P from r_X to x that lastly passes through v between v and w . Since $I_v \setminus I_w$ is not empty, there is a path Q from r_X to v that does not contain w . Thus, we can construct a path from r_X to x that does not contain w by combining a subpath of Q from r_X to v and a subpath of P from v to x . This contradicts the fact that x is a vertex of I_w .

Let \mathcal{I}_m be a collection of maximal members of \mathcal{I} (see Figure 4). Let I_0 be the set of vertices of V_X that is not separated from r_X by any r_X -cut vertex in \overline{H}_X .

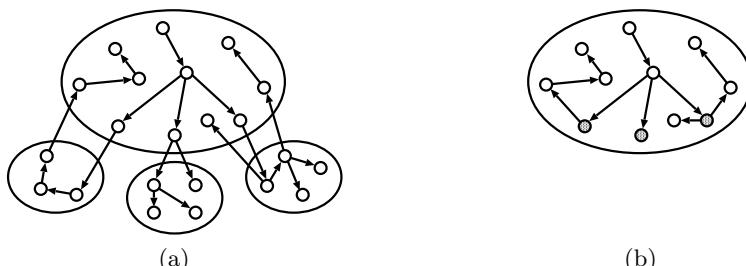


Fig. 4. A relation between arborescences in (a) H_X^* and (b) H_X^o . In (a), the biggest circle represents I_0 and other circles represent members of \mathcal{I}_m .

Notice that $\mathcal{I}_m \cup \{I_0\}$ partitions V_X , all sources of H_X^* are contained in I_0 , v is contained in I_0 for any member I_v of \mathcal{I}_m , and only arcs from v enters a member I_v of \mathcal{I}_m . Let H_X^o be the graph obtained from H_X^* by contracting $I_v \cup \{v\}$ to a single vertex for each member I_v of \mathcal{I}_m . It can be easily checked that H_X^o is strongly connected since H_X is strongly connected.

Assume that a is induced by $I_v \cup \{v\}$ for a member I_v of \mathcal{I}_m . Let G be the subgraph of H_X^* induced by $I_v \cup \{v\}$. Recall that only arcs from v enters I_v . Thus, since we assume that at least two arcs of A_F^* enters any non-empty subset of U , at least two arcs of A_X^* enters any non-empty subset of I_v in G . Thus, even if we remove a , there is a v -arborescence in G . Since H_X^o is strongly connected, we can construct an arborescence in H_X^o rooted at some source of H_X^o . Furthermore, there is an arborescence in a subgraph induced by $I_w \cup \{w\}$ for any member I_w of $\mathcal{I}_m \setminus \{I_v\}$. Therefore, since the head of an arc entering a vertex corresponding to $I_w \cup \{w\}$ in H_X^o is w in H_X^* for any member I_w of \mathcal{I}_m , we can construct a desired arborescence in H_X^* by combining these arborescences (see Figure 4).

Next we consider the case where a is not induced by $I_v \cup \{v\}$ for any member I_v of \mathcal{I}_m , i.e., a is contained in H_X^o . Let H_X^- be a graph obtained by removing a from H_X^o . If H_X^- is strongly connected, the proof is done. Assume that H_X^- is not strongly connected. Let \mathcal{S} be a collection of maximal strongly components in H_X^- . Notice that exactly one component S_0 of \mathcal{S} has no entering arc of H_X^- . Otherwise, we can not make H_X^- strongly connected by adding a single arc a . By the assumption that at least two arcs of A_F^* enters any non-empty subset of U , an arc of A_F^* that is not induced by X enters a subset of U corresponding to S_0 , i.e., S_0 contain a source s of H_X^* . Thus, since we can construct an arborescence rooted at a vertex corresponding to S_0 in a graph obtained from H_X^- by contracting each members of \mathcal{S} to a single vertex, we can construct an s -arborescence in H_X^- like the second phase of Fulkerson's algorithm. We can extending this arborescence to an arborescence in H_X^* by combining arborescences in subgraphs induced by $I_v \cup \{v\}$ for all members I_v of \mathcal{I}_m . This completes the proof. \square

Time complexity. Finally, we consider the time required for checking whether or not the optimal objective value of MCAP changes by removing some arc of A . If we naively use the fastest algorithm of [7] for MCAP, the time complexity is $O(m^2 + mn \log n)$. However, we can do this in $O(mn)$ time by using the characterization of Theorem 3 as follows. By Theorem 2, it suffices to check whether at least two arcs of A_F^* enters any non-empty subset of X of U . By Menger's theorem, this is equivalent to checking whether there are at least two arc-disjoint paths from r to any vertex of U in D_F^* . Since it is known [9] that we can check in $O(m)$ time whether there are two arc-disjoint path from r to some vertex of U , we check this in $O(mn)$ time for all vertices of U . Furthermore, we can compute A_F in $O(mn)$ time by Fulkerson's algorithm. Thus, what remains is to prove that we can compute A_{red} in $O(mn)$ time. Since it is well-known that $|\mathcal{F}| = O(n)$, we can construct H_X in $O(mn)$ time for all classes X of \mathcal{F} . Furthermore, we can compute all redundant arcs of H_X in $O(|A_X|n)$ time for each class X of \mathcal{F} as follows. We first check whether there are two arc-disjoint paths from r_X to any vertex of V_X in \overline{H}_X . Then, we transform H_X by replacing

a vertex by a new arc in a standard way. Pick a vertex v of H_X . If there are at least two arc-disjoint paths from r_X to v in the transformed graph, the operation is done. Assume that there is exactly one path P . Then, we reverse arcs of P . In this graph let W be the set of vertices reachable from r_X . An arc of P entering W corresponds to a cut vertex. Hence, arcs entering this vertex in H_X whose tail is not in W is redundant. Then, we repeat this operation by removing W . This can be finished in $O(|A_X|)$ for one vertex of H_X . We do this for all vertices of H_X , and the time complexity is $O(|A_X|n)$. Since $\sum_{X \in \mathcal{F}} |A_X| = O(m)$, we can compute A_{red} in $O(mn)$ time.

References

1. Li, Y., Thai, M.T., Wang, F., Du, D.Z.: On the construction of a strongly connected broadcast arborescence with bounded transmission delay. *IEEE Transactions on Mobile Computing* 5(10), 1460–1470 (2006)
2. Kamiyama, N., Katoh, N., Takizawa, A.: An efficient algorithm for the evacuation problem in a certain class of networks with uniform path-lengths. *Discrete Applied Mathematics* 157(17), 3665–3677 (2009)
3. Chu, Y., Liu, T.: On the shortest arborescence of a directed graph. *Scientia Sinica* 14, 1396–1400 (1965)
4. Edmonds, J.: Optimum branchings. *J. Res. Nat. Bur. Standards Sect. B* 71B, 233–240 (1967)
5. Bock, F.: An algorithm to construct a minimum directed spanning tree in a directed network. In: *Developments in Operations Research*, pp. 29–44. Gordon and Breach (1971)
6. Fulkerson, D.R.: Packing rooted directed cuts in a weighted directed graph. *Mathematical Programming* 6, 1–13 (1974)
7. Gabow, H.N., Galil, Z., Spencer, T., Tarjan, R.E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6, 109–122 (1986)
8. Gravin, N., Chen, N.: A note on k -shortest paths problem. *Journal of Graph Theory* 67(1), 34–37 (2011)
9. Schrijver, A.: *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Heidelberg (2003)
10. Edmonds, J.: Edge-disjoint branchings. In: *Combinatorial Algorithms*, pp. 91–96. Academic Press (1973)
11. Bang-Jensen, J., Gutin, G.Z.: *Digraphs: Theory, Algorithms and Applications*, 2nd edn. Springer, Heidelberg (2008)
12. Edmonds, J.: Submodular functions, matroids, and certain polyhedra. In: Guy, R., Hanani, H., Sauer, N., Schönheim, J. (eds.) *Combinatorial Structures and their Applications*, pp. 69–87. Gordon and Breach, New York (1970)
13. Frank, A.: A weighted matroid intersection algorithm. *J. Algorithms* 2(4), 328–336 (1981)

Path Queries in Weighted Trees*

Meng He¹, J. Ian Munro², and Gelin Zhou²

¹ Faculty of Computer Science, Dalhousie University, Canada
mhe@cs.dal.ca

² David R. Cheriton School of Computer Science, University of Waterloo, Canada
{imunro,g5zhou}@uwaterloo.ca

Abstract. We consider the problem of supporting several different path queries over a tree on n nodes, each having a weight drawn from a set of σ distinct values, where $\sigma \leq n$. One query we support is the path median query, which asks for the median weight on a path between two given nodes. For this and the more general path selection query, we present a linear space data structure that answers queries in $O(\lg \sigma)$ time under the word RAM model. This greatly improves previous results on the same problem, as previous data structures achieving $O(\lg n)$ query time use $O(n \lg^2 n)$ space, and previous linear space data structures require $O(n^\epsilon)$ time to answer a query for any positive constant ϵ [16]. Our linear space data structure also supports path counting queries in $O(\lg \sigma)$ time. This matches the result of Chazelle [8] when σ is close to n , but has better performance when σ is significantly smaller than n . Finally, the same data structure can also support path reporting queries in $O(\lg \sigma + occ \lg \sigma)$ time, where occ is the size of output. In addition, we present a data structure that answers path reporting queries in $O(\lg \sigma + occ \lg \lg \sigma)$ time, using $O(n \lg \lg \sigma)$ space. These are the first data structures that answer path reporting queries.

1 Introduction

Trees are fundamental structures in computer science, being widely used in modeling and representing different types of data in numerous computer applications. In many cases, properties of objects being modeled are stored as weights or labels on the nodes of trees. Thus researchers have studied the preprocessing of weighted trees in which each node is assigned a weight, in order to support different *path queries*, for which a certain function over the weights of the nodes along a given query path in the tree is computed [18,13,16].

In this paper, we design data structures to maintain a weighted tree T on n nodes such that we can support the following path queries:

- **Path Median Query:** Given two nodes u and v , return the median weight on the path from u to v . If there are m nodes on this path, then the median weight is the $\lceil m/2 \rceil$ th smallest one among the weights of these nodes.

* This work was supported by NSERC and the Canada Research Chairs Program.

- **Path Selection Query:** This kind of query is a natural extension of path median queries. We are given two nodes u, v and a positive integer k , and we need return the k th smallest weight on the path from u to v . We assume that k is not larger than the number of nodes on this path.
- **Path Counting Query:** Given two nodes u, v and a range $[s, t]$, return the number of nodes on the path from u to v whose weights are in this range.
- **Path Reporting Query:** Given two nodes u, v and a range $[s, t]$, return the set of nodes on the path from u to v that have a weight in this range.

When the given tree T is a path, the above queries become range median, range selection, two-dimensional range counting and range reporting queries. Thus the path queries we consider generalize these fundamental queries to weighted trees.

1.1 Previous Work

The path median problem was first presented by Krizanc et al. [16], who gave two solutions for this problem. The first one supports queries in $O(\lg n)$ time¹, and occupies $O(n \lg^2 n)$ words of space. The second one requires $O(b \lg^3 n / \lg b)$ query time and $O(n \lg_b n)$ space, for any $2 \leq b \leq n$. If we set b to be $n^\epsilon / \lg^2 n$ for some small constant ϵ , then the space cost becomes linear, but the query time is $O(n^\epsilon)$. These are the best known results for the path median problem.

The path counting problem was studied by Chazelle in [8]. In his formulation, weights are assigned to edges instead of nodes, and a query asks for the number of edges on a given path whose weights are in a given range. He designed a linear space data structure to support queries in $O(\lg n)$ time. His technique is based on tree partition, so it requires $O(\lg n)$ time for arbitrary set of weights.

Range median and selection queries. In the range median and selection problems, we are given an unsorted array of n elements, each having a weight. A query asks for the median or the k th smallest weight in a range.

For the static version, to obtain constant query time, the known solutions require near-quadratic space [16][19][20]. For linear space data structures, Gfeller and Sanders [11] presented a solution to answer a query in $O(\lg n)$ time. Gagie et al. [9] considered this problem in terms of σ , the number of distinct weights. They designed a data structure based on wavelet trees that supports range selection in $O(\lg \sigma)$ time. The best upper bound was achieved by Brodal et al. [45], which has $O(\lg n / \lg \lg n)$ query time. Later, Jørgensen and Larsen showed that Brodal et al.’s result is optimal by proving a lower bound of $\Omega(\lg n / \lg \lg n)$ on query time, providing that data structures for the static range selection problem use $O(n \lg^{O(1)} n)$ bits of space [15].

Orthogonal range counting and reporting queries. The space/time trade-off of 2-d orthogonal range counting is well studied. Pătrașcu [17][18] showed that any data structure using $O(n \lg^{O(1)} n)$ space requires $\Omega(\lg n / \lg \lg n)$ query time. In fact, it is possible to achieve this query time with linear space [7][14].

¹ In this paper we use $\lg n$ to denote $\log_2 n$.

For the 2-d range reporting problem, the best known results were given by Chan et al. [6]. Let occ denote the output size. Chan et al.’s first solution achieves $O(\lg \lg n + occ \lg \lg n)$ query time with $O(n \lg \lg n)$ space. Their second data structure supports queries in $O(\lg^\epsilon n + occ \lg^\epsilon n)$ time using linear space.

1.2 Our Results

Let σ be the number of distinct weights, clearly $\sigma \leq n$. Without loss of generality, we assume that weights are drawn from $[1..n]$. In the remainder of this paper, we analyze the time and space costs in terms of n and σ .

For path median and path selection queries, under the pointer machine model, our data structure requires $O(\lg \sigma)$ query time and $O(n \lg \sigma)$ words of space (Section 3). Under the word RAM model, the space cost can be reduced to $O(n)$, preserving the same query time (Section 4.1). The word RAM result significantly improves the best known result [16], in which the data structure achieving $O(\lg n)$ query time uses $O(n \lg^2 n)$ space, and the linear space data structure requires $O(n^\epsilon)$ query time for any positive constant ϵ .

For path counting queries, our data structure for the pointer machine model supports queries in $O(\lg \sigma)$ time, using $O(n \lg \sigma)$ words of space (Section 3). Our data structure for the word RAM model also supports queries in $O(\lg \sigma)$ time, using linear space only (Section 4.1). The best previous result for the path counting problem is due to Chazelle [8], which requires linear space and $O(\lg n)$ query time. The author showed that any path in a tree T can be partitioned into $O(\lg |T|)$ canonical paths, where $|T|$ is the number of nodes in T . Thus, even if the size of the set of weights is very small, Chazelle’s data structure still requires $O(\lg n)$ time to answer a query. Our word RAM result matches the result of Chazelle when σ is close to n , and improves it when σ is much smaller than n . In addition, our techniques are conceptually simple.

To the best of our knowledge, path reporting queries have never been studied before. Assuming that occ is the size of output, we give three solutions in this paper. The first one, under the pointer machine model, requires $O(n \lg \sigma)$ words of space and $O(\lg \sigma + occ)$ query time (Section 3). The second one, under the word RAM model, requires $O(n)$ words of space and $O(\lg \sigma + occ \lg \sigma)$ query time (Section 4.1). The last one, also under the word RAM model, requires $O(n \lg \lg \sigma)$ words of space but only $O(\lg \sigma + occ \lg \lg \sigma)$ query time (Section 4.2).

To achieve the above results, we generalize powerful techniques such as the wavelet trees [12] and the technique for the ball-inheritance problem [6]. Previously, these techniques were applied to arrays and 2-d point sets only. Our work is the first that successfully generalizes them to answer path queries, and we expect these to be useful for other similar queries over trees in the future.

2 Properties of Ordinal Trees

In this paper, we take T as an ordinal tree. That is, we choose a node to be the root, and define a left-to-right order among siblings in T . In this section we

prove several properties of ordinal trees that are useful in designing our data structures for path queries.

For any nodes u, v in T , let $P_{u,v}$ denote the set of nodes on the path from u to v . If u and v are the same node, then $P_{u,v}$ contains this node only. For any node u and its ancestor c , we define $A_{u,c}$ to be the set of nodes on the path from u to c , excluding the top node c . We assume a node is also its own ancestor. Thus, $A_{u,u}$ is a valid but empty set. It is clear that, for any nodes u, v in T , $P_{u,v}$ is the disjoint union of $A_{u,c}, A_{v,c}$ and $\{c\}$, where c is the lowest common ancestor (LCA) of u and v .

Let $[l, r]$ be a range, where $1 \leq l \leq r \leq \sigma$. Let $R_{l,r}$ denote the set of nodes in T that have a weight in $[l, r]$. For any node-weighted ordinal tree S and any node x in S , we define $d_{l,r}(S, x)$, or the $[l, r]$ -depth of x in S , to be the number of ancestors of x that have a weight in $[l, r]$. The $[1, \sigma]$ -depth of x is equivalent to the depth of x in the tree, so we define $d(S, x)$ to be $d_{1,\sigma}(S, x)$ for simplicity. Also, we define $anc_{l,r}(S, x)$ to be the nearest ancestor of x that has a weight in $[l, r]$. If x has no such ancestor, then $anc_{l,r}(S, x)$ is the root of S . An obvious fact is that $d_{l,r}(S, x) = d_{l,r}(S, anc_{l,r}(S, x))$.

Now consider how to compute the intersection of $R_{l,r}$ and $P_{u,v}$. Providing that c is the lowest common ancestor of u and v , we have

$$\begin{aligned} R_{l,r} \cap P_{u,v} &= R_{l,r} \cap (A_{u,c} \cup A_{v,c} \cup \{c\}) \\ &= (R_{l,r} \cap A_{u,c}) \cup (R_{l,r} \cap A_{v,c}) \cup (R_{l,r} \cap \{c\}); \end{aligned} \quad (1)$$

and its cardinality is

$$\begin{aligned} |R_{l,r} \cap P_{u,v}| &= |(R_{l,r} \cap A_{u,c}) \cup (R_{l,r} \cap A_{v,c}) \cup (R_{l,r} \cap \{c\})| \\ &= |R_{l,r} \cap A_{u,c}| + |R_{l,r} \cap A_{v,c}| + |R_{l,r} \cap \{c\}| \\ &= d_{l,r}(T, u) - d_{l,r}(T, c) + d_{l,r}(T, v) - d_{l,r}(T, c) + \mathbf{1}_{R_{l,r}}(c), \end{aligned} \quad (2)$$

where $\mathbf{1}_{R_{l,r}}(c)$ is equal to 1 if $c \in R_{l,r}$, or equal to 0 if not. In order to compute the cardinality efficiently, we need a fast way to compute $d_{l,r}(T, u)$'s. A naive solution is to store a $d_{l,r}$ value for each node in T . This method takes $O(n)$ space for any range $[l, r]$ so that the overall space cost will be $O(n\sigma)$. To save space, we prove several useful properties of ordinal trees.

We introduce the deletion operation of tree edit distance [3], which can be performed at any non-root node of an ordinal tree. Let u be a non-root node and v be its parent. To delete u , we insert its children in place of u into the list of children of v , keeping the original left-to-right order. For range $[l, r]$, we define $T_{l,r}$ to be the ordinal tree after deleting all the non-root nodes that are not in $R_{l,r}$ from T , where the nodes are deleted in level order. Note that $T_{l,r}$ contains $|R_{l,r}|$ or $|R_{l,r}| + 1$ nodes only, depending on whether the root of T is in $R_{l,r}$. The following lemmas play central roles in our data structures. Their proofs are omitted due to space constraints.

Lemma 1. *For any node u and its arbitrary ancestor c , and any range $[l, r]$, $d_{l,r}(T, u) - d_{l,r}(T, c) = d(T_{l,r}, anc_{l,r}(T, u)) - d(T_{l,r}, anc_{l,r}(T, c))$.*

Lemma 2. For any node u in T , and any two nested ranges $[l, r] \subseteq [l', r']$, $\text{anc}_{l,r}(T, u) = \text{anc}_{l,r}(T_{l',r'}, \text{anc}_{l',r'}(T, u))$.

Lemma 3. For any two ranges $[l_1, r_1]$ and $[l_2, r_2]$, the nodes in both T_{l_1,r_1} and T_{l_2,r_2} have the same relative positions in the pre-order traversal sequences of T_{l_1,r_1} and T_{l_2,r_2} .

3 A Pointer Machine Data Structure

In this section, we present our data structure for the pointer machine model. Our basic idea is to build a conceptual range tree on $[1, \sigma]$: Starting with $[1, \sigma]$, we keep splitting each range into two child ranges differ by at most 1 in length. Formally, providing that $l < r$, the range $[l, r]$ will be split into child ranges $[l_1, r_1]$ and $[l_2, r_2]$, where $l_1 = l$, $r_1 = \lfloor (l+r)/2 \rfloor$, $l_2 = r_1 + 1$ and $r_2 = r$. This procedure stops when $[1, \sigma]$ has been split into σ leaf ranges of length 1, which are located at the bottom level of the range tree. It is easy to see that each leaf node corresponds to a single value in $[1, \sigma]$.

For each range $[l, r]$ in the range tree, we construct and store $T_{l,r}$ explicitly. On each node in $T_{l,r}$ we store its depth, and a pointer linking to the corresponding node in T . For each non-leaf range $[l, r]$, let $[l_1, r_1]$ and $[l_2, r_2]$ be the child ranges of $[l, r]$. We pre-compute and store $\text{anc}_{l_1,r_1}(T_{l,r}, u)$ and $\text{anc}_{l_2,r_2}(T_{l,r}, u)$ for each node u in $T_{l,r}$.

We now show how to support path queries using the above data structure.

Lemma 4. The data structure in this section supports path counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + \text{occ})$ time, where occ denotes the output size of the query.

Proof. Let u and v be the endpoints of the query path, and let $[s, t]$ be the query range. In the path counting and reporting problems, our objective is to compute $R_{s,t} \cap P_{u,v}$ and its cardinality. We first consider how to compute the cardinality for path counting queries. By the property of range trees, each query range $[s, t] \subseteq [1, \sigma]$ can be represented by the union of m disjoint ranges in the range tree, say $[l_1, r_1], \dots, [l_m, r_m]$, where $m = O(\lg \sigma)$. Because $R_{s,t} \cap P_{u,v} = \bigcup_{1 \leq i \leq m} (R_{l_i,r_i} \cap P_{u,v})$, we need only compute the cardinality of $|R_{l_i,r_i} \cap P_{u,v}|$ efficiently, for $1 \leq i \leq m$. As shown in Algorithm II, our algorithm accesses the range tree from top to bottom, ending at the ranges completely included in $[s, t]$. When visiting range $[l, r]$, $\text{anc}_{l,r}(T, \delta)$ is computed for $\delta \in \{u, v, c\}$. Thus, line 12 computes $|R_{l,r} \cap P_{u,v}|$ in $O(1)$ time when $[l, r] \subseteq [s, t]$. In line 16, the algorithm iteratively computes the nearest ancestors of u, v, c for child ranges. Finally, in line 20, the algorithm recurses on child ranges that intersect with $[s, t]$.

For path reporting queries, as shown in lines 8 to 11, our algorithm traverses from x to y , reporting all the nodes on this path except z . Note that for each node being reported, we report its corresponding node in T by following the pointer saved in it. Finally, we report node c if its weight is in this range.

Algorithm 1. The algorithm for path counting and reporting queries.

```

1: procedure QUERY( $u, v, [s, t]$ ) ▷  $[s, t] \subseteq [1, \sigma]$ .
2:    $c \leftarrow LCA(u, v);$ 
3:   return SEARCH( $[1, \sigma], u, v, c, c, [s, t]$ );
4: end procedure
5: procedure SEARCH( $[l, r], x, y, z, c, [s, t]$ )
6:    $\triangleright x = anc_{l,r}(T, u), y = anc_{l,r}(T, v) \text{ and } z = anc_{l,r}(T, c).$ 
7:   if  $[l, r] \subseteq [s, t]$  then
8:     if the given query is a path reporting query then
9:       report all nodes on the path from  $x$  to  $y$  except  $z$ ;
10:      report  $c$  if its weight is in  $[l, r]$ ;
11:    end if
12:    return  $d(T_{l,r}, x) + d(T_{l,r}, y) - 2d(T_{l,r}, z) + \mathbf{1}_{R_{l,r}}(c);$ 
13:    ▷  $|R_{l,r} \cap P_{u,v}|$ , by Lemma 1 and Equation 2
14:  end if
15:  Let  $[l_1, r_1]$  and  $[l_2, r_2]$  be the child ranges of  $[l, r]$ ;
16:   $\delta_i \leftarrow anc_{l_i, r_i}(T_{l,r}, \delta)$  for  $\delta = \{x, y, z\}$  and  $i = 1, 2$ ; ▷ By Lemma 2
17:   $count \leftarrow 0;$ 
18:  for  $i \leftarrow 1, 2$  do
19:    if  $[l_i, r_i] \cap [s, t] \neq \emptyset$  then
20:       $count \leftarrow count + \text{SEARCH}([l_i, r_i], x_i, y_i, z_i, c, [s, t]);$ 
21:    end if
22:  end for
23:  return  $count;$ 
24: end procedure

```

Now we analyze the time cost of Algorithm 1. First of all, the LCA queries can be supported in constant time and linear space (for a simple implementation, see 2). We thus need only consider our range tree. For range counting queries, Algorithm 1 accesses $O(\lg \sigma)$ ranges, and it spends constant time on each range. For range reporting queries, Algorithm 1 uses $O(1)$ additional time to report each occurrence. Hence, the query time of range counting is $O(\lg \sigma)$, and the query time of range reporting is $O(\lg \sigma + occ)$, where occ is the output size. □

Lemma 5. *The data structure in this section supports path median and selection queries in $O(\lg \sigma)$ time.*

Proof. Let u and v be the nodes given in the query. Our algorithm for the path median and selection problems is shown in Algorithm 2. The algorithm traverses the range tree from top to bottom, ending at some range of length 1. The procedure SELECT computes the p th smallest weight in $R_{l,r} \cap P_{u,v}$, providing that $1 \leq p \leq |R_{l,r} \cap P_{u,v}|$. If $l = r$, this weight must be l . Otherwise, let $[l_1, r_1]$ and $[l_2, r_2]$ be the child ranges of $[l, r]$, where $r_1 < r_2$. The algorithm computes $count = |R_{l_1, r_1} \cap P_{u,v}|$ and compares it with p in line 15. If p is not larger than $count$, then the algorithm recurses on $[l_1, r_1]$ in line 18; otherwise the algorithm deducts $count$ from p and recurses on $[l_2, r_2]$ in line 20. The time analysis is similar to Lemma 4. □

Algorithm 2. The algorithm for path median and selection queries.

```

1: procedure QUERY( $u, v$  or  $u, v, k$ ) ▷  $1 \leq k \leq |P_{u,v}|$ .
2:   if the given query is a path median query then
3:      $k \leftarrow \lceil |P_{u,v}|/2 \rceil$ ;
4:   end if
5:    $c \leftarrow LCA(u, v)$ ;
6:   return SELECT([1,  $\sigma$ ],  $u, v, c, c, k$ );
7: end procedure
8: procedure SELECT([ $l, r$ ],  $x, y, z, c, p$ )
9:   ▷  $x = anc_{l,r}(T, u), y = anc_{l,r}(T, v)$  and  $z = anc_{l,r}(T, c)$ .
10:  if  $l = r$  then
11:    return  $l$ ;
12:  end if
13:  Let  $[l_1, r_1]$  and  $[l_2, r_2]$  be the child ranges of  $[l, r]$ , where  $r_1 < l_2$ ;
14:   $\delta_i \leftarrow anc_{l_i, r_i}(T_{l,r}, \delta)$  for  $\delta = \{x, y, z\}$  and  $i = 1, 2$ ; ▷ By Lemma 2
15:   $count \leftarrow d(T_{l_1, r_1}, x_1) + d(T_{l_1, r_1}, y_1) - 2d(T_{l_1, r_1}, z_1) + \mathbf{1}_{R_{l_1, r_1}}(c)$ ;
16:   ▷  $|R_{l_1, r_1} \cap P_{u,v}|$ , by Lemma 1 and Equation 2
17:  if  $p \leq count$  then
18:    return SELECT([ $l_1, r_1$ ],  $x_1, y_1, z_1, c, p$ );
19:  else
20:    return SELECT([ $l_2, r_2$ ],  $x_2, y_2, z_2, c, p - count$ );
21:  end if
22: end procedure

```

With Lemmas 4 and 5, we can present our result on supporting path queries under the pointer machine model.

Theorem 1. *Under the pointer machine model, a tree on n weighted nodes can be represented in $O(n \lg \sigma)$ words of space to support path median, selection and counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + occ)$ time, where the weights are drawn from $[1, \sigma]$, and occ is the output size of the path reporting query.*

Proof. The claim of query time follows from Lemma 4 and Lemma 5. It suffices to analyze the space cost of our data structure. For a range $[l, r]$ in the range tree, our data structure uses $O(|R_{l,r}|)$ words of space to store $T_{l,r}$, depths of nodes, and the pointers to T and child ranges. Thus the adjunct data structures constructed for each level of the range tree occupy $O(n)$ words in total, and the overall space cost of our data structure is $O(n \lg \sigma)$. \square

4 Optimizations in the RAM model

In this section we show how to reduce the space cost of the data structure presented in Section 3. We adopt the word RAM model of computation with word size $w = \Omega(\lg n)$. For the path median, selection and counting problems, we achieve $O(\lg \sigma)$ query time with linear space. For the path reporting problem, we either require $O(\lg \sigma + occ \lg \sigma)$ query time with linear space, or $O(\lg \sigma + occ \lg \lg \sigma)$ query time with $O(n \lg \lg \sigma)$ space.

4.1 Linear Space, but Slower Reporting

Our starting point for space optimization is the succinct representation of ordinal trees. As shown in Lemma 6, there exists a data structure that encodes a tree on n labeled nodes in $O(n)$ bits of space when the number of distinct labels is constant, and supports a set of basic operations in constant time.

Lemma 6 ([10]). *Let S be an ordinal tree on n nodes, each having a label from an alphabet Σ . S can be represented in $n(\lg |\Sigma| + 2) + O(|\Sigma|n \lg \lg \lg n / \lg \lg n)$ bits to support the following queries in constant time. For simplicity, we assume that S contains node x , and x precedes itself in pre-order.*

- PRE-RANK(S, x): Return the number of nodes that precede x in pre-order;
- PRE-RANK(S, x, c): Return the number of nodes that are labeled with c and precede x in pre-order;
- PRE-SELECT(S, i): Return the i th node in pre-order;
- PRE-SELECT(S, i, c): Return the i th node in pre-order that is labeled with c ;
- DEPTH(S, x): Return the depth of x ;
- ANCESTOR(S, x, c): Return the lowest ancestor of node x labeled with c .

We now present our linear space data structure.

Theorem 2. *Under the word RAM model with word size $w = \Omega(\lg n)$, a tree on n weighted nodes can be represented in $O(n)$ words of space to support path median, selection and counting queries in $O(\lg \sigma)$ time, and path reporting queries in $O(\lg \sigma + occ \lg \sigma)$ time, where the weights are drawn from $[1, \sigma]$, and occ is the output size of the path reporting query.*

Proof. For our new data structure, we still build a conceptual range tree as in Section 3. Unlike the previous data structure, we store only the succinct representation of $T_{l,r}$ for each range $[l, r]$ in the range tree. We assign a label to each node in $T_{l,r}$ if range $[l, r]$ is not a leaf range. Let $[l_1, r_1]$ and $[l_2, r_2]$ be child ranges of $[l, r]$, where $r_1 < l_2$. We assign label 0 to the root node of $T_{l,r}$. For each non-root node u in $T_{l,r}$, we assign it label i if $u \in T_{l_i, r_i}$ for $i = 1, 2$. By Lemma 6, the succinct representation for range $[l, r]$ occupies $O(|R_{l,r}|)$ bits. Thus, the space cost of our new data structure is $O(n \lg \sigma / w) = O(n)$ words.

Now consider how to answer path queries. Note that the nodes in the succinct representation are indexed by their ranks in pre-order. Let $[l, r]$ be a non-leaf range, and let $[l_1, r_1]$ and $[l_2, r_2]$ be the child ranges of $[l, r]$, where $r_1 < l_2$. Suppose node u is in $T_{l,r}$, and node u' is the node in T_{l_i, r_i} that corresponds to u for $i = 1$ or 2 . We show that, once the pre-order rank of u in $T_{l,r}$ is known, the pre-order of u' in T_{l_i, r_i} can be computed in constant time, and vice versa. By the construction of our linear space data structure, T_{l_i, r_i} contains all the nodes in $T_{l,r}$ that have label 0 or i . By Lemma 3, these nodes have the same relative positions in the pre-order sequences of T_{l_i, r_i} and $T_{l,r}$. We thus have that

$$\text{PRE-RANK}(T_{l,r}, u, 0) + \text{PRE-RANK}(T_{l,r}, u, i) = \text{PRE-RANK}(T_{l_i, r_i}, u'). \quad (3)$$

Applying this formula, it takes constant time to convert the pre-order rank of the same node between two adjacent levels in the range tree.

Since the DEPTH operation is provided, we need only consider how to compute the nearest ancestors of u, v, c for child ranges (Line 16 in Algorithm 1 and line 14 in Algorithm 2). For $\delta = \{x, y, z\}$ and $i = 1, 2$, we can compute $anc_{l_i, r_i}(T_{l,r}, \delta)$ by ANCESTOR($T_{l,r}, \delta, i$). If δ has no ancestor with label i in $T_{l,r}$, then $anc_{l_i, r_i}(T_{l,r}, \delta)$ is the root node of $T_{l,r}$.

It is more complicated to deal with path reporting queries. Unlike the data structure described in Section 3, given a node u in some $T_{l,r}$ that need be reported, we cannot directly find its corresponding node in T by following an appropriate pointer, as we cannot afford storing these pointers. Instead, we compute the pre-order rank of u in the tree constructed for the parent range of $[l, r]$ using Equation 3, and repeat this process until we reach the root range in the range tree. This procedure takes $O(\lg \sigma)$ time for each node to report.

To analyze the query time, we observe that, for path median, selection and counting queries, our linear space data structure still uses constant time on each visited range. Hence, these queries can be answered in $O(\lg \sigma)$ time. For path reporting queries, this data structure requires $O(\lg \sigma)$ additional time for each node to report. Thus, path reporting queries can be answered in $O(\lg \sigma + occ \lg \sigma)$ time, where occ is the output size. \square

4.2 Slightly More Space, Much Faster Reporting

As shown above, the bottleneck of path reporting queries in our linear space data structure is that it requires $O(\lg \sigma)$ time to find the corresponding node in T for each node in the answer. We apply the technique in [6] to speed up this process. Due to space limitations, we only provide a sketch of the proof.

Theorem 3. *Under the word RAM model with word size $w = \Omega(\lg n)$, a tree on n weighted nodes can be represented in $O(n \lg \lg \sigma)$ words of space to support path reporting queries in $O(\lg \sigma + occ \lg \lg \sigma)$ time, where the weights are drawn from $[1, \sigma]$, and occ is the output size of the path reporting query.*

Proof (Sketch). We maintain a second index for the nodes in T . For each node u in T , we index u by its weight b and its pre-order rank in $T_{b,b}$. Suppose the algorithm decides to report u when accessing range $[l, r]$. We need only find the leaf range corresponding to the weight and the pre-order rank of u in the tree constructed for this leaf range.

Consider the ranges at levels i and $i + \Delta$, where $1 \leq i < i + \Delta \leq \lg \sigma + 1$. Using the space efficient array in [6], which occupies $O(n\Delta)$ bits of space, we can compute in constant time the pre-order rank of each node u at level $i + \Delta$, providing that its pre-order rank at level i is known. For positive integer i , we define $f(i)$ to be the position in the binary representation of i such that the $(f(i) + 1)$ th least significant bit is the rightmost 1. For level $1 \leq i \leq \lg \sigma$, using the above method, we maintain an array with $\Delta = 2^{f(\lg \sigma + 1 - i)}$. Given a node u to report, we access these arrays to compute the pre-order rank of u at the leaf range. Since each jumping between two levels increases $f(\lg \sigma + 1 - i)$ by 1, this procedure takes $O(\lg \lg \sigma)$ time. On the other hand, the space cost of these arrays is at most $\frac{1}{w} \sum_{1 \leq i \leq \lg \sigma} O(n \cdot 2^{f(\lg \sigma + 1 - i)}) = O(n \lg \lg \sigma)$ words. \square

References

1. Alon, N., Schieber, B.: Optimal preprocessing for answering on-line product querie. Tech. rep., Tel Aviv University (1987)
2. 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)
3. Bille, P.: A survey on tree edit distance and related problems. *Theor. Comput. Sci.* 337(1-3), 217–239 (2005)
4. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. *Theor. Comput. Sci.* 412(24), 2588–2601 (2011)
5. Brodal, G.S., Jørgensen, A.G.: Data Structures for Range Median Queries. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 822–831. Springer, Heidelberg (2009)
6. Chan, T.M., Larsen, K.G., Pătraşcu, M.: Orthogonal range searching on the RAM, revisited. In: Symposium on Computational Geometry, pp. 1–10 (2011)
7. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: SODA, pp. 161–173 (2010)
8. Chazelle, B.: Computing on a free tree via complexity-preserving mappings. *Algorithmica* 2, 337–361 (1987)
9. Gagie, T., Puglisi, S.J., Turpin, A.: Range Quantile Queries: Another Virtue of Wavelet Trees. In: Karlgren, J., Tarhio, J., Hyryrø, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 1–6. Springer, Heidelberg (2009)
10. Geary, R.F., Raman, R., Raman, V.: Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms* 2(4), 510–534 (2006)
11. Gfeller, B., Sanders, P.: Towards Optimal Range Medians. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 475–486. Springer, Heidelberg (2009)
12. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: SODA, pp. 841–850 (2003)
13. Hagerup, T.: Parallel preprocessing for path queries without concurrent reading. *Inf. Comput.* 158(1), 18–28 (2000)
14. JáJá, J., Mortensen, C.W., Shi, Q.: Space-Efficient and Fast Algorithms for Multidimensional Dominance Reporting and Counting. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 558–568. Springer, Heidelberg (2004)
15. Jørgensen, A.G., Larsen, K.G.: Range selection and median: Tight cell probe lower bounds and adaptive data structures. In: SODA, pp. 805–813 (2011)
16. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. *Nord. J. Comput.* 12(1), 1–17 (2005)
17. Pătraşcu, M.: Lower bounds for 2-dimensional range counting. In: STOC, pp. 40–46 (2007)
18. Pătraşcu, M.: (Data) Structures. In: FOCS, pp. 434–443 (2008)
19. Petersen, H.: Improved Bounds for Range Mode and Range Median Queries. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 418–423. Springer, Heidelberg (2008)
20. Petersen, H., Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space. *Inf. Process. Lett.* 109(4), 225–228 (2009)

Dynamic Range Majority Data Structures*

Amr Elmasry¹, Meng He², J. Ian Munro³, and Patrick K. Nicholson³

¹ Department of Computer Science, University of Copenhagen, Denmark

² Faculty of Computer Science, Dalhousie University, Canada

³ David R. Cheriton School of Computer Science, University of Waterloo, Canada
elmasry@diku.dk, mhe@cs.dal.ca, {imunro,p3nichol}@uwaterloo.ca

Abstract. Given a set P of n coloured points on the real line, we study the problem of answering range α -majority (or “heavy hitter”) queries on P . More specifically, for a query range Q , we want to return each colour that is assigned to more than an α -fraction of the points contained in Q . We present a new data structure for answering range α -majority queries on a dynamic set of points, where $\alpha \in (0, 1)$. Our data structure uses $O(n)$ space, supports queries in $O((\lg n)/\alpha)$ time, and updates in $O((\lg n)/\alpha)$ amortized time. If the coordinates of the points are integers, then the query time can be improved to $O(\lg n/(\alpha \lg \lg n))$. For constant values of α , this improved query time matches an existing lower bound, for any data structure with polylogarithmic update time. We also generalize our data structure to handle sets of points in d -dimensions, for $d \geq 2$, as well as dynamic arrays, in which each entry is a colour.

1 Introduction

Many problems in computational geometry deal with point sets that have information encoded as colours assigned to the points. In this paper, we design dynamic data structures for the *range α -majority problem*, in which we want to report colours that appear *frequently* within an axis-aligned query rectangle. This problem is useful in database applications in which we would like to know typical attributes of the data points in a query range [14][15]. For the one-dimensional case, where the points represent time stamps, this problem has data mining applications for network traffic logs, similar to those of coloured range counting [10].

Formally, we are given a set, P , of n points, where each point $p \in P$ is assigned a colour c from a set, C , of colours. Let $\text{col}(p) = c$ denote the colour of p . We are also given a fixed parameter $\alpha \in (0, 1)$, that defines the threshold for determining whether a colour is to be considered frequent. Our goal is to design a *dynamic range α -majority data structure* that can perform the following operations:

- **QUERY(Q):** We are given an axis-aligned hyper-rectangle Q as a query. Let $P(Q)$ be the set $\{p \mid p \in Q, p \in P\}$, and $P(Q, c)$ be the set $\{p \mid p \in P(Q), \text{col}(p) = c\}$. The answer to the query Q is the set of colours M

* This work was supported by NSERC and the Canada Research Chairs Program.

such that for each colour $c \in M$, $|P(Q, c)| > \alpha|P(Q)|$, and for all $c \notin M$, $|P(Q, c)| \leq \alpha|P(Q)|$. We refer to a colour $c \in M$ as an α -majority for Q , and the query Q as an α -majority query. When $\alpha = \frac{1}{2}$, the problem is to identify the majority colour in Q , if such a colour exists.

- $\text{INSERT}(p, c)$: Insert a point p with colour c into P .
- $\text{DELETE}(p)$: Remove the point p from P .

1.1 Previous Work

Static and Dynamic α -Majority: In all of the following results, the threshold $\alpha \in (0, 1)$ is fixed at construction time, rather than given for each query.

Karpinski and Nekrich [14] studied the problem of answering range α -majority queries, which they call *coloured α -domination* queries. In the static case, they gave an $O(n/\alpha)$ space data structure that supports one-dimensional queries in $O((\lg n \lg \lg n)/\alpha)$ time, and an $O((n \lg \lg n)/\alpha)$ space data structure that supports queries in $O((\lg n)/\alpha)$ time. In the dynamic case, they gave an $O(n/\alpha)$ space data structure for one-dimensional queries which supports queries and insertions in $O((\lg^2 n)/\alpha)$ time, and deletions in $O((\lg^2 n)/\alpha)$ amortized time. They also gave an alternative $O((n \lg n)/\alpha)$ space data structure which supports queries and insertions in $O((\lg n)/\alpha)$ time, and deletions in $O((\lg n)/\alpha)$ amortized time. For points in d -dimensions, they gave a static $O((n \log^{d-1} n)/\alpha)$ space data structure that supports queries in $O((\log^d n)/\alpha)$ time, as well as a dynamic $O((n \log^{d-1} n)/\alpha)$ space data structure that supports queries and insertions in $O((\log^{d+1} n)/\alpha)$ time, and deletions in $O((\log^{d+1} n)/\alpha)$ amortized time.

Recently, Durocher et al. [8] described a static $O(n(\lg(1/\alpha) + 1))$ space data structure that answers range α -majority queries in an array in $O(1/\alpha)$ time.

Lai et al. [15] studied the dynamic problem, using the term *heavy-hitters* instead of α -majorities. They presented a dynamic data structure based on sketching, which provides an approximate solution with probabilistic guarantees for constant values of α . For one-dimension their data structure uses $O(hn)$ space and supports queries and updates in $O(h \log n)$ time, where the parameter $h = O(\frac{\lg m}{\varepsilon} \lg(\frac{\lg m}{\alpha\delta}))$ depends on the threshold α , the approximation factor ε , the total number of colours m , and the probability of failure δ . They also note that the space can be reduced to $O(n)$, at the cost of increasing the query time to $O(h \lg n + h^2)$. Thus, for constant values of ε , δ , and α , their data structure uses $O(n)$ space and has $O((\lg n \lg \lg n)^2)$ query and update time in the worst case when $\lg m = \Omega(\lg n)$. Another approximate data structure based on sketching was proposed by Wei and Yi [17]. Their data structure uses linear space, answers queries in $O(\log n + 1/\varepsilon)$ time, and may return false positives with relative frequency between $\alpha - \varepsilon$ and α . The cost of updates is $O(\mu \log n \log(1/\varepsilon))$, where μ is the cost of updating the sketches.

Finally, a cell-probe lower bound of Husfeldt and Rauhe [12, Prop. 11] implies that $t_q = \Omega(\lg n / (\lg(t_u b \lg n)))$, for any dynamic range α -majority data structure with query time t_q and update time t_u , when α is a constant and our machine has cell size b .

Other Related Work: Several other results exist for finding α -majorities in the streaming model [16,6,13]. De Berg and Haverkort [5] studied the problem of reporting τ -significant colours. For this problem, the goal is to output all colours c such that at least a τ -fraction of *all* of the points with colour c lie in the query rectangle.

There are other data structure problems that also deal with coloured points. In *coloured range reporting problems*, we are interested in reporting the set of distinct colours assigned to the points contained in an axis-aligned rectangle. Similarly, in *the coloured range counting problem* we are interested in returning the number of such distinct colours. Gupta et al. [11], Bozanis et al. [4], and, more recently, Gagie and Kärkkäinen [10] studied these problems and presented several interesting results.

1.2 Our Results

In this paper we present new data structures for the dynamic range α -majority problem in the word-RAM model with word size $\Omega(\lg n)$, where n is the number of points in the set P , and $\alpha \in (0, 1)$. Our results are summarized and compared to the previous best results in Table 1. The *input* column indicates the type of data we are considering. We use *points* to denote a set of points on a line with real-valued coordinates that we can compare in constant time, *integers* to denote a set of points on a line with word sized integer coordinates, and *array* to denote that the input is to be considered a dynamic array, where the positions of the points are in rank space.

Table 1. Comparison of the results in this paper to the previous best results. For the entries marked with “*” the running times are amortized.

Source	Input	Space	Query	Insert	Delete
[14, Thm. 3]	points	$O\left(\frac{n}{\alpha}\right)$	$O\left(\frac{\lg^2 n}{\alpha}\right)$	$O\left(\frac{\lg^2 n}{\alpha}\right)$	$O\left(\frac{\lg^2 n}{\alpha}\right)*$
[14, Thm. 3]	points	$O\left(\frac{n \lg n}{\alpha}\right)$	$O\left(\frac{\lg n}{\alpha}\right)$	$O\left(\frac{\lg n}{\alpha}\right)$	$O\left(\frac{\lg n}{\alpha}\right)*$
New	points	$O(n)$	$O\left(\frac{\lg n}{\alpha}\right)$	$O\left(\frac{\lg n}{\alpha}\right)*$	$O\left(\frac{\lg n}{\alpha}\right)*$
New	integers	$O(n)$	$O\left(\frac{\lg n}{\alpha \lg \lg n}\right)$	$O\left(\frac{\lg n}{\alpha}\right)*$	$O\left(\frac{\lg n}{\alpha}\right)*$
New	array	$O(n)$	$O\left(\frac{\lg n}{\alpha \lg \lg n}\right)$	$O\left(\frac{\lg^3 n}{\alpha \lg \lg n}\right)*$	$O\left(\frac{\lg n}{\alpha}\right)*$

Our results improve upon previous results in several ways. Most noticeably, all of our data structures require linear space. In order to provide fast query and update times for our linear space structures, we prove several interesting properties of α -majority colours. We note that the lower bound from Section 1.1 implies that, for constant values of α , an $O(\lg n / \lg \lg n)$ query time for integer point sets is optimal for any data structure with polylogarithmic update time, when the word size is $b = \Theta(\lg n)$. Our data structure for points on a line with integer coordinates achieves this optimal query time. Our data structures can also be generalized to handle d -dimensional points, improving upon previous results [14].

Road Map. In Section 2 we present a dynamic range α -majority data structure for points in one dimension. In Section 3 we show how to speed up the query time of our data structure in the case where the points have integer coordinates. Finally, in Section 4 we generalize our one dimensional data structures to higher dimensions. We defer the details of our dynamic array data structure to the full version of this paper.

Assumptions About Colours. In the following sections, we assume that we can compare colours in constant time. In order to support a dynamic set of colours, we employ the techniques described by Gupta et al. [11]. These techniques allow us to maintain a mapping from the set of colours to integers in the range $[1, 2n]$, where n is the number of points *currently* in our data structure. This allows us to index into an array using a colour in constant time.

For the dynamic problems discussed, this mapping is maintained using a method similar to global rebuilding to ensure that the integer identifiers of the colours do not grow too large [11, Section 2.3]. When a coloured point is inserted, we must first determine whether we have already assigned an integer to that colour. By storing the set of known colours in a balanced binary search tree, this can be checked in $O(\lg n_c)$ time, where n_c is the number of distinct colours currently assigned to points in our data structure. Therefore, from this point on, we assume that we are dealing with integers in the range $[1, 2n]$ when we discuss colours.

2 Dynamic Data Structures in One Dimension

In one-dimension we can interchange the idea of points and x -coordinates in P , since they are equivalent. Depending on the context we may use either term. Our data structure is a modification of the approach of Karpinski and Nekrich [14], and we prove several interesting combinatorial properties in order to provide more efficient support for queries and updates.

We create a weight-balanced B-tree [2] T with branching parameter 8 and leaf parameter 1 such that each leaf represents an x -coordinate in P . From left to right the leaves are sorted in ascending order of the x -coordinate that they represent. Let T_u be the subtree rooted at node u . Each internal node u in the tree represents a range $R_u = [x_s, x_e]$, where x_s and x_e are the x -coordinates represented by the leftmost and rightmost leaves in T_u , respectively. We number the levels of the tree T $0, 1, \dots, \lg n$ from top to bottom. If a node is h levels above the leaf level, we say that this node is of *height* h . By the properties of weight-balanced B-trees, the range represented by an internal node of height h contains at least $8^h/2$ (except the root) and at most 2×8^h points, and the degree of each internal node is at least 2 and at most 32.

2.1 Supporting Queries

Given a query $Q' = [x'_a, x'_b]$, we perform a top-down traversal on T to map Q' to the range $Q = [x_a, x_b]$, where x_a and x_b are the points in P with x -coordinates

that are the successor and the predecessor of x'_a and x'_b in P , respectively. We call the query range Q *general* if Q is not represented by a single node of T . We first define the notion of *representing* a general query range by a set of nodes:

Definition 1. *Given a general query range $Q = [x_a, x_b]$, T partitions Q into a set of blocks, in which each block is represented by an node of T , and the range represented by the parent of each such node is not entirely contained in Q . The set, I , of all the nodes representing these blocks is the set of nodes of T representing Q .*

For each node $v \in T$, we keep a list, $C(v)$, of k *candidate* colours, i.e., the k most frequent colours in the range R_v represented by v (breaking ties arbitrarily); we will fix k later. Let $L = \cup_{v \in I} C(v)$. For each colour c , we keep a separate range counting data structure, F_c , containing all of the points of colour c in P , and also a range counting data structure, F , containing all of the points in P . Let m be the total number of points in $[x_a, x_b]$, which can be answered by F . For each $c \in L$, we query F_c with the range $[x_a, x_b]$ letting f be the result. If $f > \alpha m$, then we report f .

It is clear that I contains at most $O(\lg n)$ nodes. Furthermore, if a colour c is an α -majority for Q , then it must be an α -majority for at least one of the ranges in I [4, Observation 1]. If we set $k = \lceil 1/\alpha \rceil$ and store $\lceil 1/\alpha \rceil$ colours in each internal node as candidate colours, then, by the procedure just described, we will perform a range counting query on $O((\lg n)/\alpha)$ colours. If we use balanced search trees for our range counting data structures, then this takes $O((\lg^2 n)/\alpha)$ time overall. However, in the sequel we show how to improve this query time by exploiting the fact that the blocks in I that are closer to the root of T contain more points in the ranges that they represent.

We now prove useful properties of a general query range Q and the set, I , of nodes representing it in Lemmas 1, 2, 3, and 4. In these lemmas, m denotes the number of points in Q , and i_1, i_2, \dots denote the distinct values of the heights of the nodes in I , where $i_1 > i_2 > \dots \geq 0$. We first give an upper bound on the number of points contained in the ranges represented by the nodes of I of a given height:

Lemma 1. *The total number of points in the ranges represented by all the nodes in I of height i_j is less than $m \times \min(1, 8^{1-j} \times 31)$.*

Proof. Since Q is general and it contains at least one node of height i_1 , m is greater than the minimum number of points that can be contained in a node of height i_1 , which is $8^{i_1}/2$. The nodes of I whose height is i_j must have at least one sibling that is not in I , and the number of points contained in the interval represented by this sibling is at least $8^{i_j}/2$. Therefore, the number, m_j , of points in the ranges represented by the nodes of I at level i_j is at most $2 \times 8^{i_j+1} - 8^{i_j}/2 = 8^{i_j} \times (31/2)$. Thus, $m_j/m < 8^{i_j-i_1} \times 31 < 8^{1-j} \times 31$. \square

We now use the above lemma to bound the number of points whose colours are not among the candidate colours stored in the corresponding nodes in I .

Lemma 2. Given a node $v \in I$ of height i_j and a colour c , let $n_v^{(c)}$ denote the number of points with colour c in the range R_v , if c is not among the first $k_j = k/2^{j-1}$ most frequent candidate colours in the candidacy list of v , and $n_v^{(c)} = 0$ otherwise. Then $\sum_{v \in I} n_v^{(c)} < \frac{5.59m}{k+1}$.

Proof. If c is not among the first k_j candidate colours stored in v , then the number of points with colour c in R_v is at most $1/(k_j + 1)$ times the number of points in R_v . Thus,

$$\begin{aligned} \sum_{v \in I} n_v^{(c)} &< \sum_{j=1}^2 \frac{1}{k_j + 1} \times m + \sum_{j \geq 3} \frac{1}{k_j + 1} \times m \times 31 \times 8^{1-j} \\ &< \frac{m}{k+1} \times (1 + 2 + 31 \times (\frac{2^2}{8^2} + \frac{2^3}{8^3} + \dots)) \\ &< \frac{5.59m}{k+1} \end{aligned}$$

□

We next consider the nodes in I that are closer to the leaf levels. Let I_t denote the nodes in I that are at one of the top $t = \lceil \frac{\lg(\frac{1}{\alpha})}{3} + 2.05 \rceil$ (not necessarily consecutive) levels of the nodes in I . We prove the following property:

Lemma 3. The number of points contained in the ranges represented by the nodes in $I \setminus I_t$ is less than $\alpha m/2$.

Proof. By Lemma 1, the number of points contained in the ranges represented by the nodes in $I \setminus I_t$ is less than:

$$\begin{aligned} m \times 31 \times \sum_{j \geq t+1} 8^{1-j} &< m \times 31 \times (\frac{1}{8^t} + \frac{1}{8^{t+1}} + \dots) \\ &< m \times 31 \times \frac{8}{7} \times \frac{1}{8^t} \end{aligned}$$

Since $t \geq \frac{\lg(\frac{1}{\alpha})}{3} + 2.05$, the above value is less than $\alpha m/2$. □

With the above lemmas, we can now choose an appropriate value for k to achieve the following property that is critical to achieve improved query time:

Lemma 4. When $k = \lceil \frac{11.18}{\alpha} \rceil - 1$, any α -majority, c , of Q is among the union of the first $\lceil k/2^{j-1} \rceil$ candidates stored in each node of height i_j representing a range in I_t .

Proof. The total number of points with colour c in the ranges represented by the nodes in $I \setminus I_t$ is less than $\alpha m/2$ by Lemma 3. By Lemma 2 and our choice for the value of k , less than $\alpha m/2$ points in the ranges represented by the nodes in I_t for which c is not a candidate can have colour c . The lemma thus follows. □

For each node $v \in T$, we keep a semi-ordered list of the k candidate colours in the range R_v represented by v . The order on the colours for any candidacy list is maintained such that the most frequent $k/2^{j-1}$ colours come first, for all $j = 2, 3, \dots$, arbitrarily ordered within their positions. Note that such a semi-ordering can be obtained in linear time by repeated median findings. Thus, by setting $k = \lceil 11.18/\alpha \rceil - 1$, Lemma 4 implies that the colours that we have checked are the only possible α -majority colours for the query. Furthermore, Lemma 3 implies that we need only check the nodes on the top $O(\lg(1/\alpha))$ levels in T . Let I_t denote the set of nodes in these levels. We present the following lemma:

Lemma 5. *The data structures described in this section occupy $O(n)$ words, and can be used to answer a range α -majority query in $O((\lg n)/\alpha)$ time with the help of an additional array of size $2n$.*

Proof. To support α -majority queries, we only consider the nontrivial case in which the query range Q is general. By Lemma 4, the α -majorities can be found by examining the first $\lceil k/2^{j-1} \rceil$ candidate colours stored in each node representing a range in I_t . Thus, there are at most $O(\lceil \frac{1}{\alpha} \rceil + \lceil \frac{1}{2\alpha} \rceil + \lceil \frac{1}{4\alpha} \rceil + \dots + \lceil \frac{1}{2^{t-1}\alpha} \rceil) = O(\frac{1}{\alpha})$ relevant colours to check. Let L_t denote the set of these colours. For each $c \in L_t$ we query our range counting data structures F_c and F in $O(\lg n)$ time to determine whether c is an α -majority. Thus, the overall query time is $O((\lg n)/\alpha)$.

There are $O(n)$ nodes in the weight-balanced B-tree. Therefore, we would expect the space to be $O(n/\alpha)$ words, since each node stores $O(1/\alpha)$ colours. We use a pruning technique on the lower levels of the tree in order to reduce the space to $O(n)$ words overall. If a node u contains at most $1/\alpha$ points, then we need not store $C(u)$, since every colour in T_u is an α -majority for R_u . Instead, during a query, we can do a linear scan of the leaves of T_u in order to determine the unique colours. To make this efficient, we require an array D of size $2n$ to count the frequencies of the colours in R_u . As mentioned in Section 1.2, we can map a colour into an index of the array D , which allows us to increment a frequency counter in $O(1)$ time. Thus, we can extract the unique colours in R_u in $O(|T_u|) = O(\frac{1}{\alpha})$ time. The number of nodes whose subtrees have more than $1/\alpha$ leaves is $O(n\alpha)$. Thus, we store $O(k) = O(1/\alpha)$ words in $O(n\alpha)$ nodes, and the total space used by our B-tree T is $O(n)$ words. The only other data structures we make use of are the array D and the range counting data structures F and F_c for each $c \in C$, and together these occupy $O(n)$ words. \square

2.2 Supporting Updates

We now establish how much time is required to maintain the lists $C(v)$ in node v under insertions and deletions. We begin by observing that it is not possible to lazily maintain the list of the top $k = \lceil \frac{11.18}{\alpha} \rceil - 1$ most frequent colours in each range: many of these colours could have low frequencies, and the list $C(v)$ would have to be rebuilt after very few insertions or deletions. To circumvent this problem, we relax our requirements on what is stored in $C(v)$, only guaranteeing that *all* of the β -majorities of range R_v must be present in $C(v)$,

where $\beta = \lceil \frac{11.18}{\alpha} \rceil^{-1}$. With this alteration, we can still make use of the lemmas from the previous section, since they depend only on the fact that there are no colours $c \notin C(v)$ with frequency greater than $\beta|T_v|$. The issue now is how to maintain the β -majorities of R_v during insertions and deletions of colours.

Karpinski and Nekrich noted that, if we store the $(\beta/2)$ -majorities for each node v in T , then it is only after $|T_v|\beta/2$ deletions that we must rebuild $C(v)$ [14]. For the case of insertions and deletions, their data structure performs a range counting query at each node v along the path from the root of T to the leaf representing the inserted or deleted colour c . This counting query is used to determine if the colour c should be added to, or removed from, the list $C(v)$.

In contrast, our strategy is to be lazy during insertions and deletions, waiting as long as possible before recomputing $C(v)$, and to avoid performing range counting queries for each node in the update path. We provide a tighter analysis (to constant factors) of how many insertions and deletions can occur before the list $C(v)$ must be rebuilt. One caveat is that the results in this section only apply when $\beta \in (0, \frac{1}{2}]$. However, since $\alpha < 1$, our choice of β satisfies this condition. We present the following lemma, deferring its proof until a later version of this paper:

Lemma 6. *Suppose the list $C(v)$ for node v contains the $\lceil \frac{1-\beta+\sqrt{1-\beta}}{\beta} \rceil$ most frequent colours in the range R_v , breaking ties arbitrarily. For $\beta \in (0, \frac{1}{2}]$, this value is upper bounded by $\lceil \frac{2}{\beta} \rceil$. Let ℓ be the number of points contained in R_v . Only after $\lceil \frac{\beta\ell}{\sqrt{1-\beta}(1+\sqrt{1-\beta})} \rceil \geq \lceil \frac{\beta\ell}{2} \rceil$ insertions or deletions into T_v can a colour $c \notin C(v)$ become a β -majority for the range spanned by node v .*

By Lemma 6, our lazy updating scheme only requires each list $C(v)$ to have size $\lceil \frac{1-\beta+\sqrt{1-\beta}}{\beta} \rceil = O(1/\alpha)$. This leads to the following theorem:

Theorem 1. *Given a set P of n points in one dimension and a fixed $\alpha \in (0, 1)$, there is an $O(n)$ space data structure that supports range α -majority queries on P in $O((\lg n)/\alpha)$ time, and insertions and deletions in $O((\lg n)/\alpha)$ amortized time.*

Proof. Query time follows from Lemma 5. In order to get the desired space, we combine Lemmas 5 and 6 implying that each list $C(v)$ containing $O(1/\alpha)$ colours. This allows us to use the same pruning technique described in Lemma 5 in order to reduce the space to $O(n)$.

When an update occurs, we follow the path from the root of T to the updated node u . Suppose, without loss of generality, the update is an insertion of a point of colour c . For each vertex v in the path, if v contains a list $C(v)$ and c is in $C(v)$, we increment the count of colour c , which takes $O(1/\alpha)$ time. We also increment the counter for v that keeps track of the number of updates into T_v that have occurred since $C(v)$ was rebuilt. Thus, modifying the counters along the path requires $O((\lg n)/\alpha)$ time in the worst case.

We now look at the costs of maintaining the lists $C(v)$. The list $C(v)$ can be rebuilt in $O(|T_v|)$ time, using the array D . Note that D can be maintained under updates using the same scheme described in Section 1.2. First, we use D

to compute the frequency of all the colours in R_v in $O(|T_v|)$ time. Since there are at most $O(|T_v|)$ colours, we can select the top k most frequent colours in $O(|T_v|)$ time, where k is the value from Lemma 6. We can then enforce the necessary semi-ordering on this list in $O(k) = O(\frac{1}{\alpha})$ time. Thus, each leaf in T_v pays $O(1)$ cost every $O(|T_v|\alpha)$ insertions, or $O(1/\alpha)$ cost for $O(|T_v|)$ insertions. Since each update may cause $O(\lg n)$ lists to be rebuilt, this increases the cost to $O((\lg n)/\alpha)$ amortized time per update.

We make use of standard local rebuilding techniques to keep the tree balanced, rebuilding the lists in nodes that are merged or split during an update. Since a node v will only be merged or split after $O(|T_v|)$ updates by the properties of weight-balanced B-trees, these local rebuilding operations require $O(\lg n)$ amortized time. Thus, updates require $O((\lg n)/\alpha)$ amortized time overall, and are dominated by the costs of maintaining the lists $C(v)$ in each node v . \square

3 Speedup for Integer Coordinates

We now describe how to improve the query time of the data structure from Theorem 1 from $O((\lg n)/\alpha)$ to $O(\lg n/(\alpha \lg \lg n))$ for the case in which the x -coordinates of the points in P are word-sized integers.

To accomplish this goal, we require an improved one dimensional range counting data structure, which we get by combining two existing data structures. The fusion tree of Fredman and Willard [9] is an $O(n)$ space data structure that supports predecessor and successor queries, insertion, and deletion in $O(\lg n / \lg \lg n)$ time. The list indexing data structure of Dietz [2] uses $O(n)$ space and supports rank queries, insertion, and deletion in $O(\lg n / \lg \lg n)$ time. In Andersson et al. [1], it was observed that these data structures could be combined to support dynamic one-dimensional range counting queries in $O(\lg n / \lg \lg n)$ time per operation. We refer to this data structure as an *augmented fusion tree*.

In order to achieve $O(\lg n/(\alpha \lg \lg n))$ query time, we implement all of our range counting data structures as augmented fusion trees: i.e., the data structures F , and F_c for each $c \in C$. Immediately we get that we can perform a query in $O(\lg n/(\alpha \lg \lg n) + \lg n)$ time: $O(\lg n/(\alpha \lg \lg n))$ time for the range counting queries, and $O(\lg n)$ time to find the nodes in I_t . To remove the additive $O(\lg n)$ term, we modify our weight balanced B-tree to support dynamic lowest common ancestor queries. We defer the details to a later version of this paper, and present the following theorem:

Theorem 2. *Given a set P of n points in one dimension with integer coordinates and a fixed $\alpha \in (0, 1)$, there is an $O(n)$ space data structure that supports range α -majority queries on P in $O(\lg n/(\alpha \lg \lg n))$ time, and both insertions and deletions into P in $O((\lg n)/\alpha)$ amortized time.*

4 Dynamic Data Structures in Higher Dimensions

As an immediate application of Theorems 1 and 2, we can generalize our data structure to higher dimensions in the same manner as Karpinski and Nekrich [14], who used standard range tree techniques [3]. We present the following theorem:

Theorem 3. *Given a set P of n points in d dimensions ($d \geq 1$) and a fixed $\alpha \in (0, 1)$, there is an $O(n \lg^{d-1} n)$ space data structure that supports range α -majority queries on P in $O((\lg^d n)/\alpha)$ time, and insertions and deletions into P in $O((\lg^d n)/\alpha)$ amortized time. If the points have integer coordinates the query can be improved to $O((\lg^d n)/(\alpha \lg \lg n))$ time, without changing the update time.*

References

1. Andersson, A., Miltersen, P., Thorup, M.: Fusion trees can be implemented with AC⁰ instructions only. *Theoretical Computer Science* 215(1-2), 337–344 (1999)
2. Arge, L., Vitter, J.S.: Optimal external memory interval management. *SIAM J. Comput.* 32(6), 1488–1508 (2003)
3. Bentley, J.: Multidimensional divide-and-conquer. *Communications of the ACM* 23(4), 214–229 (1980)
4. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New Upper Bounds for Generalized Intersection Searching Problems. In: Fülpö, Z. (ed.) ICALP 1995. LNCS, vol. 944, pp. 464–474. Springer, Heidelberg (1995)
5. De Berg, M., Haverkort, H.: Significant-presence range queries in categorical data. *Algorithms and Data Structures*, 462–473 (2003)
6. Demaine, E.D., López-Ortiz, A., Munro, J.I.J.: Frequency Estimation of Internet Packet Streams with Limited Space. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 348–360. Springer, Heidelberg (2002)
7. Dietz, P.: Optimal algorithms for list indexing and subset rank. *Algorithms and Data Structures*, 39–46 (1989)
8. Durocher, S., He, M., Munro, J.I., Nicholson, P.K., Skala, M.: Range Majority in Constant Time and Linear Space. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6755, pp. 244–255. Springer, Heidelberg (2011)
9. Fredman, M., Willard, D.: Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences* 47(3), 424–436 (1993)
10. Gagie, T., Kärkkäinen, J.: Counting Colours in Compressed Strings. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 197–207. Springer, Heidelberg (2011)
11. Gupta, P., Janardan, R., Smid, M.: Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization. *Journal of Algorithms* 19(2), 282–317 (1995)
12. Husfeldt, T., Rauhe, T.: New lower bound techniques for dynamic partial sums and related problems. *SIAM Journal on Computing* 32(3), 736–753 (2003)
13. Karp, R., Shenker, S., Papadimitriou, C.: A simple algorithm for finding frequent elements in streams and bags. *ACM TODS* 28(1), 51–55 (2003)
14. Karpinski, M., Nekrich, Y.: Searching for frequent colors in rectangles. In: Proc. CCCG, pp. 11–14 (2008), <http://cccc.ca/proceedings/2008/paper02.pdf>
15. Lai, Y., Poon, C., Shi, B.: Approximate colored range and point enclosure queries. *Journal of Discrete Algorithms* 6(3), 420–432 (2008)
16. Misra, J., Gries, D.: Finding repeated elements. *Science of Computer Programming* 2(2), 143–152 (1982)
17. Wei, Z., Yi, K.: Beyond simple aggregates: indexing for summary queries. In: Proc. PODS, pp. 117–128. ACM (2011)

Dynamic Range Selection in Linear Space*

Meng He¹, J. Ian Munro², and Patrick K. Nicholson²

¹ Faculty of Computer Science, Dalhousie University, Canada

² David R. Cheriton School of Computer Science, University of Waterloo, Canada
`mhe@cs.dal.ca, {imunro,p3nichol}@uwaterloo.ca`

Abstract. Given a set S of n points in the plane, we consider the problem of answering range selection queries on S : that is, given an arbitrary x -range Q and an integer $k > 0$, return the k -th smallest y -coordinate from the set of points that have x -coordinates in Q . We present a linear space data structure that maintains a dynamic set of n points in the plane with real coordinates, and supports range selection queries in $O((\lg n / \lg \lg n)^2)$ time, as well as insertions and deletions in $O((\lg n / \lg \lg n)^2)$ amortized time. The space usage of this data structure is an $\Theta(\lg n / \lg \lg n)$ factor improvement over the previous best result, while maintaining asymptotically matching query and update times. We also present a succinct data structure that supports range selection queries on a dynamic array of n values drawn from a bounded universe.

1 Introduction

The problem of finding the *median* value in a data set is a staple problem in computer science, and is given a thorough treatment in modern textbooks [6]. In this paper we study a dynamic data structure variant of this problem in which we are given a set S of n points in the plane. The *dynamic range median problem* is to construct a data structure to represent S such that we can support *range median queries*: that is, given an arbitrary range $Q = [x_1, x_2]$, return the median y -coordinate from the set of points that have x -coordinates in Q . Furthermore, the data structure must support insertions of points into, as well as deletions from, the set S . We may also generalize our data structure to support *range selection queries*: that is, given an arbitrary x -range $Q = [x_1, x_2]$ and an integer $k > 0$, return the k -th smallest y -coordinate from the set of points that have x -coordinates in Q .

In addition to being a challenging theoretical problem, the range median and selection problems have several practical applications in the areas of image processing [9], Internet advertising, network traffic analysis, and measuring real-estate prices in a region [10].

In previous work, the data structures designed for the above problems that support queries and updates in polylogarithmic time require superlinear space [5]. In this paper, we focus on designing linear space dynamic range selection data

* This work was supported by NSERC and the Canada Research Chairs Program.

structures, without sacrificing query or update time. We also consider the problem of designing *succinct data structures* that support range selection queries on a dynamic array of values, drawn from a bounded universe: here “succinct” means that the space occupied by our data structure is close to the information-theoretic lower bound of representing the array of values [13].

1.1 Previous Work

Static Case: The static range median and selection problems have been studied heavily in recent years [3][5][10][18][19][7][8][4][5][14]. In these problems we consider the n points to be in an array: that is, the points have x -coordinates $\{1, \dots, n\}$. We now summarize the upper and lower bounds for the static problem. In the remainder of this paper we assume the word-RAM model of computation with word size $w = \Omega(\lg n)$ bits.

For exact range medians in constant time, there have been several iterations of near-quadratic space data structures [15][18][19]. For linear space data structures, Gfeller and Sanders [8] showed that range median queries could be supported in $O(\lg n)$ time¹, and Gagie et al. [7] showed that selection queries could be supported in $O(\lg \sigma)$ time using a wavelet tree, where σ is the number of distinct y -coordinates in the set of points. Optimal upper bounds of $O(\lg n / \lg \lg n)$ time for range median queries have since been achieved by Brodal et al. [4][5], and lower bounds by Jørgensen and Larsen [14]; the latter proved a cell-probe lower bound of $\Omega(\lg n / \lg \lg n)$ time for any static range selection data structure using $O(n \lg^{O(1)} n)$ bits of space. In the case of range selection when k is fixed for all queries, Jørgensen and Larsen proved a cell-probe lower bound of $\Omega(\lg k / \lg \lg n)$ time for any data structure using $O(n \lg^{O(1)} n)$ space [14]. Furthermore, they presented an adaptive data structure for range selection, where k is given at query time, that matches their lower bound, except when $k = 2^{o(\lg^2 \lg n)}$ [14]. Finally, Bose et al. [3] studied the problem of finding *approximate range medians*. A c -approximate median of range $[i..j]$ is a value of rank between $\frac{1}{c} \times \lceil \frac{j-i+1}{2} \rceil$ and $(2 - \frac{1}{c}) \times \lceil \frac{j-i+1}{2} \rceil$, for $c > 1$.

Dynamic Case: Gfeller and Sanders [8] presented an $O(n \lg n)$ space data structure for the range median problem that supports queries in $O(\lg^2 n)$ time and insertions and deletions in $O(\lg^2 n)$ amortized time. Later, Brodal et al. [4][5] presented an $O(n \lg n / \lg \lg n)$ space data structure for the dynamic range selection problem that answers range queries in $O((\lg n / \lg \lg n)^2)$ time and insertion and deletions in $O((\lg n / \lg \lg n)^2)$ amortized time. They also show a reduction from the *marked ancestor problem* [1] to the dynamic range median problem. This reduction shows that $\Omega(\lg n / \lg \lg n)$ query time is required for any data structure with polylogarithmic update time. Thus, there is still a gap of $\Theta(\lg n / \lg \lg n)$ time between the upper and lower bounds for linear and near linear space data structures.

¹ In this paper we use $\lg n$ to denote $\log_2 n$.

In the restricted case where the input is a dynamic array A of n values drawn from a bounded universe, $[1, \sigma]$, it is possible to answer range selection queries using a dynamic wavelet tree, such as the succinct dynamic string data structure of He and Munro [11]. This data structure uses $nH_0(A) + o(n \lg \sigma) + O(w)$ bits² of space, the query time is $O(\frac{\lg n \lg \sigma}{\lg \lg n})$, and the update time is $O(\frac{\lg n}{\lg \lg n}(\frac{\lg \sigma}{\lg \lg n} + 1))$.

1.2 Our Results

In Section 2, we present a *linear space* data structure for the dynamic range selection problem that answers queries in $O((\lg n / \lg \lg n)^2)$ time, and performs updates in $O((\lg n / \lg \lg n)^2)$ amortized time. This data structure can be used to represent point sets in which the points have *real coordinates*. In other words, we only assume that the coordinates of the points can be compared in constant time. This improves the space usage of the previous best data structure by a factor of $\Theta(\lg n / \lg \lg n)$ [5], while maintaining query and update time.

In Section 3, we present a succinct data structure that supports range selection queries on a dynamic array A of values drawn from a bounded universe, $[1.. \sigma]$. The data structure occupies $nH_0(A) + o(n \lg \sigma) + O(w)$ bits, and supports queries in $O(\frac{\lg n}{\lg \lg n}(\frac{\lg \sigma}{\lg \lg n} + 1))$ time, and insertions and deletions in $O(\frac{\lg n}{\lg \lg n}(\frac{\lg \sigma}{\lg \lg n} + 1))$ amortized time. This is a $\Theta(\lg \lg n)$ improvement in query time over the dynamic wavelet tree, and thus closes the gap between the dynamic wavelet tree solution and that of Brodal et al. [5].

2 Linear Space Data Structure

In this section we describe a linear space data structure for the dynamic range selection problem. Our data structure follows the same general approach as the dynamic data structure of Brodal et al. [5]. However, we make several important changes, and use several other auxiliary data structures, in order to improve the space by a factor of $\Theta(\lg n / \lg \lg n)$.

The main data structure is a weight balanced B-tree [2], T , with branching parameter $\Theta(\lg^{\varepsilon_1} n)$, for $0 < \varepsilon_1 < 1/2$, and leaf parameter 1. The tree T stores the points in S at its leaves, sorted in non-decreasing order of y -coordinate [9]. The height of T is $h_1 = \Theta(\lg n / \lg \lg n)$ levels, and we assign numbers to the levels starting with level 1 which contains the root node, down to level h_1 which contains the leaves of T . Inside each internal node $v \in T$, we store the smallest and largest y -coordinates in $T(v)$. Using these values we can acquire the path from the root of T to the leaf representing an arbitrary point contained in S in $O(\lg n)$ time; a binary search over the values stored in the children of an arbitrary internal node requires $O(\lg \lg n)$ time per level.

² $H_0(A)$ denotes the 0th-order empirical entropy of the multiset of values stored in A . Note that $H_0(A) \leq \lg \sigma$ always holds.

³ Throughout this paper, whenever we order a list based on y -coordinate, it is assumed that we break ties using the x -coordinate, and vice versa.

Following Brodal et al. [5], we store a *ranking tree* $R(v)$ inside each internal node $v \in T$. The purpose of the ranking tree $R(v)$ is to allow us to efficiently make a branching decision in the main tree T , at node v . Let $T(v)$ denote the subtree rooted at node v . The ranking tree $R(v)$ represents all of the points stored in the leaves of $T(v)$, sorted in non-decreasing order of x -coordinate. The fundamental difference between our ranking trees, and those of Brodal et al. [5], is that ours are more space efficient. Specifically, in order to achieve linear space, we must ensure that the ranking trees stored in each level of T occupy no more than $O(n \lg \lg n)$ bits in total, since there are $O(\lg n / \lg \lg n)$ levels in T . We describe the ranking trees in detail in Section 2.1, but first discuss some auxiliary data structures we require in addition to T .

We construct a red-black tree S_x that stores the points in S at its leaves, sorted in non-decreasing order of x -coordinate. As in [5], we augment the red-black tree S_x to store, in each node v , the count of how many points are stored in $T(v_1)$ and $T(v_2)$, where v_1 and v_2 are the two children of v . Using these counts, S_x can be used to map any query $[x_1, x_2]$ into r_1 , the rank of the successor of x_1 in S , and r_2 , the rank of the predecessor of x_2 in S . These ranking queries, as well as insertions and deletions into S_x , take $O(\lg n)$ time.

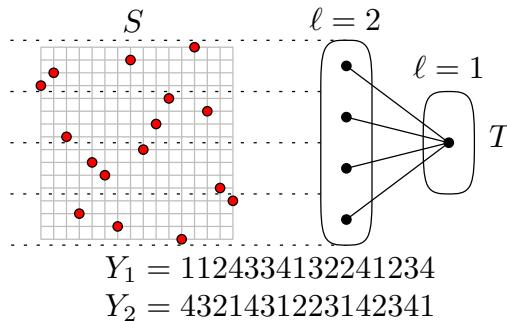


Fig. 1. The top two levels of an example tree T , and the corresponding strings Y_1 and Y_2 for these levels. Each node at level 2 has exactly 4 children.

We also store a string $Y(v)$ for each node v in T . This string consists of all of the points in $T(v)$ sorted in non-decreasing order of x -coordinate, where each point is represented by the index of the child of node v 's subtree in which they are contained, i.e., an integer bounded by $O(\lg^{\varepsilon_1} n)$. However, for technical reasons, instead of storing each string with each node $v \in T$, we concatenate all the strings $Y(v)$ for each node v at level ℓ in T into a string of length n , denoted by Y_ℓ . Each chunk of string Y_ℓ from left to right represents some node v in level ℓ of T from left to right within the level. See Figure 1 for an illustration. We represent each string Y_ℓ using the succinct dynamic string data structure of He and Munro [12]. Depending on the context, we refer to both the string, and also the data structure that represents the string, as Y_ℓ . Consider the following operations on the string Y_ℓ :

- **access**(Y_ℓ, i), which returns the i -th integer, $Y_\ell[i]$, in Y_ℓ ;
- **rank** $_\alpha(Y_\ell, i)$, which returns the number of occurrences of integer α in $Y_\ell[1..i]$;
- **range_count**($Y_\ell, x_1, x_2, y_1, y_2$), which returns the total number of entries in $Y_\ell[x_1..x_2]$ whose values are in the range $[y_1..y_2]$;
- **insert** $_\alpha(Y_\ell, i)$, which inserts integer α between $Y_\ell[i - 1]$ and $Y_\ell[i]$;
- **delete**(Y_ℓ, i), which deletes $Y_\ell[i]$ from Y_ℓ .

Let $W = \lceil \frac{\lceil \lg n \rceil^2}{\lg \lceil \lg n \rceil} \rceil$. The following lemma summarizes the functionality of these data structures for succinct *dynamic strings* over small universe:

Lemma 1 ([12]). *Under the word RAM model with word size $w = \Omega(\lg n)$, a string $Y_\ell[1..n]$ of values from a bounded universe $[1..\sigma]$, where $\sigma = O(\lg^\mu n)$ for any constant $\mu \in (0, 1)$, can be represented using $nH_0(Y_\ell) + O(\frac{n \lg \sigma \lg \lg n}{\sqrt{\lg n}}) + O(w)$ bits to support **access**, **rank**, **range_count**, **insert** and **delete** in $O(\frac{\lg n}{\lg \lg n})$ time. Furthermore, we can perform a batch of m update operations in $O(m)$ time on a substring $Y_\ell[i..i + m - 1]$ in which the j -th update operation changes the value of $Y_\ell[i + j - 1]$, provided that $m > \frac{5W}{\lg \sigma}$.*

The data structure summarized by the previous lemma is, roughly, a B-tree constructed over the string $Y_\ell[1..n]$, in which each leaf stores a *superblock*, which is a substring of $Y_\ell[1..n]$ of length at most $2W$ bits. We mention this because the ranking tree stored in each node of T will implicitly reference these superblocks instead of storing leaves. Thus, the leaves of the dynamic string at level ℓ are *shared* with the ranking trees stored in nodes at level ℓ .

As for their purpose, these dynamic string data structures Y_ℓ are used to translate the ranks r_1 and r_2 into ranks within a restricted subset of the points when we navigate a path from the root of T to a leaf. The space occupied by these strings is $O((n \lg(\lg^{\varepsilon_1} n) + w) \times \lg n / \lg \lg n)$ bits, which is $O(n)$ words. We present the following lemma:

Lemma 2. *Ignoring the ranking trees stored in each node of T , the data structures described in this section occupy $O(n)$ words.*

In the next section we discuss the technical details of our space-efficient ranking tree. The key idea to avoid using linear space per ranking tree is to *not* actually store the points in the leaves of the ranking tree, sorted in non-decreasing order of x -coordinate. Instead, for each point p in ranking tree $R(v)$, we implicitly reference the the string $Y(v)$, which stores the index of the child of v that contains p .

2.1 Space Efficient Ranking Trees

Each ranking tree $R(v)$ is a weight balanced B-tree with branching parameter $\lg^{\varepsilon_2} n$, where $0 < \varepsilon_2 < 1 - \varepsilon_1$, and leaf parameter $\Theta(W/\lg \lceil \lg n \rceil) = \Theta((\lg n / \lg \lg n)^2)$. Thus, $R(v)$ has height $\Theta(\lg n / \lg \lg n)$, and each leaf of $R(v)$ is implicitly represented by a substring of $Y(v)$, which is stored in one of the dynamic strings, Y_ℓ .

Internal Nodes: Inside each internal node u in $R(v)$, let q_i denote the number of points stored in the subtree rooted at the i -th child of u , for $1 \leq i \leq f_2$, where f_2 is the degree of u . We store a *searchable partial sums structure* [20] for the sequence $Q = \{q_1, \dots, q_{f_2}\}$. This data structure will allow us to efficiently navigate from the root of $R(v)$ to the leaf containing the point of x -coordinate rank r . The following lemma summarizes the functionality of this data structure:

Lemma 3 ([20]). *Suppose the word size is $\Omega(\lg n)$. A sequence Q of $O(\lg^\mu n)$ nonnegative integers of $O(\lg n)$ bits each, for any constant $\mu \in (0, 1)$, can be represented in $O(\lg^{1+\mu} n)$ bits and support the following operations in $O(1)$ time:*

- $\text{sum}(Q, i)$ which returns $\sum_{j=1}^i Q[j]$,
- $\text{search}(Q, x)$ which returns the smallest i such that $\text{sum}(Q, i) \geq x$,
- $\text{modify}(Q, i, \delta)$ which sets $Q[i]$ to $Q[i] + \delta$, where $|\delta| \leq \lg n$.

This data structure can be constructed in $O(\lg^\mu n)$ time, and it requires a pre-computed universal table of size $O(n^{\mu'})$ bits for any fixed $\mu' > 0$.

We also store the *matrix structure* of Brodal et al. [5] in each internal each node u of the ranking tree. Let $f_1 = \Theta(\lg^{\varepsilon_1} n)$ denote the out-degree of node $v \in T$, and let $T(v_1), \dots, T(v_{f_1})$ denote the subtrees rooted at the children of v from left to right. Similarly, recall that $f_2 = \Theta(\lg^{\varepsilon_2} n)$ denotes the out-degree of $u \in R(v)$, and let $T'(u_1), \dots, T'(u_{f_2})$ be the subtrees rooted at each child of u from left to right. These matrix structures are a kind of partial sums data structure defined as follows; we use roughly the same notation as [5]:

Definition 1 (Summarizes [5]). A matrix structure M^u is an $f_1 \times f_2$ matrix, where entry $M_{p,q}^u$ stores the number of points from $\cup_{i=1}^q T'(u_i)$ that are contained in $\cup_{i=1}^p T(v_i)$. The matrix structure M^u is stored in two ways. The first representation is a standard table, where each entry is stored in $O(\lg n)$ bits. In the second representation, we divide each column into sections of $\Theta(\lg^{\varepsilon_1} n)$ bits — leaving $\Theta(\lg \lg n)$ bits of overlap between the sections for technical reasons — and we number the sections s_1, \dots, s_g , where $g = \Theta(\lg^{1-\varepsilon_1} n)$. In the second representation, for each column q , there is a packed word $w_{q,i}^u$, storing section s_i of each entry in column q . Again, for technical reasons, the most significant bit of each section stored in the packed word $w_{q,i}^u$ is padded with a zero bit.

We defer the description of how the matrix structures are used to guide queries until Section 2.2. For now, we just treat these structures as a black box and summarize their properties with the following lemma:

Lemma 4 ([5]). *The matrix structure M^u for node u in the ranking tree $R(v)$ occupies $O(\lg^{1+\varepsilon_1+\varepsilon_2} n)$ bits, and can be constructed in $O(\lg^{1+\varepsilon_1+\varepsilon_2} n)$ time. Furthermore, consider an update path that goes through node u when we insert a value into or delete a value from $R(v)$. The matrix structures in each node along an update path can be updated in $O(1)$ amortized time per node.*

Shared Leaves: Now that we have described the internal nodes of the ranking tree, we describe how the leaves are shared between $R(v)$ and the dynamic string over Y_ℓ . To reiterate, we do not actually store the leaves of $R(v)$: they are only conceptual. We present the following lemma that will be crucial to performing queries on the ranking trees:

Lemma 5. *Let u be a leaf in $R(v)$ and S be the substring of $Y(v)$ that u represents, where each value in S is in the range $[1..σ]$, and $σ = \Theta(\lg^{\varepsilon_1} n)$. Using a universal table of size $O(\sqrt{n} \times \text{polylog}(n))$ bits, for any $z \in [1..|S|]$, an array $C_z = \{c_1, \dots, c_\sigma\}$ can be computed in $O(\lg n / \lg \lg n)$ time, where $c_i = \text{rank}_i(S, z)$, for $1 \leq i \leq \sigma$.*

We end our discussion of ranking trees by presenting the following lemma regarding their space and construction time:

Lemma 6. *Each ranking tree $R(v)$ occupies $O\left(\frac{m(\lg \lg n)^2}{\lg^{1-\varepsilon_1} n} + w\right)$ bits of space if $|T(v)| = m$, and requires $O(m)$ time to construct, assuming that we have access to the string $Y(v)$.*

Remark 1. Note that the discussion in this section implies that we need not store ranking trees for nodes $v \in T$, where $|T(v)| = O(\lg n / \lg \lg n)^2$. Instead, we can directly query the dynamic string Y_ℓ using Lemma 5 in $O(\lg n / \lg \lg n)$ time to make a branching decision in T . This will be important in Section 3, since it significantly reduces the number of pointers we need.

2.2 Answering Queries

In this section, we explain how to use our space efficient ranking tree in order to guide a range selection query in T . We are given a query $[x_1, x_2]$ as well as a rank k , and our goal is to return the k -th smallest y -coordinate in the query range. We begin our search at the root node v of the tree T . In order to guide the search to the correct child of v , we determine the canonical set of nodes in $R(v)$ that represent the query $[x_1, x_2]$. Before we query $R(v)$, we search for x_1 and x_2 in S_x . Let r_1 and r_2 denote the ranks of the successor of x_1 and predecessor of x_2 in S , respectively. We query $R(v)$ using $[r_1, r_2]$, and use the searchable partial sum data structures stored in each node of $R(v)$, to identify the canonical set of nodes in $R(v)$ that represent the interval $[r_1, r_2]$. At this point we outline how to use the matrix structures in order to decide how to branch in T .

Matrix Structures: We discuss a straightforward, *slow method* of computing the branch of the child of v to follow. The full details of how to use the matrix structures to speed up the query can be found in the original paper [5].

In order to determine the child of v that contains the k -th smallest y -coordinate in the query range, recall that T is sorted by y -coordinate. Let f_i denote the degree of v , and q'_i denote the number of points that are contained in the range $[x_1, x_2]$ in the subtree rooted at the i -th child of v , for $1 \leq i \leq d$. Determining the child that contains the k -th smallest y -coordinate in $[x_1, x_2]$ is equivalent to computing

the value τ such that $\sum_{i=1}^{\tau-1} q'_i < k$ and $\sum_{i=1}^{\tau} q'_i \geq k$. In order to compute τ , we examine the set of canonical nodes of $R(v)$ that represent $[x_1, x_2]$, denoted C . The set C contains $O(\lg n / \lg \lg n)$ internal nodes, as well as at most two leaf nodes.

Consider any internal node $u \in C$, and without loss of generality, suppose u was on the search path for r_1 , but not the search path for r_2 , and that u has degree f_2 . If the search path for r_1 goes through child c_q in u , then consider the difference between columns f_2 and q in the first representation of matrix M^u . We denote this difference as M'^u , where $M'_i = M^u_{i,f_2} - M^u_{i,q}$, for $1 \leq i \leq f_1$. For each internal node $u \in C$ we add each M'^u to a running total, and denote the overall sum as M' . Next, for each of the — at most — two leaves on the search path, we query the superblocks of Y_ℓ to get the relevant portions of the sums, and add them to M' . At this point, $M'_i = q'_i$, and it is a simple matter to scan each entry in M' to determine the value of τ . Since each matrix structure has f_1 entries in its columns, and by Lemma 5, this overall process takes $O(f_1 \times \lg n / \lg \lg n) = O(\lg^{1+\varepsilon_1} n / \lg \lg n)$ time, since there are $O(\lg n / \lg \lg n)$ levels in $R(v)$. Since there are $O(\lg n / \lg \lg n)$ levels in T , this costs $O((\lg n / \lg \lg n)^2 \times \lg^{\varepsilon_1} n)$ time. This time bound can be reduced by a factor of $f_1 = O(\lg^{\varepsilon_1} n)$, using word-level parallelism and the second representation of the matrix structures [5].

Recursively Searching in T : Let v_τ denote the τ -th child of v . The final detail to discuss is how we translate the ranks $[r_1, r_2]$ into ranks in the tree $R(v_\tau)$. To do this, we query the string $Y(v)$ before recursing to v_τ . We use two cases to describe how to find $Y(v)$ within Y_ℓ . In the first case, if v is the root of T , then $Y_\ell = Y(v)$. Otherwise, suppose the range in $Y_{\ell-1}$ that stores the parent v_p of node v begins at position z , and v is the i -th child of v_p . Let $c_j = \text{range_count}(Y(v), z, z + |Y(v)|, 1, j)$ for $1 \leq j \leq f_1$. Then, the range in Y_ℓ that stores $Y(v)$ is $[z + c_{i-1}, z + c_i]$. We then query $Y(v)$, and set $r_1 = \text{rank}_\tau(Y(v), r_1)$, $r_2 = \text{rank}_\tau(Y(v), r_2)$, $k = k - q'_{\tau-1}$, and recurse to v_τ . We present the following lemma, summarizing the arguments presented thus far:

Lemma 7. *The data structures described in this section allow us to answer range selection queries in $O((\lg n / \lg \lg n)^2)$ time.*

2.3 Handling Updates

In this section, we sketch the algorithm for updating the data structures. We start by describing how insertions are performed. First, we insert p into S_x and look up the rank, r_x , of p 's x -coordinate in S_x . Next, we use the values stored in each internal node in T to find p 's predecessor by y -coordinate, p' . We update the path from p' to the root of T . If a node v on this path splits, we must rebuild the ranking tree in the parent node v_p at level ℓ , and the dynamic string Y_ℓ .

Next, we update T in a top-down manner; starting from the root of T and following the path to the leaf storing p . Suppose that at some arbitrary node v in this path, the path branches to the j -th child of v , which we denote v_j . We insert the symbol j into its appropriate position in Y_ℓ . After updating Y_ℓ — its leaves in particular — we insert the symbol j into the ranking tree $R(v)$,

at position r_x , where r_x is the rank of the x -coordinate of p among the points in $T(v)$. As in T , each time a node splits in $R(v)$, we must rebuild the data structures in the parent node. We then update the nodes along the update path in $R(v)$ in a top-down manner: each update in $R(v)$ must be processed by all of the auxiliary data structures in each node along the update path. Thus, in each internal node, we must update the searchable partial sums data structures, as well as the matrix structures.

After updating the structures at level ℓ , we use Y_ℓ to map r_x to its appropriate rank by x -coordinate in $T(v_j)$. At this point, we can recurse to v_j . In the case of deletions, we follow the convention of Brodal et al. [5] and use node marking, and rebuild the entire data structure after $\Theta(n)$ updates. We present the following theorem; the technical details will be deferred to the full version of this paper.

Theorem 1. *Given a set S of points in the plane, there is a linear space dynamic data structure representing S that supports range selection queries for any range $[x_1, x_2]$ in $O((\lg n / \lg \lg n)^2)$ time, and supports insertions and deletions in $O((\lg n / \lg \lg n)^2)$ amortized time.*

3 Dynamic Arrays

In this section, we adapt Theorem 1 for problem of maintaining a dynamic array A of values drawn from a bounded universe $[1..\sigma]$. A query consists of a range in the array, $[i..j]$, along with an integer $k > 0$, and the output is the k -th smallest value in the subarray $A[i..j]$. Inserting a value into position i shifts the position of the values in positions $A[i..n]$ to $A[i+1..n+1]$, and deletions are analogous. We present the following theorem; the omitted proof uses standard techniques to reduce the number of word-sized pointers that are required to a constant:

Theorem 2. *Given an array $A[1..n]$ of values drawn from a bounded universe $[1..\sigma]$, there is an $nH_0(A) + o(n \lg \sigma) + O(w)$ bit data structure that can support range selection queries on A in $O(\frac{\lg n}{\lg \lg n} (\frac{\lg \sigma}{\lg \lg n} + 1))$ time, and insertions into, and deletions from, A in $O(\frac{\lg n}{\lg \lg n} (\frac{\lg \sigma}{\lg \lg n} + 1))$ amortized time. Thus, when $\sigma = O(\text{polylog}(n))$ this is $O(\frac{\lg n}{\lg \lg n})$ time for queries, and $O(\frac{\lg n}{\lg \lg n})$ amortized time for insertions and deletions.*

4 Concluding Remarks

In the same manner as Brodal et al. [5], the data structure we presented can also support orthogonal range counting queries in the same time bound as range selection queries. We note that the cell-probe lower bounds for the static range median and static orthogonal range counting match [17, 14], and — very recently — dynamic weighted orthogonal range counting was shown to have a cell-probe lower bound of $\Omega((\lg n / \lg \lg n)^2)$ query time for any data structure with polylogarithmic update time [16]. In light of these bounds, it is likely that $O((\lg n / \lg \lg n)^2)$ time for range median queries is optimal for linear space data

structures with polylogarithmic update time. However, it may be possible to do better in the case of dynamic range selection, when $k = o(n^\varepsilon)$ for any constant $\varepsilon > 0$, using an adaptive data structure as in the static case [14].

References

1. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked ancestor problems. In: Proc. 39th Annual Symposium on Foundations of Computer Science, pp. 534–543. IEEE (1998)
2. Arge, L., Vitter, J.S.: Optimal external memory interval management. SIAM J. Comput. 32(6), 1488–1508 (2003)
3. Bose, P., Kranakis, E., Morin, P., Tang, Y.: Approximate Range Mode and Range Median Queries. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 377–388. Springer, Heidelberg (2005)
4. Brodal, G., Jørgensen, A.: Data Structures for Range Median Queries. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 822–831. Springer, Heidelberg (2009)
5. Brodal, G., Gfeller, B., Jørgensen, A., Sanders, P.: Towards optimal range medians. Theoretical Computer Science (2010)
6. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill Higher Education (2001)
7. Gagie, T., Puglisi, S.J., Turpin, A.: Range Quantile Queries: Another Virtue of Wavelet Trees. In: Karlgren, J., Tarhio, J., Hyryö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 1–6. Springer, Heidelberg (2009)
8. Gfeller, B., Sanders, P.: Towards Optimal Range Medians. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 475–486. Springer, Heidelberg (2009)
9. Gil, J., Werman, M.: Computing 2-d min, median, and max filters. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(5), 504–507 (1993)
10. Har-Peled, S., Muthukrishnan, S.: Range Medians. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 503–514. Springer, Heidelberg (2008)
11. He, M., Munro, J.I.: Succinct Representations of Dynamic Strings. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 334–346. Springer, Heidelberg (2010)
12. He, M., Munro, J.I.: Space Efficient Data Structures for Dynamic Orthogonal Range Counting. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 500–511. Springer, Heidelberg (2011)
13. Jacobson, G.: Space-efficient static trees and graphs. In: Proc. SFCS, pp. 549–554 (1989)
14. Jørgensen, A., Larsen, K.: Range selection and median: Tight cell probe lower bounds and adaptive data structures. In: Proc. SODA (2011)
15. Krizanc, D., Morin, P., Smid, M.: Range mode and range median queries on lists and trees. Nordic Journal of Computing 12, 1–17 (2005)
16. Larsen, K.: The cell probe complexity of dynamic range counting. Arxiv preprint arXiv:1105.5933 (2011)
17. Pătrașcu, M.: Lower bounds for 2-dimensional range counting. In: Proc. 39th ACM Symposium on Theory of Computing (STOC), pp. 40–46 (2007)
18. Petersen, H.: Improved Bounds for Range Mode and Range Median Queries. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 418–423. Springer, Heidelberg (2008)
19. Petersen, H., Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space. Inf. Process. Lett. 109, 225–228 (2009)
20. Raman, R., Raman, V., Rao, S.S.: Succinct Dynamic Data Structures. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, pp. 426–437. Springer, Heidelberg (2001)

A Dynamic Stabbing-Max Data Structure with Sub-Logarithmic Query Time

Yakov Nekrich*

Department of Computer Science, University of Chile, Chile
yakov.nekrich@googlemail.com

Abstract. In this paper we describe a dynamic data structure that answers one-dimensional stabbing-max queries in optimal $O(\log n / \log \log n)$ time. Our data structure uses linear space and supports insertions and deletions in $O(\log n)$ and $O(\log n / \log \log n)$ amortized time respectively.

We also describe a $O(n(\log n / \log \log n)^{d-1})$ space data structure that answers d -dimensional stabbing-max queries in $O((\log n / \log \log n)^d)$ time. Insertions and deletions are supported in $O((\log n / \log \log n)^d \log \log n)$ and $O((\log n / \log \log n)^d)$ amortized time respectively.

1 Introduction

In the stabbing-max problem, a set of rectangles is stored in a data structure, and each rectangle s is assigned a priority $p(s)$. For a query point q , we must find the highest priority rectangle s that contains (or is stabbed by) q . In this paper we describe a dynamic data structure that answers stabbing-max queries on a set of one-dimensional rectangles (intervals) in optimal $O(\log n / \log \log n)$ time. We also show how this result can be extended to $d > 1$ dimensions.

Previous Work. The stabbing-max problem has important applications in networking and geographic information systems. Solutions to some special cases of the stabbing-max problem play a crucial role in classification and routing of Internet packets; we refer to e.g., [9, 11, 18] for a small selection of the previous work and to [10, 19] for more extensive surveys. Below we describe the previous works on the general case of the stabbing-max problem.

The semi-dynamic data structure of Yang and Widom [24] maintains a set of one-dimensional intervals in linear space; stabbing-max queries and insertions are supported in $O(\log n)$ time. Agarwal et al. [1] showed that stabbing-max queries on a set of one-dimensional intervals can be answered in $O(\log^2 n)$ time; the data structure of Agarwal et al. [1] uses linear space and supports updates in $O(\log n)$ time. The linear space data structure of Kaplan et al. [15] supports one-dimensional queries and insertions in $O(\log n)$ time, but deletions take $O(\log n \log \log n)$ time. In [15], the authors also consider the stabbing-max problem for a nested set of intervals: for any two intervals $s_1, s_2 \in S$, either

* Supported in part by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

$s_1 \subset s_2$ or $s_2 \subset s_1$ or $s_1 \cap s_2 = \emptyset$. Their data structure for a nested set of one-dimensional intervals uses $O(n)$ space and supports both queries and updates in $O(\log n)$ time. Thorup [22] described a linear space data structure that supports very fast queries, but needs $\log^{\omega(1)} n$ time to perform updates. His data structure supports stabbing-max queries in $O(\ell)$ time and updates in $O(n^{1/\ell})$ time for any parameter $\ell = o(\log n / \log \log n)$. However the problem of constructing a data structure with poly-logarithmic update time is not addressed in [22]. Agarwal et al. [2] described a data structure that uses linear space and supports both queries and updates in $O(\log n)$ time for an arbitrary set of one-dimensional intervals. The results presented in [15] and [2] are valid in the pointer machine model [20].

The one-dimensional data structures can be extended to $d > 1$ dimensions, so that space usage, query time, and update time increase by a factor of $O(\log^{d-1} n)$. Thus the best previously known data structure [2] for d -dimensional stabbing-max problem uses $O(n \log^{d-1} n)$ space, answers queries in $O(\log^d n)$ time, and supports updates in $O(\log^d n)$ time. Kaplan et al. [15] showed that d -dimensional stabbing-max queries can be answered in $O(\log n)$ time for any constant d in the special case of nested rectangles.

Our Result. The one-dimensional data structure described in [2] achieves optimal query time in the pointer machine model. In this paper we show that we can achieve sublogarithmic query time without increasing the update time in the word RAM model of computation. Our data structure supports deletions and insertions in $O(\log n / \log \log n)$ and $O(\log n)$ amortized time respectively. As follows from the lower bound of [3], any fully-dynamic data structure with poly-logarithmic update time needs $\Omega(\log n / \log \log n)$ time to answer a stabbing-max query.¹ Thus our data structure achieves optimal query time and space usage.

Our result can be also extended to $d > 1$ dimensions. We describe a data structure that uses $O(n(\log n / \log \log n)^{d-1})$ space and answers stabbing-max queries in $O((\log n / \log \log n)^d)$ time; insertions and deletions are supported in $O((\log n / \log \log n)^d \log \log n)$ and $O((\log n / \log \log n)^d)$ amortized time respectively. Moreover, our construction can be modified to support stabbing-sum² queries: we can count the number of one-dimensional intervals stabbed by a query point q in $O(\log n / \log \log n)$ time. The stabbing-sum data structure also supports insertions and deletions in $O(\log n)$ and $O(\log n / \log \log n)$ amortized time.

Overview. We start by describing a simple one-dimensional stabbing-max data structure in section 2. This data structure achieves the desired query and update times but needs $\omega(n^2)$ space. All intervals are stored in nodes of the base tree of height $O(\log n / \log \log n)$; the base tree is organized as a variant of the segment tree data structure. Intervals in a node u are stored in a variant of the van Emde Boas (VEB) data structure [8]. We can answer a stabbing-max query by

¹ In fact, the lower bound of [3] is valid even for existential stabbing queries: Is there an interval in the set S that is stabbed by a query point q ?

² The stabbing-sum problem considered in this paper is to be distinguished from the more general stabbing-group problem, in which every interval is associated with a weight drawn from a group G .

traversing a leaf-to-root path in the base tree; the procedure spends $O(1)$ time in each node on the path.

In section 3, we show how all secondary data structures in the nodes of the base tree can be stored in $O(n \log n)$ bits of space. The main idea of our method is the compact representation of intervals stored in each node. Similar compact representations were also used in data structures for range reporting queries [6][16] and some other problems [5]. However the previous methods are too slow for our goal: we need $O(\log \log n)$ time to obtain the representation of an element e in a node u if the representation of e in a child of u is known. Therefore it would take $O(\log n)$ time to traverse a leaf-to-root path in the base tree. In this paper we present a new, improved compact storage scheme. Using our representation, we can traverse a path in the base tree and spend $O(1)$ time in each node. We believe that our method is of independent interest and can be also applied to other problems.

The results for multi-dimensional stabbing-max queries and stabbing-sum queries are described in Theorems 2 and 3; an extensive description of these results is provided in Appendices C and D.

2 A Data Structure with Optimal Query Time

Base Tree. In this section we describe a data structure that answers stabbing-max queries in optimal time. Endpoints of all intervals from the set S are stored in the leaves of the base tree \mathcal{T} . Every leaf of \mathcal{T} contains $\Theta(\log^\varepsilon n)$ endpoints. Every internal node, except of the root, has $\Theta(\log^\varepsilon n)$ children; the root has $O(\log^\varepsilon n)$ children. Throughout this paper ε denotes an arbitrarily small positive constant. The range $rng(u)$ of a node u is an interval bounded by the minimal and maximal values stored in the leaf descendants of u .

For a leaf node u_l , the set $S(u_l)$ contains all intervals s , such that at least one endpoint of s belongs to u_l . For an internal node u , the set $S(u)$ contains all intervals s , such that $rng(u_i) \subset s$ for at least one child u_i of u but $rng(u) \not\subset s$. Thus each interval is stored in $O(\log n / \log \log n)$ sets $S(u)$. For every pair $i \leq j$, $S_{ij}(u)$ denotes the set of all intervals $s \in S(u)$ such that $rng(u_f) \subset s$ for a child u_f of u if and only if $i \leq f \leq j$. For simplicity, we will sometimes not distinguish between intervals and their priorities.

Secondary Data Structures. For each internal node u , we store a data structure $D(u)$ described in the following Proposition.

Proposition 1. *Suppose that priorities of all intervals in $S(u)$ are integers in the interval $[1, p_{\max}]$ for $p_{\max} = O(n)$. There exists a data structure $D(u)$ that uses $O(p_{\max} \cdot \log^{2\varepsilon} n)$ words of space and supports the following queries: for any $l \leq r$, the predecessor of q in $S_{lr}(u)$ can be found in $O(\log \log n)$ time and the maximum element in $S_{lr}(u)$ can be found in $O(1)$ time. The data structure supports insertions in $O(\log \log n)$ time. If a pointer to an interval $s \in S(u)$ is given, s can be deleted in $O(1)$ amortized time.*

Proof: It suffices to store all intervals from $S_{lr}(u)$ in a VEB data structure $D_{lr}(u)$. Each $D_{lr}(u)$ uses $O(p_{\max})$ words of space and answers queries in $O(\log \log p_{\max}) = O(\log \log n)$ time [8]. It is a folklore observation that we can modify the VEB data structure so that maximum queries are supported in constant time. \square

We also store a data structure $M(u)$ that contains the interval $\max_{ij}(u)$ with maximal priority among all intervals in $S_{ij}(u)$ for each $i \leq j$. Since each internal node has $\Theta(\log^\varepsilon n)$ children, $M(u)$ contains $O(\log^{2\varepsilon} n)$ elements. For any query index f , $M(u)$ reports the largest element among all \max_{ij} , $i \leq f \leq j$. In other words for any child u_f of u , $M(u)$ can find the interval with the highest priority that covers the range of the f -th child of u . Using standard techniques, we can implement $M(u)$ so that queries and updates are supported in $O(1)$ time. For completeness, we will describe the data structure $M(u)$ in the full version of this paper.

Queries and Updates. Let π denote the search path for the query point q in the base tree \mathcal{T} . The path π consists of the nodes v_0, v_1, \dots, v_R where v_0 is a leaf node and v_R is the root node. Let $s(v_i)$ be the interval with the highest priority among all intervals that are stored in $\cup_{j \leq i} S(v_j)$ and are stabbed by q . The interval $s(v_0)$ can be found by examining all $O(\log^\varepsilon n)$ intervals stored in $S(v_0)$. Suppose that we reached a node v_i and the interval $s(v_{i-1})$ is already known. If q stabs an interval s stored in $S(v_i)$, then $\text{rng}(v_{i-1}) \subset s$. Therefore q stabs an interval $s \in S(v_i)$ if and only if s is stored in some set $S_{lr}(v_i)$ such that $l \leq f \leq r$ and v_{i-1} is the f -th child of v_i . Using the data structure $M(v_i)$, we can find in constant time the maximal interval s_m , such that $s_m \in S_{lr}(v_i)$ and $l \leq f \leq r$. Then we just set $s(v_i) = \max(s_m, s(v_{i-1}))$ and proceed in the next node v_{i+1} . The total query time is $O(\log n / \log \log n)$.

When we insert an interval s , we identify $O(\log n / \log \log n)$ nodes v_i such that s is to be inserted into $S(v_i)$. For every such v_i , we proceed as follows. We identify l and r such that s belongs to $S_{lr}(v_i)$. Using $D(v_i)$, we find the position of s in $S_{lr}(v_i)$, and insert s into $S_{lr}(v_i)$. If s is the maximal interval in $S_{lr}(v_i)$, we delete the old interval $\max_{lr}(v_i)$ from the data structure $M(v_i)$, set $\max_{lr}(v_i) = s$, and insert the new $\max_{lr}(v_i)$ into $M(v_i)$.

When an interval s is deleted, we also identify nodes v_i , such that $s \in S(v_i)$. For each v_i , we find the indices l and r , such that $s \in S_{lr}(v_i)$. Using the procedure that will be described in the next section, we can find the position of s in $S_{lr}(v_i)$. Then, s is deleted from the data structure $D(v_i)$. If $s = \max_{lr}(v_i)$, we remove $\max_{lr}(v_i)$ from $M(v_i)$, find the maximum priority interval in $S_{lr}(v_i)$, and insert it into $M(v_i)$. We will show in section [3] that positions of the deleted interval s in $S_{lr}(v_i)$ for all nodes v_i can be found in $O(\log n / \log \log n)$ time. Since all other operations take $O(1)$ time per node, the total time necessary for a deletion of an interval is $O(\log n / \log \log n)$.

Unfortunately, the space usage of the data structure described in this section is very high: every VEB data structure $D_{lr}(u)$ needs $O(p_{\max})$ space, where p_{\max} is the highest possible interval priority. Even if $p_{\max} = O(n)$, all data structures $D(u)$ use $O(n^2 \log^{2\varepsilon} n)$ space. In the next section we show how all data structures $D(u)$, $u \in \mathcal{T}$, can be stored in $O(n \log n)$ bits without increasing the query and update times.

3 Compact Representation

The key idea of our compact representation is to store only interval identifiers in every node u of \mathcal{T} . Our storage scheme enables us to spend $O(\log \log n)$ bits for each identifier stored in a node u . Using the position of an interval s in a node u , we can obtain the position of s in the parent w of u . We can also compare priorities of two intervals stored in the same node by comparing their positions. These properties of our storage scheme enable us to traverse the search path for a point q and answer the query as described in section 2.

Similar representations were also used in space-efficient data structures for orthogonal range reporting [6][16] and orthogonal point location and line-segment intersection problems [5]. Storage schemes of [16][5] also use $O(\log \log n)$ bits for each interval stored in a node of the base tree. The main drawback of those methods is that we need $O(\log \log n)$ time to navigate between a node and its parent. Therefore, $\Theta(\log n)$ time is necessary to traverse a leaf-to-root path and we need $\Omega(\log n)$ time to answer a query. In this section we describe a new method that enables us to navigate between nodes of the base tree and update the lists of identifiers in $O(1)$ time per node. The main idea of our improvement is to maintain identifiers for a set $\overline{S}(u) \supset S(u)$ in every $u \in \mathcal{T}$. When an interval is inserted in $S(u)$, we also add its identifier to $\overline{S}(u)$. But when an interval is deleted from $S(u)$, its identifier is not removed from $\overline{S}(u)$. When the number of deleted interval identifiers in all $\overline{S}(u)$ exceeds the number of intervals in $S(u)$, we re-build the base tree and all secondary structures (global re-build).

Compact Lists. We start by defining a set $S'(u) \supset S(u)$. If u is a leaf node, then $S'(u) = S(u)$. If u is an internal node, then $S'(u) = S(u) \cup (\cup_i S'(u_i))$ for all children u_i of u . An interval s belongs to $S'(u)$ if at least one endpoint of s is stored in a leaf descendant of u . Hence, $|\cup_v S'(v)| = O(n)$ where the union is taken over all nodes v that are situated on the same level of the base tree \mathcal{T} . Since the height of \mathcal{T} is $O(\log n / \log \log n)$, the total number of intervals stored in all $S'(u)$, $u \in \mathcal{T}$, is $O(n \log n / \log \log n)$.

Let $\overline{S}(u)$ be the set that contains all intervals from $S'(u)$ that were inserted into $S'(u)$ since the last global re-build. We will organize global re-builds in such way that at most one half of elements in all $\overline{S}(u)$ correspond to deleted intervals. Therefore the total number of intervals in $\cup_{u \in \mathcal{T}} \overline{S}(u)$ is $O(n \log n / \log \log n)$. We will show below how we can store sets $\overline{S}(u)$ in compact form, so that an element of $\overline{S}(u)$ uses $O(\log \log n)$ bits in average. Since $S(u) \subset \overline{S}(u)$, we can also use the same method to store all $S(u)$ and $D(u)$ in $O(n \log n)$ bits.

Sets $S'(u)$ and $\overline{S}(u)$ are not stored explicitly in the data structure. Instead, we store a list $\text{Comp}(u)$ that contains a compact representation for identifiers of intervals in $\overline{S}(u)$. $\text{Comp}(u)$ is organized as follows. The set $\overline{S}(u)$ is sorted by interval priorities and divided into blocks. If $|\overline{S}(u)| > \log^3 n / 2$, then each block of $\overline{S}(u)$ contains at least $\log^3 n / 2$ and at most $2 \log^3 n$ elements. Otherwise all $e \in \overline{S}(u)$ belong to the same block. Each block B is assigned an integer block label $\text{lab}(B)$ according to the method of [14][23]. Labels of blocks are monotone with respect to order of blocks in $\text{Comp}(u)$: The block B_1 precedes B_2 in $\text{Comp}(u)$

if and only if $\text{lab}(B_1) < \text{lab}(B_2)$. Besides that, all labels assigned to blocks of $\text{Comp}(u)$ are bounded by a linear function of $|\text{Comp}(u)|/\log^3 n$: for any block B in $\text{Comp}(u)$, $\text{lab}(B) = O(|\text{Comp}(u)|/\log^3 n)$. When a new block is inserted into a list, we may have to change the labels of $O(\log^2 n)$ other blocks.

For every block B , we store its block label as well as the pointers to the next and the previous blocks in the list $\text{Comp}(u)$. For each interval \bar{s} in a block B of $\overline{\mathcal{S}}(u)$, the list $\text{Comp}(u)$ contains the *identifier* of \bar{s} in $\text{Comp}(u)$. The identifier is simply the list of indices of children u_i of u , such that $\bar{s} \in \text{Comp}(u_i)$. As follows from the description of the base tree and the sets $\overline{\mathcal{S}}(u)$, we store at most two child indices for every interval \bar{s} in a block; hence, each identifier uses $O(\log \log n)$ bits. To simplify the description, we sometimes will not distinguish between a segment s and its identifier in a list $\text{Comp}(u)$.

We say that the position of a segment s in a list $\text{Comp}(u)$ is known if the block B that contains s and the position of s in B are known. If we know positions of two intervals s_1 and s_2 in $\text{Comp}(u)$, we can compare their priorities in $O(1)$ time. Suppose that s_1 and s_2 belong to blocks B_1 and B_2 respectively. Then $p(s_1) > p(s_2)$ if $\text{lab}(B_1) > \text{lab}(B_2)$, and $p(s_1) < p(s_2)$ if $\text{lab}(B_1) < \text{lab}(B_2)$. If $\text{lab}(B_1) = \text{lab}(B_2)$, we can compare priorities of s_1 and s_2 by comparing their positions in the block $B_1 = B_2$.

The rest of this section has the following structure. First, we describe auxiliary data structures that enable us to search in a block and navigate between nodes of the base tree. Each block B contains a poly-logarithmic number of elements and every identifier in B uses $O(\log \log n)$ bits. We can use this fact and implement block data structures, so that queries and updates are supported in $O(1)$ time. If the position of some \bar{s} in a list $\text{Comp}(u)$ is known, we can find in constant time the positions of \bar{s} in the parent of u . Next, we show how data structures $D(u)$ and $M(u)$, defined in section 2, are modified. Finally, we describe the search and update procedures.

Block Data Structures. We store a data structure $F(B)$ that supports rank and select queries in a block B of $\text{Comp}(u)$: A query $\text{rank}(f, i)$ returns the number of intervals that also belong to $\overline{\mathcal{S}}(u_f)$ among the first i elements of B . A query $\text{select}(f, i)$ returns the smallest j , such that $\text{rank}(f, j) = i$; in other words, $\text{select}(f, i)$ returns the position of the i -th interval in B that also belongs to $\overline{\mathcal{S}}(u_f)$. We can answer rank and select queries in a block in $O(1)$ time. We can also count the number of elements in a block of $\text{Comp}(u)$ that are stored in the f -th child of u , and determine for the r -th element of a block in which children of u it is stored. Implementation of $F(B)$ will be described in the full version. .

For each block $B_j \in \text{Comp}(u)$ and for any child u_f of u , we store a pointer to the largest block B_j^f before B_j that contains an element from $\text{Comp}(u_f)$. These pointers are maintained with help of a data structure $P_f(u)$ for each child u_f . We implement $P_f(u)$ as an incremental split-find data structure [13]. Insertion of a new block label into $P_f(u)$ takes $O(1)$ amortized time; we can also find the block B_j^f for any block $B_j \in \text{Comp}(u)$ in $O(1)$ worst-case time. Using the data structure $F(B_j)$ for a block B_j , we can identify for any element e in a block B_j the largest $e_f \leq e$, $e_f \in B_j$, such that e_f belongs to $\text{Comp}(u_f)$. Thus we can

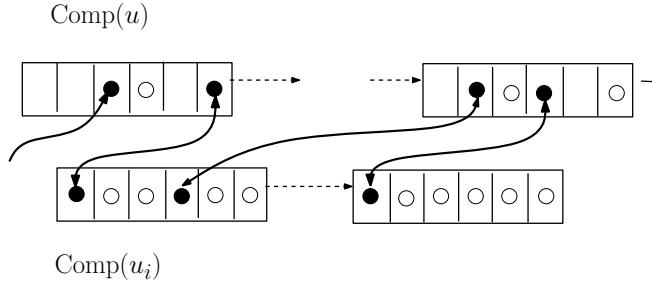


Fig. 1. Parent and child pointers in blocks of $\text{Comp}(u)$ and $\text{Comp}(u_i)$. Segments that are stored in $\text{Comp}(u_i)$ are depicted by circles; filled circles are segments with pointers. Only relevant segments and pointers are shown in $\text{Comp}(u)$. In $\text{Comp}(u_i)$, only parent pointers are shown.

identify for any $e \in \text{Comp}(u)$ the largest element $e_f \in \text{Comp}(u)$, such that $e_f \leq e$ and $e_f \in \text{Comp}(u_f)$, in $O(1)$ time.

Navigation in Nodes of the Base Tree. Finally, we store pointers to selected elements in each list $\text{Comp}(u)$. Pointers enable us to navigate between nodes of the base tree: if the position of some $e \in \text{Comp}(u)$ is known, we can find the position of e in $\text{Comp}(w)$ for the parent w of u and the position of e in $\text{Comp}(u_i)$ for any child u_i of u such that $\text{Comp}(u_i)$ contains e .

We associate a *stamp* $t(e)$ with each element stored in a block B ; every stamp is a positive integer bounded by $O(|B|)$. A pointer to an element e in a block B consists of the block label $\text{lab}(B)$ and the stamp of e in B . When an element is inserted into a block, stamps of other elements in this block do not change. Therefore, when a new interval is inserted into a block B we do not have to update all pointers that point into B . Furthermore, we store a data structure $H(B)$ for each block B . Using $H(B)$, we can find the position of an element e in B if its stamp $t(e)$ in B is known. If an element e must be inserted into B after an element e_p , then we can assign a stamp t_e to e and insert it into $H(B)$ in $O(1)$ time. Implementation of $H(B)$ is very similar to the implementation of $F(B)$ and will be described in the full version of this paper.

If e is the first element in the block B of $\text{Comp}(u)$ that belongs to $\text{Comp}(u_i)$, then we store the pointer from the copy of e in $\text{Comp}(u)$ to the copy of e in $\text{Comp}(u_i)$. If an element $e \in \text{Comp}(u)$ is also stored in $\text{Comp}(u_i)$ and e is the first element in a block B' of $\text{Comp}(u_i)$, then there is a pointer from the copy of e in $\text{Comp}(u)$ to the copy of e in $\text{Comp}(u_i)$. Such pointers will be called child pointers. For any pointer from $e \in \text{Comp}(u)$ to $e \in \text{Comp}(u_i)$, we store a pointer from $e \in \text{Comp}(u_i)$ to $e \in \text{Comp}(u)$. Such pointers will be called parent pointers. See Fig. 1 for an example.

We can store each pointer in $O(\log n)$ bits. The total number of pointers in $\text{Comp}(u)$ equals to the number of blocks in $\text{Comp}(u)$ and $\text{Comp}(u_i)$ for all children u_i of u . Hence, all pointers and all block data structures use $O(n \log n)$ bits.

If we know the position of some interval s in $\text{Comp}(v)$, we can find the position of s in the parent w of v as follows. Suppose that an interval s is stored in the block B of $\text{Comp}(v)$ and v is the f -th child of w . We find the last segment s' in B stored before s , such that there is a parent pointer from s' . Let m denote the number of elements between s' and s in B . Let B' be the block in $\text{Comp}(w)$ that contains s' . Using $H(B')$, we find the position m' of s' in the block B' of $\text{Comp}(w)$. Then the position of s in B' can be found by answering the query $\text{select}(f, \text{rank}(f, m') + m)$ to a data structure $F(B')$. Using a symmetric procedure, we can find the position of s in $\text{Comp}(v)$ if its position in $\text{Comp}(w)$ is known.

Root and Leaves of the Base Tree. Elements of $\overline{S}(v_R)$ are explicitly stored in the root node v_R . That is, we store a table in the root node v_R that enables us to find for any interval s the block B that contains the identifier of s in $\text{Comp}(v_R)$ and the stamp of s in B . Conversely, if the position of s in a block B of $\text{Comp}(v_R)$ is known, we can find its stamp in B in $O(1)$ time. If the block label and the stamp of a segment s are known, we can identify s in $O(1)$ time. In the same way, we explicitly store the elements of $\overline{S}(u_l)$ for each leaf node u_l . Moreover, we store all elements of $\overline{S}(v_R)$ in a data structure R , so that the predecessor and the successor of any value x can be found in $O(\log n / \log \log n)$ time.

Data Structures $M(u)$ and $D(u)$. Each set $S(u)$ and data structure $D(u)$ are also stored in compact form. $D(u)$ consists of structures $D_{lr}(u)$ for every pair $l \leq r$. If a block B contains a label of an interval $s \in S_{lr}(u)$, then we store the label of B in the VEB data structure $D_{lr}(u)$. The data structure $G(B)$ contains data about intervals in $S(u) \cap B$. For every $s \in G(B)$, we store indices l, r if $s \in S_{lr}(u)$; if an element $s \in B$ does not belong to S (i.e., s was already deleted), then we set $l = r = 0$. For a query index f , $G(B)$ returns in $O(1)$ time the highest priority interval $s \in B$, such that $s \in S_{lr}(u)$ and $l \leq f \leq r$. A data structure $G(B)$ uses $O(|B| \log \log n)$ bits of space and supports updates in $O(1)$ time. Implementation of $G(B)$ is very similar to the implementation of $F(B)$ and will be described in the full version of this paper. We store a data structure $M(u)$ in every node $u \in \mathcal{T}$, such that $\text{Comp}(u)$ consists of at least two blocks. For each pair $l \leq r$, the data structure $M(u)$ stores the label of the block that contains the highest priority interval in $S_{lr}(u)$. For a query f , $M(u)$ returns the label of the block that contains the highest priority interval in $\cup_{l \leq f \leq r} S_{lr}(u)$. Such queries are supported in $O(1)$ time; a more detailed description of the data structure $M(u)$ will be given in the full version.

Search Procedure. We can easily modify the search procedure of section 2 for the case when only lists $\text{Comp}(u)$ are stored in each node. Let v_i be a node on the path $\pi = v_0, \dots, v_R$, where π is the search path for the query point q . Let $s(v_i)$ denote the interval that has the highest priority among all intervals that belong to $\cup_{j \leq i} S(v_j)$ and are stabbed by q . We can examine all segments in $S(v_0)$ and find $s(v_0)$ in $O(\log^\varepsilon n)$ time. The position of $s(v_0)$ in $\text{Comp}(v_0)$ can also be found in $O(1)$ time. During the i -th step, $i \geq 1$, we find the position of $s(v_i)$ in $\text{Comp}(v_i)$. An interval s in $S(v_i)$ is stabbed by q if and only if $s \in S_{lr}(v_i)$, $l \leq f \leq r$, and

v_{i-1} is the f -th child of v_i . Using $M(v_i)$, we can find the block label of the maximal interval s_m among all intervals stored in $S_{lr}(v_i)$ for $l \leq f \leq r$. Let B_m be the block that contains s_m . Using the data structure $G(B_m)$, we can find the position of s_m in B_m .

By definition, $s(v_i) = \max(s_m, s(v_{i-1}))$. Although we have no access to s_m and $s(v_{i-1})$, we can compare their priorities by comparing positions of s_m and $s(v_{i-1})$ in $\text{Comp}(v_i)$. Since the position of $s(v_{i-1})$ in $\text{Comp}(v_{i-1})$ is already known, we can find its position in $\text{Comp}(v_i)$ in $O(1)$ time. We can also determine, whether s_m precedes or follows $s(v_{i-1})$ in $\text{Comp}(v_i)$ in $O(1)$ time. Since a query to $M(v_i)$ also takes $O(1)$ time, our search procedure spends constant time in each node v_i for $i \geq 1$. When we know the position of $s(v_R)$ in $\text{Comp}(v_R)$, we can find the interval $s(v_R)$ in $O(1)$ time. The interval $s(v_R)$ is the highest priority interval in S that is stabbed by q . Hence, a query can be answered in $O(\log n / \log \log n)$ time.

We will describe how our data structure can be updated in the full version of this paper. Our result is summed up in the following Theorem.

Theorem 1. *There exists a linear space data structure that answers orthogonal stabbing-max queries in $O(\log n / \log \log n)$ time. This data structure supports insertions and deletions in $O(\log n)$ and $O(\log n / \log \log n)$ amortized time respectively.*

Our data structure can be extended to the case of d -dimensional stabbing-max queries for $d > 1$.

Theorem 2. *There exists a data structure that uses $O(n(\log n / \log \log n)^{d-1})$ space and answers d -dimensional stabbing-max queries in $O((\log n / \log \log n)^d)$ time. Insertions and deletions are supported in $O((\log n / \log \log n)^d \log \log n)$ and $O((\log n / \log \log n)^d)$ amortized time respectively.*

Theorem 2 will be proved in the full version of this paper.

We can also modify our data structure so that it supports stabbing-sum queries: count the number of intervals stabbed by a query point q .

Theorem 3. *There exists a linear space data structure that answers orthogonal stabbing-sum queries in $O(\log n / \log \log n)$ time. This data structure supports insertions and deletions in $O(\log n)$ and $O(\log n / \log \log n)$ amortized time respectively.*

We will provide the proof of Theorem 3 in the full version of this paper.

References

1. Agarwal, P.K., Arge, L., Yang, J., Yi, K.: I/O-Efficient Structures for Orthogonal Range-Max and Stabbing-Max Queries. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 7–18. Springer, Heidelberg (2003)
2. Agarwal, P.K., Arge, L., Yi, K.: An Optimal Dynamic Interval Stabbing-Max Data Structure? In: Proc. SODA 2005, pp. 803–812 (2005)

3. Alstrup, S., Husfeldt, T., Rauhe, T.: Marked Ancestor Problems. In: Proc. FOCS 1998, pp. 534–544 (1998)
4. Arge, L., Vitter, J.S.: Optimal External Memory Interval Management. SIAM J. Comput. 32(6), 1488–1508 (2003)
5. Blelloch, G.E.: Space-Efficient Dynamic Orthogonal Point Location, Segment Intersection, and Range Reporting. In: Proc. SODA 2008, pp. 894–903 (2008)
6. Chazelle, B.: A Functional Approach to Data Structures and its Use in Multidimensional Searching. SIAM J. on Computing 17, 427–462 (1988)
7. Edelsbrunner, H.: A New Approach to Rectangle Intersections, part I. Int. J. Computer Mathematics 13, 209–219 (1983)
8. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and Implementation of an Efficient Priority Queue. Mathematical Systems Theory 10, 99–127 (1977)
9. Gupta, P., McKeown, N.: Dynamic Algorithms with Worst-Case Performance for Packet Classification. In: Pujolle, G., Perros, H.G., Fdida, S., Körner, U., Stavrakakis, I. (eds.) NETWORKING 2000. LNCS, vol. 1815, pp. 528–539. Springer, Heidelberg (2000)
10. Gupta, P., McKeown, N.: Algorithms for Packet Classification. IEEE Network 15(2), 24–32 (2001)
11. Feldmann, A., Muthukrishnan, S.: Tradeoffs for Packet Classification. In: Proc. INFOCOM 2000, pp. 1193–1202 (2000)
12. Fredman, M.L., Willard, D.E.: Trans-Dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. J. Comput. Syst. Sci. 48(3), 533–551 (1994)
13. Imai, H., Asano, T.: Dynamic Orthogonal Segment Intersection Search. Journal of Algorithms 8(1), 1–18 (1987)
14. Itai, A., Konheim, A.G., Rodeh, M.: A Sparse Table Implementation of Priority Queues. In: Even, S., Kariv, O. (eds.) ICALP 1981. LNCS, vol. 115, pp. 417–431. Springer, Heidelberg (1981)
15. Kaplan, H., Molad, E., Tarjan, R.E.: Dynamic Rectangular Intersection with Priorities. In: Proc. STOC 2003, pp. 639–648 (2003)
16. Nekrich, Y.: Orthogonal Range Searching in Linear and Almost-Linear Space. Comput. Geom. 42(4), 342–351 (2009)
17. Pătrașcu, M., Demaine, E.D.: Tight Bounds for the Partial-Sums Problem. In: Proc. 15th SODA 2004, pp. 20–29 (2004)
18. Sahni, S., Kim, K.: $O(\log n)$ Dynamic Packet Routing. In: Proc. IEEE Symposium on Computers and Communications, pp. 443–448 (2002)
19. Sahni, S., Kim, K., Lu, H.: Data Structures for One-Dimensional Packet Classification Using Most Specific Rule Matching. In: Proc. ISPAN 2002, pp. 3–14 (2002)
20. Tarjan, R.E.: A Class of Algorithms which Require Nonlinear Time to Maintain Disjoint Sets. J. Comput. Syst. Sci. 18(2), 110–127 (1979)
21. Thorup, M.: Undirected Single-Source Shortest Paths with Positive Integer Weights in Linear Time. J. ACM 46(3), 362–394 (1999)
22. Thorup, M.: Space Efficient Dynamic Stabbing with Fast Queries. In: Proc. STOC 2003, pp. 649–658 (2003)
23. Willard, D.E.: A Density Control Algorithm for Doing Insertions and Deletions in a Sequentially Ordered File in Good Worst-Case Time. Information and Computation 97, 150–204 (1992)
24. Yang, J., Widom, J.: Incremental Computation and Maintenance of Temporal Aggregates. In: Proc. IEEE Intl. Conf. on Data Engineering 2001, pp. 51–60 (2001)

Encoding 2D Range Maximum Queries

Mordecai Golin¹, John Iacono^{2,*}, Danny Krizanc³,
Rajeev Raman^{4,**}, and S. Srinivasa Rao^{5,***}

¹ Hong Kong University of Science and Technology

² Polytechnic Institute of New York University

³ Wesleyan University

⁴ University of Leicester

⁵ Seoul National University

Abstract. We consider the *two-dimensional range maximum query (2D-RMQ)* problem: given an array A of ordered values, to pre-process it so that we can find the position of the largest element in a (user-specified) range of rows and range of columns. We focus on determining the *effective* entropy of 2D-RMQ, i.e., how many bits are needed to encode A so that 2D-RMQ queries can be answered *without* access to A . We give tight upper and lower bounds on the expected effective entropy for the case when A contains independent identically-distributed random values, and new upper and lower bounds for arbitrary A , for the case when A contains few rows. The latter results improve upon upper and lower bounds by Brodal et al. (ESA 2010). We also give some efficient data structures for 2D-RMQ whose space usage is close to the effective entropy.

1 Introduction

In this paper, we study the two-dimensional *range maximum query* problem (2D-RMQ). The input to this problem is a two dimensional m by n array A of $N = m \cdot n$ elements from a totally ordered set. We assume w.l.o.g. that $m \leq n$ and that all the entries of A are distinct (identical entries of A are ordered lexicographically by their index). We consider queries of the following types. A 1-sided query consists of the positions in the array in the range $q = [1 \cdots m] \times [1 \cdots j]$, where $1 \leq j \leq n$. (For the case $m = 1$ these may also be referred to as *prefix maximum* queries.) For a 2-sided query the range is $q = [1 \cdots i] \times [1 \cdots j]$, where $1 \leq i \leq m$ and $1 \leq j \leq n$; for a 3-sided query, $q = [1 \cdots i] \times [j_1 \cdots j_2]$, where $1 \leq i \leq m$ and $1 \leq j_1 \leq j_2 \leq n$ and for a 4-sided query, the query range is $q = [i_1 \cdots i_2] \times [j_1 \cdots j_2]$, where $1 \leq i_1 \leq i_2 \leq m$ and $1 \leq j_1 \leq j_2 \leq n$. In each case, the response to a query is the position of the maximum element in the query range, i.e., $\text{RMQ}(A, q) = \text{argmax}_{(i,j) \in q} A[i, j]$. If the number of sides is not specified we assume the query is 4-sided.

* Supported in part by NSF grant CCF-1018370 and by an Alfred P. Sloan fellowship.

** Research done while on study leave from the University of Leicester.

*** Supported by the Seoul National University Foundation Research Expense.

We focus on the space requirements for answering this query in the *encoding* model [4], where the aim is to pre-process A and produce a representation of A which allows 2D-RMQ queries to be answered *without* accessing A any further. We now briefly motivate this particular question. Lossless data compression is concerned with the information content of data, and how effectively to compress/decompress the data so that it uses space close to its information content. However, there has been an explosion of interest in operating *directly* (without decompression) on compressed data via *succinct* or *compressed* data structures [128]. In such situations, a fundamental issue that needs to be considered is the “information content of the data *structure*,” formalized as follows. Given a set of objects S , and a set of queries Q , consider the equivalence class \mathcal{C} on S induced by Q , where two objects from S are equivalent if they provide the same answer to all queries in Q . Whereas traditional succinct data structures are focussed on storing a given $x \in S$ using at most $\lceil \log_2 |S| \rceil$ bits — the *entropy* of S , we consider the problem of storing x in $\lceil \log_2 |\mathcal{C}| \rceil$ bits — the *effective entropy* of S with respect to Q ¹ — while still answering queries from Q correctly.

Although this term is new, the question is not: a classical result, using *Cartesian trees* [16], shows that given an array A with n values from $\{1, \dots, n\}$, only $2n - O(\log n)$ bits are required to answer 1D-RMQ without access to A , as opposed to the $\Theta(n \log n)$ bits needed to represent A itself. The low effective entropy of 1D-RMQ is useful in many applications, e.g. it is used to simulate access to LCP information in compressed suffix arrays (see e.g. [15]). This has motivated much research into data structures whose space usage is close to the $2n - O(\log n)$ lower bound and which can answer RMQ queries quickly (see [9] and references therein). In addition to being a natural generalization of the 1D-RMQ, the 2D-RMQ query is also a standard kind of range reporting query.

Previous Work. The 2D-RMQ problem, as stated here, was proposed by Amir et al. [13]. Building on work by Atallah and Yuan [2], Brodal et al. [4] gave a hybrid data structure that combined a compressed “index” of $O(N)$ bits along with the original array A . Queries were answered using the index along with $O(1)$ accesses to A . They showed that this is an optimal point on the trade-off between the number of accesses and the memory used. In contrast, Brodal et al. refined Demaine et al.’s [6] earlier lower bound to show that the effective entropy of 2D RMQ is $\Omega(N \log m)$ bits, thus resolving in the negative Amir et al.’s open question regarding the existence of an $O(N)$ -bit encoding for the 2D-RMQ problem. Brodal et al. also gave an $O(N \min\{m, \log n\})$ bit encoding of A . Recalling that m is the smaller of the two dimensions, it is clear that Brodal et al.’s encoding is non-optimal unless $m = n^{\Omega(1)}$.

Our Results. We primarily consider two cases of the above problem: (a) the *random* case, where the input A comprises N independent, uniform (real) random numbers from $[0, 1]$, and (b) the case of *fixed* m , where A is worst-case,

¹ We will abbreviate this as “the effective entropy of Q . ”

² The variant where elements are associated with a sparse set of points in 2D, introduced by [10], is fundamentally different and is not discussed further here.

but m is taken to be a (fairly small) constant. Random inputs are of interest in practical situations and provide insights into the lower bounds of [64] — for instance, we show that the 2D-RMQ can be encoded in $O(N)$ expected bits as opposed to $\Omega(N \log m)$ bits for the worst case — that could inform the design of *adaptive* data structures which could use significantly less space for practical inputs. For the case of fixed m , we determine the precise constants in the effective entropy for particular values of m — applying the techniques of Brodal et al. directly yields significantly non-optimal lower and upper bounds. These results use ideas that may be relevant to solving the asymptotic version of the problem. The majority of our effort is directed towards determining the effective entropy and providing concrete encodings that match the effective entropy, but we also in some cases provide data structures that support range maximum queries space- and time-efficiently on the RAM model with logarithmic word size.

Effective Entropy on Random Inputs. We first consider the 1D-RMQ problem for an array $A[1 \cdots n]$ (i.e. $m = 1$) and show that, in contrast to the worst case lower bound of $2n - O(\log n)$ bits, the expected effective entropy of RMQ is $\leq cn$ bits for $c = 1.9183 \dots < 1.92$. In the 2D case, A is an $m \times n$ array with $2 \leq m \leq n$. We show bounds on the expected effective entropy of RMQ as below:

1-sided	2-sided	3-sided	4-sided
$\Theta((\log n)^2)$ bits	$\Theta((\log n)^2 \log m)$ bits	$\Theta(n(\log m)^2)$ bits	$\Theta(nm)$ bits

The 2D bounds are considerably lower than the known worst-case bounds of $O(n \log m)$ for the 1-sided case, $O(nm)$ for the 2-sided case, and known lower bounds of $\Omega(nm)$ and $\Omega(nm \log m)$ for the 3-sided and 4-sided cases respectively. The above results also hold in the weaker model where we assume all permutations of A are equally likely. We also give a data structure that supports (4-sided) RMQ queries in $O(1)$ time using expected $O(nm)$ bits of space.

Effective Entropy for Small m . Our results for the 2D RMQ problem (4-sided queries) with worst-case inputs are as follows:

1. We give an encoding based on “merging” Cartesian trees³. While this encoding uses $\Theta(nm^2)$ bits, the same as that of Brodal et al. [4], it has lower constant factors: e.g., it uses $5n - O(\log n)$ bits when $m = 2$ rather than $7n - O(\log n)$ bits [4]. We also give a data structure for the case $m = 2$ that uses $(5 + \epsilon)n$ bits and answers queries in $O(\frac{\log(1/\epsilon)}{\epsilon})$ time for any $\epsilon > 0$.
2. We give a lower bound on the effective entropy based on “merging” Cartesian trees. This lower bound is not asymptotically superior to the lower bound of Brodal et al. [4], but for all fixed $m < 2^{12}$ it gives a better lower bound than that of Brodal et al. For example, we show that for $m = 2$, the effective entropy is $5n - O(\log n)$ bits, *exactly* matching the upper bound, but the method of Brodal et al. yields only a lower bound of $n/2$ bits.

³ This encoding has also been discovered by Brodal (personal communication).

3. For the case $m = 3$, we give an encoding that requires $(6 + \log 5)n - O(\log n) \approx 8.32n$ bits⁴. Brodal et al.'s approach requires $(12 + \log 5)n - O(\log n) \approx 14.32n$ bits and the method in (1) above would require about $9n$ bits. Our lower bound from (2) is $8n - O(\log n)$ for this case.

The paper is organized as follows: in Section 2 we give bounds on the expected entropy for random inputs, in Section 3 we consider the case of small m and Section 4 gives the new data structures. Due to page limitations, several proofs are omitted from this extended abstract and can be found in [11].

2 Random Input

In this section we consider the case of “random” inputs, where the array A is populated with $N = m \cdot n$ independent uniform random real numbers in the range $[0, 1]$. We first consider the 1D-RMQ problem (Theorem 1), then the 2D cases, beginning with 1-sided queries (Theorem 2), 2-sided (Theorem 3) and finally 4- and 3-sided queries (Theorem 4).

Theorem 1. *The expected effective entropy of 2-sided queries on a 1D array of size n is at most cn bits for $c = 1.9183\dots < 1.92$.*

Proof. We study the distribution of different kinds of nodes in the Cartesian tree [16] of a random array. Given an array A containing n distinct numbers, its Cartesian tree is a binary tree in which each node corresponds to a unique position in the array, and is defined recursively as follows: the root of the tree corresponds to position i , where $A[i]$ is the maximum element in A , and the left and right subtrees of the root are the Cartesian trees for the sub-arrays $A[1\dots i-1]$ and $A[i+1\dots n]$ respectively (the Cartesian tree of a null array is the empty binary tree). A Cartesian tree for an array A can be used to answer 1D-RMQ on A via a lowest common ancestor query on the Cartesian tree.

Each node in a Cartesian tree can be of four types – it can have two children (type-2), only a left or right child (type-L/type-R), or it can be a leaf (type-0). Consider an element $A[i]$ for $1 \leq i \leq n$ and observe that the type of the i -th node in the Cartesian tree in inorder (which corresponds to $A[i]$) is determined by the relative values of $l = A[i-1]$, $m = A[i]$ and $r = A[i+1]$ (adding dummy random elements in $A[0]$ and $A[n+1]$). Specifically:

1. if $r > m > l$ then node i is type-L;
2. if $l > m > r$ then node i is type-R;
3. if $l > r > m$ or $r > l > m$ then node i is type-0 and
4. if $m > l > r$ or $m > r > l$ then node i is type-2.

In a random array, the probabilities of the alternatives above are clearly $1/6$, $1/6$, $1/3$ and $1/3$. By linearity of expectation, if N_x is the random variable that denotes the number of type- x nodes, we have that $E[N_0] = E[N_2] = n/3$ and

⁴ All logarithms are to the base 2.

$E[N_L] = E[N_R] = n/6$. The encoding consists in traversing the Cartesian tree in either level-order or in depth-first order (pre-order) and writing down the label of each node in the order it is visited: it is known that this suffices to reconstruct the Cartesian tree [143]. The sequence of labels is encoded using arithmetic coding, choosing the probability of type-0 and type-2 to be $1/3$ and that of type-L and type-R to be $1/6$. The coded output would be of size $\log 6(N_R + N_L) + \log 3(N_0 + N_2)$ (to within lower-order terms) [13]; plugging in the expected values of the random variables N_x gives the result. \square

Theorem 2. *The expected effective entropy of 1-sided queries on an $m \times n$ array is $\Theta(\log^2 n)$ bits.*

Proof. For the upper bound observe that we can recover the answers to the 1-sided queries by storing the positions of the prefix maxima, i.e., those positions, (i, j) , such that the value stored at (i, j) is the maximum among those in positions $[1 \dots m] \times [1 \dots j]$. Since the position (i, j) is a prefix maximum with probability $1/jm$ and can be stored using $\lceil \log(nm + 1) \rceil$ bits, the expected number of bits used is at most $\sum_{i=1}^m \sum_{j=1}^n (\lceil \log(nm + 1) \rceil)/jm = O(\log^2 n)$ bits.

Consider a random source that generates n elements of $\{0, 1, \dots, m\}$ as follows: the i th element of the source is j if the answer to the query $[1 \dots m] \times [1 \dots i]$ is (j, i) for some j , and 0 otherwise. Clearly the entropy of this source is a lower bound on the expected size of the encoding. This source produces n independent elements of $\{0, 1, \dots, m\}$ with the i th equal to j , $j = 1, \dots, m$, with probability $1/im$ and is equal to 0 with probability $1 - 1/i$. I.e., its entropy is $\sum_{i=1}^n [(1 - 1/i) \log(\frac{i+1}{i}) + \sum_{j=1}^m \log(im)/im] = \Omega(\log^2 n)$ bits. \square

Theorem 3. *The expected effective entropy of 2-sided queries on an $m \times n$ array is $\Theta(\log^2 n \log m)$ bits.*

Proof. Omitted from this extended abstract.

Theorem 4. *The expected effective entropies of 4-sided and 3-sided queries on an $m \times n$ array are $\Theta(nm)$ bits and $\Theta(n(\log m)^2)$ respectively.*

Proof. We begin with the 4-sided case⁵. For each position (i, j) we store a region which has the property that for any query containing (i, j) and lying entirely within that region, (i, j) is the answer to that query. This contiguous region is delimited by a monotone (along columns or rows) sequence of positions in each of the quadrants defined by (i, j) . A position (k, l) delimits the boundary of the region of (i, j) if the value in position (k, l) is the largest and the value in position (i, j) is the second largest, in the sub-array defined by (i, j) and (k, l) , i.e., any query in this sub-array not including positions on row k or column l is answered with (i, j) (dealing with boundary conditions appropriately). The answer to any query is the (unique) position inside the query whose region entirely contains the query. For each position, we store a clockwise ordered list of the positions delimiting the region (starting with the position above i in column j) by giving the position's column and row offset from (i, j) .

⁵ NB: positions are given as row index then column index, not x - y coordinates.

24	37	76	95	20	3	90	79
80	17	53	25	59	85	96	30
48	12	11	50	38	68	63	97
9	57	37	36	79	5	59	82

93	56	14	59	66	28	82	45
70	1	69	71	50	2	13	27
51	73	13	29	97	39	9	16
46	68	86	98	65	55	24	17

Fig. 1. Regions for the italicised values for 4-sided queries (left) and 3-sided queries (right), with region delimiters in blue

The expected number of bits required to store any region is at most 4 times the expected number bits required to store the region of $(1, 1)$, whose boundary runs diagonally from the first column to the first row. A position (i, j) delimits the boundary of $(1, 1)$ ($i = 1, \dots, m + 1$, $j = 1, \dots, n + 1$ excluding the case $i = j = 1$) with probability $1/ij(ij - 1)$ (if it contains the largest value and $(1, 1)$ is the second largest in the sub-array $[1 \dots i] \times [1 \dots j]$) and its offsets require at most $2(\lceil \log(i+1) \rceil + \lceil \log(j+1) \rceil)$ bits to store. I.e., the expected number of bits stored per position is at most

$$4 \cdot \left(\sum_{i=2}^{m+1} \frac{2\lceil \log(i+1) \rceil}{i(i-1)} + \sum_{j=2}^{n+1} \sum_{i=1}^{m+1} \frac{2(\lceil \log(i+1) \rceil + \lceil \log(j+1) \rceil)}{ij(ij-1)} \right) = O(1).$$

By linearity of expectation, the expected number of bits stored is $O(nm)$. The bound is tight as we can generate $\lceil n/2 \rceil m$ equiprobable independent random bits (of entropy $\Omega(nm)$) from A by reporting 1 iff the answer to the query consisting of the two positions $(2i-1, j)$ and $(2i, j)$ is $(2i, j)$, for $i = 1, \dots, \lceil n/2 \rceil$, $j = 1, \dots, m$. (The proof for the 3-sided case is omitted.) \square

3 Small m

Brodal et al. [4] gave a 2D-RMQ encoding of size $\left(\frac{m(m+3)}{2} - O(\log m)\right) \cdot 2n \approx n \cdot m(m+3)$ bits for a $m \times n$ array, and showed that the effective entropy of 2D-RMQ is at least $\log((\frac{m}{2}!)^{\lfloor \frac{n-m/2+1}{2} \rfloor})$ bits. In this section we improve upon these results for small m . Our main tool is the following lemma:

Lemma 1. *Let A be an arbitrary $m \times n$ array, $m \geq 2$. Given an encoding capable of answering range maximum queries of the form $[1 \dots (m-1)] \times [j_1 \dots j_2]$ ($1 \leq j_1 \leq j_2 \leq n$) and an encoding answering range maximum queries on the last row of A , n additional bits are necessary and sufficient to construct an encoding answering queries of the form $[1 \dots m] \times [j_1 \dots j_2]$ ($1 \leq j_1 \leq j_2 \leq n$) on A .*

Proof. The proof has two parts, one showing sufficiency (upper bound) and the other necessity (lower bound).

Upper Bound: We construct a joint Cartesian tree that can be used in answering queries of the form $[1 \dots m] \times [j_1 \dots j_2]$ for $1 \leq j_1 \leq j_2 \leq n$, using an additional n bits. The root of the joint Cartesian tree is either the answer to the query

$[1 \cdots (m-1)] \times [1 \cdots n]$ or $[m] \times [1 \cdots n]$. We store a single bit indicating the larger of these two values. We now recurse on the portions of the array to the left and right of the column with the maximum storing a single bit, which indicates which sub-problem the winner comes from, at each level of the recursion. Following this procedure, using the n additional bits it created, we can construct the joint Cartesian tree. To answer queries of the form $[1 \cdots m] \times [i \cdots j]$, the lowest common ancestor x (in the joint Cartesian tree) of i and j gives us the column in which the maximum lies. However, the comparison that placed x at the root of its subtree also tells us if the maximum lies in the m -th row or in rows $1 \cdots m-1$; in the latter case, query the given data structure for rows $1 \cdots m-1$.

Lower Bound: For simplicity we consider the case $m = 2$ — it is easy to see that by considering the maxima of the first $m-1$ elements of each column the general problem can be reduced to that of an array with two rows. Let the elements of the top and bottom rows be t_1, t_2, \dots, t_n and b_1, b_2, \dots, b_n . Given two arbitrary Cartesian trees T and B that describe the answers to the top and bottom rows, the procedure described in the upper bound for constructing the Cartesian tree for the $2 \times n$ array from T and B makes exactly n comparisons between some t_i and b_j . Let c_1, \dots, c_n be a bit string that describes the outcomes of these comparisons in the order which they are made. We now show how to assign values to the top and bottom rows that are consistent with any given T, B , and comparison string c_1, \dots, c_n . Notice this is different from (and stronger than) the trivial observation that there exists a $2 \times n$ array A such that merging T and B must use n comparisons: we show that T, B and the n bits to merge the two rows are independent components of the $2 \times n$ problem.

If the i -th comparison compares the maximum in t_{l_i}, \dots, t_{r_i} with the maximum in b_{l_i}, \dots, b_{r_i} , say that the range $[l_i, r_i]$ is *associated* with the i -th comparison. The following properties of ranges follow directly from the algorithm for constructing the Cartesian tree for both rows:

- (a) for a fixed T and B , $[l_i, r_i]$ is uniquely determined by c_1, \dots, c_{i-1} ;
- (b) if $j > i$ then the range associated with j is either contained in the range associated with i , or is disjoint from the range associated with i .

By (a), given distinct bit strings c_1, \dots, c_n and c'_1, \dots, c'_n that differ for the first time in position i , the i -th comparison would be associated with the same interval $[l_i, r_i]$ in both cases, and the query $[1..2] \times [l_i..r_i]$ then gives different answers for the two bit strings. Thus, each bitstring gives distinguishable $2 \times n$ arrays, and we now show that each bitstring gives valid $2 \times n$ arrays.

First note that if the i -th bit in a given bit string c_1, \dots, c_n is associated with the interval $[l, r]$, then it enforces the condition that $t_j > b_k$, where $j = \text{argmax}_{i \in [l, r]} \{t_i\}$ and $k = \text{argmax}_{i \in [l, r]} \{b_i\}$ or vice-versa. Construct a digraph G with vertex set $\{t_1, \dots, t_n\} \cup \{b_1, \dots, b_n\}$ which contains all edges in T and B , as well as edges for conditions $t_j > b_k$ (or vice versa) enforced by the bit string. All arcs are directed from the larger value to the smaller. We show that G is a DAG and therefore there is a partial order of the elements satisfying T and B as well as the constraints enforced by the bit string (proof omitted). \square

Using Lemma 1 we show by induction (proofs omitted):

Theorem 5. *There exists an encoding solving the 2D-RMQ problem on a $m \times n$ array requiring at most $n \cdot \frac{m(m+3)}{2}$ bits.*

Theorem 6. *The minimum space required for any encoding for the 2D-RMQ problem on a $m \times n$ array is at least $n \cdot (3m - 1) - O(m \log n)$ bits.*

For fixed $m < 2^{12}$ this lower bound is better (in n) than that of Brodal et al. [4]. For the case $m = 2$ our bounds are tight:

Corollary 1. *$5n - O(\log n)$ bits are necessary and sufficient for an encoding answering range maximum queries on a $2 \times n$ array.*

For the case $m = 3$, our upper bound is $9n$ bits and our lower bound is $8n - O(\log n)$ bits. We can improve the upper bound slightly:

Theorem 7. *The 2D-RMQ problem can be solved using at most $(6 + \log 5)n + o(n) \approx 8.322n$ bits on a $3 \times n$ array.*

Proof. We refer to the three rows of the array as T (top), M (middle) and B (bottom). We store Cartesian trees for each of the three rows (using $6n$ bits). We now show how to construct data structures for answering queries for TM (the array consisting of the top and middle rows), MB (the middle and bottom rows) and TMB (all three rows) using an additional $n \log 5 + o(n)$ bits. Let $0 \leq x \leq 1$ be the fraction of nodes in the Cartesian tree for TMB such that the maximum lies in the middle row. Given the trees for each row, and a sequence indicating for each node in the Cartesian tree for TMB in in-order, which row contains the maximum for that node, we can construct a data structure for TMB. The sequence of row maxima is coded using arithmetic coding, taking $\Pr[M] = x$ and $\Pr[B] = \Pr[T] = (1 - x)/2$; the output takes $(-x \log x - (1 - x) \log((1 - x)/2))n + o(n)$ bits [13].

We now apply the same procedure as in Lemma 1 to construct the Cartesian trees for TM and MB, storing a bit to answer whether the maximum is in the top or middle row for TM (middle or bottom row for MB) for each query made in the construction of the tree starting with the root. However, before comparing the maxima in T and M in some range, we check the answer in TMB for that range; if TMB reports the answer is in either T or M, we do not need to store a bit for that range for TM. It is easy to see that every maximum in TMB that comes from T or B saves one bit in TM or MB, and every maximum in TMB that comes from M saves one bit in both TM and MB. Thus, a total of $2n - (1 - x)n - 2xn = (1 - x)n$ bits are needed for TM and MB. The total number of bits needed for all three trees (excluding the $o(n)$ term) is $(2(1 - x) - x \log x - (1 - x) \log(1 - x))n$. This takes a maximum at $x = 1/5$ of $n \log 5$. \square

4 Data Structures for 2D-RMQ

In this section, we give efficient data structures for 2D-RMQ.

Theorem 8. *There is a data structure for 2D-RMQ on a random $m \times n$ array A which answers queries in $O(1)$ time using $O(mn)$ expected bits of storage.*

Proof. Take $N = mn$ and $\lambda = \lceil 2 \log \log N \rceil + 1$, and define the *label* of an element $z = A[i, j]$ as $\min\{\lceil \log(1/(1-z)) \rceil, \lambda\}$ if $z \neq 0$. The labels bucket the elements into λ buckets of exponentially decreasing width. We store the following:

- (a) An $m \times n$ array L , where $L[i, j]$ stores the label of $A[i, j]$.
- (b) For labels $x = 1, 2, \dots, \lambda - 1$, take $r = 2^{2x}$ and partition A into $r \times r$ submatrices (called *grid boxes* or *grid sub-boxes* below) using a regular grid with lines r apart. Partition A four times, with the origin of the grid once each at $(0, 0), (0, r/2), (r/2, 0)$ and $(r/2, r/2)$. For each grid box, and for all elements labelled x in it, store their relative ranks within the grid box.
- (c) For all values with label λ , store their global ranks in the entire array.

The query algorithm is as follows:

1. Find an element with the largest label in the query rectangle.
2. If the query contains an element with label λ , or if the maximum label is x and the query fits into a grid box at the granularity associated with label x , then use (c) or (b) respectively to answer the query.

A query fails if the maximal label in the query rectangle is $x < \lambda$ but the rectangle does not fit into any grid box associated with label x . For this case:

- (d) Explicitly store the answer for all queries that fail.

Implementation details and the proof of space usage are omitted. \square

We now show how to support 2D-RMQ queries efficiently on a $2 \times n$ array, using space close to that of Corollary [II](#)

Theorem 9. *There is a data structure for 2D-RMQ on an arbitrary $2 \times n$ array A , which answers queries in $O(k)$ time using $5n + O(n \log k/k)$ bits of space, for any parameter $k = (\log n)^{O(1)}$.*

Proof. We use the 2D-maxHeap structure of Fischer and Heun [\[9\]](#) (which is essentially a space- and query-efficient representation of a Cartesian tree) to support queries on each of the two rows, using a total of $4n + o(n)$ bits. The upper bound of Lemma [II](#) shows how to encode the joint Cartesian tree using a bit vector M of length n , to answer queries involving both the rows.

The main idea is to represent an augmented version of the joint Cartesian tree (obtained by adding leaves to each node in the Cartesian tree), referred as ACT below, using the succinct tree representation of Farzan and Munro [\[7\]](#) which is based on tree decomposition. The representation decomposes the ACT into $O(n/k)$ microtrees, each of size at most k , and stores several auxiliary structures of total size $O(n \log k/k)$ bits. It can support various queries (such as LCA) in constant time by accessing a constant number of microtrees and reading a constant number of words from the auxiliary structures. Instead of storing the representations of microtrees, we show how reconstruct any microtree in $O(k)$ time using the bit vector M (together with additional auxiliary structures of size $O(n \log k/k)$ bits). This achieves the stated bounds (details omitted). \square

5 Conclusions and Open Problems

We have given new explicit encodings for the RMQ problem, as well as (in some cases) efficient data structures whose space usage is close to the sizes of the encodings. We have focused on the cases of random matrices (which may have relevance to practical applications such as OLAP cubes [5]) and the case of small values of m . Obviously, the problem of determining the asymptotic complexity of encoding RMQ for general m remains open, as does the problem of determining the precise effective entropy of 1D-RMQ for a random array.

References

1. Amir, A., Fischer, J., Lewenstein, M.: Two-Dimensional Range Minimum Queries. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 286–294. Springer, Heidelberg (2007)
2. Atallah, M.J., Yuan, H.: Data structures for range minimum queries in multi-dimensional arrays. In: Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 150–160. SIAM (2010)
3. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. *Algorithmica* 43(4), 275–292 (2005)
4. Brodal, G.S., Davoodi, P., Rao, S.S.: On Space Efficient Two Dimensional Range Minimum Data Structures. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6347, pp. 171–182. Springer, Heidelberg (2010)
5. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. *SIGMOD Rec.* 26, 65–74 (1997)
6. Demaine, E.D., Landau, G.M., Weimann, O.: On Cartesian Trees and Range Minimum Queries. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 341–353. Springer, Heidelberg (2009)
7. Farzan, A., Munro, J.I.: A Uniform Approach Towards Succinct Representation of Trees. In: Gudmundsson, J. (ed.) SWAT 2008. LNCS, vol. 5124, pp. 173–184. Springer, Heidelberg (2008)
8. Ferragina, P., Manzini, G.: Indexing compressed text. *JACM* 52, 552–581 (2005)
9. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM J. Comput.* 40(2), 465–492 (2011)
10. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proc. 16th Annual ACM Symposium on Theory of Computing, pp. 135–143. ACM (1984)
11. Golin, M.J., Iacono, J., Krizanc, D., Raman, R., Rao, S.S.: Encoding 2D range maximum queries. Preprint (2011), <http://arxiv.org/abs/1109.2885v1>
12. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SICOMP* 35(2), 378–407 (2005)
13. Howard, P.G., Vitter, J.S.: Arithmetic coding for data compression. In: Encyclopedia of Algorithms (2008)
14. Jacobson, G.: Space-efficient static trees and graphs. In: FOCS, pp. 549–554. IEEE (1989)
15. Sadakane, K.: Compressed suffix trees with full functionality. *Theory of Computing Systems* 41(4), 589–607 (2007)
16. Vuillemin, J.: A unifying look at data structures. *Communications of the ACM* 23(4), 229–239 (1980)

Diameter and Broadcast Time of Random Geometric Graphs in Arbitrary Dimensions

Tobias Friedrich¹, Thomas Sauerwald¹, and Alexandre Stauffer²

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany

² Computer Science Division, University of California, Berkeley, USA

Abstract. A random geometric graph (RGG) is defined by placing n points uniformly at random in $[0, n^{1/d}]^d$, and joining two points by an edge whenever their Euclidean distance is at most some fixed r . We assume that r is larger than the critical value for the emergence of a connected component with $\Omega(n)$ nodes. We show that, with high probability (w.h.p.), for any two connected nodes with a minimum Euclidean distance of $\omega(\log n)$, their graph distance is only a constant factor larger than their Euclidean distance. This implies that the diameter of the largest connected component is $\Theta(n^{1/d}/r)$ w.h.p.

We also analyze the following randomized broadcast algorithm on RGGs. At the beginning, only one node from the largest connected component of the RGG is informed. Then, in each round, each informed node chooses a neighbor independently and uniformly at random and informs it. We prove that w.h.p. this algorithm informs every node in the largest connected component of an RGG within $\Theta(n^{1/d}/r + \log n)$ rounds.

1 Introduction

We study Random Geometric Graphs (RGGs) in $d \geq 2$ dimensions. An RGG is a graph resulting from placing n nodes independently and uniformly at random on $[0, n^{1/d}]^d$ and creating edges between pairs of nodes if and only if their Euclidean distance is at most r . These graphs have been studied intensively in relation to subjects such as cluster analysis, statistical physics, and wireless sensor networks [25]. Traditionally, most work on RGGs is restricted to two dimensions. However, wireless sensor networks also expand in three dimensions. Examples are sensors in water bodies [1] and sensor networks based on the use of flying anchors [19]. Another motivation for RGGs in arbitrary dimensions is multivariate statistics of high-dimensional data [23]. In this case the coordinates of the nodes of the RGG represent different attributes of the data. The metric imposed by the RGG then depicts the similarity between data elements in the high-dimensional space. Also in bioinformatics, RGGs in up to four dimensions have been observed to give an excellent fit for various global and local measures of protein-protein interaction networks [16].

Several algorithms and processes have been studied on RGGs. One prominent example is the cover time of random walks. [3] considered RGGs in two

dimensions when the coverage radius is a constant larger than the one that assures the RGG to be connected with probability $1 - o(1)$. They proved that in this regime, the cover time of an RGG is $\Theta(n \log n)$ with probability $1 - o(1)$, which is optimal up to constant factors. This has been improved by [5] who gave a more precise estimate of the cover time on RGGs that also extends to larger dimensions. However, all these works are restricted to the case where the probability that the RGG is connected goes to 1 as $n \rightarrow \infty$.

We are interested in a wider range for r . All the following results hold for the regime where the RGG is likely to contain a connected component with $\Omega(n)$ nodes. [4] proved for RGGs in $d = 2$ dimensions that with probability $1 - \mathcal{O}(n^{-1})$ any two connected nodes with a minimum Euclidean distance of $\Omega(\log^{3.5} n / r^2)$, their graph distance is only a constant factor larger than their Euclidean distance. We establish this result for all dimensions $d \geq 2$ under the weaker condition that the minimum Euclidean distance is $\omega(\log n)$. For this, we have to employ a different proof technique since the approach of [4] strongly depends on restrictions imposed by the geometry in two dimensions. Our result implies that the diameter of the largest connected component is $\mathcal{O}(n^{1/d}/r)$ with high probability¹, which was open for $d \geq 3$ and matches the corresponding bound for $d = 2$ [4, 7]. Our techniques are inspired by percolation theory and we believe them to be useful for other problems, like estimating the cover time for the largest connected component of RGGs.

Broadcasting Information

We use the forementioned structural result of RGGs to study the problem of broadcasting information in RGGs. We study the well known randomized rumor spreading algorithm which is also known as push algorithm [10]. In this algorithm, in each round each informed node chooses a neighbor independently and uniformly at random and informs it. We are interested in the runtime, i.e., how long it takes to spread a piece of information from an arbitrary node of the largest connected component to all other connected nodes.

The obvious lower bound of this process on an arbitrary graph G is $\Omega(\text{diam}(G) + \log n)$, where $\text{diam}(G)$ denotes the diameter of the largest connected component. A matching upper bound of $\mathcal{O}(\text{diam}(G) + \log n)$ is known for complete graphs [13, 24], hypercubes [10], expander graphs [27, 12], several Cayley graphs [8], and RGGs in two dimensions [4]. In this paper we prove that also RGGs in $d \geq 3$ dimensions allow an optimal broadcast time of $\mathcal{O}(\text{diam}(G) + \log n) = \mathcal{O}(n^{1/d}/r + \log n)$ w.h.p. This generalizes the two-dimensional result of [4] and significantly improves upon the general bound of $\mathcal{O}(\Delta \cdot (\text{diam}(G) + \log n))$ [10] since for sparse RGGs (where $r = \Theta(1)$) the maximum degree is $\Delta = \Theta(\log n / \log \log n)$. Note that also for connected RGGs our result implies that all nodes get informed after $\mathcal{O}(n^{1/d}/r + \log n)$ rounds.

¹ By with high probability (short: w.h.p.), we refer to an event that holds with probability at least $1 - \mathcal{O}(n^{-1})$.

2 Precise Model and Results

We consider the following random broadcast algorithm also known as the push algorithm (cf. [10]). We are given an undirected graph G . At the beginning, called round 0, a node s of G owns a piece of information, i.e., it is informed. In each subsequent round $1, 2, \dots$, every informed node chooses a neighbor independently and uniformly at random and informs that neighbor. We are interested in the runtime of this algorithm, which is the time until every node in G gets informed; in case of G being disconnected, we require every node in the same connected component as s to get informed. The runtime of this algorithm is a random variable denoted by $\mathcal{R}(s, G)$. Our aim is to prove bounds on $\mathcal{R}(s, G)$ that hold with high probability, i.e., with probability $1 - \mathcal{O}(n^{-1})$.

We study $\mathcal{R}(s, G)$ for the case of a random geometric graph G in arbitrary dimension $d \geq 2$. We define the random geometric graph in the space $\Omega := [0, n^{1/d}]^d$ equipped with the Euclidean norm, which we denote by $\|\cdot\|_2$. The most natural definition of RGG is stated as follows.

Definition 1 (cf. [23]). *Let $\mathcal{X}_n = \{X_1, X_2, \dots, X_n\}$ be points in Ω chosen independently and uniformly at random. The random geometric graph $\mathcal{G}(\mathcal{X}_n; r)$ has node set \mathcal{X}_n and edge set $\{(x, y) : x, y \in \mathcal{X}_n, \|x - y\|_2 \leq r\}$.*

In our analysis, it is more advantageous to use to the following definition.

Definition 2 (cf. [23]). *Let N_n be a Poisson random variable with mean n and let $\mathcal{P}_n = \{X_1, X_2, \dots, X_{N_n}\}$ be points chosen independently and uniformly at random from Ω ; i.e., \mathcal{P}_n is a Poisson Point Process over Ω with intensity 1. The random geometric graph $\mathcal{G}(\mathcal{P}_n; r)$ has node set \mathcal{P}_n and edge set $\{(x, y) : x, y \in \mathcal{P}_n, \|x - y\|_2 \leq r\}$.*

The following basic lemma says that any result that holds in the setting of Definition 2 with sufficiently large probability holds with similar probability in the setting of Definition II.

Lemma 1. *Let \mathcal{A} be any event that holds with probability at least $1 - \alpha$ in $\mathcal{G}(\mathcal{P}_n; r)$. Then, \mathcal{A} also holds in $\mathcal{G}(\mathcal{X}_n; r)$ with probability $1 - \mathcal{O}(\alpha\sqrt{n})$.*

Henceforth, we consider an RGG given by $G = \mathcal{G}(\mathcal{P}_n; r)$, and refer to r as the coverage radius of G . It is known that, for $d \geq 2$, there exists a critical value $r_c = r_c(d) = \Theta(1)$ such that if $r > r_c$, then with high probability the largest connected component of G has cardinality $\Omega(n)$. On the contrary, if $r < r_c$, each connected component of G has $\mathcal{O}(\log n)$ nodes with probability $1 - o(1)$ [23]. The exact value of r_c is not known, though some bounds have been derived in [18]. In addition, if $r^d \geq \frac{\log n + \omega(1)}{b_d}$, where b_d is the volume of the d -dimensional ball of radius 1, then G is connected with probability $1 - o(1)$ [21, 22].

Our main result is stated in the next theorem. It shows that if $r > r_c$, then for all s inside the largest connected component of G , $\mathcal{R}(s, G) = \mathcal{O}(n^{1/d}/r + \log n)$ with probability $1 - \mathcal{O}(n^{-1})$. Note that r_c does not depend on n , but if r is regarded as a function of n , then here and in what follows, $r > r_c$ means that this strict inequality must hold in the limit as $n \rightarrow \infty$.

Theorem 2. For a random geometric graph $G = \mathcal{G}(\mathcal{P}_n; r)$ in $d \geq 2$ dimensions, if $r > r_c$, then $\mathcal{R}(s, G) = \mathcal{O}(n^{1/d}/r + \log n)$ with probability $1 - \mathcal{O}(n^{-1})$ for all nodes s inside the largest connected component of G .

The proof of Theorem 2, which is similar to the proof of [4, Theorem 2.2] and is given in [], requires an upper bound on the length of the shortest path between nodes of G . Our result on this matter, which is stated in the next theorem, gives that for any two nodes that are sufficiently distant in Ω , the distance between them in the metric induced by G is only a constant factor larger than the optimum with probability $1 - \mathcal{O}(n^{-1})$. In particular, this result implies that the diameter of the largest connected component of G is $\mathcal{O}(n^{1/d}/r)$, a result previously known only for two dimensions and values of r that give a connected G with probability $1 - o(1)$.

For all $v_1, v_2 \in G$, we say that v_1 and v_2 are connected if there exists a path in G from v_1 to v_2 , and define $d_G(v_1, v_2)$ as the distance between v_1 and v_2 on G , that is, $d_G(v_1, v_2)$ is the length of the shortest path from v_1 to v_2 in G . Also, we denote the Euclidean distance between the locations of v_1 and v_2 by $\|v_1 - v_2\|_2$. Clearly, the length of the shortest path between two nodes v_1 and v_2 in G satisfies $d_G(v_1, v_2) \geq \|v_1 - v_2\|_2/r$.

Theorem 3. If $d \geq 2$ and $r > r_c$, for any two connected nodes v_1 and v_2 in $G = \mathcal{G}(\mathcal{P}_n; r)$ such that $\|v_1 - v_2\|_2 = \omega(\log n)$, we obtain $d_G(v_1, v_2) = \mathcal{O}(\|v_1 - v_2\|_2/r)$ with probability $1 - \mathcal{O}(n^{-1})$.

Corollary 4. If $r > r_c$, the diameter of the largest connected component of $G = \mathcal{G}(\mathcal{P}_n; r)$ is $\mathcal{O}(n^{1/d}/r)$ with probability $1 - \mathcal{O}(n^{-1})$.

The statement of Theorem 3 generalizes and improves upon Theorem 2.3 of [4] which only holds for $d = 2$ and $\|v_1 - v_2\|_2 = \Omega(\log^{3.5} n/r^2)$. The current paper not only improves upon the previous results, but also employs different proof techniques which are necessary to tackle the geometrically more involved case $d \geq 3$.

3 The Diameter of the Largest Connected Component

We devote this section to prove Theorem 3. We consider $G = \mathcal{G}(\mathcal{P}_n; r)$. Recall that we assume $r > r_c$ and $r = \mathcal{O}(\log^{1/d} n)$. (When $r = \omega(\log^{1/d} n)$, G is connected with probability $1 - o(1)$ and Theorem 3 becomes a slightly different version of [7, Theorem 8].) Note also that $r = \Omega(1)$ since $r_c = \Theta(1)$. We show that, for any two connected nodes v_1 and v_2 of G such that $\|v_1 - v_2\|_2 = \omega(\log n)$, we have $d_G(v_1, v_2) = \mathcal{O}(\|v_1 - v_2\|_2/r)$ with probability $1 - \mathcal{O}(n^{-1})$.

Before going to the proof, we establish some notation and discuss results for discrete lattices that we will use later. For $m \geq 0$, whose value we will set later, let \mathbb{S}_m be the elements of \mathbb{Z}^d contained in the cube of side length m centered at the origin (i.e., $\mathbb{S}_m = \{i \in \mathbb{Z}^d : \|i\|_\infty \leq m/2\}$). Let L be the graph with vertex set \mathbb{S}_m such that an edge between two vertices $i, j \in \mathbb{S}_m$ exists if and only if

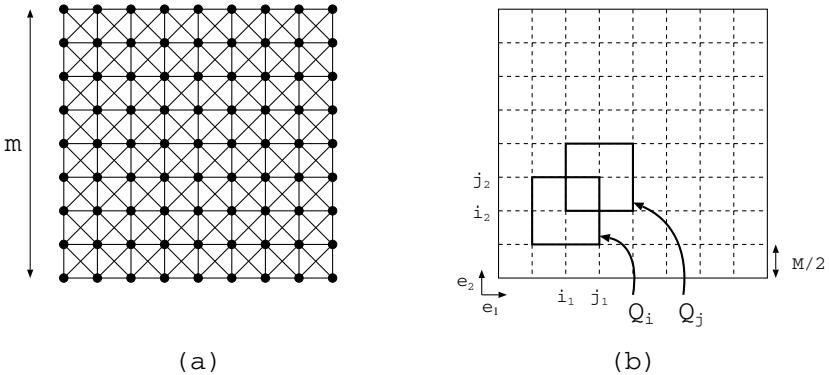


Fig. 1. (a) The graph L over \mathbb{S}_m . (b) Illustration of the neighboring cubes Q_i and Q_j .

$\|i - j\|_\infty = 1$ (see Figure 1(a)). It is easy to see that the maximum degree Δ of L is $\Delta = 3^d - 1$. Let $X = (X_i)_{i \in \mathbb{S}_m}$ be a collection of binary random variables. For two vertices $i, j \in \mathbb{S}_m$, let $d_L(i, j)$ be their graph distance in L . Also, for any $i \in \mathbb{S}_m$ and $k \geq 0$, let $\mathcal{F}_k(i)$ be the σ -field generated by all X_j with $d_L(i, j) > k$. Then, for $k \geq 0$ and $p \in (0, 1)$, we say that X is a *k -dependent site percolation process on L with probability p* if, for any $i \in \mathbb{S}_m$, we have $\Pr[X_i = 1] \geq p$ and $\Pr[X_i = 1 | \mathcal{F}_k(i)] = \Pr[X_i = 1]$; i.e., X_i is independent of any collection $(X_j)_j$ for which the distance between i and j in L is larger than k for all j in the collection. Let $L(X)$ be the subgraph of L induced by the vertices i with $X_i = 1$. The following lemma is a direct application of a result by Liggett, Schonmann and Stacey [17, Theorem 1.3] that gives that $L(X)$ stochastically dominates an independent site percolation process.

Lemma 5 ([17, Theorem 1.3]). *For given constants $p \in (0, 1)$ and $k \geq 0$, let $L(X)$ be the subgraph of L obtained via a k -dependent site percolation process X with probability p . If p is large enough, then there exists a value $p' \in (0, p]$ depending only on k and p so that $L(X)$ stochastically dominates a collection of independent Bernoulli random variables with mean p' . Moreover, p' can be made arbitrarily close to 1 by increasing p while k is kept fixed.*

We will, from now on, assume that p is so large that from Lemma 5 we can let p' be arbitrarily large. In particular, p' will be larger than the critical value for independent site percolation on the square lattice. Then if $(Y_i)_{i \in \mathbb{S}_m}$ is a collection of independent Bernoulli random variables with mean p' , we have that $L(Y)$ contains a giant component since the square lattice is contained in L [15]. We will henceforth say that a vertex $i \in \mathbb{S}_m$ is *open* if $Y_i = 1$ and *closed* if $Y_i = 0$.

Proof of Theorem 3. We take two fixed nodes v_1 and v_2 satisfying the conditions of Theorem 3 and show that the probability that v_1 and v_2 are connected by a path and $d_G(v_1, v_2) = \omega(\|v_1 - v_2\|_2/r)$ is $\mathcal{O}(n^{-3})$. Then, we would like to take the union bound over all pairs of nodes v_1 and v_2 to conclude the proof of Theorem 3; however, the number of nodes in G is a random variable and hence

the union bound cannot be employed directly. We employ the following lemma from [4] to extend the result to all pairs of nodes v_1 and v_2 .

Lemma 6 ([4, Lemma 3.1]). *Let $\mathcal{E}(w_1, w_2)$ be an event associated to a pair of nodes $w_1, w_2 \in G = \mathcal{G}(\mathcal{P}_n, r)$. Assume that, for all pairs of nodes, $\Pr[\mathcal{E}(w_1, w_2)] \geq 1 - p$, with $p > 0$. Then,*

$$\Pr \left[\bigcap_{w_1, w_2 \in G} \mathcal{E}(w_1, w_2) \right] \geq 1 - 9n^2p - e^{-\Omega(n)}.$$

Before establishing the result for two fixed nodes v_1, v_2 , we describe our renormalization argument. Fix a sufficiently large constant $M > 0$. For each $i = (i_1, i_2, \dots, i_d) \in \mathbb{Z}^d$, we define the cube Q_i centered at $(i_1M/2, i_2M/2, \dots, i_dM/2)$ whose sides have length M and are parallel to the bases of \mathbb{R}^d (see Figure II(b)). Let \mathcal{Q} be the set of Q_i having center inside Ω and set m so that $\Omega \cap \mathbb{Z}^d = \mathbb{S}_m$ (thus $Q_i \in \mathcal{Q}$ if and only if $i \in \mathbb{S}_m$). Note that $m = \Theta(n^{1/d})$ and the cubes in \mathcal{Q} cover the whole of Ω . We call two cubes Q_i and Q_j neighbors if $\|i - j\|_\infty \leq 1$. Note that in this case i and j are also neighbors in L . Therefore each cube has at most $\Delta = 3^d - 1$ neighbors, and there are at most $K = \lceil 2n^{1/d}/M \rceil^d = \Theta(n)$ cubes in \mathcal{Q} .

We say that a parallelogram R in \mathbb{R}^d has a *crossing component* if there exists a connected component inside R such that, for each face of R , there exists at least one node of the component within distance r of the face. Then, for each $i \in \mathbb{S}_m$, let \mathcal{E}_i be defined as the event where all the following happen:

- (i) For each neighbor Q_j of Q_i , the parallelogram $Q_i \cap Q_j$ contains a crossing component.
- (ii) Q_i contains only one connected component with diameter larger than $M/5$.

Note that, when \mathcal{E}_i happens for some Q_i , then (ii) above gives that the largest component of Q_i intersects the crossing components of all parallelograms $Q_i \cap Q_j$, where Q_j is a neighbor of Q_i . Moreover, for two i and j neighbors in L , we have that, if \mathcal{E}_i and \mathcal{E}_j happen, then the crossing components of Q_i and Q_j intersect. It is a direct consequence of a result of Penrose and Pisztora [20, Theorem 2] that, when $r > r_c$, for any $\varepsilon > 0$ there exists a M_0 depending only on ε and d such that, for any fixed i ,

$$\Pr[\mathcal{E}_i] \geq 1 - \varepsilon \tag{1}$$

for all $M \geq M_0$. Since M can be arbitrarily large, we set M to be larger than M_0 and $2r$. So, in general, we can assume that M/r is a constant.

Now we set $X_i = \mathbf{1}(\mathcal{E}_i)$ for all $i \in \mathbb{S}_m$. By construction, \mathcal{E}_i does not depend on the events \mathcal{E}_j for which $d_L(i, j) \geq 2$ since, in this case, the set of nodes in Q_i and the set of nodes in Q_j are disjoint. Therefore, $(X_i)_{i \in \mathbb{S}_m}$ is a 1-dependent site percolation process with probability $1 - \varepsilon$. Since ε can be made arbitrarily small, we can apply Lemma 5 to find a collection of independent Bernoulli random variables $Y = (Y_i)_{i \in \mathbb{S}_m}$ with mean p' so that $L(Y)$ is a subset of $L(X)$. Moreover,

p' can be made arbitrarily close to 1 so that $L(Y)$ has a giant component with probability $1 - \exp(-\Theta(n^{1-1/d}))$ [15].

We now show that, for any fixed pair of nodes v_1, v_2 of G such that $\|v_1 - v_2\|_2 = \omega(\log n)$, we have that either v_1 and v_2 are in different connected components or $d_G(v_1, v_2) = \mathcal{O}(\|v_1 - v_2\|_2/r)$. Let i_1 be the closest vertex of \mathbb{S}_m from v_1 and i_2 be the closest vertex of \mathbb{S}_m from v_2 . Clearly, $v_1 \in Q_{i_1}$ and $v_2 \in Q_{i_2}$. We use some ideas from Antal and Pisztora [2]. For any connected subset H of \mathbb{S}_m , let ∂H be the set of vertices of $\mathbb{S}_m \setminus H$ from which there exists an edge to a vertex in H ; that is, ∂H is the outer boundary of H . Note that $|\partial H| \leq \Delta|H|$. Let $L'(Y)$ be the graph induced by the *closed* vertices of L . For each $j \in \mathbb{S}_m$, if j is closed, let Z_j be the connected component of $L'(Y)$ containing j and let $\widehat{Z}_j = \partial Z_j$. If j is open, then set $Z_j = \emptyset$ and $\widehat{Z}_j = \{j\}$. Note that Z_j only contains closed vertices and \widehat{Z}_j only contains open vertices. Moreover, \widehat{Z}_j separates Z_j from $\mathbb{S}_m \setminus (Z_j \cup \widehat{Z}_j)$ in the sense that any path in L from a vertex in $\{j\} \cup Z_j$ to a vertex in $\mathbb{S}_m \setminus (Z_j \cup \widehat{Z}_j)$ must contain a vertex of \widehat{Z}_j . Now, let $A_j = \bigcup_{k: \|k-j\|_\infty \leq 1} Z_k$. If $A_j \neq \emptyset$, set $\widehat{A}_j = \partial A_j$; otherwise set $\widehat{A}_j = \{k \in \mathbb{S}_m : \|k-j\|_\infty \leq 1\}$.

Now we give an upper bound for the tails of $|Z_j|$ and $|A_j|$.

Lemma 7. *Let $j \in \mathbb{S}_m$. Then, if $p' > 1 - 3^{-\Delta}$, there exists a positive constant c such that, for all large enough $z > 0$, we have*

$$\Pr[|Z_j| \geq z] \leq \exp(-cz) \quad \text{and} \quad \Pr[|A_j| \geq z] \leq \exp(-cz).$$

Therefore, for some sufficiently large constant c_1 , we obtain $\Pr[|Z_j| \geq c_1 \log m] = \mathcal{O}(m^{-3d})$ and $\Pr[|A_j| \geq c_1 \log m] = \mathcal{O}(m^{-3d})$, and using the union bound over the m^d choices for j , we conclude that, for all $j \in \mathbb{S}_m$, we have $|Z_j| \leq c_1 \log m$ and $|A_j| \leq c_1 \log m$ with probability $1 - o(1/m)$.

Now we take an arbitrary path j_1, j_2, \dots, j_ℓ in L such that $j_1 = i_1$, $j_\ell = i_2$ and $\ell \leq \|i_1 - i_2\|_1$. For $2 \leq k \leq \ell - 1$ we consider the set \widehat{Z}_{j_k} . Note that, since \widehat{Z}_j separates Z_j from $\mathbb{S}_m \setminus (Z_j \cup \widehat{Z}_j)$, we have that $\bigcup_{k \in [2, \ell-1]} \widehat{Z}_{j_k}$ contains a connected component with at least one vertex from each \widehat{Z}_{j_k} , $2 \leq k \leq \ell - 1$. We call this component the *bridging* component and denote it by $B(i_1, i_2)$. For i_1 and i_2 we consider the sets \widehat{A}_{i_1} and \widehat{A}_{i_2} .

We will show how to find a path from v_1 to v_2 in G in three parts. We will bound the length of these parts by F_1 , F_2 , and F_3 so that this path from v_1 to v_2 in G contains $F_1 + F_2 + F_3$ edges. Note that, since v_1 and v_2 are such that $\|v_1 - v_2\|_2 = \omega(\log n)$ and $|A_j| \leq c_2 \log m = \mathcal{O}(\log n)$ for all j , we have that A_{i_1} and A_{i_2} are disjoint. Note that, by definition, i_1 is not a neighbor of any vertex that is not in A_{i_1} , which gives that, intuitively, \widehat{A}_{i_1} envelops the region Q_{i_1} . This property gives that, if there exists a path from v_1 to v_2 in G , this path must cross the region $\bigcup_{k \in \widehat{A}_{i_1}} Q_k$. Now, since \widehat{A}_{i_1} is a set of open vertices, we have that, for each $j \in \widehat{A}_{i_1}$, the cube Q_j has a crossing component. For any connected set $V \subseteq \mathbb{S}_m$ of open vertices, where connectivity is defined with respect to L , let $C(V)$ be the set of vertices of G that belong to the crossing component of at least one Q_j with $j \in V$. With this definition, we have that the path from v_1

to v_2 must have a node in $\mathcal{C}(\widehat{A}_{i_1})$. Let F_1 be the length of the shortest path between v_1 and a node of $\mathcal{C}(\widehat{A}_{i_1}) \cap \mathcal{C}(\widehat{Z}_{j_2})$. Note that this node must exist since $\widehat{A}_{i_1} \cap \widehat{Z}_{j_2} \neq \emptyset$ by construction. If we denote by R the set $A_{i_1} \cup \widehat{A}_{i_1}$, we have that this path is completely contained inside $\cup_{k \in R} Q_k$. Therefore, we can bound F_1 using the following geometric lemma.

Lemma 8. *Let I be a set of vertices of \mathbb{S}_m and $Q = \cup_{i \in I} Q_i$. Let w_1 and w_2 be two nodes of G inside Q . If there exists a path between w_1 and w_2 entirely contained in Q , then there exists a constant $c > 0$ depending only on d such that*

$$d_G(w_1, w_2) \leq \frac{c|I|M^d}{r^d}.$$

Remark 9. *The result in Lemma 8 also holds when I is replaced by any bounded subset of \mathbb{R}^d composed of the union of parallelograms; in this case, the term $|I|M^d$ gets replaced by the volume of this set.*

Lemma 8 then establishes that there exists a constant c_2 such that

$$F_1 \leq \frac{c_2|A_{i_1} \cup \widehat{A}_{i_1}|M^d}{r^d} \leq \frac{c_2(1 + \Delta)(|A_{i_1}| + 1)M^d}{r^d} = \mathcal{O}(\log m),$$

since $|\widehat{A}_j| \leq \Delta|A_j| + \Delta = \mathcal{O}(\log m)$ for all j , and M/r is constant. Similarly, there is a path from v_2 to a node inside $\mathcal{C}(\widehat{A}_{i_2}) \cap \mathcal{C}(\widehat{A}_{j_{\ell-1}})$, whose length we denote by F_2 . An analogous derivation then gives that $F_2 = \mathcal{O}(\log m)$. These paths must intersect $\mathcal{C}(B(i_1, i_2))$ since they intersect $\mathcal{C}(\widehat{A}_{j_2})$ and $\mathcal{C}(\widehat{A}_{j_{\ell-1}})$, respectively. Denote the length of the path in $\mathcal{C}(B(i_1, i_2))$ that connects the two paths we found above by F_3 . Using Lemma 8 we obtain a constant c_3 such that

$$F_3 \leq \frac{c_3|B(i_1, i_2)|M^d}{r^d}. \quad (2)$$

In order to bound $|B(i_1, i_2)|$, we use a coupling argument by Fontes and Newman [11] and a result of Deuschel and Pisztora [6, Lemma 2.3], which gives $\Pr\left[\sum_{k=2}^{\ell-1} |\widehat{Z}_{j_k}| \geq \ell\alpha\right] \leq \Pr\left[\sum_{k=2}^{\ell-1} |Z_{j_k}| \geq (\ell\alpha - 1)/\Delta\right] \leq \Pr\left[\sum_{k=2}^{\ell-1} |\tilde{Z}_{j_k}| \geq (\ell\alpha - 1)/\Delta\right]$, where the first inequality follows since $|\widehat{Z}_j| \leq 1 + \Delta|Z_j|$ for all j , and the \tilde{Z} 's are defined to be independent random variables such that \tilde{Z}_{j_k} has the same distribution as Z_{j_k} . From Lemma 7 we know that \tilde{Z}_{j_k} is stochastically dominated by an exponential random variable with mean $\mu = \Theta(1)$. Then, applying a Chernoff bound for exponential random variables in the equation above, we obtain a constant c_4 such that, for any large enough α , we have $\Pr\left[\sum_{k=2}^{\ell-1} |\widehat{Z}_{j_k}| \geq \alpha\ell\right] \leq \exp\left(-\frac{c_4\alpha\ell}{\mu}\right)$. Since $\ell = \omega(\log n)$ we have that $\Pr\left[\sum_{k=2}^{\ell-1} |\widehat{Z}_{j_k}| \geq \alpha\ell\right] = \mathcal{O}(m^{-3d})$ for some large enough α . Then, using (2), we have that, with probability $1 - \mathcal{O}(m^{-3d})$,

$$F_3 \leq \frac{c_3\alpha\ell M^d}{r^d} = \mathcal{O}(\|i_1 - i_2\|_1) = \mathcal{O}\left(\frac{\|v_1 - v_2\|_2}{M}\right) = \mathcal{O}\left(\frac{\|v_1 - v_2\|_2}{r}\right).$$

Putting everything together, with probability $1 - \mathcal{O}(m^{-3d})$, we obtain a path from v_1 to v_2 with length at most

$$F_1 + F_2 + F_3 = \mathcal{O} \left(\log n + \frac{\|v_1 - v_2\|_2}{r} \right). \quad (3)$$

By Lemma 6, the result above holds for all connected pairs of nodes v_1, v_2 such that $\|v_1 - v_2\|_2 = \omega(\log n)$. Then using $m = \Theta(n^{1/d})$ completes the proof. \square

References

- [1] Akyildiz, I., Pompili, D., Melodia, T.: Underwater acoustic sensor networks: research challenges. *Ad Hoc Networks* 3, 257–279 (2005)
- [2] Antal, P., Pisztora, A.: On the chemical distance for supercritical Bernoulli percolation. *The Annals of Probability* 24, 1036–1048 (1996)
- [3] Avin, C., Ercal, G.: On the Cover Time and Mixing Time of Random Geometric Graphs. *Theoretical Computer Science* 380(1-2), 2–22 (2007)
- [4] Bradonjić, M., Elsässer, R., Friedrich, T., Sauerwald, T., Stauffer, A.: Efficient broadcast on random geometric graphs. In: 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 1412–1421 (2010)
- [5] Cooper, C., Frieze, A.: The cover time of random geometric graphs. In: 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), pp. 48–57 (2009)
- [6] Deuschel, J.-D., Pisztora, A.: Surface order large deviations for high-density percolation. *Probability Theory and Related Fields* 104, 467–482 (1996)
- [7] Ellis, R.B., Martin, J.L., Yan, C.: Random geometric graph diameter in the unit ball. *Algorithmica* 47(4), 421–438 (2007)
- [8] Elsässer, R., Sauerwald, T.: Broadcasting vs. Mixing and Information Dissemination on Cayley Graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 163–174. Springer, Heidelberg (2007)
- [9] Elsässer, R., Sauerwald, T.: On the runtime and robustness of randomized broadcasting. *Theoretical Computer Science* 410(36), 3414–3427 (2009)
- [10] Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized Broadcast in Networks. *Random Structures and Algorithms* 1(4), 447–460 (1990)
- [11] Fontes, L., Newman, C.: First passage percolation for random colorings of \mathbb{Z}^d . *The Annals of Applied Probability* 30(3), 746–762 (1993)
- [12] Fountoulakis, N., Panagiotou, K.: Rumor Spreading on Random Regular Graphs and Expanders. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX 2010, LNCS, vol. 6302, pp. 560–573. Springer, Heidelberg (2010)
- [13] Frieze, A., Grimmett, G.: The shortest-path problem for graphs with random-arc-lengths. *Discrete Applied Mathematics* 10, 57–77 (1985)
- [14] Frieze, A., Molloy, M.: Broadcasting in random graphs. *Discrete Applied Mathematics* 54, 77–79 (1994)
- [15] Grimmett, G.: Percolation, 2nd edn. Springer, Heidelberg (1999)
- [16] Higham, D., Rašajski, M., Pržulj, N.: Fitting a geometric graph to a protein–protein interaction network. *Bioinformatics* 24(8), 1093 (2008)
- [17] Liggett, T., Schonmann, R., Stacey, A.: Domination by product measures. *The Annals of Probability* 25(1), 71–95 (1997)
- [18] Meester, R., Roy, R.: Continuum percolation. Cambridge University Press (1996)

- [19] Ou, C.-H., Ssu, K.-F.: Sensor position determination with flying anchors in three-dimensional wireless sensor networks. *IEEE Transactions on Mobile Computing* 7, 1084–1097 (2008)
- [20] Penrose, M., Pisztora, A.: Large deviations for discrete and continuous percolation. *Advances in Applied Probability* 28, 29–52 (1996)
- [21] Penrose, M.D.: The longest edge of the random minimal spanning tree. *The Annals of Applied Probability* 7(2), 340–361 (1997)
- [22] Penrose, M.D.: On k -connectivity for a geometric random graph. *Random Struct. Algorithms* 15(2), 145–164 (1999)
- [23] Penrose, M.D.: *Random Geometric Graphs*. Oxford University Press (2003)
- [24] Pittel, B.: On spreading rumor. *SIAM Journal on Applied Mathematics* 47(1), 213–223 (1987)
- [25] Pottie, G.J., Kaiser, W.J.: Wireless integrated network sensors. *Commun. ACM* 43(5), 51–58 (2000)
- [26] Ravelomanana, V.: Extremal properties of three-dimensional sensor networks with applications. *IEEE Transactions on Mobile Computing* 3, 246–257 (2004)
- [27] Sauerwald, T.: On mixing and edge expansion properties in randomized broadcasting. *Algorithmica* 56, 51–88 (2010)
- [28] Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., et al.: A macroscope in the redwoods. In: 3rd ACM International Conference on Embedded Networked Sensor Systems (SenSys), pp. 51–63 (2005)

Broadcasting in Heterogeneous Tree Networks with Uncertainty*

Cheng-Hsiao Tsou¹, Gen-Huey Chen¹, and Ching-Chi Lin²

¹ Department of Computer Science and Information Engineering,
National Taiwan University, Taipei 10617, Taiwan
{f97922063,ghchen}@csie.ntu.edu.tw

² Department of Computer Science and Engineering,
National Taiwan Ocean University, Keelung 20224, Taiwan
lincc@mail.ntou.edu.tw

Abstract. We consider the broadcasting problem in heterogeneous tree networks $T = (V(T), E(T))$ with edge weight uncertainty, where the edge weight $w_{u,v}$ can take any value from $[w_{u,v}^-, w_{u,v}^+]$ with unknown distribution. The broadcasting problem with uncertainty is to find a “minmax-regret” broadcast center to minimize the worst-case loss in the objective function that may occur due to the uncertainty. In this paper, we propose an $O(n \log n \log \log n)$ -time algorithm for the broadcasting problem with uncertainty in heterogeneous tree networks following the postal model.

Keywords: algorithm, heterogeneous tree network, broadcasting, minmax-regret, uncertainty.

1 Introduction

A heterogeneous network connects computers (and other devices) together that use different operating systems and/or protocols. It is convenient to represent a heterogeneous network with a graph G , where the set of vertices, denoted by $V(G)$, represents the set of computers, and the set of edges, denoted by $E(G)$, represents the set of communication links. Associated with each edge $(u, v) \in E(G)$, there is a non-negative weight, denoted by $w_{u,v}$, whose value represents the number of time units required for transmitting a message from computer u to computer v over the link (u, v) .

The postal model [4] and telephone model [9, 12, 14] were used to characterize the message-passing systems that used the packet-switching technique and circuit-switching technique, respectively. Suppose that a message is to be transmitted from a sender u to a receiver v , where $(u, v) \in E(G)$. The postal model asks u first to connect with v and then to transmit the message to v . Let t_c and t_r be the numbers of time units spent on the connection and transmission, respectively. It was assumed that $t_c = 1$ and $t_r = w_{u,v}$.

* This work is partially supported by the National Science Council under the Grants No. NSC98-2221-E-019-029, and NSC99-2221-E-019-022-MY2.

Contrary to the simultaneous connections to u under the postal model, the telephone model allows at most one connection to each computer at a time. That is, if u is connected to v , it cannot be connected to another computer unless u and v are disconnected. Under the telephone model, only the connection time is considered, i.e., $t_c > 0$; the transmission time is negligible, i.e., $t_r = 0$.

Recently, the broadcasting problem [4, 9, 12, 14] has received much attention, due to the prompt advances of multimedial and network technologies. Given a heterogeneous network G , the broadcasting problem is to find a vertex $\kappa \in V(G)$ from which broadcasting messages to the other vertices of G requires minimum time. The vertex κ is called a *broadcast center* of G . The broadcasting problem considered the value of each $w_{u,v}$ deterministic. However, the information-carrying capacity of a communication link may vary with time. We suppose that the value of each $w_{u,v}$ is randomly generated from $[w_{u,v}^-, w_{u,v}^+]$, where $w_{u,v}^-$ and $w_{u,v}^+$ are two non-negative real numbers. Thus, the broadcasting problem with edge weight uncertainty results, as a consequence of the nondeterministic value of $w_{u,v}$.

Previously, the concept of uncertainty was introduced into some optimization problems [1, 3, 11]. There was a widely used approach, named *minmax regret* [1–3, 11], for solving them (i.e., optimization problems with edge (or vertex) weight uncertainty). Given an optimization problem with edge (or vertex) weight uncertainty, the minmax regret approach aims to minimize the worst-case loss in the objective function, without specifying the probability distribution of edge (or vertex) weights. A comprehensive discussion on the motives and applications of the minmax regret approach can be found in [11, 13].

When G is a general graph, the broadcasting problem, either under the postal model or under the telephone model, is NP-hard [5]. However, it requires polynomial time to find a broadcast center, if G is a tree [10, 14, 15]. In [14] ([10]), an $O(n)$ time ($O(n \log n)$ time) algorithm was proposed under the telephone model with uniform (nonuniform) connection time, where n is the number of vertices. In [15], an $O(n)$ time algorithm was proposed under the postal model. In this paper, considering the broadcasting problem with edge weight uncertainty under the postal model, we adopt the minmax regret approach for finding a minmax-regret broadcast center on a tree and an $O(n \log n \log \log n)$ time algorithm is proposed.

The rest of this paper is organized as follows. In Section 2, some notations and definitions, together with some useful properties, are introduced. In Section 3, an $O(n \log n \log \log n)$ time algorithm for finding a minmax-regret broadcast center on a tree is presented. In Section 4, the correctness proofs of the algorithms are provided. The time complexity is then analyzed in Section 5. Finally, in Section 6, this paper concludes with some further research directions.

2 Preliminaries

Let T be a tree and $w_{u,v}$ be the weight of $(u, v) \in E(T)$. We suppose that $n = |V(T)|$ and the value of $w_{u,v}$ is randomly generated from $[w_{u,v}^-, w_{u,v}^+]$, where

$w_{u,v}^-$ and $w_{u,v}^+$ are two non-negative real numbers. Define C to be a Cartesian product of all $n - 1$ intervals $[w_{u,v}^-, w_{u,v}^+]$ for T . A feasible weight assignment $s \in C$ of all edges in $E(T)$ is called a *scenario* of T . Define $w_{u,v}^s$ to be the weight of $(u, v) \in E(T)$, $b_time^s(u, G)$ to be the minimum time required for u to broadcast a message to all other vertices of G , and B_Ctr^s to be the set of broadcast centers of T , all under the scenario s .

Further, for any $x, y \in V(T)$, we define $r_{x,y}^s = b_time^s(x, T) - b_time^s(y, T)$ and $\max_r(x) = \max\{r_{x,y}^s \mid y \in V(T) \text{ and } s \in C\}$, where $r_{x,y}^s$ is called the *relative regret* of x with respect to y and $\max_r(x)$ is called the *maximum regret* of x . Intuitively, $r_{x,y}^s$ represents the time lost, if x was chose, instead of y , to broadcast a message over T under the scenario s . Notice that if $\max_r(x) = r_{x,y'}^{s'}$ for some $y' \in V(T)$ and $s' \in C$, then $y' \in B_Ctr^{s'}$, and s' is called a *worst-case scenario* of T with respect to x . The *minmax-regret broadcasting problem on T with edge weight uncertainty* is to find a *minmax-regret broadcast center* $\kappa^* \in V(T)$ such that its maximum regret is minimized, i.e., $\max_r(\kappa^*) \leq \max_r(v)$ for all $v \in V(T)$.

Suppose $x, y \in V(T)$. Removing x from T will result in some subtrees of T . We use $B_{x,y}$ to denote the subtree that contains y , and let $\bar{B}_{x,y} = T - B_{x,y}$.

Lemma 1 ([15]). *Suppose $s \in C$ and $(u, v) \in E(T)$. If $b_time^s(u, \bar{B}_{u,v}) \leq b_time^s(v, \bar{B}_{v,u})$, then the following hold:*

- $b_time^s(u, T) = 1 + w_{u,v}^s + b_time^s(v, \bar{B}_{v,u})$;
- $b_time^s(v, T) \leq b_time^s(u, T)$.

Lemma 2 ([15]). *The subgraph of T induced by B_Ctr^s is a star.*

Define $N_T(u) = \{v \mid (u, v) \in E(T)\}$ to be the set of neighboring vertices of u in T . A vertex $\hat{\kappa} \in B_Ctr^s$ is a *prime broadcast center* of T under the scenario s if and only if $b_time^s(\hat{\kappa}, \bar{B}_{\hat{\kappa},u}) \geq b_time^s(u, \bar{B}_{u,\hat{\kappa}})$ for all $u \in N_T(\hat{\kappa})$.

Lemma 3. *A prime broadcast center of T always exists under any scenario.*

Suppose $x, y \in V(T)$. Let $P_{x,y}$ be the set of edges contained in the x -to- y path of T , $d_{x,y} = |P_{x,y}|$, and $\tilde{w}_{x,y}^s$ be the total weight of all edges in $P_{x,y}$ under the scenario s , i.e. $\tilde{w}_{x,y}^s = \sum_{(u,v) \in P_{x,y}} w_{u,v}^s$. The following lemma shows a relation between $b_time^s(x, T)$ and $b_time^s(\hat{\kappa}, \bar{B}_{\hat{\kappa},x})$, where $\hat{\kappa}$ is a prime broadcast center of T under the scenario s .

Lemma 4. *Suppose that $s \in C$ and $\hat{\kappa}$ is a prime broadcast center of T under s . If $x \in V(T) - \{\hat{\kappa}\}$, then $b_time^s(x, T) = d_{x,\hat{\kappa}} + \tilde{w}_{x,\hat{\kappa}}^s + b_time^s(\hat{\kappa}, \bar{B}_{\hat{\kappa},x})$.*

Also notice that $d_{x,\hat{\kappa}} + \tilde{w}_{x,\hat{\kappa}}^s$ in Lemma 4 is the minimum time requirement for x to broadcast a message along the x -to- $\hat{\kappa}$ path. In general, for any $x, y \in V(T)$ and $x \neq y$, then we have the following lemma.

Lemma 5. *Suppose $s \in C$ and $x, y \in V(T)$. If $x \neq y$, then $b_time^s(x, T) \geq d_{x,y} + \tilde{w}_{x,y}^s + b_time^s(y, \bar{B}_{y,x})$.*

Lemma 6. Suppose $s \in C$. A prime broadcast center $\hat{\kappa}$ of T under s is a center of the star induced by B_Ctr^s .

3 Finding a Minmax-Regret Broadcast Center

In subsequent discussion, we use $\ddot{s}(u)$ to denote some worst-case scenario of T with respect to u . It was shown in [15] that given any $x \in V(T)$ and $s \in C$, both B_Ctr^s and $b_time^s(x, T)$ can be determined in $O(n)$ time. Suppose $Q(n)$ is the time requirement for finding $\ddot{s}(u)$, then inspired with the following lemma, we describe an $O(n \log n + Q(n) \cdot \log n)$ time algorithm below.

Lemma 7. Suppose $x \in V(T)$. If $x \in B_Ctr^{\ddot{s}(x)}$, then x is a minmax-regret broadcast center of T . Otherwise, a minmax-regret broadcast center of T can be found in $V(B_{x,\kappa}) \cup \{x\}$, where $\kappa \in B_Ctr^{\ddot{s}(x)}$.

With Lemma 7, a minmax-regret broadcast center of T can be found by the prune-and-search strategy. The search starts on T . A centroid z is first determined such that the largest open z -branch has minimum size. If $z \in B_Ctr^{\ddot{s}(z)}$, then z is a minmax-regret broadcast center of T and the search terminates. Otherwise, a broadcast center $\kappa \in B_Ctr^{\ddot{s}(z)}$ of T under $\ddot{s}(z)$ is found, and the search continues on the subtree of T induced by $V(B_{z,\kappa}) \cup \{z\}$. The search will proceed until z is a minmax-regret broadcast center of T or the size of the induced subtree is small enough. The detailed execution is expressed as Algorithm 1.

Algorithm 1. Determining a minmax-regret broadcast center of a tree

Input: a tree T .

Output: a minmax-regret broadcast center of T .

```

1:  $T' \leftarrow T$ ;
2: while  $|V(T')| \geq 3$  do
3:   determine a centroid  $z$  of  $T'$ ;
4:   find  $\ddot{s}(z)$  and  $B\_Ctr^{\ddot{s}(z)}$ ;
5:   if  $z \in B\_Ctr^{\ddot{s}(z)}$  then
6:     return  $z$ ;
7:   else
8:     find  $\kappa \in B\_Ctr^{\ddot{s}(z)}$ ;
9:      $T' \leftarrow$  the subtree of  $T'$  induced by  $V(T') \cap \{V(B_{z,\kappa}) \cup \{z\}\}$ ;
10:  end if
11: end while
12: determine  $v \in V(T')$  whose  $max\_r(v)$  is minimum;
13: return  $v$ .

```

It was shown in [8] that a centroid z of T can be found in $O(n)$ time. Besides, the largest open z -branch contains at most $\lfloor \frac{n}{2} \rfloor$ vertices. However, it is not easy to find $\ddot{s}(z)$, because there are an infinite number of scenarios. Suppose $V(T) = \{v_1, v_2, \dots, v_n\}$. For $x \in V(T)$, we would like to construct a finite set

$\ddot{S}(x) = \bigcup_{1 \leq \ell \leq n} \Psi_\ell(x)$ of scenarios such that $\Psi_\ell(x)$ contains a worst-case scenario of T with respect to x , i.e., $\ddot{s}(x) \in \Psi_\ell(x)$, if v_ℓ is a prime broadcast center of T under some worst-case scenario of T with respect to x . Then, we have $\ddot{s}(x) \in \ddot{S}(x)$ according to Lemma 3. The construction of $\ddot{S}(x)$ is described below.

Without losing generality, we assume $x = v_n$. We set $\Psi_n(x) = \{c\}$, where $c \in C$ is an arbitrary scenario of T . Define α_{x,v_i} to be the *base scenario* of T , where $1 \leq i \leq n-1$, which satisfies the following:

- $w_{a,b}^{\alpha_{x,v_i}} = w_{a,b}^+$, if $(a,b) \in P(x, v_i) \cup E(\bar{B}_{v_i, x})$;
- $w_{a,b}^{\alpha_{x,v_i}} = w_{a,b}^-$, else.

To construct $\Psi_i(x)$, we modify α_{x,v_i} to create new scenarios. The edge weights under α_{x,v_i} are to be changed according to a predefined sequence that is suggested by the following lemma.

Lemma 8. [15] Suppose $x \in V(T)$, $s \in C$, and $N_T(x) = \{u_1, u_2, \dots, u_h\}$, where $h = |N_T(x)|$. If $w_{x,u_k}^s + b_time^s(u_k, \bar{B}_{u_k, x})$ is nonincreasing as k increases from 1 to h , then u_1, u_2, \dots, u_h is an optimal sequence (with respect to minimum broadcast time) for x to broadcast a message to $N_T(x)$ under s . That is, $b_time^s(x, T) = \max\{k + w_{x,u_k}^s + b_time^s(u_k, \bar{B}_{u_k, x}) \mid 1 \leq k \leq h\}$.

Let $h_i = |\bar{N}_{\bar{B}_{v_i, x}}(v_i)|$. If $h_i = 0$, then we set $\Psi_i(x) = \emptyset$. Otherwise, the vertices of $\bar{N}_{\bar{B}_{v_i, x}}(v_i)$ are arranged as $u_{i,1}, u_{i,2}, \dots, u_{i,h_i}$ so that $w_{v_i, u_{i,k}}^{\alpha_{x,v_i}} + b_time^{\alpha_{x,v_i}}(u_{i,k}, \bar{B}_{u_{i,k}, v_i})$ is nonincreasing as k increases from 1 to h_i . Then, we set $\Psi_i(x) = \{\beta_{x,v_i}^1, \beta_{x,v_i}^2, \dots, \beta_{x,v_i}^{h_i}\}$, where each β_{x,v_i}^j , $1 \leq j \leq h_i$, is a scenario of T that satisfies the following:

- $w_{a,b}^{\beta_{x,v_i}^j} = w_{a,b}^-$, if $(a,b) \in \bigcup_{j < k \leq h_i} (\{(v_i, u_{i,k})\} + E(\bar{B}_{u_{i,k}, v_i}))$;
- $w_{a,b}^{\beta_{x,v_i}^j} = w_{a,b}^{\alpha_{x,v_i}}$, else.

After $\ddot{S}(x)$ ($= \bigcup_{1 \leq \ell \leq n} \Psi_\ell(x)$) is constructed, $\ddot{s}(x)$ can be determined from $\ddot{S}(x)$, as a consequence that $\max\{r_{x,y}^s \mid s \in \ddot{S}(x) \text{ and } y \in B_Ctr^s\}$ occurs when $s = \ddot{s}(x)$. The detailed execution for finding $\ddot{s}(x)$ is expressed as Algorithm 2.

The main result of this paper is stated as the following theorem.

Theorem 1. A minmax-regret broadcast center on a tree of n vertices with edge weight uncertainty can be found in $O(n \log n \log \log n)$ time.

4 Correctness

Clearly, with Lemma 7, a minmax-regret broadcast center κ^* of T can be found by Algorithm 1, provided $\ddot{s}(x)$ can be determined for any $x \in V(T)$. So, it suffices to determine $\ddot{s}(x)$, i.e., to verify Algorithm 2, in order to prove the correctness of Algorithm 1. Recall that we have $\Psi_n(x) = \{c\}$, where $c \in C$. If x is a prime broadcast center of T under $\ddot{s}(x)$, then we have $\max_r(x) = 0$. Since $r_{x,y}^c \geq 0 = \max_r(x)$ for any $y \in B_Ctr^c$, we have $c = \ddot{s}(x) \in \Psi_n(x)$. In the rest

Algorithm 2. Finding $\ddot{s}(x)$

Input: a tree T with vertices v_1, v_2, \dots, v_n and a vertex $x (= v_n)$ of T .
Output: $\ddot{s}(x)$.

```

1: set  $\Psi_n(x) = \{c\}$ , where  $c$  is an arbitrary scenario of  $T$ ;
2: for each  $v_i \in V(T) - \{x\}$  do
3:   construct  $\alpha_{x,v_i}$ ;
4:    $h_i \leftarrow |N_{\bar{B}_{v_i,x}}(v_i)|$ ;
5:   if  $h_i = 0$  then
6:     set  $\Psi_i(x) = \emptyset$ ;
7:   else
8:     arrange the vertices of  $N_{\bar{B}_{v_i,x}}(v_i)$  as  $u_{i,1}, u_{i,2}, \dots, u_{i,h_i}$  so that  $w_{v_i,u_{i,k}}^{\alpha_{x,v_i}} +$ 
       $b\_time^{\alpha_{x,v_i}}(u_{i,k}, \bar{B}_{u_{i,k},v_i})$  is nonincreasing as  $k$  increases from 1 to  $h_i$ ;
9:     construct  $\beta_{x,v_i}^1, \beta_{x,v_i}^2, \dots, \beta_{x,v_i}^{h_i}$ ;
10:    set  $\Psi_i(x) = \{\beta_{x,v_i}^1, \beta_{x,v_i}^2, \dots, \beta_{x,v_i}^{h_i}\}$ ;
11:   end if
12: end for
13: determine  $s' \in \bigcup_{1 \leq \ell \leq n} \Psi_\ell(x)$  so that  $r_{x,y'}^{s'} = \max\{r_{x,y}^s \mid s \in \bigcup_{1 \leq \ell \leq n} \Psi_\ell(x)$  and
     $y \in B\_Ctr^s\}$  for some  $y' \in B\_Ctr^{s'}$ ;
14: return  $s'$ .

```

of this section, we assume that x is not a prime broadcast center of T under any worst-case scenario of T with respect to x (which implies $|V(T)| \geq 3$). Besides, unless specified particularly, whenever a worst-case scenario c is mentioned, it means that c is a worst-case scenario of T with respect to x .

Suppose that v_ℓ is a prime broadcast center of T under $\ddot{s}(x)$, where $\ell \neq n$. Clearly, v_ℓ is not a leaf node of T . We attempt to identify a worst-case scenario in $\Psi_\ell(x)$, by means of performing a series of scenario transformations, starting with an arbitrary worst-case scenario, denoted by $\ddot{s}(x)$. First, we transform $\ddot{s}(x)$ into another worst-case scenario by specifying some edge weights. This transformation can be realized by the following fact (let $y = v_\ell$).

Fact 1. Suppose $x, y \in V(T)$ and y is a prime broadcast center of T under $\ddot{s}(x)$. Let s be a scenario of T that satisfies $w_{a,b}^s = w_{a,b}^{\alpha_{x,y}}$ if $(a,b) \in P_{x,y} \cup E(B_{y,x})$, and $w_{a,b}^s = w_{a,b}^{\ddot{s}(x)}$ else. Then, s is a worst-case scenario of T with respect to x . Besides, we have $y \in B_Ctr^s$.

According to Fact 1, $\ddot{s}(x)$ is transformed into s , which is also a worst-case scenario. The weights of edges $(a,b) \in P_{x,y} \cup E(B_{y,x})$ are specified under s . However, y is not necessarily a prime broadcast center under s . The following fact finds out a worst-case scenario and an associated prime broadcast center. To simplify notations, we use $\ddot{s}(x)$ and y to represent the resulting worst-case scenario and prime broadcast center, respectively. They may differ from those used in Fact 1.

Fact 2. Suppose $x \in V(T)$. There exist $\ddot{s}(x)$ and $y \in V(T)$ satisfying $w_{a,b}^{\ddot{s}(x)} = w_{a,b}^{\alpha_{x,y}}$ for each $(a,b) \in P_{x,y} \cup E(B_{y,x})$. Besides, y is a prime broadcast center of T under $\ddot{s}(x)$.

Finally, we transform $\ddot{s}(x)$ into another worst-case scenario by specifying the weights of the other edges, i.e., all edges $(a, b) \in E(\bar{B}_{y,x}) (= E(T) - \{P_{x,y} \cup E(B_{y,x})\})$. The resulting scenario is identical with some $\beta_{x,y}^j$ ($\in \Psi_\ell(x)$, as $y = v_\ell$), and thus Algorithm 2 is verified. The transformation, which depends on two cases, can be carried out by the following two facts, where μ is the neighbor of y in $B_{y,x}$, q_{0,τ_0} is a particular vertex in $N_{\bar{B}_{y,x}}(y)$, $h = |N_{\bar{B}_{y,x}}(y)|$, and $1 \leq t^* \leq h$.

Fact 3. If $w_{y,\mu}^{\ddot{s}(x)} + b_time^{\ddot{s}(x)}(\mu, B_{y,x}) \geq w_{y,q_{0,\tau_0}}^{\ddot{s}(x)} + b_time^{\ddot{s}(x)}(q_{0,\tau_0}, \bar{B}_{q_{0,\tau_0},y})$, then $\beta_{x,y}^h$ is a worst-case scenario of T with respect to x .

Fact 4. If $w_{y,\mu}^{\ddot{s}(x)} + b_time^{\ddot{s}(x)}(\mu, B_{y,x}) < w_{y,q_{0,\tau_0}}^{\ddot{s}(x)} + b_time^{\ddot{s}(x)}(q_{0,\tau_0}, \bar{B}_{q_{0,\tau_0},y})$, then $\beta_{x,y}^{t^*}$ is a worst-case scenario of T with respect to x .

5 Time Complexity

In this section, we show that the time complexity of Algorithm 2 is $O(n \log \log n)$, provided an $O(n \log n)$ time preprocessing is made. Consequently, Algorithm 1 requires $O(n \log n) + O(n \log \log n) \cdot O(\log n) = O(n \log n \log \log n)$ time. We assume below that x is not a prime broadcast center of T under $\ddot{s}(x)$, i.e., $\ddot{s}(x) \in \Psi_1(x) \cup \Psi_2(x) \cup \dots \cup \Psi_{n-1}(x)$ (refer to the first paragraph of Section 4). The following lemma is useful to find $\ddot{s}(x)$.

Lemma 9. Suppose $x \in V(T)$. If $d_{x,y} + \tilde{w}_{x,y}^{\beta_{x,y}^j} + b_time^{\beta_{x,y}^j}(y, \bar{B}_{y,x}) - b_time^{\beta_{x,y}^j}(y, T)$ has maximum value as $y = y'$ and $j = j'$, then we have $\beta_{x,y'}^{j'} = \ddot{s}(x)$.

With Lemma 9, it suffices to find a scenario β_{x,v_i}^j that can maximize the value of $d_{x,v_i} + \tilde{w}_{x,v_i}^{\beta_{x,v_i}^j} + b_time^{\beta_{x,v_i}^j}(v_i, \bar{B}_{v_i,x}) - b_time^{\beta_{x,v_i}^j}(v_i, T)$, in order to determine $\ddot{s}(x)$. By a depth-first traversal of T , d_{x,v_i} and $\tilde{w}_{x,v_i}^{\alpha_{x,v_i}}$ ($= \tilde{w}_{x,v_i}^{\beta_{x,v_i}^j}$ for all $1 \leq j \leq h_i$) for all $1 \leq i < n$ can be determined in $O(n)$ time. On the other hand, as shown below, $b_time^{\beta_{x,v_i}^j}(v_i, \bar{B}_{v_i,x})$ and $b_time^{\beta_{x,v_i}^j}(v_i, T)$ for all $1 \leq i < n$ and $1 \leq j \leq h_i$ can be determined in $O(n \log \log n)$ time. Therefore, $\ddot{s}(x)$ can be determined in $O(n \log \log n)$ time.

We suppose $N_{\bar{B}_{v_i,x}}(v_i) = \{u_{i,1}, u_{i,2}, \dots, u_{i,h_i}\}$ in the rest of this section. Then, we have the following fact.

Fact 5. With an $O(n \log n)$ -time preprocessing, $b_time^{\beta_{x,v_i}^j}(u_{i,k}, \bar{B}_{u_{i,k},v_i})$ can be determined in constant time, where $1 \leq j \leq h_i$ and $1 \leq k \leq h_i$, and a vertex ordering of $N_{\bar{B}_{v_i,x}}(v_i)$ can be determined in $O(h_i)$ time such that $w_{v_i,u_{i,k}}^{\beta_{x,v_i}^j} + b_time^{\beta_{x,v_i}^j}(u_{i,k}, \bar{B}_{u_{i,k},v_i})$ is nonincreasing as k increases from 1 to h_i .

Suppose $s \in C$, and let $bucket[1], bucket[2], \dots, bucket[h_i]$ be h_i buckets. Like [15], we insert $u_{i,k}$ into $bucket[\ell]$, if $\ell - 1 \leq (w_{v_i,u_{i,1}}^s + b_time^s(u_{i,1}, \bar{B}_{u_{i,1},v_i})) - (w_{v_i,u_{i,k}}^s + b_time^s(u_{i,k}, \bar{B}_{u_{i,k},v_i})) < \ell$, where $1 \leq k \leq h_i$ and $1 \leq \ell \leq h_i$. Define

$acc(\ell) = \sum_{1 \leq r \leq \ell} |bucket[r]|$ and $min_v(\ell) = \min\{w_{v_i, u_{i,k}}^s + b_time^s(u_{i,k}, \bar{B}_{u_{i,k}, v_i}) \mid u_{i,k} \in bucket[\ell]\}$. Then, we have $b_time^s(v_i, \bar{B}_{v_i, x}) = \max\{min_v(\ell) + acc(\ell) \mid 1 \leq \ell \leq h_i\}$, which can be computed in $O(h_i)$ time.

It follows that $b_time^{\beta_{x,v_i}^j}(v_i, \bar{B}_{v_i, x})$ for all $1 \leq j \leq h_i$ can be determined in additional $O(h_i^2)$ time. As elaborated below, the time complexity can be reduced to $O(h_i \log \log h_i)$ (notice that $\sum_{1 \leq i < n} O(h_i \log \log h_i) = O(n \log \log n)$). Suppose $u_{i,j} \in bucket[\tau_j]$ under β_{x,v_i}^j . Let $pred(j)$ ($succ(j)$) be a list of non-empty buckets such that $bucket[\ell]$ is included in $pred(j)$ ($succ(j)$) if and only if $bucket[\ell]$ is not empty, $\ell < \tau_j$ ($\ell > \tau_j$), and $min_v(\ell) + acc(\ell) \geq min_v(t) + acc(t)$ for all $1 \leq t \leq \ell$ ($\ell \leq t \leq h_i$). Assume $pred(j) = (bucket[\pi_{j,1}], bucket[\pi_{j,2}], \dots, bucket[\pi_{j,p_j}])$ and $succ(j) = (bucket[\sigma_{j,1}], bucket[\sigma_{j,2}], \dots, bucket[\sigma_{j,q_j}])$, where $\pi_{j,1} < \pi_{j,2} < \dots < \pi_{j,p_j}$ and $\sigma_{j,1} < \sigma_{j,2} < \dots < \sigma_{j,q_j}$. Then, we have $b_time^{\beta_{x,v_i}^j}(v_i, \bar{B}_{v_i, x}) = \max\{min_v(\pi_{j,p_j}) + acc(\pi_{j,p_j}), min_v(\tau_j) + acc(\tau_j), min_v(\sigma_{j,1}) + acc(\sigma_{j,1})\}$. Hence, it suffices to find the values of $min_v(\ell) + acc(\ell)$ for all $\ell \in \{\pi_{j,p_j}, \tau_j, \sigma_{j,1}\}$, in order to determine $b_time^{\beta_{x,v_i}^j}(v_i, \bar{B}_{v_i, x})$.

First, with Fact 5, additional $O(h_i)$ time is enough to find the buckets (and hence τ_{h_i}) where $u_{i,1}, u_{i,2}, \dots, u_{i,h_i}$ reside and determine the values of $min_v(\ell)$ and $acc(\ell)$ for all $1 \leq \ell \leq h_i$, under $\beta_{x,v_i}^{h_i}$. Another $O(h_i)$ time is required to find $pred(h_i)$ and $succ(h_i)$ (and hence $\pi_{h_i, p_{h_i}}$ and $\sigma_{h_i, 1}$). As explained below, it requires $O(h_i \log \log h_i)$ time to determine $b_time^{\beta_{x,v_i}^j}(v_i, \bar{B}_{v_i, x})$, sequentially, for $j = h_i - 1, h_i - 2, \dots, 1$.

Suppose $u_{i,j+1} \in bucket[\ell]$ under β_{x,v_i}^j , which can be determined in constant time (as a consequence of Fact 5).

Lemma 10. β_{x,v_i}^j and β_{x,v_i}^{j+1} induce the same $bucket[\ell]$ for all $\ell \in \{1, 2, \dots, h_i\} - \{\tau_{j+1}, \hat{\ell}\}$.

Lemma 10 implies the following results. First, $u_{i,j}$ resides in the same bucket under $\beta_{x,v_i}^j, \beta_{x,v_i}^{j+1}, \dots, \beta_{x,v_i}^{h_i}$ and hence τ_j can be determined in constant time. Second, it takes constant time to obtain $min_v(1), min_v(2), \dots, min_v(h_i), |bucket[1]|, |bucket[2]|, \dots, |bucket[h_i]|$ under β_{x,v_i}^j from $min_v(1), min_v(2), \dots, min_v(h_i), |bucket[1]|, |bucket[2]|, \dots, |bucket[h_i]|$ under β_{x,v_i}^{j+1} . Third, we have $\tau_j \leq \tau_{j+1} \leq \hat{\ell}$, implying β_{x,v_i}^j and β_{x,v_i}^{j+1} induce the same $bucket[\ell]$ for all $1 \leq \ell \leq \tau_j - 1$, i.e., the same $acc(\tau_j - 1)$. It follows that $acc(\tau_j)$ can be determined in constant time. Therefore, it takes $O(h_i)$ time to determine the values of $min_v(\tau_j) + acc(\tau_j)$ under β_{x,v_i}^j for all $1 \leq j \leq h_i - 1$.

Also, as a consequence of $\tau_j \leq \tau_{j+1}$ and $\beta_{x,v_i}^j, \beta_{x,v_i}^{j+1}$ inducing the same $bucket[\ell]$ for all $1 \leq \ell \leq \tau_j - 1$, $pred(j)$ is a sublist of $pred(j+1)$ for all $1 \leq j \leq h_i - 1$. Since $\pi_{h_i, p_{h_i}}$ is available, $O(h_i)$ time is enough to determine π_{j,p_j} for all $1 \leq j \leq h_i - 1$. Therefore, it takes $O(h_i)$ time to determine the values of $min_v(\pi_{j,p_j}) + acc(\pi_{j,p_j})$ under β_{x,v_i}^j for all $1 \leq j \leq h_i - 1$.

To determine the values of $min_v(\sigma_{j,1}) + acc(\sigma_{j,1})$ under β_{x,v_i}^j for all $1 \leq j \leq h_i - 1$, we claim that $succ(j) = (bucket[\sigma_{j,1}], bucket[\sigma_{j,2}], \dots, bucket[\sigma_{j,q_j}])$ for all $1 \leq j < h_i$ can be constructed in $O(h_i \log \log h_i)$ time, by the aid of the van Emde Boas priority queue [6]. Since the values of $min_v(\ell)$ under β_{x,v_i}^j for all

$1 \leq \ell \leq h_i$ and $1 \leq j < h_i$ can be determined in $O(h_i)$ time, we show in the next paragraph that the values of $acc(\sigma_{j,1})$ under β_{x,v_i}^j for all $1 \leq j < h_i$ can be determined in additional $O(h_i)$ time.

Define $\Delta_{j,t} = (acc(\sigma_{j,t}) - acc(\sigma_{j,t-1}))$ under β_{x,v_i}^j – $(acc(\sigma_{j,t}) - acc(\sigma_{j,t-1}))$ under $\beta_{x,v_i}^{h_i}$, where $1 \leq t \leq q_j$ and $\sigma_{j,0}$ is set to τ_j . The values of $\Delta_{j,t}$ for all $1 \leq t \leq q_j$ can be obtained, while constructing $succ(j)$. When $t = 1$, we have

$$\Delta_{j,1} = (acc(\sigma_{j,1}) - acc(\tau_j)) \text{ under } \beta_{x,v_i}^j - (acc(\sigma_{j,1}) - acc(\tau_j)) \text{ under } \beta_{x,v_i}^{h_i}. \quad (1)$$

Now that the values of $acc(\tau_j)$ under β_{x,v_i}^j for all $1 \leq j < h_i$ and $acc(\ell)$ under $\beta_{x,v_i}^{h_i}$ for all $1 \leq \ell \leq h_i$ can be determined in $O(h_i)$ time, the values of $acc(\sigma_{j,1})$ under β_{x,v_i}^j for all $1 \leq j < h_i$ can be determined in additional $O(h_i)$ time according to (II).

6 Conclusion

In this paper, we considered the broadcasting problem on the heterogeneous network with edge weight uncertainty, i.e., the weight $w_{u,v}$ of each edge (u, v) is randomly generated from $[w_{u,v}^-, w_{u,v}^+]$. Our problem generalizes the classical broadcasting problem, in which each edge weight is deterministic. Given a heterogeneous tree T with edge weight uncertainty, we first proved that a worst-case scenario of T exists within a finite subset of scenarios of T . Then, an $O(n \log n \log \log n)$ -time algorithm was proposed to find a minmax-regret broadcast center of T from these scenarios.

The broadcasting problem we considered was inspired from the traditional 1-center problem. If the connection time in the broadcasting problem is ignored, then the two problems are equivalent. While searching for an optimal solution to the broadcasting problem, an optimal broadcast sequence of the neighbors of the starting vertex must be determined, as a consequence of the connection time. The difficulty of the broadcasting problem mainly arises from the edge weight uncertainty. Thus far, the most efficient algorithm [5] for the minmax-regret 1-center problem on trees requires $O(n \log n)$ time. The proposed algorithm runs in $O(n \log n \log \log n)$ time. That is, a multiplier of $O(\log \log n)$ is incurred for the parameter of connection time.

Below we present some open problems related to the broadcasting problems on the heterogeneous network. As discussed above, it would be of interest to improve the time complexity for solving the minmax-regret broadcasting problem to $O(n \log n)$ time, or to prove the lower bound is $O(n \log n \log \log n)$. Furthermore, we can consider the broadcasting problem with uncertain vertex weights, which represent the demands of the vertices. It would also be interesting to consider the p -center and p -median broadcasting problems, and the bi-criteria broadcasting problems such as minimizing the sum of communication times from the center to all vertices in T with bounded broadcasting time. Meanwhile, since the broadcasting problem is NP-hard on general graphs, one can try to design polynomial-time approximation algorithms for general graphs.

References

1. Averbakh, I., Berman, O.: Minimax regret p-center location on a network with demand uncertainty. *Location Science* 5(4), 247–254 (1997)
2. Averbakh, I., Berman, O.: Algorithms for the robust 1-center problem on a tree. *European J. Oper. Res.* 123(2), 292–302 (2000)
3. Averbakh, I., Berman, O.: Minmax regret median location on a network under uncertainty. *INFORMS J. Comput.* 12(2), 104–110 (2000)
4. Bar-Noy, A., Kipnis, S.: Designing broadcasting algorithms in the postal model for message-passing systems. *Math. Systems Theory* 27(5), 431–452 (1994)
5. Burkard, R.E., Dollani, H.: A note on the robust 1-center problem on trees. *Ann. Oper. Res.* 110, 69–82 (2002)
6. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. *Theory Comput. Syst.* 10, 99–127 (1976)
7. Garey, M.R., Johnson, D.S.: Approximation algorithms for combinatorial problems: an annotated bibliography. In: *Algorithms and Complexity: New Directions and Recent Results*, pp. 41–52. Academic Press, New York (1976)
8. Goldman, A.J.: Optimal center location in simple networks. *Transportation Science* 5, 212–221 (1971)
9. Khuller, S., Kim, Y.A., Wan, Y.C.: On generalized gossiping and broadcasting. *J. Algorithms* 59(2), 81–106 (2006)
10. Koh, J.m., Tcha, D.w.: Information dissemination in trees with nonuniform edge transmission times. *IEEE Trans. Comput.* 40(10), 1174–1177 (1991)
11. Kouvelis, P., Yu, G.: Robust discrete optimization and its applications. In: *Non-convex Optimization and its Applications*, vol. 14, pp. xvi+356. Kluwer Academic Publishers, Dordrecht (1997)
12. Lee, H.-M., Chang, G.J.: Set to set broadcasting in communication networks. *Discrete Appl. Math.* 40(3), 411–421 (1992)
13. Nikulin, Y.: Robustness in combinatorial optimization and scheduling theory: An extended annotated bibliography (August 2006)
14. Slater, P.J., Cockayne, E.J., Hedetniemi, S.T.: Information dissemination in trees. *SIAM J. Comput.* 10(4), 692–701 (1981)
15. Su, Y.-H., Lin, C.-C., Lee, D.T.: Broadcasting in Heterogeneous Tree Networks. In: Thai, M.T., Sahni, S. (eds.) *COCOON 2010. LNCS*, vol. 6196, pp. 368–377. Springer, Heidelberg (2010)

Optimal File Distribution in Peer-to-Peer Networks

Kai-Simon Goetzmann^{1,*}, Tobias Harks¹, Max Klimm^{1,*}, and Konstantin Miller²

¹ Technische Universität Berlin, Institut für Mathematik

{goetzmann, harks, klimm}@math.tu-berlin.de

² Technische Universität Berlin, Telecommunication Networks Group

miller@tkn.tu-berlin.de

Abstract. We study the problem of distributing a file initially located at a server among a set of peers. Peers who downloaded the file can upload it to other peers. The server and the peers are connected to each other via a core network. The upload and download rates to and from the core are constrained by user and server specific upload and download capacities. Our objective is to minimize the makespan. We derive exact polynomial time algorithms for the case when upload and download capacities per peer and among peers are equal. We show that the problem becomes strongly NP-hard for equal upload and download capacities per peer that may differ among peers. For this case we devise a polynomial time $(1 + 2\sqrt{2})$ -approximation algorithm. To the best of our knowledge, neither NP-hardness nor approximation algorithms were known before for this problem.

1 Introduction

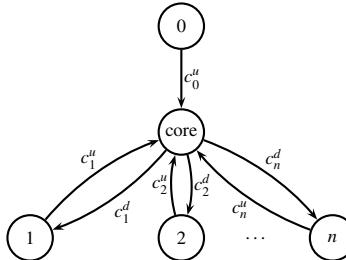
In the past decade, the concept of data distribution based on peer-to-peer overlay networks has become increasingly popular. A significant fraction of Internet traffic is nowadays generated by peer-to-peer file sharing applications [3]. The key principle underlying peer-to-peer file sharing is that peers who downloaded parts of the file (chunks), start to assist the server in uploading them to other peers. A chunk is the smallest indivisible unit w.r.t. the download source, that is, a peer cannot download parts of a chunk from different sources. It is assumed that the server and the peers are connected via a core network (Internet). The upload and download rates to and from the core are constrained by peer and server specific upload and download capacities. The core itself is usually overprovisioned, thus, there are no capacity constraints present.

While in the past many algorithms and protocols have been studied in the literature for the important problem of optimizing the download process [1,9,10], a systematic performance evaluation of the proposed solutions with respect to *optimal* solutions has not received similar attention. In this paper, we address the fundamental problem of minimizing the total completion time (makespan) of distributing a file among a set of peers in a peer-to-peer network. Due to the intrinsic difficulty of the problem, in this work we restrict ourselves to the case of a single chunk only. To the best of our knowledge, even for this restricted case there are no algorithms with provable performance

* Supported by the Deutsche Forschungsgemeinschaft within the research training group ‘Methods for Discrete Structures’ (GRK 1408).

guarantee known. While the case of multiple chunks still eludes us, we see our study as an important first step towards understanding the general peer-to-peer file distribution problem.

The Model. An instance of this problem is described by a tuple $I = (N, c^d, c^u)$, where $N = \{0, \dots, n\}$ is the set of peers, $c^d = (c_0^d, \dots, c_n^d) \in \mathbb{Q}_{\geq 0}^{n+1}$ is the vector of download capacities from the core network and $c^u = (c_0^u, \dots, c_n^u) \in \mathbb{Q}_{\geq 0}^{n+1}$ is the vector of upload capacities to the core network. We will identify the server with peer 0 and assume that only the server initially owns a file of unit size, which is not divided into several chunks. See Figure 1 for an illustration. A feasible solution $S = (s_{i,j})_{i,j \in N}$ is a family of integrable functions $s_{i,j} : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, $i, j \in N$, where $s_{i,j}(t)$ denotes the sending rate of peer i to peer j at time t . We require that every peer $i \in N$ receives its data from a unique peer, denoted by $p(i) \neq i$: $s_{k,i}(t) = 0$ for all $k \neq p(i)$ and all $t \in \mathbb{R}_{\geq 0}$. In addition, only peers that possess the file can send with a positive rate: $s_{i,j}(t) = 0$ for all $i, j \in N \setminus \{0\}$, $t \in \mathbb{R}_{\geq 0}$ with $\int_0^t \sum_{k \in N} s_{k,i}(\tau) d\tau < 1$. Finally, the sending rates have to obey download and upload capacity constraints: $\sum_{j \in N} s_{i,j}(t) \leq c_i^d$ for all $i \in N$, $t \in \mathbb{R}_{\geq 0}$ and $s_{p(j),j}(t) \leq c_j^u$ for all $j \in N$, $t \in \mathbb{R}_{\geq 0}$. We denote by $x_i(t) = \int_0^t s_{p(i),i}(\tau) d\tau$ the proportion of the file owned by peer $i \in N \setminus \{0\}$ at time t . For notational convenience, we set $x_0(t) = 1$ for all $t \in \mathbb{R}_{\geq 0}$. We denote by $C_i = \inf\{t \in \mathbb{R}_{\geq 0} : x_i(t) = 1\}$ the *completion time* of peer i . The makespan of a solution S is then defined as $M = \max_{i \in N \setminus \{0\}} C_i$. If an instance satisfies $c_i^u = c_i^d = c_j^u = c_j^d$ for all $i, j \in N \setminus \{0\}$ we speak of an instance with *homogeneous symmetric capacities*. If an instance satisfies $c_i^u = c_i^d$ for all $i \in N$ (but possibly $c_i^u \neq c_j^u$ for some $i, j \in N \setminus \{0\}$) we speak of *heterogeneous symmetric capacities*. In both cases, we only write $I = (N, c)$.



it takes one unit of time (round) to pass a message. For complete graphs on n nodes, this process terminates in $\lceil \log_2(n + 1) \rceil$ rounds. It is known that for arbitrary communication graphs, the problem of computing an optimal broadcast in the telephone model is NP-hard [5], even for 3-regular planar graphs [12].

Khuller et al. [7] study the problem of broadcasting in heterogeneous networks (see Bar-Noy et al. [2] for the corresponding multicast problem). They extend the telephone model by allowing the transmission time of a message to depend on the sender. They prove that it is NP-hard to minimize the makespan and present an approximation scheme.

Note that in contrast to both models, in our model a peer may upload the file to an arbitrary number of peers simultaneously provided the capacity constraints are satisfied. Moreover, the transfer time between any two peers is not given as part of the instance but determined by the sending rates of a feasible solution. In fact, the model by Khuller et al. [7] for broadcasting in heterogeneous networks reduces to a special case of our model. If $c_i^d \geq \max_{j \in N} c_j^u$ for all $i \in N$, there is always an optimal solution in which no peer will send the file to more than one other peer at a time, thus, the time to transfer the file from one peer to another only depends on the sender's upload capacity. In general, however, it may be beneficial to serve multiple peers simultaneously as illustrated in the following example. Peer 0 with capacity $c_0^u = 2$ initially owns the file, and there are two further peers with capacities $c_i^u = c_i^d = 1$, $i = 1, 2$. The optimal solution is $s_{0,1}(t) = s_{0,2}(t) = 1 \forall t \in [0, 1]$ with a makespan of $M^* = 1$. On the other hand, restricting peers to upload to at most one other peer at a time results in an optimal makespan of 2.

Mundinger et al. [13] studied a peer-to-peer file distribution problem, where a file is subdivided in multiple parts and the goal is to disseminate the complete file to every peer as fast as possible. The crucial difference to our model is that they assume that the upload capacity of a peer is equally shared among concurrent uploads. Under this fair sharing assumption they prove that for homogeneous capacities a simple greedy algorithm is optimal. For our model we prove a similar result (though for a single indivisible file) *without* the fair-sharing assumption. Our proof is quite involved since dropping the fair sharing assumption considerably complicates matters. For heterogeneous capacities, to the best of our knowledge, neither approximation algorithms nor hardness results were known before.

Also related are works on the so called fluid limit model, where the number of file parts tends to infinity [4,8,11,13,14].

Summary of the Results and Used Techniques. We first study the peer-to-peer file distribution problem for the case of homogeneous symmetric (unit) capacities. We show that a greedy algorithm computes an optimal solution with makespan $\lceil \log_2(n + 1) \rceil$. Although similar results for the same algorithm have been obtained before, e.g. by Mundinger et al. [13], or in the broadcast literature, our result holds for a more general model (dropping the fair sharing assumption of [13] and dropping the telephone model). For the case of unit peer capacities and an arbitrary integer server capacity, we propose a polynomial time algorithm (that possibly splits up server capacity) and prove its optimality. We also give a closed-form expression of the minimal makespan. If peer capacities are heterogeneous and symmetric we show that the peer-to-peer file distribution

problem becomes strongly NP-hard. A key ingredient of the reduction (from 3-partition) is the *capacity expansion lemma* (Lemma 3.1) that provides an upper bound on the total amount of data downloaded at any point in time. In light of the hardness, we then study approximation algorithms. We first devise a polynomial time $2\sqrt{2}$ -approximation algorithm for instances with heterogeneous, symmetric capacities in which the upload capacity of the server is larger than the download capacity of any other peer. For smaller server capacities a slight modification of our algorithm gives a $(1+2\sqrt{2})$ -approximation. Our algorithm proceeds in two phases; in a first phase, we use a time-varying resource augmentation to construct a so called $\sqrt{2}$ -augmented solution that violates the capacity constraints of the peers at any point in time by a factor of at most $\sqrt{2}$. By exploiting again our capacity expansion lemma, we prove that the makespan of the thus constructed augmented solution is at most a factor of 2 away from the optimal makespan of the original instance. We then rescale the relaxed solution to obtain a feasible solution with makespan less than a factor $2\sqrt{2}$ away from the optimal makespan.

2 Homogeneous Symmetric Capacities

In this section we consider the homogeneous symmetric setting, i.e. $c_i^u = c_i^d = c_i = 1$ for all $i \in N \setminus \{0\}$. The server has a capacity of $c_0^u = c$. For the simplest case, $c = 1$, we propose a GREEDY procedure: At each point in time, any peer that already owns the file uploads it to exactly one other peer, which takes one unit of time. Thus in each step, the number of peers owning the file is doubled, resulting in a makespan of $\lceil \log_2(n + 1) \rceil$.

Lemma 2.1. *If $c_i = 1$ for all $i \in N$, GREEDY computes an optimal solution. The optimal makespan is $\lceil \log_2(n + 1) \rceil$.*

To prove this, we consider an arbitrary solution and modify it in such a way that in the time interval $[0, 1)$ only one peer is served, without increasing the makespan. Iterating this argument for later points in time and other peers proves that there is an optimal solution with the GREEDY structure. More specifically, let peer 1 be a peer that finishes its download first in the given solution. We serve this peer in $[0, 1)$ and change the sending rates from the server to the other peers in such a way that the fraction of the file they own at time C_1 is the same as in the given solution. This can be done without violating the capacities, and obviously the makespan is not increased. A detailed proof will be included in the full version of the paper.

We now consider the case where c is an arbitrary integer, and start with a Lemma stating that in this case there always is an optimal solution that uses *fair-sharing*, i.e. whenever peer 0 serves several peers simultaneously, all of them are served with equal rate. The proof is quite involved, and the integrality condition on c is crucial. To see this, consider an example with $c = 3/2$ and $n = 4$. In an optimal solution, the server serves peer 1 in $[0, 1)$ and peer 2 in $[1, 2)$ with a rate of 1 each and peer 3 with a rate of $1/2$ in $[0, 2)$. Peer 4 is served by peer 1 in $[1, 2)$, resulting in a makespan of $M^* = 2$. The best solution that uses fair-sharing, however, has a makespan of $M = 7/3$: The server serves peers 1 and 2 in $[0, 4/3)$ with a rate of $3/4$ each, and peer 3 and 4 are both served at a rate of 1 in $[4/3, 7/3)$ by the server and peer 1.

Lemma 2.2. *For any instance $I = (N, (c_i)_{i \in N})$ with $c_0 = c \in \mathbb{N}$ and $c_i = 1 \forall i = 1, \dots, n$, there always exists an optimal solution that uses fair sharing. Also, the sending rates $s_{i,j}$ are piecewise constant with discontinuities only at points in time where some peer finishes its download.*

[Proof will be included in the full version of the paper.]

In the same setting, we also know that the server always serves groups of c peers jointly, except in the beginning, where a group of at least c and most $2c$ peers is served.

Lemma 2.3. *For any instance $I = (N, (c_i)_{i \in N})$ with $c_0 = c \in \mathbb{N}$ and $c_i = 1 \forall i = 1, \dots, n$, there always exists an optimal solution where only at the beginning more than c peers are served simultaneously.*

[Proof will be included in the full version of the paper.]

Summing up, we know so far that there always is an optimal solution where peer 0 at the beginning serves a set of peers N_1 , $c \leq |N_1| < 2c$, in $|N_1| / c$ time units, then some sets N_2, \dots, N_{k-1} with $|N_j| = c \forall j \geq 2$, in one time unit each, and finally a set N_k containing at most c peers, in another unit of time.

We are now ready to present an exact algorithm for this special case and prove its correctness. The algorithm is an extended GREEDY procedure: Set $h = \lceil \log_2(n/c + 1) \rceil$. If $n < c(2^h - 1 + 2^{h-1})$, let $|N_1| = \lceil (n - c(2^{h-1} - 1)) / 2^{h-1} \rceil$ and $k = h$, otherwise let $|N_1| = c$ and $k = h + 1$. Let $|N_j| = c$ for $j = 2, \dots, k$. The server serves N_1 in $[0, |N_1|/c)$ and N_j in $[|N_1|/c + j - 2, |N_1|/c + j - 2)$. Meanwhile, any peer that finishes its download starts serving other peers greedily.

Theorem 2.4. *For any instance $I = (N, (c_i)_{i \in N})$ with $c_0 = c \in \mathbb{N}$ and $c_i = 1 \forall i = 1, \dots, n$, the extended GREEDY yields an optimal solution. The optimal makespan is*

$$M^* = \begin{cases} h - 1 + \frac{1}{c} \left\lceil \frac{n - c(2^{h-1} - 1)}{2^{h-1}} \right\rceil & \text{if } n \in [c(2^h - 1), c(2^h - 1 + 2^{h-1})) \\ h + 1 & \text{if } n \in [c(2^h - 1 + 2^{h-1}), c(2^{h+1} - 1)), \end{cases}$$

where $h = \lceil \log_2(n/c + 1) \rceil$.

Proof. For $j = 1, \dots, k$, we denote by \tilde{N}_j the set of peers that are directly or indirectly served by a peer in N_j :

$$\tilde{N}_j = \{i \in N : p^\ell(i) = i' \text{ for some } i' \in N_j, \ell \in \mathbb{N}_0\}$$

Here $p^\ell(j)$ means that the function p is applied ℓ times to j . W.l.o.g. we can assume that in the solution produced by our algorithm $|\tilde{N}_j| = c \cdot 2^{k-j}$ for all $j = 2, \dots, k$, i.e. all sets \tilde{N}_j for $j \geq 2$ are as big as possible. If this is not the case we can move peers from \tilde{N}_1 to these sets. The lemmas above state that there also is an optimal solution with this structure and $c \leq |N_1| < 2c$. In the argumentation below, we therefore restrict to solutions of this structure.

The integer h is chosen such that $c(2^h - 1) \leq n < 2^{h+1} - 1$. We first consider the case where $n \geq c(2^h - 1 + 2^{h-1})$. Our algorithm sets $k = h + 1$ and $|N_1| = c$. The maximum number of peers that can be served like this is $c(2^{h+1} - 1) > n$, so indeed all peers are served within a makespan of $h + 1$. Assume that in an optimal solution

only $k = h$ sets are served by the server. The maximum number of peers served in this way is $|N_1| \cdot 2^{h-1} + c(2^{h-1} - 1) < c(2^h + 2^{h-1} - 1) \leq n$, so not all peers can be served, contradiction! Therefore in the optimal solution it must hold that $k = h + 1$ and $|N_1| \geq c$, proving that the solution produced by our algorithm is optimal.

If $n < c(2^h - 1 + 2^{h-1})$, the size of N_1 in an optimal solution (with $|\tilde{N}_1| = c \cdot 2^{k-j}$ for $j \geq 2$) has to be chosen such that $|\tilde{N}_1| = n - c(2^{h-1} - 1) \leq |N_1| \cdot 2^{h-1}$, i.e.

$$|N_1| = \min \left\{ \ell \in \mathbb{N} : \ell \geq \frac{n - c(2^{h-1} - 1)}{2^{h-1}} \right\} = \left\lceil \frac{n - c(2^{h-1} - 1)}{2^{h-1}} \right\rceil,$$

which is exactly how the algorithm chooses N_1 . It is obvious that choosing $k \leq h - 1$ does not lead to a solution where all peers are served, and choosing $k \geq h + 1$ leads to a solution with a makespan of at least $h + 1$. Since the solution produced by our algorithm has a makespan of $t_1 + k - 1 \leq h + 1$ this proves the correctness of our algorithm. \square

3 Heterogeneous Symmetric Capacities

In this section, we consider the case of heterogenous and symmetric capacities. First, we show that the peer-to-peer file distribution for heterogenous symmetric peers is strongly NP-hard. Then, we devise an algorithm that approximates the optimal makespan by a factor $1 + 2\sqrt{2}$. Our hardness and approximation results rely on a useful lemma that bounds the work done in any feasible solution. For a feasible solution, let $u_i(t_1, t_2)$ and $z_i(t_1, t_2)$ denote the total upload and the idleness of peer i in time interval $[t_1, t_2]$, defined as $u_i(t_1, t_2) = \int_{t_1}^{t_2} \sum_{j \in N} s_{i,j}(\tau) d\tau$ and $z_i(t_1, t_2) = \int_{\max\{t_1, C_i\}}^{t_2} (c_i - \sum_{j \in N} s_{i,j}(\tau)) d\tau$. For $t \geq 0$, we define $X(t) = \sum_{i \in N} x_i(t)$ and $Z(t) = \sum_{i \in N} z_i(0, t)$.

Lemma 3.1 (Capacity Expansion Lemma). *Let $I = (N, (c_i)_{i \in N})$ be an instance of the peer-to-peer file distribution problem with $c = \max_{i \in N} c_i$. Then, for all solutions S of I , the following two hold:*

1. $u_i(t_1, t_2) + z_i(t_1, t_2) \leq \max \left\{ 0, \left(t_2 - t_1 - \frac{1 - x_i(t_1)}{c_i} \right) c_i \right\} = \max \left\{ 0, (t_2 - t_1) c_i - 1 + x_i(t_1) \right\}$ for all peers $i \in N$ and all times $0 \leq t_1 < t_2$.
2. $X(k/c) + Z(k/c) \leq 2^k$ for all $k \in \mathbb{N}$. This inequality is strict if there is $i \in N$ with $c_i < c$ and $0 < C_i \leq k/c$.

Proof. To see 1., note that for any peer i with $C_i \leq t_1$, the upload rate (integrand of the total upload) and the idle rate (integrand of the idleness) sum up to c_i for all $t \in [t_1, t_2]$, thus $u_i(t_1, t_2) + z_i(t_1, t_2) \leq (t_2 - t_1) c_i$. If $x_i(t_1) < 1$, peer i needs at least $(1 - x_i(t_1))/c_i$ time units to finish its download, thus only a time interval of length $t_2 - t_1 - (1 - x_i(t_1))/c_i$ remains for the upload and the claimed inequality follows.

We prove 2. by induction over k . The inequality is trivial for $k = 0$. So, let us assume that for $k \in \mathbb{N}$, we have $X((k-1)/c) + Z((k-1)/c) \leq 2^{k-1}$. Using 1., we obtain the inequality

$$\begin{aligned} X(k/c) - X((k-1)/c) + Z(k/c) - Z((k-1)/c) &= \sum_{i \in N} u_i((k-1)/c, k/c) + z_i((k-1)/c, k/c) \\ &\leq \sum_{i \in N} \max \{ 0, c_i/c - 1 + x_i((k-1)/c) \} \leq X((k-1)/c), \end{aligned}$$

where we use $c_i \leq c$. Note that the latter inequality is satisfied strictly if there is a peer i with $C_i \leq k/c$ and $c_i < c$. Rearranging terms and using the induction hypothesis, we obtain $X(k/c) + Z(k/c) \leq 2X((k-1)/c) + Z((k-1)/c) \leq 2^k$, as claimed. \square

Hardness. We are now ready to prove strong NP-hardness of the peer-to-peer file distribution problem.

Theorem 3.2. *The peer-to-peer file distribution problem for heterogeneous symmetric peers is strongly NP-hard.*

We reduce from 3-PARTITION, where a multiset $P = \{k_1, \dots, k_n\}$ of $n = 3m$ numbers has to be partitioned into sets P_0, \dots, P_{m-1} such that $\sum_{k_i \in P_j} k_i = B$ for all $j = 0, \dots, m-1$. W.l.o.g. we can assume $m = 2^\ell$. The idea is to introduce m *subset peers* with capacity B and n *master element peers* with capacity k_i . If we have a yes-instance of 3-PARTITION, we can first serve all subset peers, and then each subset peer serves those master element peers that correspond to elements in one of the sets P_j with full capacity, resulting in a makespan of $\ell + \ell/B$. To show that for no-instances the makespan is strictly greater than $\ell + \ell/B$, we have to introduce $2\ell k_i - 2$ *element peers* for all $k_i \in P$. This is also the point where the capacity expansion lemma is used. The proof of the hardness result is quite involved and will be included in the full version of the paper.

Approximation. In this section, we devise an algorithm that runs in time $O(n \log n)$ and computes a solution with makespan no larger than $(1 + 2\sqrt{2})M^*$, where M^* is the optimal makespan. We first consider the case $c_0 \geq c_i$ for all $i = 1, \dots, n$, for which we will show a $2\sqrt{2}$ -approximation. Then, we will use this result to obtain a $(1 + 2\sqrt{2})$ -approximation for arbitrary server capacities. Before we present details of the algorithm we give a high-level picture of our approach. The intrinsic difficulty in proving any approximation bound is to obtain lower bounds on the optimal makespan. Let us recall the inequality shown in Lemma 3.1: for *any* solution, peer i 's contribution to the upload in time interval $[t_1, t_2]$ is bounded by $u_i(t_1, t_2) \leq \max\{0, c_i(t_2 - t_1) - 1 + x_i(t_1)\}$. To exploit this capacity bound, our algorithm should fulfill two properties: it should finish peers with large capacity as soon as possible and it should avoid idle time as long as possible. To achieve exactly this, we use the concept of *time-varying resource augmentation*. We call a possibly infeasible solution \tilde{S} a $\sqrt{2}$ -*augmented* solution if at any point in time the original peer capacities are not exceeded by more than a factor of $\sqrt{2}$, i.e. $\sum_{j \in N} s_{i,j}(t) \leq \sqrt{2}c_i^u$ and $s_{p(i),i}(t) \leq \sqrt{2}c_i^d$ for all $i \in N, t \in \mathbb{R}_{\geq 0}$. We will show that there is an efficiently computable $\sqrt{2}$ -*augmented* solution that satisfies the above mentioned properties. This allows to apply the capacity expansion lemma, and we get that the makespan \tilde{M} of \tilde{S} is not larger than $2M^*$. Rescaling the augmented solution \tilde{S} to obtain a feasible solution S we get an additional factor of $\sqrt{2}$.

Now, we explain our algorithm (termed SCALE-FIT) in more detail. Let the peers be labeled such that $c_0 \geq \dots \geq c_n$. We choose an index k such that

$$c_0/\sqrt{2} \leq \sum_{j=1}^k c_j \leq \sqrt{2}c_0 \quad (1)$$

is satisfied, and assign peers $1, \dots, k$ to peer 0. Note that there always exists such an index k unless peer 0 can serve all peers simultaneously at full download rate. This is

formally proven in the following lemma, which we give for the slightly weaker assumption $c_0 \geq c_1/\sqrt{2}$.

Lemma 3.3. *Let $c_1 \geq \dots \geq c_n$ and let $c_0 \geq c_1/\sqrt{2}$. If $\sum_{j=1}^n c_j > c_0$, then there is $k < n$ such that $c_0/\sqrt{2} \leq \sum_{j=1}^k c_j \leq \sqrt{2}c_0$.*

Proof. If $c_0/\sqrt{2} \leq c_1$, there is nothing left to show. Otherwise, let $k' = \max\{\ell \in \{1, \dots, n\} : \sum_{j=1}^{\ell} c_j < c_0/\sqrt{2}\}$. Since we assume $c_1 < c_0/\sqrt{2}$ and $\sum_{j=1}^n c_j > c_0$, we have $1 \leq k' \leq n - 1$. We set $k = k' + 1$. By definition, $\sum_{j=1}^k c_j \geq c_0/\sqrt{2}$. In addition, we have $\sum_{j=1}^k c_j \leq 2 \sum_{j=1}^{k'} c_j < \sqrt{2}c_0$. \square

Given the choice of k according to (1), we now explain the scaling of the capacities. We distinguish two cases. If the total capacity of the downloading peers $1, \dots, k$ exceeds the capacity of the uploading peer 0, we augment the upload capacity to match the total download capacity. Otherwise the downloaders' capacities are scaled by a common factor in order to fit the uploader's capacity. Formally, the capacities of peers $1, \dots, k$ are increased by a factor of $\alpha = \max\{1, c_0 / \sum_{j=1}^k c_j\}$ and that of peer 0 by a factor of $\beta = \sum_{j=1}^k \alpha c_j / c_0$. Note that by (1) the scaling factor is not greater than $\sqrt{2}$ in either case.

At time 0, all k peers start downloading from peer 0 with their full (possibly augmented) capacity. Whenever a peer finishes its download, the algorithm rescales the augmented capacity (either that of the downloader or that of the uploader) to the original level. After rescaling, we proceed serving new peers by the respective unused capacities of the parent and the finished peer according to (1), breaking ties arbitrarily. The process stops as soon as the total capacity of the remaining peers is less than the currently available upload capacity. From this point on, all remaining peers are served simultaneously at full download rate (this last step takes at most M^* time). The choice of the augmentation factors and the rescaling procedure ensures the invariant that the augmented capacities do not exceed the original capacities by a factor of $\sqrt{2}$. A formal proof of this fact will be included in the full version of this paper.

Before we formally analyse the performance of SCALE-FIT, let us illustrate the algorithm by an example:

Example 3.4. Consider an instance with 6 peers and capacities $c_0 = 5$, $c_1 = 3$, $c_2 = 3$, $c_3 = 5/2$, $c_4 = 2$, and $c_5 = 2$. We first describe how to construct the augmented solution \tilde{S} . At time 0, the upload capacity of peer 0 is augmented by a factor of $6/5$ in order to serve peers 1 and 2 at their full download capacities. These downloads are completed at time $1/3$. At that time, the rescaled upload capacities of the parent of peers 1 and 2 (peer 0) can be used separately to serve further peers. The $5/2$ units of capacity (previously assigned to peer 1) are now assigned to peer 3 that downloads with full capacity without any augmentation. The remaining $5/2$ units of capacity are assigned to peer 4, whose download capacity is augmented by a factor of $5/4$. At the same time, peer 1 starts serving peer 5 without any augmentation because peer 5 is the only remaining peer. The highest augmentation factor is $5/4$. Thus, we rescale all rates by $4/5$ and obtain the feasible solution S .

We now turn to the approximation guarantee of our algorithm. For this, let \tilde{S} be the $\sqrt{2}$ -augmented solution generated by SCALE-FIT, and let T_0 denote the first point in time

in this solution when a peer does not fully use its upload capacity, that is, $T_0 = \min\{t \geq 0 : \exists i \in N \text{ with } \tilde{C}_i \leq t \text{ and } \sum_{j \in N} \tilde{s}_{i,j}(t) < c_i\}$. We first show that T_0 is a lower bound on the optimal makespan.

Lemma 3.5. *Let I be an instance of the peer-to-peer file distribution problem with $c_0 \geq c_i \forall i = 1, \dots, n$ and let \tilde{S} be the $\sqrt{2}$ -augmented solution generated by SCALE-FIT. Then $T_0 \leq M^*$, where M^* is the optimal makespan.*

Proof. Let us first consider the case $T_0 < \tilde{M}$. For a contradiction, suppose $M^* < T_0$ and fix an optimal solution S^* . We have $\tilde{X}(T_0) < X^*(T_0) = n$. Let k_0 be the peer with smallest index that has not finished the download at time T_0 , that is, $k_0 = \min\{i \in N : \tilde{C}_i > T_0\}$. Note that there is such peer since we assume $T_0 < \tilde{M}$. If $k_0 = 1$, then we obtain the inequality $T_0 < 1/c_1$, which is a lower bound on M^* and there is nothing left to show. So we assume $k_0 > 1$ and consider the point in time $T_1 = T_0 - 1/c_{k_0}$. For a contradiction, let us assume that $\tilde{X}(T_1) \geq X^*(T_1)$.

For the $\sqrt{2}$ -augmented solution \tilde{S} returned by SCALE-FIT we have $\tilde{x}_{k_0}(T_1) = 0$, and since peers start downloading in order we also get $\tilde{x}_i(T_1) = 0$ for all $i \geq k_0$. By construction, every peer $i \in N$ with $\tilde{x}_i(T_1) > 0$ receives data with download rate $d_i = \alpha c_i \geq c_i$ until \tilde{C}_i . Using $\tilde{z}_i(T_1, T_0) = 0$, we obtain

$$\tilde{u}_i(T_1, T_0) \geq \left(\frac{1}{c_{k_0}} - \frac{1 - \tilde{x}_i(T_1)}{d_i} \right) c_i = \frac{c_i}{c_{k_0}} + \frac{c_i}{d_i} (\tilde{x}_i(T_1) - 1) \geq \frac{c_i}{c_{k_0}} + \tilde{x}_i(T_1) - 1$$

for all $i \leq k_0$ with $\tilde{x}_i(T_1) > 0$ and $u_i(T_1, T_0) \geq 0$ for all other peers. Referring to Lemma 3.1 (1.), we calculate

$$\begin{aligned} X^*(T_0) - X^*(T_1) &\leq \sum_{i \in N} \max \left\{ 0, \frac{c_i}{c_{k_0}} - 1 + x_i^*(T_1) \right\} = \sum_{i \in N: x_i^*(T_1) > 1 - c_i/c_{k_0}} \left(\frac{c_i}{c_{k_0}} - 1 + x_i^*(T_1) \right) \\ &\leq \sum_{i < k_0: x_i^*(T_1) > 1 - c_i/c_{k_0}} \left(\frac{c_i}{c_{k_0}} - 1 \right) + X^*(T_1), \end{aligned}$$

where we use $c_i/c_{k_0} \leq 1 \forall i \geq k_0$. We get $X^*(T_0) - X^*(T_1) \leq \sum_{i < k_0} (c_i/c_{k_0} - 1) + \tilde{X}(T_1) \leq \tilde{X}(T_0) - \tilde{X}(T_1)$, a contradiction. We conclude $X^*(T_1) > \tilde{X}(T_1)$. Applying the same line of argumentation for T_1 instead of T_0 , we derive the existence of $T_2 = T_1 - 1/c_{k_1}$ for some $k_1 \in N$ with $X^*(T_2) > \tilde{X}(T_1)$. We can iterate this argument until we reach T_ℓ , with the property that $X^*(T_\ell) > \tilde{X}(T_\ell)$ and $k_\ell = \min\{i \in N : \tilde{C}_i > T_\ell\} = 1$. This is a contradiction since in the time interval $[0, T_\ell]$ only the server can upload and its upload rate in \tilde{S} is not smaller than that in S^* . We conclude that our initial assumption that $T_0 > M^*$ was wrong, finishing the proof for the case $T_0 < \tilde{M}$. If $T_0 = \tilde{M}$, we can use the same line of argumentation for $T_0 - \epsilon$ instead of T_0 , where $\epsilon > 0$ is arbitrary. Thus we obtain $T_0 \leq M^*$ also for that case. \square

We are now ready to prove the approximation guarantee of SCALE-FIT.

Theorem 3.6. *For the peer-to-peer file distribution problem with heterogeneous symmetric peers, the following holds:*

1. If $c_0 \geq c_1$, SCALE-FIT is a $2\sqrt{2}$ -approximation.
2. If $c_0 < c_1$, uploading the file to the peer 1 and applying SCALE-FIT is a $(1 + 2\sqrt{2})$ -approximation.

Proof. We first show 1. By construction, peer n determines the makespan and starts its download not after T_0 . Hence, the makespan of the $\sqrt{2}$ -augmented solution \tilde{S} is not larger than $T_0 + 1/c_n \leq 2M^*$ where we use Lemma 3.5 and the fact that $1/c_n$ is a lower bound on the optimal makespan. For rational input, the largest factor with which a capacity constraint is violated is strictly smaller than $\sqrt{2}$. Dividing all sending rates by that factor, we obtain a feasible solution with makespan smaller than $2\sqrt{2}M^*$.

To see 2., note that the server needs $1/c_0 \leq M^*$ time units to transfer the file to peer 1. We then treat the instance as an instance where peer 1 is the server, i.e. we do not let peer 0 upload the file to any other peer except peer 1. Using the same arguments as in the proof of Lemma 3.5, we derive that this takes at most $2\sqrt{2}M^*$ additional time units, implying the claimed approximation factor. \square

References

1. Androulatsos-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surveys 36, 335–371 (2004)
2. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Message multicasting in heterogeneous networks. SIAM J. Comput. 30(2), 347–358 (2000)
3. Cho, K., Fukuda, K., Esaki, H., Kato, A.: The impact and implications of the growth in residential user-to-user traffic. In: Proc. ACM SIGCOMM Conf. Applications, Technologies, Architectures and Protocols for Computer Comm. (2006)
4. Ezovski, G., Tang, A., Andrew, L.: Minimizing average finish time in P2P networks. In: Proc. 30th IEEE Internat. Conf. Computer Comm., INFOCOM (2009)
5. Garey, M., Johnson, D.: Computers and Intractability (1979)
6. Hedetniemi, S.T., Hedetniemi, S.M., Liestman, A.: A survey of gossiping and broadcasting in communication networks. Networks 18, 129–134 (1998)
7. Khuller, S., Kim, Y.-A.: Broadcasting in heterogeneous networks. Algorithmica 48(1), 1–21 (2007)
8. Kumar, R., Ross, K.: Peer assisted file distribution: The minimum distribution time. In: Proc. 1st IEEE Workshop on Hot Topics in Web Syst. and Technologies (2006)
9. Li, J.: On peer-to-peer (P2P) content delivery. Peer-to-Peer Netw. Appl. 1, 45–63 (2008)
10. Lua, E., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Comm. Surveys and Tutorials 7, 72–93 (2005)
11. Mehyar, M., Gu, W., Low, S., Effros, M., Ho, T.: Optimal strategies for efficient peer-to-peer file sharing. In: Proc. IEEE Internat. Conf. on Acoustics Speech and Signal Proc., ICASSP (2007)
12. Middendorf, M.: Minimum broadcast time is NP-complete for 3-regular planar graphs and deadline 2. Inf. Process. Lett. 46(6), 281–287 (1993)
13. Mundinger, J., Weber, R., Weiss, G.: Optimal scheduling of peer-to-peer file dissemination. J. of Scheduling 11, 105–120 (2008)
14. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In: Proc. ACM SIGCOMM Conf. Applications, Technologies, Architectures and Protocols for Computer Comm. (2004)
15. Ravi, R.: Rapid rumor ramification: Approximating the minimum broadcast time. In: Proc. 35th Annual IEEE Sympos. Foundations Comput. Sci., pp. 202–213 (1994)

Animal Testing

Adrian Dumitrescu^{1,*} and Evan Hilscher^{2,**}

¹ Computer Science, University of Wisconsin–Milwaukee, USA

dumitres@uwm.edu

² Computer Science, University of Wisconsin–Milwaukee, USA

hilscher@uwm.edu

Abstract. A configuration of unit cubes in three dimensions with integer coordinates is called an *animal* if the boundary of their union is homeomorphic to a sphere. Shermer discovered several animals from which no single cube may be removed such that the resulting configurations are also animals [14]. Here we obtain a dual result: we give an example of an animal to which no cube may be added within its minimal bounding box such that the resulting configuration is also an animal. We also present two $O(n)$ -time algorithms for determining whether a given configuration of n unit cubes is an animal.

1 Introduction

An *animal* is defined as a configuration of axis-aligned unit cubes with integer coordinates in 3-space such that the boundary of their union is homeomorphic to a sphere. There are several properties that may manifest in cube configurations, but are not allowed in animals. First of all, an animal must be “connected”, that is, it must form a whole piece. Moreover, it must be *face-connected*: any two cubes are connected by a path of face-adjacent cubes. Second, an animal may not have any handles, *i.e.*, the genus of its boundary surface is zero. Third, an animal may not have any interior holes. Holes (sometimes called shells, voids, or hollow areas) are empty regions that are completely surrounded by cubes. Finally, an animal may not have any illegal adjacencies, where the boundary locally is not homeomorphic to a disk. Illegal adjacencies are defined precisely later on, but some examples are shown in Fig. 1. Three configurations that are not animals are shown in Fig. 2.

A fundamental question regarding animals is: Given two animals A_1 and A_2 , can A_1 be transformed into A_2 by a sequence of single-cube additions and removals, such that each intermediate configuration is also an animal? This problem was first communicated by Pach in 1988 at the Seventh NYU Computational Geometry Day [10][11], and has since remained open. More recently, it has been mentioned again in [4].

A tentative algorithm for computing such a sequence is as follows. First reduce A_1 to a single cube by removals only, and then expand this single cube into A_2

* Supported in part by NSF grant CCF-0444188 and NSF grant DMS-1001667.

** Supported by NSF CAREER grant CCF-0444188 and NSF grant DMS-1001667.

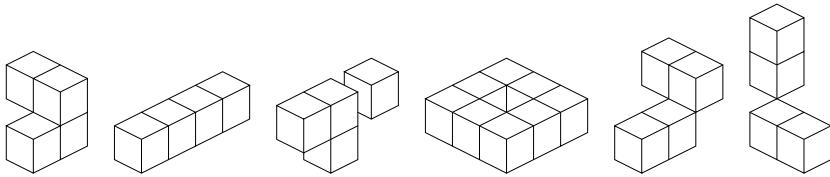


Fig. 1. Two simple animals (first two from the left), and four configurations that are not animals (3rd, 4th, 5th, 6th from the left): a disconnected configuration, a configuration with genus 1, a configuration with an illegal edge adjacency, and a configuration with an illegal vertex adjacency

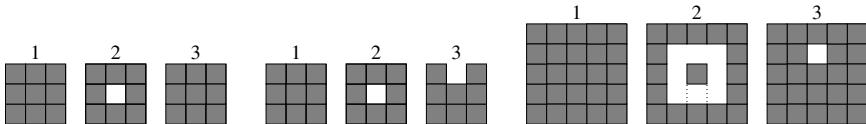


Fig. 2. Three 3-layer configurations that are not animals. Adding a cube to the configuration on the right in the cell marked with dotted lines makes it a valid animal.

by additions only. Unfortunately this algorithm was shown by Shermer [14] to be infeasible. He presented constructions of several animals such that no cube could be removed while maintaining the configuration as an animal. Such an animal is said to be *irreducible*. O'Rourke accredits the smallest irreducible animal known so far, made from 119 cubes, to Shermer [14]. We have been unable to obtain a description of this animal in the literature. The smallest irreducible animal of which we have been able to find a description consists of 128 cubes, and can fit inside of a $7 \times 7 \times 4$ box [14].

Here we study a dual algorithm for transforming A_1 to A_2 . Consider the smallest axis-aligned box containing an animal A . Let A' be the animal such that there is a cube at every integer coordinate within the box, *i.e.*, it is a solid rectangular box containing the given animal. The algorithm is as follows:

1. Transform A_1 to A_1' by addition only.
2. Transform A_1' to A_2' .
3. Transform A_2' to A_2 by removal only.

It is easy to see that A_1' can be transformed to A_2' . We simply add or remove one layer of A_1' , one cube at a time. The only question is, can any animal A be transformed to A' by addition only? If the answer is yes, then the third step above is also feasible. As it turns out, the answer is no, thus our alternative algorithm is also infeasible.

Our results. We first construct an animal to which no cube may be added within the minimal bounding box such that the resulting collection of unit cubes is an animal (Theorem 1 below); due to space constraints we omit the details. Such an animal is said to be *weakly inexpandable*. Note that any animal can be

expanded by adding a cube, but not always within the minimal bounding box. In Sections 2 and 3 we present two linear-time algorithms for deciding whether a given configuration of unit cubes is an animal, a problem which has not been studied so far. Clearly any algorithm must look at each of the n cubes to make this decision, thus our two algorithms are asymptotically optimal.

Theorem 1. *There exists an animal that is weakly inexpandable.*

Theorem 2. *Given a configuration \mathcal{C} of n unit cubes, it can be determined in $O(n)$ time whether \mathcal{C} forms an animal.*

2 Algorithm 1 Based on Fundamental Polygons and Davenport-Schinzel Sequences

In this section we present our first linear-time algorithm for determining whether a given configuration of n unit cubes is an animal and thereby prove Theorem 2.

Preprocessing. The input consists of a list of n unit cubes $\mathcal{C} = \{C_1, \dots, C_n\}$ given by their (center) coordinates: (x_i, y_i, z_i) is the coordinate of C_i . First the input configuration is translated so that it lies in the first octant of the coordinate system: $x, y, z \geq 0$, and there is a cube in each of the three planes $x = 0$, $y = 0$, and $z = 0$. This step is easily accomplished in $O(n)$ time. Test now whether there exists a cube with some coordinate that is larger or equal to n : $x_i \geq n$, $y_i \geq n$ or $z_i \geq n$. Note that any animal is face-connected, *i.e.*, for any pair of cubes there is a connecting path made of cubes, such that any two consecutive cubes are face-adjacent. Since $|\mathcal{C}| = n$, the existence of a cube with a large coordinate as above implies that \mathcal{C} is not face-connected, hence not an animal.

Next we determine the set of (at most 26) other cubes that are *adjacent* to C_i , for each $i = 1, \dots, n$. From \mathcal{C} , a “neighborhood” list $\Gamma = \Gamma(\mathcal{C}) = \{N_1, \dots\}$ of at most $27n$ cells is constructed. Γ contains \mathcal{C} and all empty cells that are adjacent to some cube in \mathcal{C} . Each element of Γ is marked as a cube or as an empty cell. During construction, pointers are constructed that link each C_i to the positions in Γ of itself and of its 26 adjacent cells. Using *counting sort*, or *radix sort*, Γ is sorted six times in the following orders: (x, y, z) , (x, z, y) , (y, x, z) , (y, z, x) , (z, x, y) , (z, y, x) . This allows finding for each cell $(x, y, z) \in \Gamma$, and for each coordinate, the at most two cells of Γ next to it in that coordinate. The results (after pruning duplicate elements) are stored in six arrays A_1, \dots, A_6 , each with $k \leq 27n$ elements. During the sorting, pointers are maintained that link each C_i to the positions in the six arrays A_1, \dots, A_6 of the six occurrences of each cell adjacent to C_i . Conversely, each cell in Γ is linked to all cubes C_i to which it is adjacent to (this can be done in the process of pruning duplicate elements). The six occurrences of each cell N_j , $j \leq k$ are also linked together. Using this data structure, one can determine for each cube C_i the set of cubes adjacent to it, by following only a constant number of pointers.

Outline. The algorithm is performed in four phases:

1. Initial scan and check for illegal adjacencies.
2. Identification of twins.
3. Circular list expansion.
4. Crossing pair detection.

In phase one, we first determine the set of (at most 26) other cubes that are adjacent to C_i , for each $i = 1, \dots, n$. We then determine the set of boundary faces of the configuration (those faces that are adjacent to some empty cell). We then scan the input array for invalid edge-adjacencies, vertex-adjacencies, and inverse-vertex-adjacencies (described in detail below). Each cube has a constant number of neighbors, and a constant number of combinations of these neighbors must be checked for these potential problems, any of which would preclude the given configuration from being an animal. Thus this phase can be performed in $O(n)$ time.

In phases two and three, we construct a circular list of edges, called a *fundamental polygon* of the boundary of the configuration of cubes. This fundamental polygon will help us determine whether the given configuration has genus > 0 , which would indicate that the configuration is not homeomorphic to a sphere or if it has genus $= 0$, which would indicate that the configuration is an animal. We represent the fundamental polygon as a circular list L of edges. Once we have a fundamental polygon, phase four of the algorithm determines the presence (or absence) of handles by finding (or failing to find) a *crossing pair* of twins (defined below) in the overall list.

To construct this list for the entire configuration, we first construct a list of four edges for each boundary face. Each small list is oriented in a clockwise direction as viewed from the interior of the cube to which the face belongs. These lists will be then merged into a larger list. Each edge on the boundary of the cube-configuration is shared by precisely two boundary faces. As each face has its own list of four edges, there are exactly two instances of this shared edge. We say that these two instances form a *twin pair* of edges, and we point them to each other. Note that each edge has exactly one twin. The task in phase two is to determine the twin for each of the edges. Once this is done we can begin phase three, expanding some initial list of four edges, one face at a time, until all small lists of the boundary faces are merged, resulting in a fundamental polygon in the form of a circular list of edges.

Cubes, faces and edges form a hierarchy and are labeled accordingly: $E_{i,j,k}$ is the k th edge of the j th face of the i th cube, $i \in \{1, \dots, n\}$, $j \in \{1, \dots, 6\}$, $k \in \{1, \dots, 4\}$. We write $T(E_{i,j,k}) = E_{i',j',k'}$, if $E_{i',j',k'}$ is the twin of $E_{i,j,k}$. Observe that $T(T(E_{i,j,k})) = E_{i,j,k}$. A *crossing pair* of twins is a pair of twins $T(E_{i_0,j_0,k_0}) = E_{i'_0,j'_0,k'_0}$ and $T(E_{i_1,j_1,k_1}) = E_{i'_1,j'_1,k'_1}$ in list order $E_{i_0,j_0,k_0}, E_{i_1,j_1,k_1}, E_{i'_0,j'_0,k'_0}, E_{i'_1,j'_1,k'_1}$. If twin edges are labeled by the same symbol, a crossing pair of twins (for symbols a and b) is a subsequence of L of the form a, b, a, b , where $a \neq b$. It is known that a fundamental polygon represents a surface of genus > 0 (resp., $= 0$), if and only if it contains (resp., does not contain) a crossing pair of

twins. It is also true that a fundamental polygon containing some adjacent twin edges is equivalent to a fundamental polygon with those adjacent edges omitted from the list. See Fig. 1 in reference to these properties.

Using these two properties together gives us a simple method for determining the existence of a crossing pair of twins. Search for a pair of twins that are adjacent in the list (if any), remove them, and check the two edges immediately adjacent to the removed pair. If these edges are twins, we can remove them as well, and repeat until the list becomes empty. If the two edges are not twins, then continue searching the remainder of the list for another pair of adjacent twin edges. Note that removal of an adjacent pair of edges can only create a new adjacent pair of edges at that same location. Therefore we only need to continue searching the remainder of the list, rather than start the search over. If such a pair is found, then repeat the above steps. If no such pair is found, and the list is non-empty, then there must exist a crossing pair of twins in the circular list, and thus the given configuration is not an animal. If we reach a point where the list becomes empty, then there cannot have been a crossing pair of twins in the original list, and thus the given configuration is an animal. The whole procedure takes linear time in the size of the list.

Phase 1. We distinguish several types of cube *adjacencies*. We say that two cubes C_1 and C_2 are *adjacent* if and only if they share either a single vertex, a single edge, or a single face. Cubes that share a single vertex are said to be *vertex-adjacent*, cubes that share a single edge are said to be *edge-adjacent*, and cubes that share a face are said to be *face-adjacent*.

Since this adjacency information is available from the preprocessing step, we next verify whether \mathcal{C} has any illegal adjacency present, and if such is found, \mathcal{C} is ruled out as an animal. If a configuration of unit cubes contains a pair of edge-adjacent cubes C_1 and C_2 , then the configuration is an animal only if there is a face-adjacent path of 3 cubes from C_1 to C_2 ; see Fig. 3.

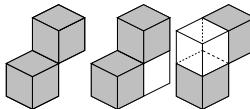


Fig. 3. An illegal edge adjacency (left), and the two minimal legal edge adjacencies (middle and right)

If a configuration of unit cubes contains a pair of vertex-adjacent cubes C_1 and C_2 , then the configuration is an animal only if there is a face-adjacent path of 4 cubes from C_1 to C_2 ; see Fig. 4.

There is one more type of adjacency, which we refer to as the *inverse-vertex-adjacency*. Two empty cells are *inverse-vertex-adjacent* if and only if they share a single vertex. There is only one situation where inverse-vertex-adjacency is illegal: in a $2 \times 2 \times 2$ block of cells, if only two of the cells are empty and inverse-vertex-adjacent; see Fig. 5.

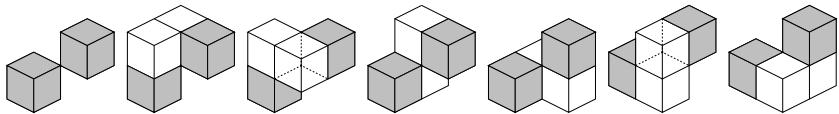


Fig. 4. An illegal vertex-adjacency (left), and the six minimal legal vertex-adjacencies

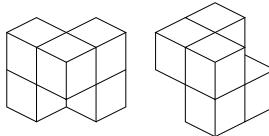


Fig. 5. Two views of the illegal inverse-vertex-adjacency configuration

These are the only three types of illegal adjacencies that could preclude locally a cube configuration from being an animal. Given any cube along with its adjacent cubes, it is easy to determine, in constant time, if all of the above conditions are met. Since the input configuration consists of n cubes, Phase 1 takes $O(n)$ time.

Phases 2,3,4. The remainder of the algorithm makes use of the concept of *fundamental polygon* from geometric topology; see [1, pp. 60–61].

Definition 1. [1 p. 61]. A fundamental polygon is an even-sided convex polygon in the plane, whose edges are ordered clockwise. If D is a fundamental polygon, a gluing scheme S for the edges of D is a labeling of each edge of D , where each letter used in the labeling appears on precisely two edges.

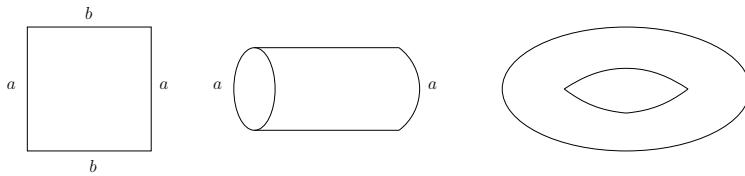


Fig. 6. A simple fundamental polygon (left), the result (a cylinder) of gluing the twin edges labeled b (middle), and the final surface (a torus) obtained by then gluing the twin edges labeled a (right); [1 p. 61]

By the following theorem, we know that we can construct a fundamental polygon for the boundary of a connected configuration of unit cubes without any illegal adjacencies.

Theorem 3. [1 p. 64]. (i) Let D be a fundamental polygon, and let S be a gluing scheme for the edges of D . Then there is a surface $Q \subset \mathbb{R}^d$ that is obtained from D and S .

- (ii) Let $Q \subset \mathbb{R}^d$ be a compact connected surface. Then there is a fundamental polygon D and a gluing scheme S for the edges of D such that Q is obtained from D and S .

We now give a more detailed description of phases 2, 3 and 4. We start by examining each boundary face of the configuration. Our objective at this stage is to identify and label every pair of twin edges uniquely. This can be done in linear time, as each cube has at most six boundary faces, each boundary face has exactly four edges, and the twin of each edge instance must be in one of three possible locations; see Fig. 7.

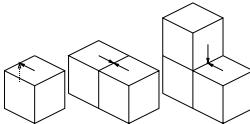


Fig. 7. The three possible locations of a twin edge

Once the boundary edges have been labeled, we begin constructing a fundamental polygon for the configuration of unit cubes, by using breadth-first search. Start with any boundary face, say F_{i_0, j_0} , and mark this face as visited. Insert the edges of F_{i_0, j_0} into a circular list in clockwise order. Examine each of the twins of the edges of F_{i_0, j_0} . For instance, say we examine $T(E_{i_0, j_0, k_0}) = E_{i_1, j_1, k_1}, E_{i_1, j_1, k_1}$ belongs to boundary face F_{i_1, j_1} . Replace E_{i_0, j_0, k_0} in our circular list with a sequence of edges: those of F_{i_1, j_1} , except for E_{i_1, j_1, k_1} , in clockwise order starting with the edge following E_{i_1, j_1, k_1} . The replacement and insertion of edges can be easily done in constant time if we maintain pointers from the edges to nodes of a circular linked list. Mark F_{i_1, j_1} as visited, and continue examining the twins of the edges of F_{i_0, j_0} . As we are doing a breadth-first search of the boundary faces, we also insert F_{i_0, j_0} into a queue so that we know to examine its edges later. It should be clear that if $T(E_{i', j', k'})$ belongs to a visited face, we return to the face containing $E_{i', j', k'}$ without doing anything, and proceed to the next edge. Let L denote the circular list (representing the fundamental polygon) obtained. Observe that L has even size, and $|L| = 2m = O(n)$. Identification of twins and circular list expansion (Phases 2 and 3) take $O(n)$ time.

Once the search is complete, we scan for unvisited faces. The existence of any such face implies that there are at least two connected components of cubes in the given configuration, and that it is therefore not an animal. Once again, this test can easily be done in $O(n)$ time. Should it happen that all faces have been visited, we then have a fundamental polygon, in the form of a circular list with an even number of edges. Recall that a fundamental polygon represents a surface of genus > 0 (resp., $= 0$), if and only if it contains (resp., does not contain) a crossing pair of twins. Finally, with the assistance of Lemma 11 below, we can determine whether the boundary of the given configuration of unit cubes has genus 0 or genus > 0 .

A sequence $a_1a_2\dots$ of integers between 1 and m is called an (m, d) -Davenport-Schinzel sequence, if (i) it has no two consecutive elements which are the same, and (ii) it has no alternating subsequence of length $d + 2$, *i.e.*, there are no indices $1 \leq i_1 < i_2 < \dots < i_{d+2}$ such that

$$a_{i_1} = a_{i_3} = a_{i_5} = \dots = a, \quad a_{i_2} = a_{i_4} = a_{i_6} = \dots = b,$$

where $a \neq b$. Let $\lambda_d(m)$ denote the maximum length of an (m, d) -Davenport-Schinzel sequence (see [3,12]). Obviously, we have $\lambda_1(m) = m$, and it is known that $\lambda_2(m) = 2m - 1$, for every m ($\lambda_d(m)$ is super-linear in m for $d \geq 3$). Here we are only interested in the (simple) case $d = 2$, *i.e.*, in the equality $\lambda_2(m) = 2m - 1$.

Lemma 1. *Either there exist two consecutive elements in L that are the same, or L has at least one crossing pair of twins.*

Proof. Observe that the circular list L has a crossing pair of twins if and only if any linear list derived from it (by breaking the circular list arbitrarily) has a subsequence of the form a, b, a, b , where $a \neq b$. Recall that L has length $2m$ and consists of m distinct symbols, and each symbol appears twice. Then either there exist two consecutive elements in L that are the same and we are done, or no two consecutive elements are the same. In the latter case, we either (i) can find a subsequence of the form a, b, a, b , with ($a \neq b$), that is, a crossing as defined above, and we are done; or (ii) there is no such subsequence, hence L forms an $(m, 2)$ -Davenport-Schinzel sequence. However, since $|L| = 2m$, this would contradict the equality $\lambda_2(m) = 2m - 1$ given above. In other words, (ii) is impossible and this concludes the proof. \square

As explained in the outline of the algorithm, crossing pair detection (Phase 4) takes $O(m) = O(n)$ time, so the entire algorithm takes $O(n)$ time.

3 Algorithm 2 Based on Euler-Poincaré Formula

In this section we present another linear-time algorithm and thereby obtain an alternative proof of Theorem 2. The idea to use Euler's polyhedral formula was suggested to us by Todd Phillips [9]. Originally, the algorithm consisted of one very short step: count all the edges, faces, and vertices, and then check Euler's formula [2,8]: $V - E + F = 2$. In this formula, V , E and F stand for the number of vertices, edges and faces of a convex polytope (or of a connected planar graph). However, there are animals for which this formula does not hold, for instance the 2-layer configuration shown in Fig. 8. We have $V = 16$, $F = 11$, $E = 24$, and $V - E + F = 16 - 24 + 11 = 3$.

Specifically, we need the extension provided by the Euler-Poincaré formula for manifolds [5, pp. 41–45]; see also [6,7,13]. To state this formula we need the following definitions:

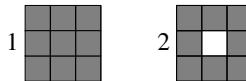


Fig. 8. A 17-cube animal for which Euler's formula does not hold

- V : the number of vertices.
- E : the number of edges.
- F : the number of faces.
- G : the number of handles (holes penetrating the solid), also referred to as *genus*.
- S : the number of *shells*. A shell is an internal void of a solid or the solid itself. A shell is bounded by a 2-manifold surface, which can have its own genus value. Since the solid itself is counted as a shell, $S \geq 1$.
- L : the number of *loops*. Each face has at least one loop (a plane cycle of edges) and all outer and inner loops of faces are counted.

Then, the Euler-Poincaré formula is the following:

$$V - E + F - (L - F) - 2(S - G) = 0.$$

Observe that for $S = 1$ the formula becomes $V + 2F - E - L + 2G = 2$. Assume that the input configuration is first checked for illegal adjacencies and it passes the test. Assume also that the input configuration is face-connected, and further that we can test for $S = 1$ (*i.e.*, no interior holes). Then the cube configuration is a 3-dimensional manifold and its boundary is a single connected component. Hence the formula $V + 2F - E - L = 2$ holds if and only if $G = 0$. The resulting algorithm is as follows:

1. Check that there are no illegal adjacencies (Phase 1 in Section 2).
2. Check that the cube configuration is face-connected.
3. Construct the *unit face graph* of the solid (polyhedron) made up from the boundary faces of the unit cubes and check whether it is connected. (Two boundary unit squares are adjacent in the unit face graph if they share a boundary unit edge.)
4. Compute V , E , F and L .
5. Check the Euler-Poincaré formula $V + 2F - E - L = 2$ for the solid made up from the unit cubes.
6. If all checks pass, return that the cube configuration is an animal, otherwise return that it is not.

For instance, on the animal in Fig. 8, all checks pass and $V + 2F - E - L = 16 + 22 - 24 - 12 = 2$. On the other hand, on the 8-cube torus in Fig. 8(left), the last check fails with $V + 2F - E - L = 16 + 20 - 24 - 12 = 0$.

Algorithm analysis. Note that if there are no illegal adjacencies and the unit face graph is connected then there are no interior holes (the second assumption alone does *not* imply this conclusion; see e.g., Fig. 2, middle). So the cube configuration is a 3-dimensional manifold and its boundary forms a single connected component. Hence the formula $V + 2F - E - L = 2$ tested by the algorithm in Step 5 holds if and only if the cube configuration is an animal, as required.

The numbers V , E and F can be computed in $O(n)$ time. By determining the set of loops associated with each face, L can be also computed in $O(n)$ time. Overall, the algorithm can be implemented so that it runs in $O(n)$ time, where n is the number of cubes.

Acknowledgment. The authors are grateful to an anonymous reviewer for uncovering an error in our earlier version of Algorithm 2.

References

1. Bloch, E.: A First Course in Geometric Topology and Differential Geometry. Birkhäuser, Basel (1997)
2. Bollobás, B.: Modern Graph Theory. Springer, Heidelberg (1998)
3. Davenport, H., Schinzel, A.: A combinatorial problem connected with differential equations. American Journal of Mathematics 87, 684–694 (1965)
4. Dumitrescu, A., Pach, J.: Pushing squares around. Graphs and Combinatorics 22(1), 37–50 (2006)
5. Hoffmann, C.: Geometric & Solid Modeling: An Introduction. Morgan Kaufmann (1989)
6. Kong, T.Y., Rosenfeld, A.: Digital topology: introduction and survey. Computer Vision, Graphics, and Image Processing 48, 357–393 (1989)
7. Nakamura, A., Aizawa, K.: On the recognition of properties of three-dimensional pictures. IEEE Transactions on Pattern Analysis and Machine Intelligence 7(6), 708–713 (1985)
8. Pach, J., Agarwal, P.K.: Combinatorial Geometry. John Wiley, New York (1995)
9. Phillips, T.: Personal communication (October 2010)
10. O'Rourke, J.: The Computational Geometry Column #4. ACM SIGGRAPH Computer Graphics 22(2), 111–112 (1988)
11. O'Rourke, J.: Computational Geometry Column 6. ACM SIGACT News 20(2), 10–11 (1989)
12. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, Cambridge (1995)
13. Ching-Kuang Shene's homepage, The Euler-Poincaré formula,
<http://www.cs.mtu.edu/shene/COURSES/cs3621/NOTES/model/euler.html>
14. Shermer, T.: A smaller irreducible animal; and a very small irreducible animal. In: Snapshots of Computational and Discrete Geometry, pp. 139–143 (1988)

Cutting Out Polygons with a Circular Saw

Adrian Dumitrescu^{1,*} and Masud Hasan^{2,**}

¹ University of Wisconsin–Milwaukee, USA

dumitres@uwm.edu

² Bangladesh University of Engineering and Technology, Dhaka, Bangladesh

masudhasan@cse.buet.ac.bd

Abstract. Given a simple polygon Q drawn on a piece of planar material R , we cut Q out of R by a circular saw with a total number of cuts no more than twice the optimal. This improves the previous approximation ratio of 2.5 obtained by Demaine *et al.* in 2001.

1 Introduction

The problem of efficiently cutting a polygon Q out of a piece of planar material R (Q is already drawn on R) is by now a well studied problem in computational geometry. This line of research was initiated by Overmars and Welzl in their seminal paper from 1985 [12]. There exist several variations of this problem, based on the cutting tools used and the measure of efficiency. The type of cutting tool also determines which polygon can be cut—a convex polygon or a non-convex polygon. The cutting tools that have been studied so far are line cuts, ray cuts, and saw cuts [1, 3–9, 12, 13]. Since a saw can be abstracted as a moving line segment, one can refer to saw cuts also as segment cuts. So the types of cutting tools can be expressed quite uniformly in geometric terms: line cuts, ray cuts, and segment cuts. The efficiency measures that have been considered are the total length of cuts and the total number of cuts. See also [2] for a survey on previous results on these variations.

A line cut is a line that does not go through Q and intersects R into two pieces. For cutting Q out of R by line cuts, Q must be convex. The most studied efficiency measure for line cutting is the total length of the cuts. Several approximation algorithms have been obtained [1, 3–9, 12, 13], including a PTAS [3].

In contrast, a ray cut comes from infinity and can stop at any point outside Q . The main focus of ray cutting problems are on cutting non-convex polygons. However, not all non-convex polygons can be cut by ray cuts. Again, the most studied efficiency measure is the total length of the ray cuts. Here the results include deciding whether a polygon is ray cuttable and several approximation algorithms [5–7, 13].

* Supported in part by NSF grant CCF-0444188 and NSF grant DMS-1001667.

** Part of the research by this author was done while visiting University of Wisconsin–Milwaukee in Fall 2010 and with support from NSF grant CCF-0444188.

Demaine *et al.* [7] studied the problem of cutting a polygon by a circular saw. A circular saw cut (also called a saw cut) is like a ray cut, but may not come from infinity. The saw is abstracted as a line segment, which cuts if moved along its supporting line. If a small free space within R is available, a saw cut can be initiated there by maneuvering the saw. The space required for maneuvering the saw is proportional to the length of the saw (line segment). For convenience and ease of analysis the length of the saw was assumed in [7] to be arbitrarily small, and its width was assumed to be zero. This means that a saw cut can be initiated from an arbitrarily small free space.

Two efficiency measures were considered in [7]: the total length of the cuts and the total number of cuts. A smaller number of cuts implies a shorter maneuvering time. The authors gave an approximation algorithm for cutting Q out of its convex hull $\text{conv}(Q)$, with both the total length of the cuts and the total number of cuts within 2.5 times their respective optimums. Here we improve the approximation guarantee in the latter measure of efficiency from 2.5 to 2. Moreover, our approximation guarantee is in a stronger sense than that achieved previously by Demaine *et al.* [7]. While theirs achieves ratio 2.5 for cutting out Q from $\text{conv}(Q)$, ours achieves ratio 2 for cutting out Q from any enclosing polygon R . We summarize our result in the following theorem.

Theorem 1. *Given a simple polygon Q fixed inside a piece of planar material $R \supset Q$, Q can be cut out of R by a circular saw of arbitrarily small maneuvering space with a number of cuts no more than twice the optimal. If R and Q have m and respectively n vertices, the cuts can be computed in $O(m + n \log n)$ time.*

Remark. More precisely we show that if Q has n vertices, it can be cut out using at most $2n - h$ cuts, where h is the size of $\text{conv}(Q)$ (the number of vertices of the convex hull of Q). In particular, if the average pocket complexity of Q is bounded from above by a constant, the approximation ratio drops below 2. Indeed, under the assumption we make that no three vertices of Q are collinear, each edge of Q requires at least one cut, hence at least n cuts are needed overall.

Outline. The idea of our algorithm is as follows. The difference between Q and its convex hull $\text{conv}(Q)$ consists of disjoint pockets. Each pocket is defined by an edge of the convex hull, which we call the *lid* of the pocket. For each edge of the convex hull we make a cut along the edge. In addition, if there is a pocket corresponding to this edge, we cut it out. We cut out each pocket of Q by using a number of cuts at most twice the number of edges of the respective pocket (excluding the lid) minus 1. To this end, for each pocket P , we shall compute the shortest path tree T of P from one of the vertices of the lid. Then we shall partition the edges of P into two types of polygonal chains. In the first type, each chain consists of only reflex edges of P and contains an internal vertex of T . In the first phase of our algorithm, we find those chains by traversing T . Along this traversal, we cut those chains by two cuts for each edge. The other type of chains are formed from the remaining edges. In the second phase of our algorithm, we cut the edges of those chains by using no more than two cuts per edge on average.

2 Preliminaries

Let Q be the polygon to be cut out from an arbitrary enclosing polygon R . We assume that the vertices of Q are in general position, i.e., no three vertices of Q are collinear. Let $n = |Q|$ and $h = |\text{conv}(Q)|$ be the number of vertices of Q and $\text{conv}(Q)$, respectively. Let (s, r) be an arbitrary edge of $\text{conv}(Q)$. We first make one long cut along (s, r) . This is similar to a line cut, and removes the part of R on the other side of the supporting line of (s, r) . If there is a pocket behind the lid (s, r) , we proceed to cut it out. The pocket is an open polygonal chain, say U , with endpoints s and r . $U \cup (s, r)$ defines a simple polygon, and we shall denote this polygon by P .

We say that a vertex v of P is *reflex* (resp. *convex*) if its angle within P is larger than π (resp. smaller than π). This definition of reflex and convex for v is opposite with respect to the interior of Q . An edge e of P is *reflex* if both its endpoints are reflex. After the first cut for P is made along the lid (s, r) , the vertices s and r become reflex in P .

A region F in the plane outside Q but contained in R is called *free* if the material has been removed from F by making cuts along F 's boundary. As cuts are executed, pieces of material that become free are automatically removed. A point x on the boundary of P is called *free* if there exists a free half-disc centered at x . A point x in the interior of P is called *free* if there exists a free disc centered at x . A *saw cut* is a line segment that starts from a free region and does not enter Q . An edge e of P is called *free* if every point of e except its convex endpoint (if any) is free.

When a saw cut is applied along an edge of Q , the edge becomes *disconnected* from Q . Clearly, a free edge is also disconnected, but the reverse is not true. We assume that after a saw cut is applied, any region of R that becomes disconnected from the remaining portion of R is automatically removed. Then the region R becomes free. The removing procedure for a disconnected region does not require a planar motion; it can be done by lifting it up. Once the lid of a pocket P has been cut along, P is considered to have been *cut* when all the edges of the polygonal chain $U = U(P)$ have been disconnected.

A convex vertex of P can never be free, since it cannot be the center of a free half-disk. Therefore, a saw cut cannot be initiated from a convex vertex. However, by definition, if a free edge e has a convex endpoint v , then a cut can be initiated from a point arbitrarily close to v . We call a convex vertex to have been *disconnected* when both its adjacent edges are disconnected. For each pocket $P = U \cup (s, r)$, we cut P using a number of cuts no more than $2|U| - 1$, where $|U|$ is the number of edges of U . Since the polygonal chains U of the pockets are disjoint, summing over all pockets of $\text{conv}(Q) \setminus Q$ will result in cutting the entire polygon Q by at most $2n - h$ cuts.

Not every polygon can be cut by a circular saw, even with a saw of arbitrarily small maneuvering space. There may be an edge of the polygon whose two endpoints are reflex within the polygon (thus convex within a pocket). For such an edge, no cut can be initiated. Moreover, Demaine *et al.* [7] showed that a polygon Q is cuttable by a small circular saw if and only if Q does not have two

consecutive reflex vertices. For brevity we call such a polygon *cuttable*. Equivalently, a polygon Q is cuttable if and only if each pocket of Q does not have two consecutive convex vertices. Again for brevity we refer to any such pocket as *cuttable*. Interestingly enough however, any simple polygon can be modified very slightly, for instance by adding very flat reflex vertices in between any two consecutive convex vertices, so that the resulting polygon becomes cuttable; see [7] for more details.

We next give some preliminaries on shortest path trees inside a simple polygon P which are key to our algorithm. Let s be a point contained in P and v be a vertex of P (in our case s is an endpoint of the pocket lid of P , hence also a vertex of P). The Euclidean shortest path from s to v inside P (according to total Euclidean length) is a unique polygonal chain (path) from s to v which can make turns only at reflex vertices of P [10, 11]. This path is denoted by $\pi(s, v)$. The union of the paths $\pi(s, v)$ over all vertices v of P forms a plane tree, called the *shortest path tree* of P from s [10, 11]. We denote this tree by T . The reflex vertices of P at which paths of T make turns are exactly the internal vertices of T . On the other hand, the vertices of P at which paths of T do not make turns are the leaves of T . While all the internal vertices of T are reflex, a leaf vertex of T can be reflex or convex. An edge connecting two internal vertices in T is referred to as an *internal edge* of T . An internal edge of T is called *reflex* if both its endpoints are reflex. An internal edge of T may be an edge of P or not; if it is an edge of P , it has to be a reflex edge of P . Otherwise we call that edge a *bridging edge* of T .

Lemma 1. *Let (u, v) be a (reflex/non-reflex) edge of P or a reflex edge of T , with u being a reflex vertex. Let $(p, v) \supseteq (u, v)$ be a line segment such that p is free. Then it is possible to make (u, v) free by using two cuts. If v is also reflex, this cut makes v free too.*

Proof. See Fig. 11 for an illustration of how the cuts are executed in each case. \square

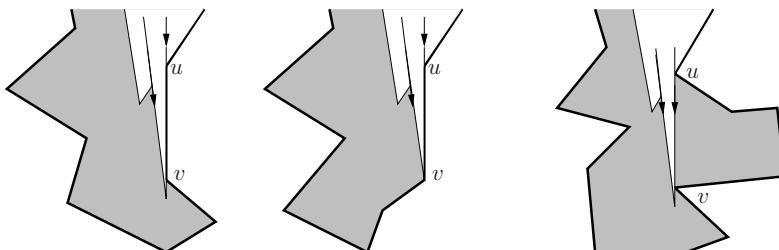


Fig. 1. Two cuts are sufficient to make free a reflex or non-reflex edge of P (left), or a reflex edge of T (right), if there is a free space to initiate the cuts

We compute the shortest path tree T of P from s . We identify two types of polygonal chains in the boundary of P : reachable chains and critical chains. These chains are edge-disjoint. We also categorize the vertices of P into three types. Let v be an internal vertex of T . If v is an endpoint of a reflex edge e of P , let C_R be the maximal polygonal chain that consists of only reflex edges of P including e . We call C_R a *reachable chain* of P . The vertices of reachable chains are called *type-1* vertices. On the other hand, the two adjacent vertices of v can be convex, and in that case, we call v a *type-2* vertex. (A type-2 vertex can be considered as a reachable chain with zero edges.) Critical chains are defined by type-2 vertices and the endpoints of reachable chains. More precisely, a *critical chain* is a polygonal chain whose two endpoints are type-2 vertices or endpoints of reachable chains. Reachable chains contain all internal vertices of T and possibly some leaves of T . Other vertices of P are the internal vertices of critical chains and we call them *type-3* vertices. Type-3 vertices are leaves of T ; these vertices can be reflex or convex. Vertex types are shown in Fig. 2(right).

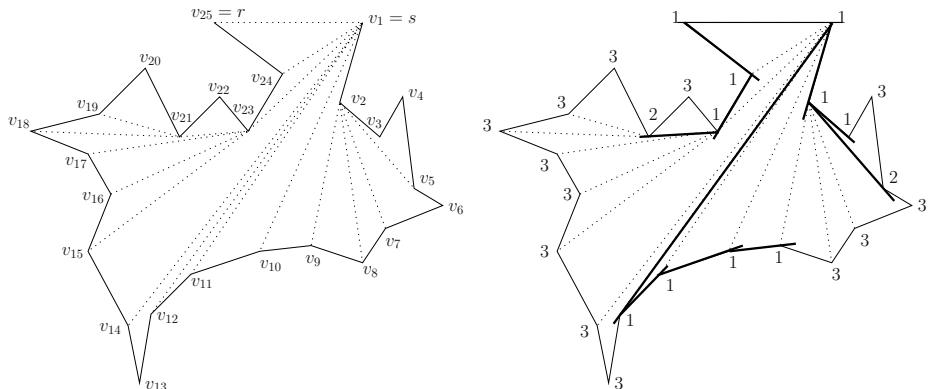


Fig. 2. Left: A pocket P with its shortest path tree T from s . The bridging edges of T are shown in dotted lines. The reachable chains of P are $\{v_1, v_2, v_3\}$, $\{v_9, v_{10}, v_{11}, v_{12}\}$ and $\{v_{23}, v_{24}, v_{25}\}$. Type-1 vertices of P are all vertices in these three chains. Type-2 vertices of P are v_5 and v_{21} . The rest of vertices of P are of type-3. The critical chains of P are $\{v_3, v_4, v_5\}$, $\{v_5, v_6, v_7, v_8, v_9\}$, $\{v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}, v_{18}, v_{19}, v_{20}, v_{21}\}$, and $\{v_{21}, v_{22}, v_{23}\}$. Right: double cuts made in Phase 1 are drawn in bold lines; the cut along the pocket lid (in Phase 0) is a single cut.

Some properties of reachable chains and critical chains are easy to follow. A reachable chain has at least one edge and does not have any convex vertex. Due to the maximality property, two reachable chains cannot be adjacent, i.e., two reachable chains are vertex disjoint. However, two critical chains may be adjacent in a type-2 vertex. A reachable chain is adjacent to a critical chain in a type-1 vertex. A critical chain has at least one convex vertex. The two adjacent edges of this convex vertex imply that a critical chain has at least two edges.

3 Algorithm

We first compute K , a minimum axis-aligned rectangle containing R . For convenience, we start all cuts that originate outside $\text{conv}(Q)$ on the boundary of K . Alternatively, these cuts can be initiated near their intersection points with the boundary of R .

Let (s, r) be an arbitrary edge of $\text{conv}(Q)$. We first make one long cut along (s, r) ; we will refer to this single cut as Phase 0. If there is no pocket of Q behind this edge, we proceed to the next edge of $\text{conv}(Q)$. If there is, i.e., (s, r) is the lid of a pocket $P = U \cup (s, r)$, we proceed to cut it out. After the cut along (s, r) the vertices s and r become reflex. Denote by t_1 , t_2 , and t_3 the number of type-1, type-2 and type-3 vertices of P . Vertices s and r are of type-1, hence $t_1 \geq 2$ in particular. Clearly we have $t_1 + t_2 + t_3 = t$, where $t \geq 2$ is the number of vertices in P , including the two endpoints of the pocket lid.

In Phase 1 of our algorithm, we shall cut along all internal edges of T and along the edges of reachable chains by at most $2(t_1 + t_2) - 4$ cuts. This will make free all type-1 and type-2 vertices and the edges of all reachable chains. In Phase 2 of our algorithm, we shall cut along the edges of critical chains by using at most $2t_3$ cuts. Completion of these two phases will give the result.

Overall, the number of cuts made for cutting out one pocket (including the initial cut along the convex hull edge) is at most $1 + 2(t_1 + t_2) - 4 + 2t_3 = 2(t_1 + t_2 + t_3) - 3 = 2t - 3$. Overall, Q is cut out of R by at most $\sum_P (2t - 3) = \sum_P (2|U| - 1) = 2n - h$ cuts. Since n is a trivial lower bound on the number of cuts needed, the approximation ratio of 2 immediately follows; see also [7].

Phase 1: Cutting reachable chains. We shall traverse T in level order. At each level $i \geq 1$, we assume that all reflex vertices in level $i - 1$ are already free. In addition we assume that all reachable chains incident to reflex vertices made free so far are also free. For level 0 we start with two reflex vertices, s and r , which are already free. In addition, if w is free and (w, v) is a reflex edge, we make free (w, v) by two cuts by Lemma 1. We repeat this step to make free all reflex edges that are reachable in this way. So the above assumptions are satisfied for level 0.

Let $i \geq 1$. We look into each reflex vertex v in level i . If v is not free, then let w be the parent of v in level $i - 1$. Observe that (w, v) is a bridging edge. Because, if (w, v) were a reflex edge of P , then it would belong to the reachable chain that contains w , and therefore we would have made free (w, v) as well as v in the previous level. Since w is already free, we make free (w, v) by two cuts by Lemma 1. Then if v belongs to a reachable chain, say C_R , then for each reflex edge e of C_R , if e is not free, we make it free by two cuts by Lemma 1.

The following lemma proves that we have made free the edges of all reachable chains and quantifies the number of cuts used for that.

Lemma 2. *After Phase 1, all internal vertices of T and the edges of all reachable chains of P are free. The number of cuts used to make them free in Phase 1 is at most $2(t_1 + t_2) - 4$, where t_1 and t_2 are the total number of type-1 and type-2 internal vertices of P . These cuts also make free the endpoints of all reachable chains.*

Proof. By our algorithm, we have traversed all internal vertices of T and made them free, if they are not free already, from their parents in the respective previous levels. Since every reachable chain contains an internal vertex of T , this guarantees that for each reachable chain we have reached one of its vertices and from there we have made free all of its edges.

Let v be an internal vertex of T to which we have reached from its parent w in the previous level. Let C_R be the reachable chain that contains v . If v was not already free, we made it free by two cuts. Then we used two more cuts to free each edge, and corresponding new vertex of C_R . If t_R is the number of vertices in C_R , then there are $t_R - 1$ edges in C_R . Therefore, we have used at most $2 + 2(t_R - 1) = 2t_R$ cuts for C_R . Since two reachable chains are disjoint, over all reachable chains we have used at most $\sum_{C_R} 2t_R = 2t_1$ cuts.

If v is a type-2 vertex, then we have used only two cuts to make it free and no cuts for a reachable chain (a reachable chain does not exist in this case). So for all these t_2 vertices, we have used a total of $2t_2$ cuts.

It follows that so far at most $2(t_1 + t_2)$ cuts have been made in Phase 1. Recall that s and r were made free by one cut in Phase 0. Observe that each pair of cuts executed in Phase 1 made free a new vertex. Thus the above expression over-counts by 4, hence we have used in fact at most $2(t_1 + t_2) - 4$ cuts. Finally, since the endpoints of reachable chains are reflex vertices, by Lemma 11, the above cuts have made free those endpoints too. \square

At the conclusion of Phase 1, all type-1 and type-2 vertices of P and the edges of all reachable chains have been made free. We are left with only the edges and internal vertices (of type-3) of critical chains. In what follows, we shall see how to make them free.

Phase 2: Cutting critical chains. Let $P = \{s = v_1, v_2, \dots, v_n = r\}$ be the clockwise vertex sequence of P . Consider a path $\pi(s, v_i)$ in T . We denote by $C_{right}(v_i) = \{v_1, v_2, \dots, v_i\}$ the polygonal chain consisting of the vertices from v_1 to v_i and by $C_{left}(v_i) = \{v_i, v_{i+1}, \dots, v_n\}$ the polygonal chain consisting of the vertices from v_i to v_n .

Let $e = (v_i, v_{i+1})$ be an edge of P where v_i and v_{i+1} are leaves of T . Let a be the lowest common ancestor of the paths $\pi(s, v_i)$ and $\pi(s, v_{i+1})$. Let $\pi'(a, v_i)$ and $\pi'(a, v_{i+1})$ be the sub-paths of $\pi(s, v_i)$ and $\pi(s, v_{i+1})$ from a to v_i and v_{i+1} , respectively.

It is known [10, 11] that both $\pi'(a, v_i)$ and $\pi'(a, v_{i+1})$ are outward convex and oppositely oriented. Together with e they define a *funnel* with *tip* a , denoted by $\mathcal{F}(v_i, v_{i+1})$.

Lemma 3. *Let f be an edge of $\pi'(a, v_i)$. If f is an edge of P , then f is an edge of $C_{right}(v_i)$.*

Proof. If we traverse $C_{right}(v_i)$ from v_i to s , then the interior of P is on the left side. Similarly, if we traverse $C_{left}(v_i)$ from v_i to s , then the interior of P is on the right side. Since T is a plane tree, T partitions P into disjoint plane regions.

The interior of the funnel $\mathcal{F}(v_i, v_{i+1})$ is one of these regions. If we traverse $\pi(s, v_i)$ from v_i to s , this funnel is on the left side. Therefore, if f is in $\pi'(a, v_i)$, f is in $C_{right}(v_i)$. \square

Lemma 4. *Consider a funnel $\mathcal{F}(v_i, v_{i+1})$ with tip a , and a point x on $\pi'(a, v_{i+1})$. Then any ray emanating from x towards the interior of $\mathcal{F}(v_i, v_{i+1})$ must hit $e = (v_i, v_{i+1})$ or an edge in $C_{right}(v_i)$.*

Proof. Let the ray emanating from x be ℓ . Since $\pi'(a, v_{i+1})$ is outward convex, ℓ cannot hit an edge of $\pi'(a, v_{i+1})$. So, ℓ hits either e or $\pi'(a, v_i)$. Assume that, ℓ hits $\pi'(a, v_i)$ at the edge e' . If e' is an edge of P , then by Lemma 3 e' is an edge of $C_{right}(v_i)$ and we are done. Otherwise, $e' = (v_j, v_k)$, for $j < k \leq i$, is a bridging edge of T . Then ℓ hits one of the edges of P in the chain $\{v_j, v_{j+1}, \dots, v_k\}$, that is, an edge in $C_{right}(v_i)$. \square

The following observation is very useful in cutting critical chains. We formulate it as a lemma.

Lemma 5. *Let $C = \{v_j, v_{j+1}, \dots, v_k\}$, where $j < k$, be a chain of edges of P , and assume that the endpoints of the chain v_j and v_k are already free due to the cuts in Phase 0 and Phase 1. Assume that all edges of this chain have been cut along and are thus disconnected. Then each edge in the chain $\{v_j, v_{j+1}, \dots, v_k\}$ is free.*

Proof. Observe that there exist paths of free points (hence corridors of free space) connecting each of v_j and v_k with the free space outside $\text{conv}(Q)$. Since all edges of C have been disconnected, the part of R adjacent to any such edge can be removed and thus the edge is made free. \square

Now, we will show how to cut along and thus make free the edges of the critical chains. We consider the critical chains one by one in clockwise sequence starting from s .

Lemma 6. *Let $C_C = \{v_j, v_{j+1}, \dots, v_k\}$, for $k - j \geq 2$, be a critical chain in clockwise sequence. Then there exist at most $2(k - j - 1)$ cuts by which the $(k - j)$ edges of C_C can be made free. Over all critical chains of P , this takes at most $2t_3$ cuts.*

Proof. We move from v_j towards v_k . Recall that v_j and v_k are reflex and free, and v_{j+1} and v_{k-1} (possibly $v_{j+1} = v_{k-1}$) are convex. No other vertices in this chain are free. There may be other convex vertices in C_C . However, since P is cuttable, between any two convex vertices there is at least one reflex vertex. If $k - j = 2$ we apply two cuts, one from v_j and one from v_k and by Lemma 5 we are done with this chain.

If $k - j \geq 3$ there are at least two convex vertices in C_C , v_{j+1} and v_{k-1} , and at least one reflex vertex other than v_j and v_k . Hence $k - j \geq 4$. We apply two cuts from v_j to free (v_j, v_{j+1}) by Lemma 1. Then, for each reflex vertex in the sequence of C_C , we proceed as follows. Let v_i , for some $j + 2 \leq i \leq k - 2$, be the next reflex vertex. (In the first round $i = j + 2$ and v_{i-1} is convex.)

If v_i is already free, make cuts along its incident edges if not yet done previously: (i) if v_{i-1} is convex apply one cut along the edge (v_i, v_{i-1}) , otherwise v_{i-1} is reflex and also already free. (ii) apply two cuts along the edge (v_i, v_{i+1}) by Lemma 4; if v_{i+1} is reflex, this will make free v_{i+1} too.

Assume now that v_i is not free. We look into the edge (v_i, v_{i+1}) . We extend this edge from v_i until it hits the boundary of P . Let ℓ be this extended line segment and p be the point on the boundary of P where ℓ hits. At this moment, we make the following claims.

- *Claim 1.* The edges of $C_{right}(v_{i-1})$ are already free.
- *Claim 2.* ℓ hits an edge of $C_{right}(v_{i-1})$.

Assuming the above claims hold, p is free. We make free the edge (v_i, v_{i+1}) by two cuts from p by Lemma 4. This will make free v_i . If v_{i+1} is also reflex, this will make free v_{i+1} too. If v_{i-1} is convex, we apply a single cut along (v_i, v_{i-1}) , which disconnects v_{i-1} . By Claim 1, all edges of $C_{right}(v_{i-1})$ were disconnected, and due to the cut along (v_i, v_{i-1}) the edges of $C_{right}(v_i)$ are now disconnected. Since v_i is now free, by Lemma 5 all edges of $C_{right}(v_i)$ are now free.

We repeat the above procedure until all reflex vertices in C_C have been made free. Finally, we apply a single cut along (v_k, v_{k-1}) which disconnects the convex vertex v_{k-1} . By induction, it follows that all edges of $C_{right}(v_k)$ are now free. We now prove the claims.

Proof of Claim 1: The reachable chains of $C_{right}(v_{i-1})$ (preceding C_C) have already been made free in Phase 1 of our algorithm. The critical chains of $C_{right}(v_{i-1})$ (preceding C_C) are also free, because we process them in clockwise sequence. Finally, within C_C , we cut its edges in clockwise sequence. Therefore, all edges preceding v_{i-1} in clockwise order are free.

Proof of Claim 2: Recall that ℓ extends the edge (v_{i+1}, v_i) from v_i until it hits the boundary of P . Since v_i is reflex (as assumed), ℓ precedes (v_i, v_{i-1}) clockwise by at most π ; see Fig. 3. To prove that ℓ hits an edge of $C_{right}(v_{i-1})$, we look into the funnel $\mathcal{F}(v_{i-1}, v_i)$. Note that $\mathcal{F}(v_{i-1}, v_i)$ is non-degenerate, in the sense that none of v_{i-1} and v_i is the parent of the other in T ; recall that both are vertices of a critical chain that are leaves in T ; see Fig. 3(left). If ℓ is in the left side of $\pi(a, v_i)$, then the path $\pi(s, v_{i+1})$ must make a turn at v_i . But that would imply that v_i is an internal vertex of T , which contradicts that v_i is a vertex of a critical chain; see Fig. 3(left). It follows that ℓ enters this funnel. Then by Lemma 4, ℓ hits $C_{right}(v_{i-1})$; see Fig. 3(right).

We now count the total number of cuts used to make free the edges of C_C . Let x be the number of convex vertices in C_C . If $x = 1$, then we have used two cuts in total for two edges of C_C . Therefore, $k - j = 2$ and the lemma holds. For $x \geq 2$, we have used two cuts for the very first edge of C_C , which is (v_j, v_{j+1}) . Then for each edge (v_i, v_{i+1}) , for $j + 1 \leq i \leq k - 1$, we have used two cuts if v_i is reflex and one cut if v_i is convex. Since $x \geq 2$, we have used a total of $2((k - j) - x) + x = 2(k - j) - x \leq 2(k - j - 1)$ cuts.

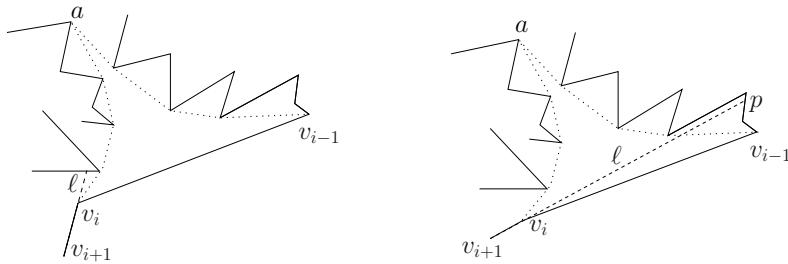


Fig. 3. Cutting the edges of a critical chain

The number of internal vertices in C_C is $k - j - 1$. Since two critical chains are edge-disjoint, they do not share any internal vertex. So we have $\sum_{C_C} 2(k - j - 1) = 2t_3$. Therefore, over all critical chains, we need at most $2t_3$ cuts. \square

Due to space limitations the time analysis of the algorithm is omitted.

References

1. Ahmed, S.I., Hasan, M., Islam, M.A.: Cutting a cornered convex polygon out of a circle. *Journal of Computers* 5(1), 4–11 (2010)
2. Ahmed, S.I., Hasan, M., Islam, M.A.: Cutting a convex polyhedron out of a sphere. *Graphs and Combinatorics* 27(3), 307–319 (2011)
3. Bereg, S., Dăescu, O., Jiang, M.: A PTAS for cutting out polygons with lines. *Algorithmica* 53, 157–171 (2009)
4. Bhadury, J., Chandrasekaran, R.: Stock cutting to minimize cutting length. *European Journal of Operations Research* 88, 69–87 (1996)
5. Chandrasekaran, R., Dăescu, O., Luo, J.: Cutting out polygons. In: *Proceedings of the Canadian Conference on Computational Geometry*, pp. 183–186 (2005)
6. Dăescu, O., Luo, J.: Cutting out polygons with lines and rays. *International Journal of Computational Geometry and Applications* 16, 227–248 (2006)
7. Demaine, E.D., Demaine, M.L., Kaplan, C.S.: Polygons cuttable by a circular saw. *Computational Geometry: Theory and Applications* 20, 69–84 (2001)
8. Dumitrescu, A.: An approximation algorithm for cutting out convex polygons. *Computational Geometry: Theory and Applications* 29, 223–231 (2004)
9. Dumitrescu, A.: The cost of cutting out convex n -gons. *Discrete Applied Mathematics* 143, 353–358 (2004)
10. Guibas, L.J., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica* 2, 209–233 (1987)
11. Lee, D.T., Preparata, F.P.: Euclidean shortest paths in the presence of rectilinear barriers. *Networks* 14(3), 393–410 (1984)
12. Overmars, M.H., Welzl, E.: The complexity of cutting paper. In: *Proceedings of the 1st Annual ACM Symposium on Computational Geometry*, pp. 316–321 (1985)
13. Tan, X.: Approximation Algorithms for Cutting Out Polygons with Lines and Rays. In: Wang, L. (ed.) *COCOON 2005. LNCS*, vol. 3595, pp. 534–543. Springer, Heidelberg (2005)

Fast Fréchet Queries

Mark de Berg¹, Atlas F. Cook IV², and Joachim Gudmundsson^{3,4}

¹ Department of Computing Science, TU Eindhoven, The Netherlands

² Dept. of Information and Computing Sciences, Utrecht University, The Netherlands

³ School of Information Technologies, University of Sydney, Australia

⁴ NICTA, Sydney, Australia

Abstract. Inspired by video analysis of team sports, we study the following problem. Let P be a polygonal path in the plane with n vertices. We want to preprocess P into a data structure that can quickly count the number of inclusion-minimal subpaths of P whose Fréchet distance to a given query segment Q is at most some threshold value ε . We present a data structure that solves an approximate version of this problem: it counts all subpaths whose Fréchet distance is at most ε , but this count may also include subpaths whose Fréchet distance is up to $(2 + 3\sqrt{2})\varepsilon$. For any parameter $n \leq s \leq n^2$, our data structure can be tuned such that it uses $O(s \text{polylog } n)$ storage and has $O((n/\sqrt{s}) \text{polylog } n)$ query time. For the special case where we wish to count all subpaths whose Fréchet distance to Q is at most $\varepsilon \cdot \text{length}(Q)$, we present a structure with $O(n \text{polylog } n)$ storage and $O(\text{polylog } n)$ query time.

1 Introduction

Motivation. Video analysis is becoming increasingly important in sports like soccer. Traditionally, video analysis is done manually: a person watches a video of a match and marks events that are statistically interesting. Manual analysis is labor intensive, however, and analyzing all league matches during an entire season is infeasible. Fortunately, computer vision and video processing have reached a level at which players and a ball can be tracked via video cameras. Existing technology generates the coordinates of all players and the ball with a frequency of 25Hz and an accuracy of 5cm [6], opening the possibility to automate parts of the process so that large collections of matches can be scrutinized interactively.

Our goal is to investigate one particular question a coach may wish to ask in an interactive analysis session: how often does a player run in a more or less straight line from a given position on the playing field to another given position. The input for our system is a large collection of polygonal paths. Each path describes the movements of one player during a single match. After the system has preprocessed these paths, a coach can use a mouse or a stylus to indicate a query segment from any start position to any end position, and the system can quickly count the number of subpaths that are similar to this query segment.

Background and problem statement. The above scenario requires a similarity measure to be defined between a query segment and a polygonal path. Much

research has been done on this topic, and numerous metrics such as the Hausdorff distance [28] and the Fréchet distance [3,7,9,10,12] have been proposed to measure the distance between two paths. Other distance measures include the longest common subsequence model [17], and a measure that combines parallel distance, perpendicular distance and angle distance [16]. Approaches have also compared curves by counting the number of common subsequences of length two [1] or using a single representative value to represent an entire curve [14].

The Fréchet distance is a distance measure for continuous shapes such as curves and surfaces. Since it takes the continuity of the shapes into account, the Fréchet distance is generally regarded as a more appropriate distance measure than the Hausdorff distance [45]. For this reason, the Fréchet distance is the distance measure that will be used in this paper.

The Fréchet distance for two directed curves $A, B \in \mathbb{R}^d$ is defined as follows. Imagine one person walks along A and another person walks along B . Each person must walk along his or her curve from start to finish. Neither person is allowed to stand still or travel backwards, but otherwise they are free to vary their speed. The *cost* of any fixed walk is the maximum distance that is attained between the two people at any time during the walk. Different walks can have different costs, and the Fréchet distance between A and B , denoted $\delta_F(A, B)$, equals the minimum possible cost over all walks. More formally, we have

$$\delta_F(A, B) = \inf_{\mu} \max_{a \in A} \text{dist}(a, \mu(a)),$$

where $\text{dist}(\cdot, \cdot)$ denotes Euclidean distance and $\mu : A \rightarrow B$ is a continuous one-to-one mapping that assigns to every point $a \in A$ a point $\mu(a) \in B$. Alt and Godau [3] have shown that the Fréchet distance can be computed in $O(nm \log nm)$ time when A and B are polygonal curves with n and m vertices, respectively.

Driemel et al. [13] recently showed how to compute a $(1 + \varepsilon)$ -approximation of the Fréchet distance for two so-called *c-packed* polygonal curves in near-linear time. Applications of the Fréchet distance exist in GIS when comparing the trajectories followed by moving objects [18] and in computational biology when comparing the backbones of large proteins [15].

In this paper, we study subpath-similarity *queries* with the Fréchet distance. We want to preprocess a polygonal path P into a data structure that can quickly count all distinct subpaths¹ of P whose Fréchet distance to a query segment Q is at most some threshold value ε . To the best of our knowledge, this problem has not been previously studied. Each *subpath* of P is a connected subset of P . Note that the endpoints of a subpath need not coincide with a vertex of P —they can lie in the interior of one of P 's segments. In general, if a subpath π has a small Fréchet distance to Q , then subpaths π' that are obtained by slightly moving the start or endpoint of π will have small Fréchet distance to Q as well. Thus, there can be many (even infinitely many) distinct subpaths with Fréchet distance to Q at most ε that are all essentially the same. We therefore count all

¹ In fact, our solution also works when we want to preprocess an entire collection of paths in order to count all subpaths in the collection whose Fréchet distance to Q is small. To simplify the presentation, we phrase our results in terms of a single path P .

inclusion-minimal subpaths whose Fréchet distance to Q is at most ε , that is, all subpaths π such that $\delta_F(Q, \pi) \leq \varepsilon$ and for which there does not exist any subpath $\pi' \subsetneq \pi$ with $\delta_F(Q, \pi') \leq \varepsilon$. We denote this collection of subpaths by $C_P(Q, \varepsilon)$. Essentially, $C_P(Q, \varepsilon)$ encodes all partial matches between P and Q that have Fréchet distance at most ε .

Our results. We start by proving that the subpaths in $C_P(Q, \varepsilon)$ are pairwise disjoint. Hence, subpaths in $C_P(Q, \varepsilon)$ are indeed distinct. This result holds provided that $\varepsilon < \text{length}(Q)/6$, a restriction we assume throughout the paper. We then proceed with the description of our data structure. Determining the exact number $|C_P(Q, \varepsilon)|$ appears to be hard. We therefore consider an approximate version of the problem. We can report a value k_ε such that $|C_P(Q, \varepsilon)| \leq k_\varepsilon \leq |C_P(Q, (2 + 3\sqrt{2})\varepsilon)|$. Our data structure uses $O(s \text{polylog } n)$ storage and has $O((n/\sqrt{s}) \text{polylog } n)$ query time, where s is a parameter with $n \leq s \leq n^2$ that can be used to trade storage for query time. For the special case where we want to count the number of inclusion-minimal subpaths whose Fréchet distance to Q is at most $\varepsilon \cdot \text{length}(Q)$, with $0 < \varepsilon < 1/6$, the data structure uses $O(n \text{polylog } n)$ storage and provides $O(\text{polylog } n)$ query time.

Due to space limitation some proofs are omitted in this version of the paper.

2 Preliminaries

We denote the start and endpoints of a path P by $\text{start}(P)$ and $\text{end}(P)$, respectively. Consider two points p, p' such that p is closer to $\text{start}(P)$ than p' along P . The subpath of P that starts at p and ends at p' is denoted $P(p, p')$.

Recall that $C_P(Q, \varepsilon)$ denotes the collection of inclusion-minimal subpaths of P with respect to a query segment Q and a threshold $\varepsilon > 0$.

Lemma 1. *Let π be a subpath of P such that $\delta_F(Q, \pi) \leq \varepsilon$. If $\varepsilon < \text{length}(Q)/2$, there is exactly one inclusion-minimal subpath $\pi^* \subseteq \pi$ with $\delta_F(Q, \pi^*) \leq \varepsilon$.*

Proof. Consider the disks D_1 and D_2 with radius ε that are, respectively, centered at the start and end points of Q . D_1 and D_2 are disjoint because $\varepsilon < \text{length}(Q)/2$. Let x_1 be the last point along π where π leaves D_1 , let x_2 be the first point after x_1 where π enters D_2 , and let $\pi^* = P(x_1, x_2)$; see Fig. II(a). We claim that $\delta_F(Q, \pi^*) \leq \varepsilon$. Indeed, consider a mapping $\mu : Q \rightarrow \pi$ that achieves² the Fréchet distance. Thus, $\max_{a \in Q} \text{dist}(a, \mu(a)) \leq \varepsilon$. Define the mapping $\mu^* : Q \rightarrow \pi^*$ to be the same mapping as μ , except that all points of Q that were mapped to points on π before x_1 are now mapped to x_1 , and all points of Q that were mapped to points on π after x_2 are now mapped to x_2 . Because $\text{dist}(\text{start}(Q), x_1) = \varepsilon$ and $\text{dist}(q, x_1) \leq \varepsilon$ for the first point $q \in Q$ that was already mapped to x_1 , all new points that are mapped to x_1 —these are the points on Q before q , have distance at most ε to x_1 . Similarly, all points that are now mapped to x_2 have

² Technically speaking there need not be such a path, since the Fréchet distance is defined as an infimum (rather than minimum) but this technicality can easily be handled.

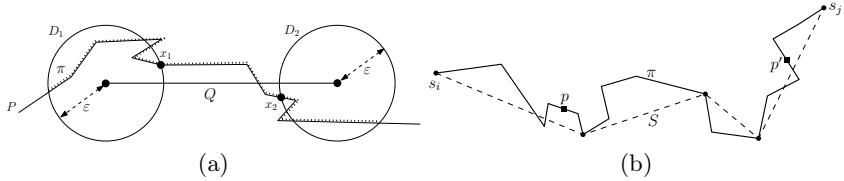


Fig. 1. (a) If $\delta_F(Q, \pi) \leq \varepsilon$, then the minimal subpath $\pi^* \subseteq \pi$ also has $\delta_F(Q, \pi^*) \leq \varepsilon$.
(b) The subpath $S(s_i, s_j)$ is called the *container* of $\pi = P(p, p')$ in S .

distance at most ε to x_2 . Moreover, μ^* corresponds to simultaneous walks along Q and π^* where we never move backwards. The mapping μ^* thus proves that $\delta_F(Q, \pi^*) \leq \varepsilon$.

It remains to prove that π^* is the *unique* inclusion-minimal subpath with this property. Since a subpath $\pi' \subseteq \pi$ with $\delta_F(Q, \pi') \leq \varepsilon$ must start in D_1 , and π does not re-enter D_1 after x_1 , the subpath π' cannot start after x_1 . Similarly, π' cannot end before x_2 . Hence, $\pi' \supseteq \pi^*$, proving that π^* is the unique inclusion-minimal subpath of π such that $\delta_F(Q, \pi^*) \leq \varepsilon$. \square

The following lemma shows the disjointness of inclusion-minimal subpaths. Note that two paths are considered to intersect even if the start of one coincides with the endpoint of the other.

Lemma 2. *The subpaths in $C_P(Q, \varepsilon)$ are pairwise disjoint.*

3 The Data Structure

Our data structure is based on the following idea: if we simplify P sufficiently, then any subpath π of the simplification whose Fréchet distance to Q is at most ε has at most three segments. We show how to determine all 3-segment, 2-segment, or 1-segment subpaths on the simplification whose Fréchet distance to Q is small. Standard data-structures can then be used to count the number of subpaths that satisfy this condition. We now make these ideas precise.

Let p_1, \dots, p_n be the sequence of vertices defining the given polygonal path P . For two vertices p_i, p_j with $i < j$, we define the error of the shortcut segment $p_i p_j$ as $\text{error}(p_i p_j) := \delta_F(p_i p_j, P(p_i, p_j))$. A *simplification* S of P is a polygonal path with vertices $\{s_1, \dots, s_m\} \subseteq \{p_1, \dots, p_n\}$ such that $s_1 = p_1$ and $s_m = p_n$.

An ε -*simplification* of P is a simplification s_1, \dots, s_m such that the error of each shortcut segment $s_i s_{i+1}$ is at most ε . We say that an ε -simplification S is *maximally simplified* if $\text{error}(s_i s_{i+2}) > \varepsilon$ for all $1 \leq i \leq m - 2$; in other words, removing any interior vertex of S will result in a shortcut with error more than ε .

Consider an arbitrary subpath $\pi \in P$ whose Fréchet distance to Q is at most ε . We next show that π consists of at most three segments when P is a maximally simplified (2ε) -simplification. We first define a correspondence between subpaths of P and subpaths in a given simplification $S = s_1, \dots, s_m$ of P . Let $\pi = P(p, p')$ be any subpath of P . Walk backwards along P , starting from p , until a vertex

s_i of S is encountered. Similarly, walk forward from p' until a vertex s_j of S is encountered. We call the subpath $S(s_i, s_j)$ the *container* of π in S , see Fig. 2(b). We define the container (in S) of a subpath $\pi \subseteq S$ in a similar way: let s_i be the vertex of S such that $\text{start}(\pi) \in s_i s_{i+1}$ and $\text{start}(\pi) \neq s_{i+1}$, and let s_j be the vertex of S such that $\text{end}(\pi) \in s_{j-1} s_j$ and $\text{end}(\pi) \neq s_{j-1}$. In other words, s_i is the first vertex of the link containing $\text{start}(\pi)$ (or $\text{start}(\pi)$ itself, if that is a vertex), and s_j is the last vertex of the link containing $\text{end}(\pi)$ (or $\text{end}(\pi)$ itself, if that is a vertex). Then the container of π is $S(s_i, s_j)$.

Lemma 3. *Let S be a maximally simplified (2ϵ) -simplification of P .*

- (i) *Let $\pi \in C_P(Q, \epsilon)$. The container of π in S has at most three segments, and the container has a subpath π^* whose Fréchet distance to Q is at most 3ϵ .*
- (ii) *If $\pi^* \subseteq S$ is a subpath whose container is $S(s_i, s_j)$ and $\delta_F(\pi^*, Q) \leq \epsilon$, then there is a subpath $\pi \in C_P(Q, 3\epsilon)$ whose container is $S(s_i, s_j)$.*

Proof. To prove (i), assume for a contradiction that the container of π in S has more than three segments. Let s_i be the first vertex of the container. Hence, the vertices $s_{i+1}, s_{i+2}, s_{i+3}$ are interior vertices of the container, so they are vertices of π , see Fig. 2(a). Consider a mapping $\mu_\pi : \pi \rightarrow Q$ such that $\max_{a \in \pi} \text{dist}(a, \mu_\pi(a)) \leq \epsilon$. Let q_1, q_2 , and q_3 be any points along Q such that $q_1 := \mu_\pi(s_{i+1})$, $q_2 := \mu_\pi(s_{i+2})$ and $q_3 := \mu_\pi(s_{i+3})$. By definition, we have:

$$\delta_F(\pi(s_{i+1}, s_{i+3}), q_1 q_3) \leq \epsilon. \quad (1)$$

Now consider the Fréchet distance between the segment $q_1 q_3 \subseteq Q$ and the shortcut segment $s_{i+1} s_{i+3}$. By Alt and Godau [3], this Fréchet distance equals $\max\{\text{dist}(q_1, s_{i+1}), \text{dist}(q_3, s_{i+3})\}$, thus $\delta_F(s_1 s_3, q_1 q_3) \leq \epsilon$. Since the Fréchet distance satisfies the triangle inequality, we can combine this with (I) and get

$$\delta_F(s_{i+1} s_{i+3}, \pi(s_{i+1} s_{i+3})) \leq \delta_F(\pi(s_{i+1} s_{i+3}), q_1 q_3) + \delta_F(s_1 s_3, q_1 q_3) \leq 2\epsilon.$$

However, this is a contradiction since S is a maximally simplified (2ϵ) -simplification of P and, hence, the assumption that the container of π in S contains at least three vertices is wrong. That completes the first part of the lemma.

To prove part (ii) of the lemma, we observe that if $\pi^* \subseteq S$ and π^* has container $S(s_i, s_j)$, then there is a subpath π of P with container $S(s_i, s_j)$ such that $\delta_F(\pi, \pi^*) \leq \epsilon$. The result now follows from the triangle inequality. \square

Let $S = s_1, \dots, s_m$ be a maximally simplified (2ϵ) -simplification, let $L_2(S) := \{S(s_i, s_{i+2}) : 1 \leq i \leq m-2\}$, and $L_3(S) := \{S(s_i, s_{i+3}) : 1 \leq i \leq m-3\}$. That is, $L_i(S)$ is the collection of all i -segment subpaths of S . Lemma 3 implies that it is sufficient to consider the sets $L_1(S), L_2(S), L_3(S)$ to find the subpaths with Fréchet distance to Q at most ϵ . Next, we show how to filter out the relevant k -segment subpaths from the $L_k(S)$'s.

Consider the query segment Q , and assume that $\epsilon < \text{length}(Q)/6$. Let $R(Q)$ denote the rectangle whose top and bottom edge are parallel to Q , whose center is the midpoint of Q , whose width is $|Q| - 6\epsilon$, and whose height is 2ϵ . We also

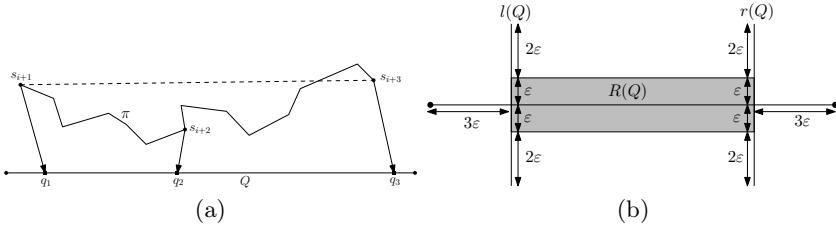


Fig. 2. (a) Whenever the container of π in S contains three or more vertices, we can use shortcircuiting to obtain a simplified curve containing at most two interior vertices. (b) The rectangle $R(Q)$ and the two antennas $l(Q)$ and $r(Q)$ for a query segment Q .

define two segments $l(Q)$ and $r(Q)$ which contain the edges of $R(Q)$ orthogonal to Q , have their midpoint on Q , and have length 6ε ; see Fig. 2(b). We call $l(Q)$ and $r(Q)$ the *antennas* of the $R(Q)$. Next, we give conditions that can be tested to check whether an i -segment subpath has a small Fréchet distance to Q .

Lemma 4. *Let $S(s_i, s_j)$ be a subpath with s_i lying to the left of the line containing $l(Q)$ and s_j lying to the right of the line containing $r(Q)$, and all other vertices lying between those lines. There is at most one subpath $\pi \in C_P(Q, \varepsilon)$ whose container is $S(s_i, s_j)$.*

Now we can state for $1 \leq k \leq 3$ the condition, denoted Condition (k) , used to filter out the relevant k -segment subpaths $S(s_i, s_{i+k})$ from $L_k(S)$.

- Condition (1): $s_i s_{i+1}$ intersects first $l(Q)$ and then $r(Q)$.
- Condition (2): $s_i s_{i+1}$ intersects $l(Q)$, $s_{i+1} s_{i+2}$ intersects $r(Q)$, and $s_{i+1} \in R(Q)$.
- Condition (3): $s_i s_{i+1}$ intersects $l(Q)$, and $s_{i+2} s_{i+3}$ intersects $r(Q)$, and $s_{i+1} \in R(Q)$, and $s_{i+2} \in R(Q)$; moreover, either $\text{length}(s_{i+1} s_{i+2}) \leq 2\varepsilon$ or the angle between the directed segments $s_{i+1} s_{i+2}$ and Q is less than $\pi/2$.

The length and angle constraints in Condition (3) ensure that $S(s_i, s_{i+3})$ does not reverse its course too far. This is necessary because otherwise the Fréchet distance to Q could depend on the width of the rectangle $R(Q)$.

Lemma 5

- (i) *If $S(s_i, s_j)$ satisfies one of the above conditions, then there is at least one minimal subpath in $C_P(Q, (2 + 3\sqrt{2})\varepsilon)$ whose container is $S(s_i, s_j)$.*
- (ii) *If $S(s_i, s_j)$ does not satisfy any of the above conditions, then no subpath π of P with container $S(s_i, s_j)$ has $\delta_F(Q, \pi) \leq \varepsilon$.*

Proof. For simplicity we assume Q is horizontal and directed left to right.

To prove (i), suppose $S(s_i, s_j)$, $1 \leq j \leq 3$, satisfies one of the above conditions. Let $y_1 = l(Q) \cap s_i s_{i+1}$ and let $y_2 = r(Q) \cap s_{i+1} s_{i+2}$. Either (a) $S(s_i, s_j)$ is x -monotone or (b) $j = 3$ and $s_{i+1} s_{i+2}$ is directed from right to left.

For case (a), consider a mapping $\mu : Q \rightarrow S(y_1, y_2)$ such that all points of Q to the left or on $l(Q)$ are mapped to y_1 , all points of Q to the right or on

$r(Q)$ are mapped to y_2 , and every other point $q \in Q$ is mapped to the point on $S(y_1, y_2)$ that has the same x -coordinate as q , as illustrated in Fig. 3(a). This is a valid mapping since $S(y_1, y_2)$ is x -monotone, and it guarantees that $\delta_F(S(y_1, y_2), Q) \leq 3\sqrt{2}\varepsilon$.

In case (b) we first define ℓ to be the vertical line splitting $s_{i+1}s_{i+2}$ into two equal length segments, see Fig. 3(b). Consider a mapping $\mu : Q \rightarrow S(y_1, y_2)$ such that all points of Q to the left or on $l(Q)$ are mapped to y_1 , all points of Q to the right or on $r(Q)$ are mapped to y_2 . Furthermore, every point q on $s_i s_{i+1}$ to the left of ℓ and every point q on $s_{i+2} s_{i+3}$ to the right of ℓ is mapped to the point on $S(y_1, y_2)$ with the same x -coordinate as q . Finally all points on Q between $\ell \cap s_i s_{i+1}$ and $\ell \cap s_{i+2} s_{i+3}$ are mapped to the point $\ell \cap S(y_1, y_2)$. This is also a valid mapping and guarantees that $\delta_F(S(y_1, y_2), Q) \leq 3\sqrt{2}\varepsilon$.

Since $\text{error}(S(s_i, s_j)) \leq 2\varepsilon$, there is a subpath $\pi \subseteq P(s_i, s_j)$ such that $\delta_F(\pi, S(y_1, y_2)) \leq 2\varepsilon$. Since the Fréchet distance satisfies the triangle inequality, we thus have $\delta_F(Q, \pi) \leq (2 + 3\sqrt{2})\varepsilon$.

To prove (ii) let π be any subpath whose container is $S(s_i, s_j)$ such that $\delta_F(Q, \pi) \leq \varepsilon$. By Lemma 3 the container $S(s_i, s_j)$ contains a subpath π^* with $\delta_F(\pi^*, Q) \leq 3\varepsilon$. In other words, there is a subsegment of $s_i s_{i+1}$ that starts in the disk of radius 3ε centered at $\text{start}(Q)$ and ends at s_{i+1} . There is also a subsegment $s_{j-1} s_j$ that starts at s_{j-1} and ends in the disk of radius 3ε centered at $\text{start}(Q)$. If $j = 3$ then there is also a segment between s_{i+1} and s_{j-1} . Thus, the path $S(s_i, s_j)$ intersects first $l(Q)$, then $r(Q)$, and all vertices in between (if any) lie within $R(Q)$. However, this means that $S(s_i, s_j)$ satisfies one of the above conditions. This contradiction completes the proof for $j = 1$ and $j = 2$.

It remains to consider the second part of the proof when $j = 3$. Assume that $\text{dist}(s_{i+1}, s_{i+2}) > 2\varepsilon$ and that the angle between the directed segments $s_{i+1}s_{i+2}$ and Q is at least $\pi/2$. Assume there exists a mapping $\mu_\pi : Q \rightarrow \pi$ such that $\max_{a \in \pi} \text{dist}(a, \mu_\pi(a)) \leq \varepsilon$. Consider the two disjoint disks D_{i+1} and D_{i+2} of radii ε with center at s_{i+1} and s_{i+2} , respectively. Let q_1 be the first point along Q such that $q_1 = \mu_\pi(s_{i+1})$. Let q_2 be any point along Q such that $q_2 = \mu_\pi(s_{i+2})$. Since D_{i+1} and D_{i+2} are disjoint, q_2 can not be encountered before q_1 along Q . However, since the angle between the directed segments $s_{i+1}s_{i+2}$ and Q is at least $\pi/2$, s_{i+2} must lie to the left of s_{i+1} . Consequently $D_{i+2} \cap Q$ must lie to the left of $D_{i+1} \cap Q$, a contradiction. This implies that the path $S(s_i s_{i+3})$ must satisfy Condition (3). \square

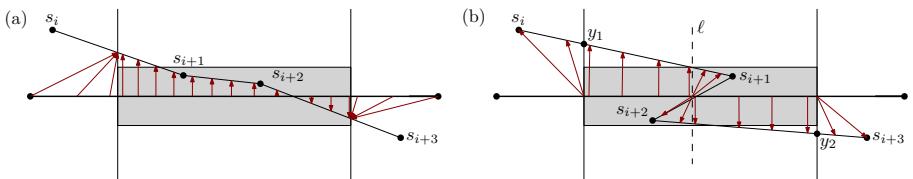


Fig. 3. Illustrating the proof of Lemma 5

Lemmas 3, 4 and 5 together imply that it is sufficient to count the 1-segment, 2-segment, and 3-segment subpaths of S that, respectively, satisfy conditions (1), (2), and (3). Suppose for now that ε is not part of the query but is fixed beforehand. First, we compute a maximally simplified (2ε) -simplification S of P . From S , we compute the sets $L_1(S)$, $L_2(S)$ and $L_3(S)$. Each $L_i(S)$ is stored in a data structure that allows us to count the number of i -segment subpaths that satisfy the relevant condition. Such a data structure can be designed using standard techniques. In particular, each of the conditions can be handled using a multi-level, 2-dimensional partition tree. The storage and query time of a multi-level 2-dimensional partition tree are $O(s \text{polylog } n)$ and $O((n/\sqrt{s}) \text{polylog } n)$, respectively, where s is a parameter with $n \leq s \leq n^2$ [11]. The following lemma summarizes the results so far.

Lemma 6. *Suppose that ε is fixed beforehand and is not part of the query. We can construct, for any $n \leq s \leq n^2$, a data structure that uses $O(s \text{polylog } n)$ storage and can, for a query segment Q with $\text{length}(Q) > 6\varepsilon$, determine in $O((n/\sqrt{s}) \text{polylog } n)$ time a value k_ε with $|C_P(Q, \varepsilon)| \leq k_\varepsilon \leq |C_P(Q, (2+3\sqrt{2})\varepsilon)|$.*

3.1 Making ε Part of the Query

Next, we explain how to deal with the case where ε is part of the query. Since we can no longer compute a fixed simplification beforehand, we start from the complete path $P = p_1, \dots, p_n$ and keep simplifying P in a greedy manner (always removing the vertex with the smallest error) until the simplification consists of the single line segment p_1p_n . During this process, various segments—links p_ip_{i+1} and shortcuts p_ip_j with $j > i + 1$ —will be used. For each segment, we record the range of values of ε for which it is part of the simplification. We call this range the *lifespan* of the segment p_ip_j and denote it by $\text{lifespan}(p_ip_j) = (\text{birth}(p_ip_j), \text{death}(p_ip_j))$.

Recall that we are not only interested in 1-segment subpaths of a simplification, but also in 2-segment and 3-segment subpaths. Thus, we also need to record the lifespan of these subpaths. Computing the lifespans is done using the following algorithm. We will use the notation $\text{pred}(p_i)$ and $\text{succ}(p_i)$ to denote the predecessor and successor of p_i in the current simplification S .

1. For each vertex p_i of P with $1 < i < n$, compute $\text{cost}(p_i) = \text{error}(p_ip_{i+2})$. Put all vertices p_i with $1 < i < n$ into a min-priority queue \mathcal{Q} with their cost as the priority. Initialize lists \mathcal{L}_1 , \mathcal{L}_2 , and \mathcal{L}_3 , where \mathcal{L}_m contains all m -segment subpaths of P . For each of these m -segment subpaths π , set $\text{birth}(\pi) = 0$. Initialize the initial simplification S by setting S equal to P .
2. While \mathcal{Q} is not empty, do the following: Remove a vertex p_i of lowest cost from \mathcal{Q} . Let $p_j := \text{pred}(p_i)$ and $p_l := \text{succ}(p_i)$. Remove p_i from S . For each \mathcal{L}_m , and for each subpath π that uses p_i —there are up to $m + 1$ such subpaths in \mathcal{L}_m —set $\text{death}(\pi) = \text{cost}(p_i)$. Moreover, create new subpaths that use p_jp_l as one of their links, and set $\text{birth}(\pi) = \text{cost}(p_i)$ for each of these subpaths π . Set $\text{cost}(p_j) := \text{error}(\text{pred}(p_j), \text{succ}(p_j))$ and $\text{cost}(p_l) := \text{error}(\text{pred}(p_l), \text{succ}(p_l))$.

- $= \text{error}(\text{pred}(p_i), \text{succ}(p_i))$. Here, $\text{pred}(\cdot)$ and $\text{succ}(\cdot)$ refer to the situation after the removal of p_i , so for example we have $\text{succ}(p_j) = p_l$.
3. Set $\text{death}(p_1 p_n) = \infty$.

The following lemma follows directly from the construction.

Lemma 7. *For any $\varepsilon > 0$ and $m = 1, 2, 3$, let $L_m(\varepsilon) \subset \mathcal{L}_m$ denote the set of m -segment subpaths π such that $\varepsilon \in \text{lifespan}(\pi)$. The segments in $L_1(\varepsilon)$ together form a maximally simplified ε -simplification $S(\varepsilon)$ of P . Moreover, the subpaths in \mathcal{L}_2 and \mathcal{L}_3 are exactly the 2-segment and 3-segment subpaths in $S(\varepsilon)$.*

Our final data structure now works as follows. We have a separate data structure for each \mathcal{L}_m . The main structure for \mathcal{L}_m is a segment tree T_m storing the lifespans of each of the m -link subpaths in \mathcal{L}_m . Each canonical subset is stored in an associated structure, which is a multi-level 2-dimensional partition tree that tests for the three conditions that we saw previously—that is, \mathcal{L}_m is used to find all m -segment matches. As compared to our data structure for a fixed ε , we just need to add one more level to identify the relevant subpaths in filter out the subpaths with valid lifespans. Adding the extra level increases the storage and query time by a logarithmic factor, leading to the following theorem.

Theorem 1. *For any $n \leq s \leq n^2$, there is a data structure of size $O(s \text{polylog } n)$ storage that can, for a query segment Q and threshold $\varepsilon > 0$ with $\text{length}(Q) > 6\varepsilon$, determine in $O((n/\sqrt{s}) \text{polylog } n)$ time a value k_ε with $|C_P(Q, \varepsilon)| \leq k_\varepsilon \leq |C_P(Q, (2 + 3\sqrt{2})\varepsilon)|$.*

3.2 Relative Error

Consider the case when the Fréchet distance to Q should be bounded by $\varepsilon \cdot \text{length}(Q)$. In this case, one can obtain a considerable speed-up by using range trees instead of partition trees. Unfortunately, range trees are only effective when answering orthogonal range queries. One way to get around this is to consider a fixed number of orientations. For example, we can choose $2\pi/\varepsilon$ orientations, and then generate a range tree for each fixed orientation.

Given a query Q , we find an orientation d whose angle to Q is minimized. If the orientations are uniformly distributed, then we can guarantee that the angle between Q and d is bounded by $\frac{\varepsilon}{2}$. Let $R_d(Q)$ be the rectangle that is orthogonal to d , with center at the midpoint of Q , width $(\text{length}(Q) \cdot (\cos \frac{\varepsilon}{2} + 2\varepsilon \sin \frac{\varepsilon}{2}))$, height $(\text{length}(Q) \cdot (\sin \frac{\varepsilon}{2} + 2\varepsilon \cos \frac{\varepsilon}{2}))$ and whose long sides are parallel to the orientation d . Define two segments $l_d(Q)$ and $r_d(Q)$ that are orthogonal to d , have their midpoint on Q , have length $\text{length}(Q) \cdot (\frac{3\varepsilon}{\cos \varepsilon/2})$, and overlap the edges of $R_d(Q)$. As before we call these segments the antennas of Q .

The results in Section 3 can be modified to hold for this case, and we obtain:

Theorem 2. *There is a data structure that uses $O(\frac{1}{\varepsilon}n \text{polylog } n)$ storage and that can, for a query segment Q , determine in $O(\frac{1}{\varepsilon} \text{polylog } n)$ time a value k_ε with $|C_P(Q, \varepsilon)| \leq k_\varepsilon \leq |C_P(Q, (2 + 3\sqrt{2})(1 + \varepsilon)\varepsilon \cdot \text{length}(Q))|$.*

Acknowledgment. We thank Herman Haverkort for discussions on this problem during the initial stages of our research.

References

1. Agrawal, R., Lin, K.-I., Sawhney, H.S., Shim, K.: Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In: 21st International Conference on Very Large Data Bases, pp. 490–501 (1995)
2. Alt, H., Behrends, B., Blomer, J.: Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence* (3-4), 251–265 (1995)
3. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. *Int. Journal of Computational Geometry & Applications* 5, 75–91 (1995)
4. Alt, H., Guibas, L.: Discrete geometric shapes: Matching, interpolation, and approximation. In: *Handbook of Comp. Geom.*, pp. 121–153. Elsevier (1999)
5. Alt, H., Knauer, C., Wenk, C.: Comparison of distance measures for planar curves. *Algorithmica* 38(2), 45–58 (2004)
6. Amisco, <http://www.sport-universal.com>
7. Aronov, B., Har-Peled, S., Knauer, C., Wang, Y., Wenk, C.: Frechet distances for curves, revisited. In: European Symposium on Algorithms, pp. 52–63 (2006)
8. Belogay, E., Cabrelli, C., Molter, U., Shonkwiler, R.: Calculating the Hausdorff distance between curves. *Information Processing Letters* 64(1), 17–22 (1997)
9. Buchin, K., Buchin, M., Gudmundsson, J.: Constrained free space diagrams: a tool for trajectory analysis. *Int. Journal of GIS* 24(7), 1101–1125 (2010)
10. Buchin, K., Buchin, M., Wang, Y.: Exact algorithms for partial curve matching via the Fréchet distance. In: 20th Symp. on Discrete Algorithm, pp. 645–654 (2009)
11. Chan, T.M.: Optimal partition trees. In: 26th Symposium on Computational Geometry (SoCG), New York, NY, USA, pp. 1–10 (2010)
12. Clausen, M., Mosig, A.: Approximately matching polygonal curves with respect to the Fréchet distance. *Comp. Geom. – Theory & Appl.* 30, 113–127 (2005)
13. Driemel, A., Har-Peled, S., Wenk, C.: Approximating the Fréchet distance for realistic curves in near linear time. 26th Symp. on Comp. Geom., SoCG (2010)
14. Kalpakis, K., Gada, D., Puttagunta, V.: Distance measures for effective clustering of arima timeseries. In: 1st IEEE Int. Conf. on Data Mining, pp. 273–280 (2001)
15. Kolodny, R., Koehl, P., Levitt, M.: Comprehensive evaluation of protein structure alignment: Scoring by geometric measures. *Journal of Molecular Biology* 346, 1173–1188 (2005)
16. Lee, J.-G., Han, J., Whang, K.-Y.: Trajectory clustering: a partition-and-group framework. In: ACM SIGMOD International Conference on Management of Data, pp. 593–604 (2007)
17. Gunopoulos, D., Vlachos, M., Kollios, G.: Discovering similar multidimensional trajectories. In: 18th Int. Conf. on Data Engineering, pp. 673–684 (2002)
18. Wenk, C., Salas, R., Pfoser, D.: Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In: 18th Conference on Scientific and Statistical Database Management (SSDBM), pp. 379–388 (2006)

Angle-Restricted Steiner Arborescences for Flow Map Layout*

Kevin Buchin, Bettina Speckmann, and Kevin Verbeek

Dep. of Mathematics and Computer Science, TU Eindhoven, The Netherlands
`{k.a.buchin,k.a.b.verbeek}@tue.nl, speckman@win.tue.nl`

Abstract. We introduce a new variant of the geometric Steiner arborescence problem, motivated by the layout of flow maps. Flow maps show the movement of objects between places. They reduce visual clutter by bundling lines smoothly and avoiding self-intersections. To capture these properties, our *angle-restricted Steiner arborescences*, or *flux trees*, connect several targets to a source with a tree of minimal length whose arcs obey a certain restriction on the angle they form with the source.

We study the properties of optimal flux trees and show that they are planar and consist of logarithmic spirals and straight lines. Flux trees have the *shallow-light property*. Computing optimal flux trees is NP-hard. Hence we consider a variant of flux trees which uses only logarithmic spirals. *Spiral trees* approximate flux trees within a factor depending on the angle restriction. Computing optimal spiral trees remains NP-hard, but we present an efficient 2-approximation, which can be extended to avoid “positive monotone” obstacles.

1 Introduction

Flow maps are a method used by cartographers to visualize the movement of objects between places [817]. One or more sources are connected to several targets by arcs whose thickness corresponds to the amount of flow between a source and a target. Good flow maps share some common properties. They reduce visual clutter by merging (bundling) lines as smoothly and frequently as possible. Furthermore, they strive to avoid crossings between lines. *Flow trees*, that is, single-source flows, are drawn entirely without crossings. Flow maps that depict trade often route edges along actual shipping routes. In addition, flow maps try to avoid covering important map features with flows to aid recognizability. Most flow maps are still drawn by hand and none of the existing algorithms (that use edge bundling), can guarantee to produce crossing-free flows.

In this paper we introduce a new variant of geometric minimal *Steiner arborescences*, which captures the essential structure of flow trees and serves as a “skeleton” upon which to build high-quality flow trees. Our input consists of

* B. Speckmann and K. Verbeek are supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707. A full version of the paper can be found at <http://arxiv.org/abs/1109.3316>

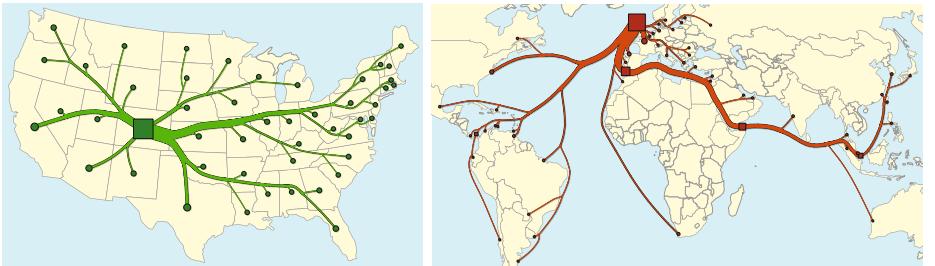


Fig. 1. Flow maps from our companion paper [6] based on angle-restricted Steiner arborescences: Migration from Colorado and whisky exports from Scotland

a point r , the *root* (source), and n points t_1, \dots, t_n , the *terminals* (targets). Visually appealing flow trees merge quickly, but smoothly. A geometric minimal Steiner arborescence on our input would result in the shortest possible tree, which naturally merges quickly. However, Steiner arborescences have angles of $2\pi/3$ at every internal node and hence are quite far removed from the smooth appearance of hand-drawn flow maps. Our goal is hence to connect the terminals to the root with a Steiner tree of minimal length whose arcs obey a certain restriction on the angle they form with the root.

Specifically, we use a *restricting angle* $\alpha < \pi/2$ to control the direction of the arcs of a Steiner arborescence T . Consider a point p on an arc e from a terminal to the root (see Fig. 2). Let γ be the angle between the vector from p to the root r and the tangent vector of e at p . We require that $\gamma \leq \alpha$ for all points p on T . We refer to a Steiner arborescence that obeys this angle restriction as *angle-restricted Steiner arborescence*, or simply *flux tree*. Here and in the remainder of the paper it is convenient to direct flux trees from the terminals to the root. Also, to simplify descriptions, we often identify the nodes of a flux tree T with their locations in the plane.

In the context of flow maps it is important that flux trees can avoid obstacles, which model important features of the underlying geographic map. Furthermore, it is undesirable that terminals become internal nodes of a flux tree. We can ensure that our trees never pass directly through terminals by placing a small triangular obstacle just behind each terminal (as seen from the root). Hence our input also includes a set of m obstacles B_1, \dots, B_m . We denote the total complexity (number of vertices) of all obstacles by M . In the presence of obstacles our goal is to find the shortest flux tree T that is planar and avoids the obstacles.

The edges of flux trees are by definition “thin”, but their topology and general structure are very suitable for flow trees. In a companion paper [6] we describe an algorithm that thickens and smoothes a given flux tree while avoiding obstacles.

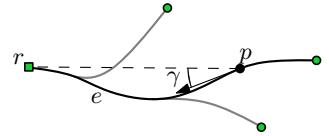


Fig. 2. The angle restriction

Fig. 11 shows two examples of the maps computed with our algorithm, further examples and a detailed discussion of our maps can be found in [6].

Related work. There is a multitude of related work on both the practical and the theoretical side of our problem and consequently we cannot cover it all.

One of the first systems for the automated creation of flow maps was developed by Tobler in the 1980s [11, 18]. His system does not use edge bundling and hence the resulting maps suffer from visual clutter. In 2005 Phan *et al.* [12] presented an algorithm, based on hierarchical clustering of the terminals, which creates flow trees with bundled edges. This algorithm uses an iterative ad-hoc method to route edges and is often unable to avoid crossings.

There are many variations on the classic Steiner tree problem which employ metrics that are related to their specific target applications. Of particular relevance to this paper is the *rectilinear Steiner arborescence* (RSA) problem, which is defined as follows. We are given a root (usually at the origin) and a set of terminals t_1, \dots, t_n in the northeast quadrant of the plane. The goal is to find the shortest rooted rectilinear tree T with all edges directed away from the root, such that T contains all points t_1, \dots, t_n . For any edge of T from $p = (x_p, y_p)$ to $q = (x_q, y_q)$ it must hold that $x_p \leq x_q$ and $y_p \leq y_q$. If we drop the condition of rectilinearity then we arrive at the *Euclidean Steiner arborescence* (ESA) problem. In both cases it is NP-hard [15, 16] to compute a tree of minimum length. Rao *et al.* [14] give a simple 2-approximation algorithm for minimum rectilinear Steiner arborescences. Córdova and Lee [7] describe an efficient heuristic which works for terminals located anywhere in the plane. Rammath [13] presents a more involved 2-approximation that can also deal with rectangular obstacles. Finally, Lu and Ruan [10] developed a PTAS for minimum rectilinear Steiner arborescences, which is, however, more of theoretical than of practical interest.

Conceptually related are *gradient-constrained minimum networks* which are studied by Brazil *et al.* [4, 5] motivated by the design of underground mines. Gradient-constrained minimum networks are minimum Steiner trees in three-dimensional space, in which the (absolute) gradients of all edges are no more than an upper bound m (so that heavy mining trucks can still drive up the ramps modeled by the Steiner tree). Krozel *et al.* [9] study algorithms for turn-constrained routing with thick edges in the context of air traffic control. Their paths need to avoid obstacles (bad weather systems) and arrive at a single target (the airport). The union of consecutive paths bears some similarity with flow maps, although it is not necessarily crossing-free or a tree.

Results and organization. In Section 2 we derive properties of optimal (minimum length) flux trees. In particular, we show that they are planar and that the arcs of optimal flux trees consist of (segments of) logarithmic spirals and straight lines. Flux trees have the *shallow-light property* [3], that is, we can bound the length of an optimal flux tree in comparison with a minimum spanning tree on the same set of terminals and we can give an upper bound on the length of a path between any point in a flux tree and the root. They also naturally induce a clustering on the terminals and smoothly bundle lines. Unfortunately we can show that it is NP-hard to compute optimal flux trees, the proof can

be found in the full version of the paper. Hence, in Section 3 we introduce a variant of flux trees, so called *spiral trees*. The arcs of spiral trees consist only of logarithmic spiral segments. We prove that spiral trees approximate flux trees within a factor depending on the restricting angle α . Our experiments show that $\alpha = \pi/6$ is a reasonable restricting angle, in this case the approximation factor is $\sec(\alpha) \approx 1.15$. Computing optimal spiral trees remains NP-hard. Hence in Section 4 we develop a 2-approximation algorithm for spiral trees which runs in $O(n \log n)$ time. Finally, in Section 5 we extend our approximation algorithm (without deteriorating the approximation factor) to include “positive monotone” obstacles. On the way, we develop a new 2-approximation algorithm for rectilinear Steiner arborescences in the presence of positive monotone obstacles. Both algorithms run in $O((n+M) \log(n+M))$ time, where M is the total complexity of all obstacles. Omitted proofs can be found in the full version of the paper.

2 Optimal Flux Trees

Recall that our input consists of a root r , terminals t_1, \dots, t_n , and a restricting angle $\alpha < \pi/2$. Without loss of generality we assume that the root lies at the origin. Recall further that an optimal flux tree is a geometric Steiner arborescence, whose arcs are directed from the terminals to the root and that satisfies the angle restriction. We show that the arcs of an optimal flux tree consist of line segments and parts of logarithmic spirals (Property 1), that any node except for the root has at most two incoming arcs (Property 2), and that an optimal flux tree is planar (Property 3). Finally, flux trees (and also spiral trees) have the shallow-light property (Property 4).

Spiral regions. For a point p in the plane, we consider the region \mathcal{R}_p of all points that are *reachable* from p with an *angle-restricted* path, that is, with a path that satisfies the angle restriction. Clearly, the root r is always in \mathcal{R}_p . The boundaries of \mathcal{R}_p consist of curves that follow one of the two directions that form exactly an angle α with the direction towards the root. Curves with this property are known as *logarithmic spirals* (see Fig. 3). Logarithmic spirals are self-similar; scaling a logarithmic spiral results in another logarithmic spiral. Logarithmic spirals are also self-approaching as defined by Aichholzer *et al.* [2], who give upper bounds on the lengths of (generalized) self-approaching curves. As all spirals in this paper are logarithmic, we simply refer to them as *spirals*. For $\alpha < \pi/2$ there are two spirals through a point. The *right spiral* \mathcal{S}_p^+ is given by the following parametric equation in polar coordinates, where $p = (R, \phi)$: $R(t) = Re^{-t}$ and $\phi(t) = \phi + \tan(\alpha)t$. The parametric equation of the *left spiral* \mathcal{S}_p^- is the same with α replaced by $-\alpha$. Note that a right spiral \mathcal{S}_p^+ can never cross another right spiral \mathcal{S}_q^+ (the same holds for left spirals). The spirals \mathcal{S}_p^+ and \mathcal{S}_p^- cross infinitely often. The reachable region \mathcal{R}_p is bounded by

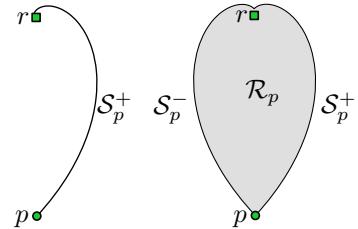


Fig. 3. Spirals and spiral regions

the parts of \mathcal{S}_p^+ and \mathcal{S}_p^- with $0 \leq t \leq \pi \cot(\alpha)$. We therefore call \mathcal{R}_p the *spiral region* of p . It follows directly from the definition that for all $q \in \mathcal{R}_p$ we have that $\mathcal{R}_q \subseteq \mathcal{R}_p$.

Lemma 1. *The shortest angle-restricted path between a point p and a point $q \in \mathcal{R}_p$ consists of a straight segment followed by a spiral segment. Either segment can have length zero.*

Property 1. An optimal flux tree consists of straight segments and spiral segments.

Property 2. Every node in an optimal flux tree, other than the root, has at most two incoming edges.

Property 3. Every optimal flux tree is planar.

Let $d^T(p)$ be the distance between p and r in a flux tree T and let $d(p)$ be the Euclidean distance between p and r .

Property 4. The length of an optimal flux tree T is at most $O((\sec(\alpha) + \csc(\alpha)) \log n)$ times the length of the minimum spanning tree on the same set of terminals. Also, for every point $p \in T$, $d^T(p) \leq \sec(\alpha)d(p)$.

3 Spiral Trees

Spiral trees are special flux trees that use only spiral segments of a given α . Below we prove that spiral trees approximate flux trees. Any arc of a spiral tree can switch between following its right spiral and following its left spiral an arbitrary number of times. The length of a spiral segment can easily be expressed in polar coordinates. Let $p = (R_1, \phi_1)$ and $q = (R_2, \phi_2)$ be two points on a spiral, then the distance $D(p, q)$ between p and q on the spiral is

$$D(p, q) = \sec(\alpha)|R_1 - R_2|. \quad (1)$$

Consider the shortest *spiral path*—using only spiral segments—between a point p and a point q reachable from p . The reachable region for p is still its spiral region \mathcal{R}_p , so necessarily $q \in \mathcal{R}_p$. The length of a shortest spiral path is given by Equation 1. The shortest spiral path is not unique, in particular, any sequence of spiral segments from p to q is shortest, as long as we move towards the root.

Theorem 1. *The optimal spiral tree T' is a $\sec(\alpha)$ -approximation of the optimal flux tree T .*

Proof. Let \mathcal{C}_R be a circle of radius R with the root r as center. A lower bound for the length of T is given by $L(T) \geq \int_0^\infty |T \cap \mathcal{C}_R| dR$, where $|T \cap \mathcal{C}_R|$ counts the number of intersections between the tree T and the circle \mathcal{C}_R . Using Equation 1,

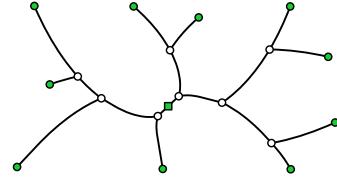


Fig. 4. An optimal flux tree ($\alpha = \pi/6$)

the length of T' is $L(T') = \sec(\alpha) \int_0^\infty |T' \cap \mathcal{C}_R| dR$. Now consider the spiral tree T'' with the same nodes as T , but where all arcs between the nodes are replaced by a sequence of spiral segments. For a given circle \mathcal{C}_R , this operation does not change the number of intersections of the tree with \mathcal{C}_R , i.e. $|T \cap \mathcal{C}_R| = |T'' \cap \mathcal{C}_R|$. So we get the following:

$$L(T') \leq L(T'') = \sec(\alpha) \int_0^\infty |T'' \cap \mathcal{C}_R| dR = \sec(\alpha) \int_0^\infty |T \cap \mathcal{C}_R| dR \leq \sec(\alpha) L(T)$$

□

Observation 1. *An optimal spiral tree is planar and every node, other than the root, has at most two incoming edges. The root has exactly one incoming edge.*

Relation with rectilinear Steiner arborescences. Both rectilinear Steiner arborescences and spiral trees contain directed paths, from the root to the terminals or vice versa. Every edge of a rectilinear Steiner arborescence is restricted to point right or up, which is similar to the angle restriction of flux and spiral trees. In fact, there exists a transformation from rectilinear Steiner arborescences into spiral trees; the details can be found in the full version of the paper. This transformation is not a bijection and does not preserve lengths. Thus the relation between the concepts cannot be used directly and algorithms developed for rectilinear Steiner arborescences cannot be simply modified to compute spiral trees. However, the same basic ideas can often be used in both settings.

4 Computing Spiral Trees

In this section we describe algorithms to compute (approximations of) optimal spiral trees. In the special case that the spiral regions of the terminals are empty, that is, if $t_i \notin \mathcal{R}_{t_j}$ for all $i \neq j$, we can use dynamic programming to compute an optimal spiral tree in $O(n^3)$ time. The details can be found in the full version of the paper; here we mention only a useful structural result.

Lemma 2. *If the spiral regions of all terminals are empty, then the leaf order of any planar spiral tree follows the radial order of the terminals.*

Rao *et al.* [14] describe a simple 2-approximation algorithm for rectilinear Steiner arborescences. The transformation mentioned above does not preserve length, so we cannot use this algorithm for spiral trees. However, below we show how to use the same global approach—sweep over the terminals from the outside in—to compute a 2-approximation for optimal spiral trees in $O(n \log n)$ time.

The basic idea is to iteratively join two nodes, possibly using a Steiner node, until all terminals are connected in a single tree T , the *greedy spiral tree*. Initially, T is a forest. We say that a node (or terminal) is *active* if it does not have a parent in T . In every step, we join the two active nodes for which the *join point* is farthest from r . The join point p_{uv} of two nodes u and v is the farthest point p from r such that $p \in \mathcal{R}_u \cap \mathcal{R}_v$. This point is unique if u , v and r are not collinear. Note that the *greedy spiral tree* is planar by construction.

The algorithm sweeps a circle \mathcal{C} , centered at r , inwards over all terminals. All active nodes that lie outside of \mathcal{C} form the *wavefront* \mathcal{W} (the black nodes in Fig. 5). \mathcal{W} is implemented as a balanced binary search tree, where nodes are sorted according to the radial order around r . We join two active nodes u and v as soon as \mathcal{C} passes over p_{uv} . For any two nodes $u, v \in \mathcal{W}$ it holds that $u \notin \mathcal{R}_v$. Because the greedy spiral tree is planar, we can apply Lemma 2 to the nodes in \mathcal{W} . Hence, when \mathcal{C} passes over p_{uv} and both nodes u and v are still active, then u and v must be neighbors in \mathcal{W} . We process the following events.

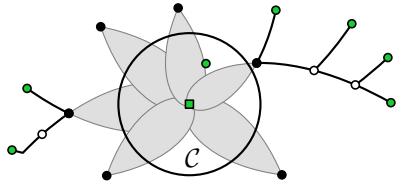


Fig. 5. The wavefront \mathcal{W}

Terminal. When \mathcal{C} reaches a terminal t , we add t to \mathcal{W} . We need to check whether there exists a neighbor v of t in \mathcal{W} such that $t \in \mathcal{R}_v$. If such a node v exists, then we remove v from \mathcal{W} and connect v to t . Finally we compute new join point events for t and its neighbors in \mathcal{W} .

Join point. When \mathcal{C} reaches a join point p_{uv} (and u and v are still active), we connect u and v to p_{uv} . Next, we remove u and v from \mathcal{W} and we add p_{uv} to \mathcal{W} as a Steiner node. Finally we compute new join point events for p_{uv} and its neighbors in \mathcal{W} .

We store the events in a priority queue \mathcal{Q} , ordered by decreasing distance to r . Initially \mathcal{Q} contains all terminal events. Every join point event adds a node to T and every node generates at most two join point events, so the total number of events is $O(n)$. We can handle a single event in $O(\log n)$ time, so the total running time is $O(n \log n)$. Next we prove that the greedy spiral tree is an approximation of the optimal spiral tree.

Lemma 3. *Let \mathcal{C} be any circle centered at r and let T and T' be the optimal spiral tree and the greedy spiral tree, respectively. Then $|\mathcal{C} \cap T'| \leq 2|\mathcal{C} \cap T|$ holds where $|\mathcal{C} \cap T'|$ is the number of intersection points between \mathcal{C} and T' .*

Proof. It is easy to see that $|\mathcal{C} \cap T'| = |\mathcal{W}|$ when the sweeping circle is \mathcal{C} . Let the nodes of \mathcal{W} be u_1, \dots, u_k , in radial order. Any node u_i is either a terminal or it is the intersection of two spirals originating from two terminals, which we call u_i^L and u_i^R (see Fig. 6). We can assume the latter is always the case, as we can set $u_i^L = u_i = u_i^R$ if u_i is a terminal. Next, let the intersections of T with \mathcal{C} be v_1, \dots, v_h , in the same radial order as u_1, \dots, u_k . As T has the same terminals as T' , every terminal u_i^L and u_i^R must be able to reach a point v_j . Let I_i^L and I_i^R be the reachable parts (intervals) of \mathcal{C} for u_i^L and u_i^R , respectively (that is $I_i^L = \mathcal{C} \cap \mathcal{R}_{u_i^L}$ and $I_i^R = \mathcal{C} \cap \mathcal{R}_{u_i^R}$). Since any two neighboring nodes u_i and u_{i+1} have not been joined by the greedy algorithm, we know that $I_i^L \cap I_{i+1}^R = \emptyset$. Now consider the collection \mathcal{S}_j of intervals that contain v_j . We always treat I_i^L and I_i^R as different intervals, even if they coincide. The union of all \mathcal{S}_j has cardinality $2k$. If $|\mathcal{S}_j| \geq 5$, then its intervals cannot be consecutive (i.e. $I_i^L, I_i^R, I_{i+1}^L, I_{i+1}^R$, etc.),

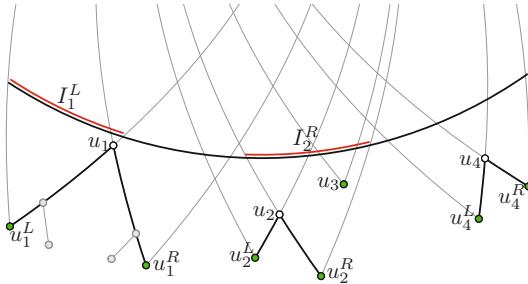


Fig. 6. Nodes $u_i \in \mathcal{W}$, terminals u_i^L, u_i^R and intervals I_i^L, I_i^R

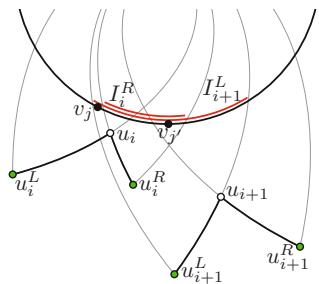


Fig. 7. $I_j^R \subset I_{j+1}^L$

as this would mean it contains both I_i^L and I_{i+1}^R for some i . So say \mathcal{S}_j contains I_i^L and I_{i+1}^L , but not I_i^R (other cases are similar). T' is planar, so this is possible only if $I_i^R \subset I_{i+1}^L$ (see Fig. 7). But then I_i^R and I_{i+1}^L are both in a collection $\mathcal{S}_{j'}$ and we can remove I_{i+1}^L from \mathcal{S}_j . This way we can construct reduced collections $\hat{\mathcal{S}}_j$ such that the union of all collections remains the same and all intervals in a collection $\hat{\mathcal{S}}_j$ are consecutive. As a result, $|\hat{\mathcal{S}}_j| \leq 4$, and hence $4h \geq 2k$ or $k \leq 2h$. \square

Theorem 2. *The greedy spiral tree is a 2-approximation of the optimal spiral tree and can be computed in $O(n \log n)$ time.*

5 Approximating Spiral Trees in the Presence of Obstacles

We now consider rectilinear Steiner arborescences and spiral trees in the presence of obstacles. Ramnath [13] gives a 2-approximation algorithm for rectilinear Steiner arborescences with rectangular obstacles. He claims that the result extends to arbitrary rectilinear obstacles. This is not the case; see the full version of this paper for details. In Section 5.1 we sketch an efficient algorithm to compute rectilinear Steiner arborescences in the presence of arbitrary polygonal obstacles. If the obstacles are *positive monotone* polygons—their boundary can be decomposed into two chains which are monotonically increasing—then the arborescence returned by our algorithm is a 2-approximation of the optimal rectilinear Steiner arborescence. In Section 5.2 we briefly explain how to use similar ideas to compute spiral trees in the presence of arbitrary polygonal obstacles. If the obstacles are *spiral monotone*—their boundary contains two points p and q such that both paths on the boundary from p to q are angle-restricted—then the spiral tree returned by our algorithm is a 2-approximation of the optimal spiral tree.

Below we sketch the algorithms for computing rectilinear Steiner arborescences and spiral trees in the presence of obstacles. We describe these algorithms in full detail in the full version of the paper.

5.1 Rectilinear Steiner Arborescences

We are given a root r at the origin, terminals t_1, \dots, t_n in the upper-right quadrant, and also m polygonal obstacles B_1, \dots, B_m with total complexity M . We place a bounding square around all terminals and the root and consider the “free space” between the obstacles as a polygonal domain P with m holes and $M+4$ vertices. We sketch a greedy algorithm that computes a rectilinear Steiner arborescence T , the *greedy arborescence*, inside P .

As before we incrementally join nodes until we have a complete arborescence. This time we sweep a diagonal line L over P towards r and maintain a wavefront \mathcal{W} with all active nodes that L has passed. If L reaches a join point p_{uv} of nodes $u, v \in \mathcal{W}$, we connect u and v to p_{uv} and add the new Steiner node to \mathcal{W} . Our greedy arborescence is restricted to grow inside the polygonal domain P , so if a point $p \in P$ cannot reach r with a monotone path in P , then p is not a suitable join point. To simplify matters we first compute a new polygonal domain P' from P , such that for every $p \in P'$, there is a monotone path from p to r in P .

To compute join points we keep track of the reachable part on L of every node $u \in \mathcal{W}$. As soon as two nodes $u, v \in \mathcal{W}$ can reach the same point p on L , then p is the join point p_{uv} and we can connect u and v to p_{uv} . We need extra information to properly connect u and v to p_{uv} , but we can easily maintain this during the sweep.

Theorem 3. *The greedy arborescence can be computed in $O((n+M) \log(n+M))$ time. If P has only positive monotone holes, then the greedy arborescence is a 2-approximation of the optimal rectilinear Steiner arborescence.*

5.2 Spiral Trees

To adapt our algorithm to compute a spiral tree, the *greedy spiral tree*, in the presence of polygonal obstacles, we need only a few changes. The sweep line is replaced by a sweeping circle \mathcal{C} . To trace out the reachable part on \mathcal{C} of a node $u \in \mathcal{W}$, we follow the intersections of \mathcal{S}_u^+ and \mathcal{S}_u^- with \mathcal{C} , instead of horizontal and vertical lines for rectilinear Steiner arborescences.

To properly trace out the reachable region of a node $u \in \mathcal{W}$ inside P , we need to deal with one subtlety. The reachable region of u can be bounded by both spirals and straight segments of P . The spirals that bound this region can originate from nodes or vertices, but also from the middle of an edge. In fact, for every edge e of P , there are at most two *spiral points* for which this can be the case. A point p on e is a spiral point if the angle between the line from p to r and the line through e is exactly α . We hence subdivide every edge of P at the spiral points. In addition we also subdivide e at the closest point to r on e to ensure that every edge of P has a single intersection with \mathcal{C} . The reachable part on \mathcal{C} of every node $u \in \mathcal{W}$ can then be maintained like for rectilinear Steiner arborescences and the algorithm proceeds in a similar fashion.

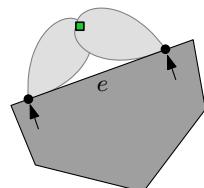


Fig. 8. Spiral points

Theorem 4. *The greedy spiral tree can be computed in $O((n + M) \log(n + M))$ time. If P has only spiral monotone holes, then the greedy spiral tree is a 2-approximation of the optimal spiral tree.*

References

1. CSISS - Spatial Tools: Tobler's Flow Mapper,
<http://www.csiss.org/clearinghouse/FlowMapper>
2. Aichholzer, O., Aurenhammer, F., Icking, C., Klein, R., Langetepe, E., Rote, G.: Generalized self-approaching curves. *Discr. Appl. Mathem.* 109(1-2), 3–24 (2001)
3. Awerbuch, B., Baratz, A., Peleg, D.: Cost-sensitive analysis of communication protocols. In: Proc. 9th ACM Symposium on Principles of Distributed Computing, pp. 177–187. ACM (1990)
4. Brazil, M., Rubinstein, J.H., Thomas, D.A., Weng, J.F., Wormald, N.C.: Gradient-constrained minimum networks. I. Fundamentals. *Journal of Global Optimization* 21, 139–155 (2001)
5. Brazil, M., Thomas, D.A.: Network optimization for the design of underground mines. *Networks* 49, 40–50 (2007)
6. Buchin, K., Speckmann, B., Verbeek, K.: Flow map layout via spiral trees. *IEEE Transactions on Visualization and Computer Graphics* (to appear, 2011) (Proceedings Visualization / Information Visualization 2011)
7. Córdova, J., Lee, Y.: A heuristic algorithm for the rectilinear Steiner arborescence problem. Technical report, Engineering Optimization (1994)
8. Dent, B.D.: *Cartography: Thematic Map Design*, 5th edn. McGraw-Hill, New York (1999)
9. Krozel, J., Lee, C., Mitchell, J.: Turn-constrained route planning for avoiding hazardous weather. *Air Traffic Control Quarterly* 14(2), 159–182 (2006)
10. Lu, B., Ruan, L.: Polynomial time approximation scheme for the rectilinear Steiner arborescence problem. *J. Comb. Optimization* 4(3), 357–363 (2000)
11. Mitchell, J.: L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica* 8, 55–88 (1992)
12. Phan, D., Xiao, L., Yeh, R., Hanrahan, P., Winograd, T.: Flow map layout. In: Proc. IEEE Symposium on Information Visualization, pp. 219–224 (2005)
13. Ramnath, S.: New approximations for the rectilinear Steiner arborescence problem. *IEEE Trans. Computer-Aided Design Integ. Circuits Sys.* 22(7), 859–869 (2003)
14. Rao, S., Sadayappan, P., Hwang, F., Shor, P.: The rectilinear Steiner arborescence problem. *Algorithmica* 7, 277–288 (1992)
15. Shi, W., Su, C.: The rectilinear Steiner arborescence problem is NP-complete. In: Proc. 11th ACM-SIAM Symposium on Discrete Algorithms, pp. 780–787 (2000)
16. Shi, W., Su, C.: The rectilinear Steiner arborescence problem is NP-complete. *SIAM Journal on Computing* 35(3), 729–740 (2005)
17. Slocum, T.A., McMaster, R.B., Kessler, F.C., Howard, H.H.: *Thematic Cartography and Geovisualization*, 3rd edn. Pearson, New Jersey (2010)
18. Tobler, W.: Experiments in migration mapping by computer. *The American Cartographer* 14(2), 155–163 (1987)

Treemaps with Bounded Aspect Ratio*

Mark de Berg¹, Bettina Speckmann¹, and Vincent van der Weele²

¹ Dep. of Mathematics and Computer Science, TU Eindhoven, The Netherlands

{mdbberg,speckman}@win.tue.nl

² Max-Planck-Institut für Informatik, Saarbrücken, Germany

vdweele@mpi-inf.mpg.de

Abstract. Treemaps are a popular technique to visualize hierarchical data. The input is a weighted tree \mathcal{T} where the weight of each node is the sum of the weights of its children. A treemap for \mathcal{T} is a hierarchical partition of a rectangle into simply connected regions, usually rectangles. Each region represents a node of \mathcal{T} and its area is proportional to the weight of the corresponding node. An important quality criterion for treemaps is the aspect ratio of its regions. One cannot bound the aspect ratio if the regions are restricted to be rectangles. In contrast, *polygonal partitions*, that use convex polygons, can have bounded aspect ratio. We are the first to obtain convex partitions with optimal aspect ratio $O(\text{depth}(\mathcal{T}))$. However, $\text{depth}(\mathcal{T})$ still depends on the input tree. Hence we introduce a new type of treemaps, namely *orthoconvex treemaps*, where regions representing leaves are rectangles, L-, and S-shapes, and regions representing internal nodes are orthoconvex polygons. We prove that any input tree, irrespective of the weights of the nodes and the depth of the tree, admits an orthoconvex treemap of constant aspect ratio.

1 Introduction

Treemaps are a very popular technique to visualize hierarchical data [12]. The input is a tree \mathcal{T} where every leaf is associated with a weight and where the weight of an internal node is the sum of the weights of its children. A treemap for \mathcal{T} is a hierarchical partition of a simple polygon, usually a rectangle, into simply connected regions, often rectangles as well. Each such region represents a node of \mathcal{T} and the area of each region is proportional to the weight of the corresponding node. To visualize the hierarchical structure the region associated with a node must contain the regions associated with its children. Shneiderman [13] and his colleagues were the first to present an algorithm for the automatic creation of rectangular treemaps. Treemaps have since been used to visualize hierarchical data from a variety of application areas, for example, stock market portfolios [8], tennis competitions trees [7], large photo collections [3], and business data [14].

* Bettina Speckmann was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.022.707. A full version of this paper can be found at <http://arxiv.org/abs/1012.1749>.

One of the most important quality criteria for treemaps is the aspect ratio of its regions; users find it difficult to compare regions with extreme aspect ratios [9]. Hence several approaches [3,4] try to “squarify” the regions of a rectangular treemap. However, one cannot bound the aspect ratio if the regions are restricted to be rectangles. (Consider a tree consisting of two leaves and a root and let the weight of one leaf tend to zero.) As a consequence, several types of treemaps using region shapes other than rectangles have been proposed. Balzer and Deussen [12] use centroidal Voronoi tessellations. Their algorithm is iterative and can give no guarantees on the aspect ratio of the regions (nor their exact size). Wattenberg [15] developed treemaps whose regions follow a space filling curve on a grid, so called Jigsaw maps. These assume the leaves to have integer weights which add up to a square number. The regions of the maps are rectilinear, but highly non-(ortho)convex. However, they do have aspect ratio 4. Onak and Sidiropoulos [11] introduced *polygonal partitions*, which use convex polygons. They proved an aspect ratio of $O((\text{depth}(\mathcal{T}) \cdot \log n)^{17})$ for a tree \mathcal{T} with n leaves. Together with De Berg, this bound has since been improved to $O(\text{depth}(\mathcal{T}) + \log n)$ [5]. That paper also gives a lower bound of $\Omega(\text{depth}(\mathcal{T}))$.

This leaves two open questions. First, is the $O(\log n)$ term in the upper bound on convex treemaps necessary, or can we guarantee a $O(\text{depth}(\mathcal{T}))$ aspect ratio? Second, is there a type of treemap that has constant aspect ratio for any input tree, irrespective of its depth and of the number of nodes and their weights?

Results and organization. We answer the two questions above affirmatively. First of all, in Section 2, we show how to construct convex partitions with optimal aspect ratio $O(\text{depth}(\mathcal{T}))$. Second, we introduce *orthoconvex treemaps*, where the regions representing internal nodes of the input tree are orthoconvex polygons, while the regions representing leaves are rectangles, L-shapes, and S-shapes (see Fig. 2). Thus our orthoconvex treemaps retain some of the schematized flavor of rectangular treemaps. In Section 3 we prove that any input tree admits an orthoconvex treemap of constant aspect ratio. Fig. 1 shows two treemaps constructed by our drawing algorithms. The hierarchy is emphasized by line thickness and color: thicker, darker lines delimit nodes higher in the hierarchy. All proofs omitted due to space constraints can be found in the full paper.

Preliminaries. Our input is a rooted tree \mathcal{T} . Following [5] we say that \mathcal{T} is *properly weighted* if each node ν of \mathcal{T} has a positive weight $\text{weight}(\nu)$ that equals the sum of the weights of the children of ν . We assume that the weights are normalized, that is, $\text{weight}(\text{root}(\mathcal{T})) = 1$. A treemap for \mathcal{T} associates a region

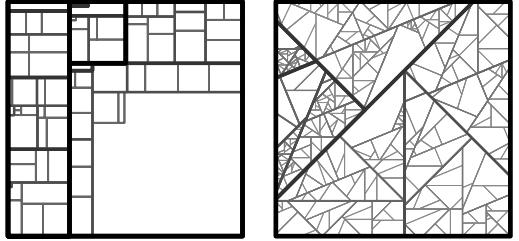


Fig. 1. Treemaps constructed by our drawing algorithms: orthoconvex and convex

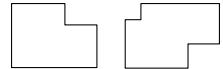


Fig. 2. An L- and an S-shape (two reflex corners)

$R(\nu)$ with each node $\nu \in \mathcal{T}$ such that (i) $R(\text{root}(\mathcal{T}))$ is the unit square, (ii) for every node we have $\text{area}(R(\nu)) = \text{weight}(\nu)$, and (iii) for any node ν , the regions associated with the children of ν form a partition of $R(\nu)$.

The aspect ratio of a treemap is the maximum aspect ratio of any of its regions. To simplify calculations, we use a different definition of aspect ratio for orthoconvex and for convex regions. Let R be a region and $\sigma(R)$ its smallest enclosing axis-aligned square, let $\text{area}(R)$ be its area and $\text{diam}(R)$ its diameter. For orthoconvex regions, we define the aspect ratio of R as $\text{asp}(R) := \text{area}(\sigma(R)) / \text{area}(R)$. For convex regions, we define it in accordance with [5] as $\text{asp}(R) := \text{diam}(R)^2 / \text{area}(R)$. Note that $\text{area}(\sigma(R)) \leq \text{diam}(R)^2 \leq 2 \cdot \text{area}(\sigma(R))$, so the aspect ratios obtained by the two definitions differ by at most a factor 2.

The following two lemmas deal with partitioning the children of a node according to weight and with partitioning a rectangular region. We denote the set of children of a node ν by $\text{children}(\nu)$.

Lemma 1. *Let all children of node ν have weight at most $t \cdot \text{weight}(\nu)$, for some $3/10 \leq t \leq 2/3$. Then we can partition $\text{children}(\nu)$ into subsets H_1 and H_2 , with*

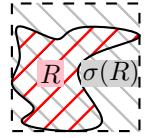
$$\text{weight}(H_2) \leq \text{weight}(H_1) \leq \begin{cases} 2t \cdot \text{weight}(\nu) & \text{if } 3/10 \leq t < 1/3; \\ 2/3 \cdot \text{weight}(\nu) & \text{if } 1/3 \leq t \leq 2/3. \end{cases}$$

Lemma 2. *Let R be a rectangle and $w_1 \geq w_2$ be weights such that $w_1 + w_2 = \text{area}(R)$. Then we can partition R into two subrectangles R_1, R_2 , such that $\text{asp}(R_i) \leq \max(\text{asp}(R), \text{area}(R)/w_i)$, for $i \in \{1, 2\}$.*

2 Convex Treemaps

We describe a recursive algorithm for computing a convex treemap (polygonal partition) of aspect ratio $O(\text{depth}(\mathcal{T}))$ for a properly weighted tree \mathcal{T} . Our algorithm has two phases. We first convert \mathcal{T} into a binary tree \mathcal{T}^* and then construct a partition for \mathcal{T}^* . Roughly speaking, at every step our algorithm finds a line to split a given convex polygon with “good” aspect ratio according to the weights of the two children of the current node. Both sub-polygons have good aspect ratio again. We cut with axis-aligned lines whenever we can. In fact we can ensure that we introduce new cutting directions only when encountering a new level of the original input tree instead of on every level of the binary tree. This is the key to obtaining $O(\text{depth}(\mathcal{T}))$ aspect ratio, rather than $O(\log n + \text{depth}(\mathcal{T}))$.

Converting to a binary tree. We recursively convert \mathcal{T} into a strictly binary tree \mathcal{T}^* , replacing each node with $k > 2$ children in \mathcal{T} by a binary subtree with $k - 1$ new internal nodes. During this process we assign a label $d(\nu)$ to each node ν , which corresponds to the depth of ν in \mathcal{T} . In a generic step, we treat a node ν with label $d(\nu)$, and our task is to convert the subtree rooted at ν . Initially $\nu = \text{root}(\mathcal{T})$ with $d(\text{root}(\mathcal{T})) = 0$. If ν is a leaf there is nothing to do. If ν has two children we recurse on these children and assign them label $d(\nu) + 1$. Otherwise



ν has k children, $\text{children}(\nu) = \{\nu_1, \dots, \nu_k\}$, for some $k > 2$. We distinguish two cases, depending on their weight.

If there is a “heavy” child, say ν_1 , such that $\text{weight}(\nu_1) \geq \text{weight}(\nu)/2$, then we proceed as follows. We turn ν into a binary node whose children are ν_1 and a new node μ_1 ; the children of μ_1 are ν_2, \dots, ν_k . We recurse on ν_1 and on μ_1 , with $d(\nu_1) = d(\nu) + 1$ and $d(\mu_1) = d(\nu)$. Otherwise all children have weight less than $\text{weight}(\nu)/2$, and hence there is a partition of $\text{children}(\nu)$ into two subsets S_1 and S_2 such that $\text{weight}(S_i) \leq 2/3 \cdot \text{weight}(\nu)$ for $i \in \{1, 2\}$. We turn ν into a binary node with children μ_1 and μ_2 , with children from S_1 and S_2 , respectively, and we recurse on μ_1 and μ_2 with $d(\mu_1) = d(\mu_2) = d(\nu)$.

Drawing a binary tree. Generalizing ϕ -separated polygons [5], we define a (k, ϕ) -polygon, with $k \geq 1$, to be a convex polygon P such that

- (i) P does not have parallel edges, except possibly two horizontal edges and two vertical edges. Moreover, each non-axis-parallel edge e makes an angle of at least ϕ with any other edge and also with the x -axis and the y -axis.
- (ii) If P has two horizontal edges, then $\text{width}(P)/\text{height}(P) \leq k$.
- (iii) If P has two vertical edges, then $\text{height}(P)/\text{width}(P) \leq k$.

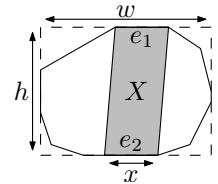
A ϕ -separated polygon can only have axis-parallel edges if these are parts of its axis-aligned bounding square [5]. Therefore, a (k, ϕ) -polygon P is a ϕ -separated polygon, if it respects the following:

- (a) if P has two horizontal edges, then $\text{height}(P) \geq \text{width}(P)$;
- (b) if P has two vertical edges, then $\text{width}(P) \geq \text{height}(P)$.

Note that a (k, ϕ) -polygon P is ϕ -separated if its bounding box is square.

Lemma 3. *Any (k, ϕ) -polygon has aspect ratio $O(\max(k, 1/\phi))$.*

Proof. Consider a (k, ϕ) -polygon P . For brevity, we write $w = \text{width}(P)$ and $h = \text{height}(P)$. Assume, without loss of generality, that $w \geq h$. Let e_1 and e_2 be the horizontal edges (possibly of length 0) and let $x = \min(|e_1|, |e_2|)$. Let X be the shaded parallelogram of width x . We distinguish two cases.



Case 1: $x > w/2$. P has two horizontal edges, so $h \geq w/k$. Clearly, $\text{area}(P) \geq \text{area}(X)$ which is $xh > w^2/(2k)$. The diameter of P is at most the diameter of the enclosing rectangle, hence $\text{diam}(P)^2 \leq w^2 + h^2 \leq 2w^2$. Combined:

$$\text{asp}(P) = \frac{\text{diam}(P)^2}{\text{area}(P)} \leq \frac{2w^2}{w^2/(2k)} = 4k = O(k).$$

Case 2: $x \leq w/2$. We obtain polygon P' from P by reducing the length of e_1 and e_2 by $\min(x, w - h)$. Clearly, $\text{area}(P') \leq \text{area}(P)$. Observe that P' is a ϕ -separated polygon since either it has at most one horizontal edge and $w - x \geq h$,

or the bounding box of P' is square. Therefore, $\text{asp}(P') = O(1/\phi)$ [5]. Using $\text{diam}(P) \leq w\sqrt{2}$ and $\text{diam}(P') \geq w - \min(x, w-h) \geq w-x \geq w/2$, we calculate

$$\text{asp}(P) = \frac{\text{diam}(P)^2}{\text{area}(P)} \leq \frac{2w^2}{\text{area}(P)} \leq 8 \cdot \frac{\text{diam}(P')^2}{\text{area}(P')} = 8 \cdot \text{asp}(P') = O(1/\phi). \quad \square$$

We construct the partition for \mathcal{T}^* in a top-down manner. Suppose we arrive at a node ν in \mathcal{T}^* , with associated region $R(\nu)$; initially $\nu = \text{root}(\mathcal{T}^*)$ and $R(\nu)$ is the unit square. We write $n(\nu)$ for the number of non-axis-parallel edges in $R(\nu)$. We maintain the following invariants:

- (Inv-1) $n(\nu) \leq d(\nu) + 4$;
- (Inv-2) $R(\nu)$ is a $(k, \phi(\nu))$ -polygon for $k = 4$ and $\phi(\nu) = \pi/(2(d(\nu) + 6))$.

Note that the invariant is satisfied for $\nu = \text{root}(\mathcal{T}^*)$. Now consider a node ν that is not the root of \mathcal{T}^* . If ν is a leaf, there is nothing to do. Otherwise, let ν_1 and ν_2 be the two children of ν . Assume without loss of generality that $\text{weight}(\nu_1) \geq \text{weight}(\nu_2)$. We distinguish two cases.

Case 1: $d(\nu_1) = d(\nu) + 1$. We consider the lines parallel to the edges of $R(\nu)$ through the origin. Moreover, we add the x - and y -axis. Since $R(\nu)$ has at most $d(\nu) + 4$ non-axis-parallel edges, we have at most $d(\nu) + 6$ lines in total. Hence, the biggest angular gap between two subsequent lines is at least $\pi/(d(\nu) + 6)$. Therefore, there is a line ℓ that makes an angle of at least $\pi/(2(d(\nu) + 6))$ with each of the edges of $R(\nu)$ and with the x - and the y -axis. Imagine placing the line ℓ such that it splits $R(\nu)$ into two halves of equal area, and define R' to be the half with the smallest number of non-axis-parallel edges. Now partition $R(\nu)$ into subpolygons $R(\nu_1)$ and $R(\nu_2)$ of the appropriate area with a cut c that is parallel to ℓ such that $R(\nu_2) \subset R'$. (Thus c lies inside R' .) We claim that both $R(\nu_1)$ and $R(\nu_2)$ satisfy the invariant.

Clearly $R(\nu_1)$ uses at most one edge more than $R(\nu)$. Since $d(\nu_1) = d(\nu) + 1$, this implies that (Inv-1) is satisfied for $R(\nu_1)$. Now consider the number of non-axis-parallel edges of $R(\nu_2)$. This is no more than the number of non-axis-parallel edges of R' . At most two non-axis-parallel edges are on both sides of ℓ , hence this number is bounded by

$$n(\nu_2) \leq \left\lfloor \frac{n(\nu) + 2}{2} \right\rfloor + 1 \leq \left\lfloor \frac{d(\nu) + 6}{2} \right\rfloor + 1 = \left\lfloor \frac{d(\nu)}{2} \right\rfloor + 4 \leq d(\nu) + 4 \leq d(\nu_2) + 4.$$

Given the choice of ℓ , and because $d(\nu_i) \geq d(\nu)$ and $R(\nu)$ satisfies (Inv-2), we know that the minimum angle between any two non-parallel edges of $R(\nu_i)$ ($i \in \{1, 2\}$) is at least $\pi/(2(d(\nu_i) + 6))$. To show that $R(\nu_1)$ and $R(\nu_2)$ satisfy (Inv-2), it thus suffices to prove the following lemma.

Lemma 4. *If $R(\nu_i)$ has two horizontal edges, then $\text{width}(R(\nu_i))/\text{height}(R(\nu_i)) \leq k$ and if $R(\nu_i)$ has two vertical edges, then $\text{height}(R(\nu_i))/\text{width}(R(\nu_i)) \leq k$, for $i \in \{1, 2\}$.*

Case 2: $d(\nu_1) = d(\nu)$. By construction of \mathcal{T}^* , $1/3 \cdot \text{weight}(\nu) \leq \text{weight}(\nu_1) \leq 2/3 \cdot \text{weight}(\nu)$. We now partition $R(\nu)$ into two subpolygons of the appropriate area with an axis-parallel cut orthogonal to the longest side of the axis-parallel bounding box of $R(\nu)$. The possible positions of this cut are limited by convexity, as specified in the following lemma.

Lemma 5. *Let P be a convex polygon with $\text{width}(P) \geq \text{height}(P)$. We can partition P with a vertical cut into subpolygons P_1, P_2 , with $\text{area}(P)/3 \leq \text{area}(P_i) \leq 2/3 \cdot \text{area}(P)$ (for $i \in \{1, 2\}$), such that $\text{width}(P)/4 \leq \text{width}(P_i) \leq 3/4 \cdot \text{width}(P)$.*

Clearly the number of non-axis-parallel edges of $R(\nu_1)$ and $R(\nu_2)$ is no more than the number of non-axis-parallel edges of $R(\nu)$. Since $d(\nu_i) \geq d(\nu)$, this implies $R(\nu_1)$ and $R(\nu_2)$ satisfy (Inv-1). As for (Inv-2), note that the cut does not introduce any new non-axis-parallel edges. It thus remains to prove the following lemma:

Lemma 6. *If $R(\nu_i)$ has two horizontal edges, then $\text{width}(R(\nu_i))/\text{height}(R(\nu_i)) \leq 4$ (for $i \in \{1, 2\}$). Similarly, if $R(\nu_i)$ has two vertical edges, $\text{height}(R(\nu_i))/\text{width}(R(\nu_i)) \leq 4$.*

Lemma 6 together with the fact that $\max_{\nu \in \mathcal{T}^*} d(\nu) = \text{depth}(\mathcal{T})$ and (Inv-2), implies the result.

Theorem 1. *Every properly weighted tree of depth d can be represented by a convex treemap (polygonal partition) which has aspect ratio $O(d)$.*

3 Ortho-Convex Treemaps

We describe a recursive algorithm for computing an orthoconvex treemap of constant aspect ratio for a properly weighted *binary* tree \mathcal{T} . If our original input tree is not binary, we can simply replace each node of degree $k > 2$ by a subtree of $k - 1$ new internal binary nodes. At every step our algorithm partitions the tree under consideration into several “well-sized” pieces which can be drawn inside rectangles, L-, and S-shapes of constant aspect ratio. The main difficulty is that subtrees may be split over several such pieces. We have to ensure that the fragments corresponding to the same subtree end up bordering each other so that the region corresponding to the subtree is orthoconvex. We solve this problem by *marking* certain corners and nodes during the recursive process, and letting our drawing algorithm be guided by the marked corners and nodes.

Our algorithm uses *staircases*: polygons defined by a horizontal edge uv , a vertical edge vw , and an xy -monotone chain of axis-parallel edges connecting u to w . The vertex v is called the *anchor* of the staircase. At each recursive step we are given a rectangle R with aspect ratio at most 8 and a tree \mathcal{T} . Exactly one node μ in \mathcal{T} and exactly one corner of R is *marked*. Initially \mathcal{T} is the input tree, R is the unit square, and the root of \mathcal{T} and the bottom-right corner of R are marked. We compute a treemap for \mathcal{T} inside R with the following properties:

- (i) every leaf is drawn as a rectangle of aspect ratio at most 8, or as an L- or S-shape of aspect ratio at most 32;

- (ii) every internal node is drawn as an orthoconvex polygon of aspect ratio at most 64;
- (iii) the marked node μ as well as its ancestors are drawn as staircases whose anchors coincide with the marked corner of R .

The third property is not a goal in itself, but it is necessary to guarantee the other two. We now describe how to draw \mathcal{T} inside R . Our algorithm distinguishes cases depending on the *relative weights* of certain nodes. The relative weight $\text{rel}(\nu)$ of a node ν is its weight as a fraction of the weight of the tree \mathcal{T} currently under consideration. We partition the relative weights into four categories:

- *tiny nodes*: nodes ν such that $\text{rel}(\nu) < 1/8$;
- *small nodes*: nodes ν such that $1/8 \leq \text{rel}(\nu) < 1/4$;
- *large nodes*: nodes ν such that $1/4 \leq \text{rel}(\nu) \leq 7/8$;
- *huge nodes*: nodes ν such that $\text{rel}(\nu) > 7/8$.

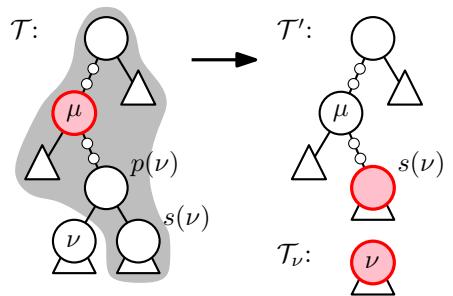
For a node ν we use \mathcal{T}_ν to denote the subtree rooted at ν . Moreover, we write $p(\nu)$ for the parent of ν and $s(\nu)$ for the sibling of ν .

Lemma 7. *Let ν be a non-tiny node in \mathcal{T} . Then \mathcal{T}_ν contains a non-tiny (in \mathcal{T}) leaf or a node that is small or large (in \mathcal{T}).*

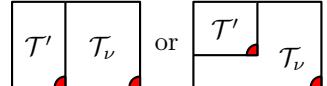
We now discuss the various cases that we distinguish, show how the algorithm handles them, and prove (using induction) that each case is handled correctly. In the base case \mathcal{T} consists of a single leaf node, so its region is simply the rectangle R . This trivially satisfies conditions (i)–(iii). So now assume \mathcal{T} has more than one node. In the following, whenever we mark a node in a tree that already has a marked node μ , we implicitly assume that the mark is removed from μ . In the description below—in particular in the figures illustrating our approach—we assume without loss of generality that the bottom-right corner of rectangle R is marked and that $\text{width}(R) \geq \text{height}(R)$.

Case (a): \mathcal{T} has a non-tiny marked node μ . By Lemma 7, \mathcal{T}_μ contains a node ν that is either a non-tiny leaf or a small or large internal node. Let \mathcal{T}' be the tree obtained from \mathcal{T} by removing \mathcal{T}_ν and contracting $s(\nu)$ into $p(\nu)$. In \mathcal{T}' we mark $s(\nu)$ and in \mathcal{T}_ν we mark ν . We distinguish two subcases.

If ν is not huge, then we split R into two subrectangles R' and $R(\nu)$, one for \mathcal{T}' and one for \mathcal{T}_ν . We put $R(\nu)$ to the right of R' and mark the bottom-right corner of both. Note that R' and $R(\nu)$ have aspect ratio at most 8, according to Lemma 2. We recursively draw the trees in their respective rectangles.

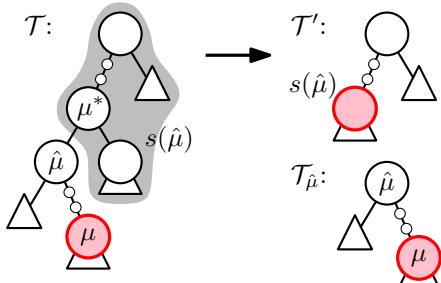


If ν is huge, then ν must be a leaf. We then draw ν as an L-shape $R(\nu)$ and recursively draw T' inside a rectangle R' which is similar to R , whose top-left corner coincides with the top-left corner of R , and whose bottom-right corner is marked.



In both subcases properties (i)–(iii) hold. The recursive calls only draw leaf regions having property (i) and the L-shaped leaf in the second subcase has aspect ratio at most $64/7$. Also, all internal nodes have property (ii). For the nodes that are not an ancestor of ν this follows by induction. For the ancestors of ν , orthoconvexity follows from the fact that the recursive call on T' has property (iii), and the relative positions of $R(\nu)$ and the marked corner of R' . Since ν is non-tiny and $\text{asp}(R) \leq 8$, the regions for the ancestors of ν have aspect ratio at most 64. The marked node μ and all of its ancestors have property (iii), because each such node is an ancestor of the marked node $s(\nu)$ in T' , the induction hypothesis, and the relative positions of $R(\nu)$ and the marked corner of R' .

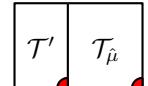
Case (b): T has a tiny marked node μ with an ancestor that is small or large. Let μ^* be the lowest huge ancestor of μ —since the root is huge, μ^* must exist—and let $\hat{\mu}$ be the child of μ^* on the path to μ . Then $\hat{\mu}$ is small or large. We obtain T' by removing $T_{\hat{\mu}}$ and contracting $s(\hat{\mu})$ into its parent. We mark $s(\hat{\mu})$ in T' and μ remains marked in $T_{\hat{\mu}}$.



We split R into two rectangles R' and $R(\hat{\mu})$, one for T' and one for $T_{\hat{\mu}}$. Since $\hat{\mu}$ is small or large, the aspect ratios of R' and $R(\hat{\mu})$ are at most 8, by Lemma 2. We put $R(\hat{\mu})$ to the right of R' , mark the bottom-right corner in both, and recursively draw T' in R' and $T_{\hat{\mu}}$ in $R(\hat{\mu})$.

The leaf regions have property (i) by induction. Property (ii) holds as well. For the nodes that are not an ancestor of $\hat{\mu}$ this follows by induction. For the ancestors of $\hat{\mu}$, orthoconvexity follows from the fact that the recursive call on T' has property (iii), and the relative positions of R' and the marked corner of $R(\hat{\mu})$. Because $\hat{\mu}$ is non-tiny, the regions for the ancestors of $\hat{\mu}$ have aspect ratio at least 64. Property (iii) follows from the fact that the recursive calls on T' and $T_{\hat{\mu}}$ have property (iii) and from the relative position of R' and the marked corners of $R(\hat{\mu})$ and R' .

Case (c): T has a tiny marked node μ without small or large ancestors, but T has a large or huge leaf λ . Define μ^* and $\hat{\mu}$ as in Case (b). Note that $\hat{\mu}$ must be tiny, since μ does not have small or large ancestors and μ^* is the lowest huge ancestor. Also note that λ must be in the other subtree of μ^* (the one not containing $\hat{\mu}$ and μ). Now we get T' from T by removing $T_{\hat{\mu}}$ and λ , and contracting $s(\hat{\mu})$ and $s(\lambda)$ into their parents. We mark $s(\lambda)$ in T' and keep μ marked in $T_{\hat{\mu}}$.



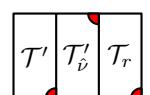
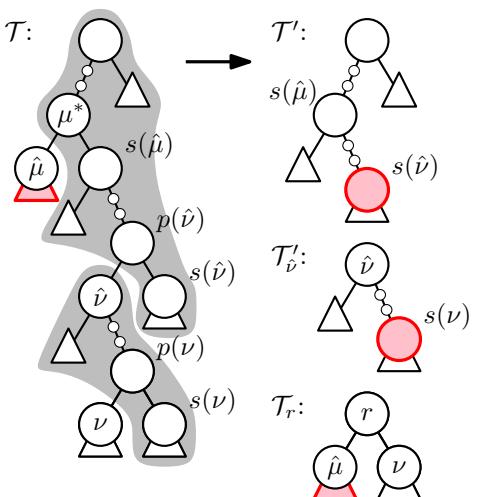
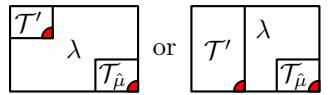
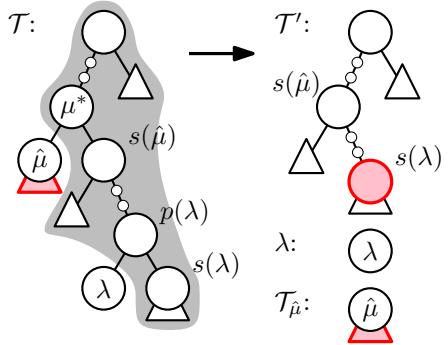
We draw $\mathcal{T}_{\hat{\mu}}$ in a rectangle $R(\hat{\mu})$ similar to R , in the bottom-right, with its bottom-right corner marked. If \mathcal{T}' is tiny we draw it as a rectangle R' similar to R , in the top-left of R ; otherwise \mathcal{T}' is drawn as a rectangle R' on the left side of R . Note that in the latter case the aspect ratio of R' is at most 8. In both cases we mark the bottom-right corner of R' . The region $R(\lambda)$ for λ is drawn in between R' and $R(\hat{\mu})$, which means it is either an S-shape or an L-shape.

All leaves in \mathcal{T}' and $\mathcal{T}_{\hat{\mu}}$ have property (i) by the induction hypothesis. Since λ is large or huge, the aspect ratio of $R(\lambda)$ is at most 32. Properties (ii) and (iii) follow using the induction hypothesis on \mathcal{T}' and $\mathcal{T}_{\hat{\mu}}$, as before.

Case (d): \mathcal{T} has a tiny marked node μ without small or large ancestors, and \mathcal{T} has no large or a huge leaf. Define μ^* and $\hat{\mu}$ as in Case (b) and (c). As in Case (c), $\hat{\mu}$ is tiny, so $\mathcal{T} \setminus \mathcal{T}_{\hat{\mu}}$ has relative weight at least $7/8$. We search for a node $\hat{\nu}$ such that $\text{rel}(\hat{\nu}) \leq 6/8$. This can be done by descending from μ^* , always proceeding to the heavier child, until we reach such a node. Observe that $\text{rel}(\hat{\nu}) \geq 3/8$. Let \mathcal{T}' be obtained by removing $\mathcal{T}_{\hat{\mu}}$ and $\mathcal{T}_{\hat{\nu}}$, marking $s(\hat{\nu})$ and contracting it in its parent. By construction, \mathcal{T}' is not tiny. Let ν in $\mathcal{T}_{\hat{\nu}}$ be a small node; such a node ν exists since there are no large or huge leaves and the relative weight of $\hat{\nu}$ is at least $3/8$. Let $\mathcal{T}'_{\hat{\nu}}$ be the tree obtained from $\mathcal{T}_{\hat{\nu}}$ by removing \mathcal{T}_{ν} , marking $s(\nu)$ and contracting it into its parent. Let \mathcal{T}_r be constructed by joining $\mathcal{T}_{\hat{\mu}}$ and \mathcal{T}_{ν} with a new root r , as shown.

We partition R into three rectangles, one for \mathcal{T}' , one for $\mathcal{T}'_{\hat{\nu}}$, and one for \mathcal{T}_r . We mark them as shown in the figure to the right, and recurse. Since all three subtrees are non-tiny the rectangles on which we recurse have aspect ratio at most 8.

All leaf regions have property (i) by induction. Moreover, all internal nodes have property (ii). For the nodes that are not an ancestor of ν this follows by



induction. For nodes on the path from $\hat{\nu}$ to ν we can argue as follows. $R(\hat{\mu})$ is a staircase anchored at the bottom-right corner of the rectangle of T_r , and so $R(\nu)$ is a staircase anchored at the opposite corner. Because the top-right corner of the rectangle of $T'_{\hat{\nu}}$ is marked, this implies that the nodes from $\hat{\nu}$ to ν are drawn as orthoconvex polygons. A similar argument applies to the ancestors of $\hat{\nu}$. The nodes that are not an ancestor of ν have aspect ratio at most 64 by induction. The ancestors of ν are not tiny, since ν is not tiny, and $\text{asp}(R) \leq 8$, hence all the regions have aspect ratio at most 64.

It remains to argue that μ as well as all of its ancestors have property (iii). For μ and its ancestors in $T_{\hat{\mu}}$ this follows by induction. For μ^* and its ancestors this follows from the fact that the regions of each of these nodes contain the rectangle of $T'_{\hat{\nu}}$ and T_r , and the induction hypothesis on T' .

Theorem 2. *Every properly weighted tree can be represented by an orthoconvex treemap in which all leaves are drawn as rectangles, L-shapes, or S-shapes, all internal nodes are drawn as orthoconvex polygons, and the aspect ratio of any of these regions is $O(1)$.*

4 Conclusions

We have settled two main theoretical questions concerning treemaps: First, we presented an algorithm for computing convex treemaps whose aspect ratio is $O(d)$, where d is the depth of the input hierarchy; this is asymptotically optimal. Second, we showed that any input hierarchy admits an orthoconvex treemap of constant aspect ratio. We also implemented the algorithms. Preliminary experiments show that the aspect ratios remain significantly below their provable worst-case bounds, and that orthoconvex treemaps hardly ever use S-shapes.

References

1. Balzer, M., Deussen, O.: Voronoi treemaps. In: Proc. IEEE Symp. Information Visualization, pp. 7–14 (2005)
2. Balzer, M., Deussen, O., Lewerentz, C.: Voronoi treemaps for the visualization of software metrics. In: Proc. ACM Symp. Software Vis., pp. 165–172 (2005)
3. Bederson, B.B., Schneiderman, B., Wattenberg, M.: Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. ACM Transactions on Graphics 21(4), 833–854 (2002)
4. Bruls, M., Huizing, K., van Wijk, J.: Squarified Treemaps. In: Proc. Joint Eurographics and IEEE TCVG Symp. Visualization, pp. 33–42. Springer, Heidelberg (2000)
5. de Berg, M., Onak, K., Sidiropoulos, A.: Fat polygonal partitions with applications to visualization and embeddings (2010) (in preparation),
<http://arxiv.org/abs/1009.1866v1>
6. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. 17, 263–269 (1969)
7. Jin, L., Banks, D.C.: Tennisviewer: A browser for competition trees. IEEE Computer Graphics and Applications 17(4), 63–65 (1997)

8. Jungmeister, W., Turo, D.: Adapting treemaps to stock portfolio visualization. Technical report UMCP-CSD CS-TR-2996, University of Maryland (1992)
9. Kong, N., Heer, J., Agrawala, M.: Perceptual guidelines for creating rectangular treemaps. *IEEE Trans. Vis. and Comp. Graphics* 16(6), 990–998 (2010)
10. Leung, J.Y.-T., Tam, T.W., Wong, C., Young, G.H., Chin, F.Y.L.: Packing squares into a square. *J. Parallel and Distributed Computing* 10, 271–275 (1990)
11. Onak, K., Sidiropoulos, A.: Circular partitions with applications to visualization and embeddings. In: Proc. 24th Symp. Computational Geometry, pp. 28–37 (2008)
12. Shneiderman, B.: Treemaps for space-constrained visualization of hierarchies, <http://www.cs.umd.edu/hcil/treemap-history/index.shtml>
13. Shneiderman, B.: Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics* 11(1), 92–99 (1992)
14. Vliegen, R., van Wijk, J., van der Linden, E.-J.: Visualizing business data with generalized treemaps. *IEEE Trans. Vis. and Comp. Graphics* 12(5), 789–796 (2006)
15. Wattenberg, M.: A note on space-filling visualizations and space-filling curves. In: Proc. IEEE Symp. Information Visualization, pp. 181–185 (2005)

Simultaneous Embedding of Embedded Planar Graphs*

Patrizio Angelini¹, Giuseppe Di Battista¹, and Fabrizio Frati^{1,2}

¹ Dipartimento di Informatica e Automazione - Università Roma Tre, Italy
`{angelini,gdb,frati}@dia.uniroma3.it`

² School of Information Technologies - The University of Sydney, Australia

Abstract. Given k planar graphs G_1, \dots, G_k , deciding whether they admit a simultaneous embedding with fixed edges (SEFE) and whether they admit a simultaneous geometric embedding (SGE) are NP-hard problems, for $k \geq 3$ and for $k \geq 2$, respectively. In this paper we consider the complexity of SEFE and of SGE when the graphs G_1, \dots, G_k have a fixed planar embedding. In sharp contrast with the NP-hardness of SEFE for three non-embedded graphs, we show that SEFE is polynomial-time solvable for three graphs with a fixed planar embedding. Furthermore, we show that, given k embedded planar graphs G_1, \dots, G_k , deciding whether a SEFE of G_1, \dots, G_k exists and deciding whether an SGE of G_1, \dots, G_k exists are NP-hard problems, for $k \geq 14$ and $k \geq 13$, respectively.

1 Introduction and Overview

Let G_1, \dots, G_k be k graphs on the same set V of vertices. A *simultaneous embedding* of G_1, \dots, G_k consists of k planar drawings $\Delta_1, \dots, \Delta_k$ of G_1, \dots, G_k , respectively, such that any vertex $v \in V$ is mapped to the same point in every drawing Δ_i . The study of simultaneous graph embeddings finds numerous applications in information visualization (see [59]). Motivated by such applications and by the natural raise of several interesting theoretical questions, simultaneous graph embeddings have received an enormous research attention during the last ten years.

Two main variants of the simultaneous embedding problem have been studied. In the geometric variant, called *simultaneous geometric embedding* (SGE), edges are required to be represented as straight-lines; in the topological variant, called *simultaneous embedding with fixed edges* (SEFE), edges that are common to distinct graphs are required to be represented by the same Jordan curve in all the drawings. Thus, if a graph admits an SGE, it also admits a SEFE, while the converse is, in general, not true.

A rich body of combinatorial results has been found for SGE and SEFE. An SGE of two paths, of two caterpillars, or of two cycles always exists [5], while there exist triples of paths [5], pairs of outerplanar graphs [5], pairs of trees [16], and tree-path pairs [3] that do not admit any SGE. Moreover, it is known that a tree and a path always have a SEFE with few bends per edge [8], an outerplanar graph and a path or a cycle always have a SEFE with few bends per edge [7], a planar graph and a tree always have a SEFE [13], while there exist pairs of outerplanar graphs that do not have any SEFE [13].

* Work partially supported by the MIUR, project AlgoDEEP 2008TFBW4, and by the ESF project 10-EuroGIGA-OP-003 “Graph Drawings and Representations”.

The algorithmic questions of deciding whether given graphs G_1, \dots, G_k admit an SGE or a SEFE have also been deeply investigated. Estrella-Balderrama et al. proved in [10] that deciding whether given graphs G_1, \dots, G_k admit an SGE is an NP-hard problem even for $k = 2$. They also show that SGE is in NP if and only if the *existential theory of the reals* and the *rectilinear crossing number* are. Establishing the membership of these two problems in NP is a long-standing open question. Concerning SEFE, Gassner et al. proved that SEFE is NP-hard for three planar graphs and that SEFE is in NP for any number of input graphs [15]. However, whether testing the existence of a SEFE of two planar graphs is doable in polynomial time or not is still an open question. A large number of results are known related to this problem. Fowler et al. characterized the planar graphs that always have a SEFE with any other planar graph and proved that SEFE is in P for two outerplanar graphs [12]; Fowler et al. showed how to test in polynomial time whether two planar graphs admit a SEFE if one of them contains at most one cycle [11]; Jünger and Schulz characterized the graphs $G_{1,2}$ that allow for a SEFE of any two planar graphs G_1 and G_2 whose intersection graph is $G_{1,2}$ [10]; Angelini et al. showed how to test whether two planar graphs admit a SEFE if one of them has a fixed embedding [2]; Angelini et al. [1] and Haeupler et al. [18] showed how to test whether two planar graphs admit a SEFE if their intersection graph is biconnected.

Motivated by the existence of several problems that are NP-hard for planar graphs and become polynomial-time solvable for planar graphs with fixed embedding (see, e.g., *testing upward planarity* [4][14] and *minimizing bends in orthogonal planar drawings* [14][20]) and by the fact that the NP-hardness proofs for SGE [10] and SEFE [15] strongly exploit the possibility of modifying the embedding of the input graphs, in this paper we study the computational complexity of deciding whether given graphs G_1, \dots, G_k admit an SGE or a SEFE when a planar (combinatorial) embedding for them is fixed in advance. We call these problems SGE-FE and SEFE-FE, respectively.

First, we study the time complexity of deciding whether *three* embedded graphs admit a SEFE-FE. Note that SEFE-FE is a linear-time solvable problem for *two* embedded graphs G_1 and G_2 . Namely, the intersection graph $G_{1,2}$ of G_1 and G_2 either has a unique planar embedding Γ coherent with the embeddings of G_1 and G_2 or it does not have such an embedding. In the former case Γ can be extended independently to the embeddings of G_1 and G_2 , while in the latter case no SEFE-FE of G_1 and G_2 exists. We show that SEFE-FE is in P for three embedded planar graphs G_1 , G_2 , and G_3 . The proof is based on a set of necessary conditions that must hold for the intersection graph $G_{1,2,3}$ of G_1 , G_2 , and G_3 to be extendable to the given embeddings of G_1 , G_2 , and G_3 . These conditions are also sufficient and testable in polynomial time. To prove that, we develop several concepts and techniques. We introduce the concept of the *zone* of a connected component of $G_{1,2,3}$ that is a set of faces in a planar embedding of that component that are equivalent with respect to the interaction with other components. Also, we show that cubic triconnected graphs with strong Hamiltonicity properties play an important role in deciding the existence of a SEFE-FE for three embedded graphs. The case of three graphs is much more complex than the case of two graphs since several embeddings of $G_{1,2,3}$ must be considered to establish the existence of a SEFE-FE.

Second, we show that SEFE-FE and SGE-FE are NP-hard for k embedded planar graphs G_1, \dots, G_k . For the proofs we exploit $k \geq 14$ and $k \geq 13$, respectively. Hence,

for sufficiently large k , the fixed embedding version of the simultaneous embedding problem becomes as hard as the variable embedding one.

The paper is organized as follows. In Sect. 2 we introduce some preliminaries; in Sect. 3, we show a polynomial-time algorithm for SEFE-FE of three embedded planar graphs; finally, in Sect. 4 we show NP-hardness proofs for SEFE-FE and SGE-FE.

2 Preliminaries

The graphs considered in this paper are not necessarily connected. A *drawing* of a graph is a mapping of each vertex to a point of the plane and of each edge to a simple Jordan curve connecting its endpoints. A drawing is *planar* if the curves representing its edges do not cross except, possibly, at common endpoints. A graph is *planar* if it admits a planar drawing. A planar drawing Δ of a graph G determines a subdivision of the plane into connected regions, called *faces*, and a clockwise ordering of the edges incident to each vertex v , called *rotation scheme* and denoted by $\rho(v)$. Let C be a simple cycle in Δ . Cycle C splits the plane into two connected parts. Denote by $V_{\Delta}^{\text{left}}(C)$ and $V_{\Delta}^{\text{right}}(C)$ the sets of vertices of G that are to the left and to the right of C in Δ , respectively, when traversing C in clockwise direction. Two drawings Δ' and Δ'' are *equivalent* if they have the same rotation schemes and, for each cycle C , $V_{\Delta'}^{\text{left}}(C) = V_{\Delta''}^{\text{left}}(C)$. A *planar embedding* is an equivalence class of planar drawings. Observe that if the graph is connected the second condition descends from the first one. See also [219].

A graph is *biconnected* (*triconnected*) if removing any vertex (any two vertices) leaves it connected. A *cut-vertex* (a *separation pair*) is a vertex (is a pair of vertices) whose removal disconnects the graph. The SPQR-tree of a biconnected graph G describes a recursive decomposition of G induced by its separation pairs and its edges [6].

3 Computing a SEFE-FE of Three Embedded Planar Graphs

In this section we show that SEFE-FE is in P when $k = 3$.

Let $\Gamma_1 = (V, E_1)$, $\Gamma_2 = (V, E_2)$, and $\Gamma_3 = (V, E_3)$ be three embedded planar graphs. We define $G_{1,2,3} = (V, E)$ as the graph such that each edge in E belongs to at least two of E_1 , E_2 , and E_3 . Let Γ be a planar embedding of $G_{1,2,3}$. We denote by $\Gamma|E_i$ the embedding obtained by restricting Γ to the edges of $E \cap E_i$. Analogously, we define the embedding $\Gamma_i|E$. We say that a planar embedding Γ of $G_{1,2,3}$ is *compatible with* Γ_i if $\Gamma|E_i = \Gamma_i|E$, where the sign “=” is used as a synonym of equivalent. We say that Γ is *compatible* if it is compatible with each Γ_i ($i = 1, 2, 3$).

Theorem 1. Γ_1 , Γ_2 , and Γ_3 have a SEFE-FE if and only if $G_{1,2,3}$ has a planar compatible embedding Γ .

Proof sketch: If $G_{1,2,3}$ has a planar compatible embedding Γ , then the edges of Γ_1 , Γ_2 , and Γ_3 can be reinserted independently, and hence such graphs have a SEFE-FE. If Γ_1 , Γ_2 , and Γ_3 have a SEFE-FE, then removing from such a SEFE-FE the edges not in $G_{1,2,3}$ yields a compatible planar embedding of $G_{1,2,3}$. \square

Observe that Theorem 1 does not generalize to more than three graphs.

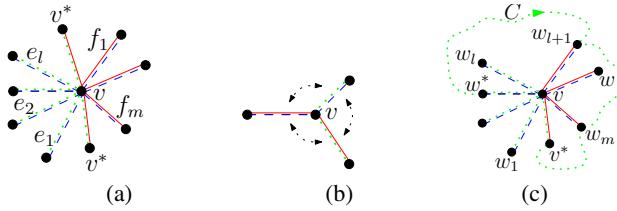


Fig. 1. Edges of E_i , of E_k , and of E_j are dashed, solid, and dotted curves, respectively. (a) A rotation-undecided vertex v . The undecided edge (v, v^*) appears in both its possible positions. (b) A degree-3 rotation-undecided vertex. (c) Lemma \square If w is to the left of C , then (v, v^*) , (v, w^*) , and (v, w) appear in this clockwise ordering in any compatible embedding of $G_{1,2,3}$.

Let $\rho(v)$ be the rotation scheme of a vertex v in any embedding Γ of $G_{1,2,3}$ and, for each $i = 1, 2, 3$, let $\rho(v, i)$ be the rotation scheme of v in Γ_i . Denote by $\rho(v)|E_i$ the rotation scheme $\rho(v)$ restricted to the edges of Γ_i . Analogously, denote by $\rho(v, i)|E$ the rotation scheme $\rho(v, i)$ restricted to the edges of $G_{1,2,3}$. We say that $\rho(v)$ is *compatible* if, for each $i = 1, 2, 3$, it holds $\rho(v)|E_i = \rho(v, i)|E$. A planar embedding Γ of $G_{1,2,3}$ is *cycle-compatible* if, for each cycle C of $\Gamma_i|E$ ($i = 1, 2, 3$), $V_{\Gamma_i|E}^{left}(C) = V_{\Gamma}^{left}(C)$. Observe that a planar embedding of $G_{1,2,3}$ is compatible if and only if it is cycle-compatible and the rotation schemes of all its vertices are compatible.

A vertex v of $G_{1,2,3}$ is *rotation-constrained (rotation-unfeasible)* if there exists a unique compatible rotation scheme (no compatible rotation scheme). A vertex v of $G_{1,2,3}$ is *rotation-undecided* if it is neither rotation-constrained nor rotation-unfeasible. Figs. \square (a) and \square (b) show two rotation-undecided vertices. If $G_{1,2,3}$ has a rotation-unfeasible vertex, then Γ_1 , Γ_2 , and Γ_3 do not admit any SEFE-FE.

We state the following property about rotation-undecided vertices. Refer to Fig. \square (a).

Property 1. A vertex v of $G_{1,2,3}$ is rotation-undecided if and only if there exists an index i such that: (i) E contains exactly one edge $(v, v^*) \notin \Gamma_i$ incident to v , (ii) $\rho(v, i)|E = e_1, \dots, e_l, f_1, \dots, f_m$; (iii) $\rho(v, j)|E = e_1, \dots, e_l, (v, v^*)$ and $\rho(v, k)|E = f_1, \dots, f_m, (v, v^*)$, with i, j, k distinct elements in $\{1, 2, 3\}$.

We say that edge (v, v_i) as in Property \square is an *undecided edge* of v . Note that if $\deg(v) = 3$ and v is rotation-undecided, then all the edges incident to v are undecided edges. See Fig. \square (b). Observe that the following holds. See Fig. \square (c):

Lemma 1. Let v be a rotation-undecided vertex of $G_{1,2,3}$. Let i, j, k be distinct values in $\{1, 2, 3\}$. Let $(v, v^*) \in E_j \cap E_k$ be an undecided edge of v . Suppose that a cycle C exists in $\Gamma_j|E$ containing (v, v^*) and an edge $(v, w^*) \in E_i \cap E_j$. Suppose also that a neighbor w of v exists not contained in C , where edge $(v, w) \in E_i \cap E_k$. Then in every compatible embedding of $G_{1,2,3}$, if any exists, the rotation scheme of v is the same.

When Lemma \square applies, we say that the rotation scheme of v is *determined* by C .

We now show how to compute a compatible planar embedding Γ of $G_{1,2,3}$. We distinguish the case in which $G_{1,2,3}$ is connected and the one in which it is not.

Assume that $G_{1,2,3}$ is connected. We characterize the compatible embeddings of $G_{1,2,3}$. Suppose that no vertex of $G_{1,2,3}$ is rotation-unfeasible. We associate to every

rotation-constrained vertex of $G_{1,2,3}$ its unique compatible rotation scheme and to each rotation-undecided vertex either one or two rotation schemes, as follows.

Let v be a rotation-undecided vertex of $G_{1,2,3}$. Let i, j, k be distinct values in $\{1, 2, 3\}$. Let $(v, v^*) \in E_j \cap E_k$ be an undecided edge of v . Suppose that there exists a cycle C in $\Gamma_j|E$ containing (v, v^*) and an edge $(v, w^*) \in E_i \cap E_j$, and suppose also that there exists a neighbor w of v that is not contained in C , where edge $(v, w) \in E_i \cap E_k$. Then, by Lemma 11 the rotation scheme of v is determined by C and it is the same in any compatible embedding of $G_{1,2,3}$. Assign such a rotation scheme to v . Again by Lemma 11 if two different rotation schemes of a vertex v are determined by two different cycles C and C' , there exists no compatible embedding of $G_{1,2,3}$. Let v be a rotation-undecided vertex of $G_{1,2,3}$. Assume that the rotation scheme of v is not determined by any cycle C . Then assign to v both its compatible rotation schemes.

An embedding of $G_{1,2,3}$ is *rotation-coherent* if every vertex has a rotation scheme that is one of the at most two rotation schemes assigned to it. We have the following:

Theorem 2. *Suppose that $G_{1,2,3}$ is connected. Then Γ_1 , Γ_2 , and Γ_3 have a SEFE-FE if and only if the following Conditions hold: (1) $G_{1,2,3}$ has no rotation-unfeasible vertex; (2) $G_{1,2,3}$ contains no vertex for which two different rotation schemes are determined by two cycles C and C' ; and (3) $G_{1,2,3}$ has a rotation-coherent planar embedding.*

Proof sketch: The necessity of Conditions (1)–(3) easily follows from Lemma 11 and Theorem 11. In order to prove the sufficiency of Conditions (1)–(3), note that in any rotation-coherent planar embedding Γ of $G_{1,2,3}$ every vertex has a compatible rotation scheme. Since $G_{1,2,3}$ is connected, this implies that Γ is also cycle-compatible. \square

Theorem 3. *Suppose that $G_{1,2,3}$ is connected. It can be tested in $O(|V|^3)$ time whether $\Gamma_1 = (V, E_1)$, $\Gamma_2 = (V, E_2)$, and $\Gamma_3 = (V, E_3)$ admit a SEFE-FE. Moreover, if such a SEFE-FE exists, it can be constructed in $O(|V|^3)$ time.*

Proof sketch: In order to prove the statement it suffices to show how to check in polynomial time whether Conditions (1), (2), and (3) of Theorem 12 are satisfied and, in case they are, how to construct a compatible embedding of $G_{1,2,3}$. To check Condition (1) we compare the rotation schemes of each vertex in Γ_1 , Γ_2 , and Γ_3 . Then, we use Lemma 11 to decide in total $O(|V|)$ time if each vertex is rotation-undecided or rotation-constrained. Condition (2) can be checked in $O(|V|^3)$ by considering, for each rotation-undecided vertex v and for each undecided edge e incident to v , a linear number of cycles passing through e . After Conditions (1) and (2) have been checked, each vertex v is equipped with either one or two compatible rotation schemes. In the latter case, these are the two circular permutations of three sequences of edges incident to v . Thus, the $O(|V|)$ -time algorithm by Gutwenger *et al.* [17] applies to test if a planar embedding exists in which each vertex has one of its assigned rotation schemes. \square

Now suppose that $G_{1,2,3}$ is *not* connected. Is it possible to compute any compatible embedding of the connected components of $G_{1,2,3}$ and to merge such embeddings to obtain a compatible embedding of $G_{1,2,3}$? Note that the rotation schemes of the vertices in $G_{1,2,3}$ are compatible if and only if the rotation schemes of the vertices in its connected components are. On the other hand, if every connected component of $G_{1,2,3}$

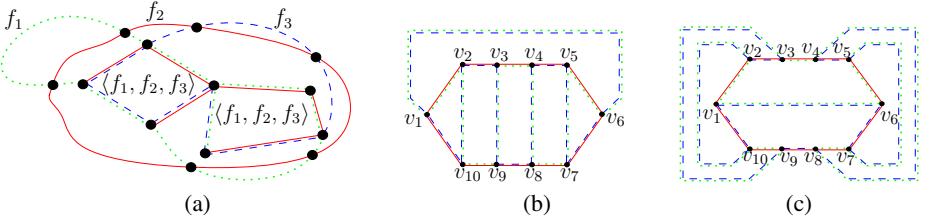


Fig. 2. (a) Two faces with the same zone-label $\langle f_1, f_2, f_3 \rangle$. (b-c) The two embeddings (b) Γ' and (c) Γ'' of a 3-Hamiltonian component G^j of $G_{1,2,3}$.

has a cycle-compatible embedding, an embedding of $G_{1,2,3}$ obtained by merging the cycle-compatible embeddings of its connected components is not necessarily cycle-compatible. Namely, the embedding constraints of Γ_1 , Γ_2 , and Γ_3 force each connected component G^i of $G_{1,2,3}$ to lie in a face f_1 of $\Gamma_1|E^j$, in a face f_2 of $\Gamma_2|E^j$, and in a face f_3 of $\Gamma_3|E^j$, for each connected component $G^j = (V^j, E^j)$ of $G_{1,2,3}$, with $j \neq i$. Since the embedding of $G_{1,2,3}$ has to be planar, then G^i can not intersect any edge of G^j . Hence, there has to be a face f of the plane in the compatible embedding Γ^j of G^j such that f is a subregion of f_1 in $\Gamma^j|E_1$, a subregion of f_2 in $\Gamma^j|E_2$, and a subregion of f_3 in $\Gamma^j|E_3$. Triplet $\langle f_1, f_2, f_3 \rangle$ is the *zone-label* of f . Any maximal set of faces having the same zone-label forms a *zone* of the embedding Γ^j . See Fig. 2(a). We say that two embeddings Γ' and Γ'' have the same zones if for every face of Γ' with zone-label $\langle f_1, f_2, f_3 \rangle$ there exists a face of Γ'' with zone-label $\langle f_1, f_2, f_3 \rangle$, and vice versa.

We study whether every compatible embedding of a connected component of $G_{1,2,3}$ has the same zones. It turns out that this is always true, except for a very special class of graphs, called *3-Hamiltonian*. A connected component G^j of $G_{1,2,3}$ is *3-Hamiltonian* if each edge of G^j belongs to exactly two of Γ_1 , Γ_2 , and Γ_3 , and for each $i = 1, 2, 3$, $G^j|E_i$ is a Hamiltonian cycle. See Fig. 2(b) and (c). Notice that 3-Hamiltonian graphs are cubic and triconnected. We show an interesting property of 3-Hamiltonian graphs.

Lemma 2. Suppose that G^j is 3-Hamiltonian. Then both the embeddings Γ' and Γ'' of G^j are compatible. Moreover, the sets Z' and Z'' of zones of Γ' and Γ'' are such that $Z' \cap Z'' = \emptyset$ and $Z' \cup Z''$ consists of all the eight zones of $\Gamma_i|E^j$ ($i = 1, 2, 3$).

Proof sketch: Since each graph $\Gamma_i|E^j$ ($i = 1, 2, 3$) is a Hamiltonian cycle, it has the same embedding in both the embeddings of G^j , that are hence compatible.

Let L_i and R_i be the two faces of $\Gamma_i|E^j$ ($i = 1, 2, 3$). Consider the embedding Γ' of G^j , the Hamiltonian cycle $\Gamma_1|E^j$, and its face L_1 . We say that an edge belonging to $\Gamma_2|E^j$ and to $\Gamma_3|E^j$ is a $(2, 3)$ -edge. Consider a $(2, 3)$ -edge e of Γ' and the two faces of Γ' incident to e . We say that e is *homogeneous* if such faces have zone-label $\langle -, L_2, L_3 \rangle$ and $\langle -, R_2, R_3 \rangle$, while e is *heterogeneous* if such faces have zone-label $\langle -, R_2, L_3 \rangle$ and $\langle -, L_2, R_3 \rangle$. We have that either all the $(2, 3)$ -edges inside L_1 are homogeneous and those inside R_1 are heterogeneous or vice-versa. This, together with the facts that (i) all the faces of Γ' inside L_1 (inside R_1) have zone-label $\langle L_1, -, - \rangle$ ($\langle R_1, -, - \rangle$) and (ii) the label of each face of Γ'' is obtained from its label in Γ' by exchanging L_i with R_i and vice versa, implies the statement. \square

We have the following:

Lemma 3. *Suppose that G^j admits a compatible embedding. Then either G^j is 3-Hamiltonian or all the compatible embeddings of G^j have the same zones.*

Proof sketch: The statement is proved first for triconnected, then for biconnected, and finally for connected graphs.

Suppose that G^j is triconnected. We use counting arguments and topological properties relating cycles of $\Gamma_i|E$ to vertices not in such cycles to prove that, assuming that both the embeddings of G^j are compatible, G^j is 3-Hamiltonian.

Suppose that G^j is biconnected. Consider any two compatible embeddings Γ' and Γ'' of G^j . Let f be a face of Γ' , let C_f be the simple cycle delimiting f , and let $\langle f_1, f_2, f_3 \rangle$ be the zone of f . We can prove the following statement: There is a face f^* of Γ'' such that any interior point p of f and any interior point p^* of f^* are on the same side of any simple cycle of $G^j|E_i$, for $i = 1, 2, 3$, unless G^j is 3-Hamiltonian. The statement implies the theorem, as it holds for all the simple cycles delimiting f_i , for $i = 1, 2, 3$. The proof of the statement exploits the SPQR-tree decomposition of G^j .

Suppose that G^j is connected. Let m be the number of maximal biconnected components of G^j . We consider an ordering of such components so that the graph composed of the first h components is connected, for each $1 \leq h \leq m$. We prove the statement by induction on h . In the inductive argument we use the rotation scheme of the cutvertices to decide the embedding of the 3-Hamiltonian components (if any exist). \square

We have the following:

Theorem 4. *It can be tested in $O(|V|^3)$ time whether $\Gamma_1 = (V, E_1)$, $\Gamma_2 = (V, E_2)$, and $\Gamma_3 = (V, E_3)$ admit a SEFE-FE. Moreover, if such a SEFE-FE exists, it can be constructed in $O(|V|^3)$ time.*

Proof sketch: For each connected component we apply Theorem 3 to decide in $O(|V|^3)$ time whether it admits a compatible embedding or not. If there exists one connected component that does not admit any compatible embedding, then Γ_1 , Γ_2 , and Γ_3 do not admit any SEFE-FE. Otherwise, each connected component admits a compatible embedding. For each connected component that is 3-Hamiltonian, we exploit Lemma 2 to choose a compatible embedding for it. For each connected component that is not 3-Hamiltonian, by Lemma 3 we can choose any of its compatible embeddings.

At this point we check if it is possible to “glue” the components into a unique compatible embedding, and in that case we compute it. Observe that for this test to succeed it is not sufficient that each two components form a graph that admits a compatible embedding. Hence, we iteratively construct an embedding of the graph composed of the first h components, for each $1 \leq h \leq m$, by merging an embedding of the graph composed of the first $h - 1$ components with the embedding of the h -th component. \square

4 SEFE-FE and SGE-FE of k Embedded Graphs Is NP-Hard

In this section we prove that SEFE-FE and SGE-FE are NP-hard when the input consists of at least 14 and 13 embedded graphs, respectively. We start with the following:

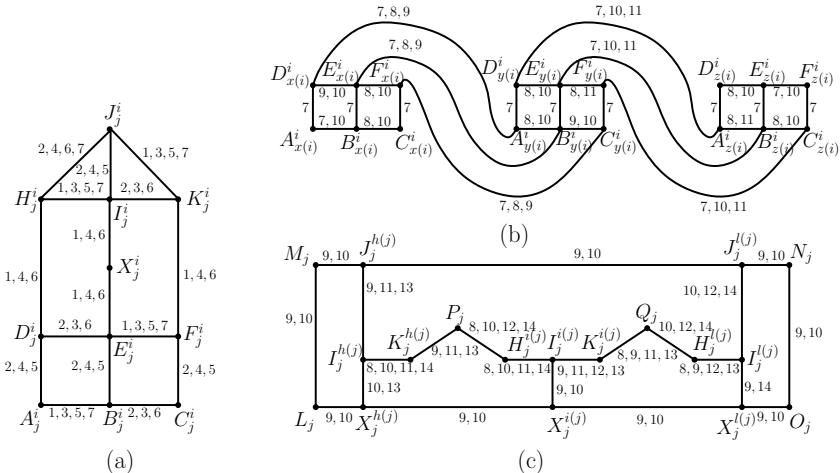


Fig. 3. (a) Variable gadget G_j^i . Edges are labeled with the graphs they belong to. (b) Graph H^i ensuring “coherence” between the embeddings of the three gadgets associated to the same variable. (c) Clause gadget when all the three literals in the clause are satisfied.

Theorem 5. SEFE-FE is an NP-hard problem.

We sketch a polynomial-time reduction from a variation of 3-SAT to SEFE-FE. Our reduction is largely inspired by a reduction from 3-SAT to SEFE described by Gassner et al. [15]. We construct 14 embedded planar graphs G_1, \dots, G_{14} starting from a 3-SAT formula F containing variables v_1, \dots, v_n , and clauses c_1, \dots, c_m . We actually show a graph G^* which is the union of G_1, \dots, G_{14} and we assign numbers to the edges of G^* , where an edge of G^* with number a belongs to graph G_a . Graphs G_1, \dots, G_{14} are constructed out of variable gadgets, clause gadgets, and variable coherence gadgets. We assume for simplicity that each variable appears in three clauses. Gadgets can be easily modified to deal with the case in which variables appear a different number of times.

Variable Gadgets: For each $1 \leq i \leq n$, let $c_{x(i)}, c_{y(i)}$, and $c_{z(i)}$ be the three clauses in which variable v_i appears. Then G^* contains three isomorphic graphs G_j^i , with $j \in \{x(i), y(i), z(i)\}$, shown in Fig. 3(a). We assign numbers to the edges of G_j^i in such a way that, in any SEFE-FE of G_1, \dots, G_{14} the drawing of G_j^i is planar and G_j^i admits at most two embeddings, where one is the flip of the other. Both such embeddings restricted to G_i coincide with any given embedding of G_i restricted to the edges of G_j^i , since all such graphs are collections of paths and isolated vertices. Thus, G_j^i admits exactly two embeddings, which are in correspondence with the two possible truth value assignments to the literal containing variable v_i occurring in clause c_j .

Clause Gadgets: For each $1 \leq j \leq m$, let $v_{h(j)}, v_{i(j)}$, and $v_{l(j)}$ be the variables that occur in c_j . Then, G^* contains a graph C_j , shown in Fig. 3(c). Because of the assignment of numbers to the edges of C_j , few edge pairs of C_j are allowed to cross and the rotation scheme of each vertex is fixed, except for the ones of vertices $I_j^{h(j)}, I_j^{i(j)}$, and $I_j^{l(j)}$. In fact, such rotation schemes are decided by the embeddings of graphs $G_j^{h(j)}, G_j^{i(j)}$, and $G_j^{l(j)}$, respectively, which are in correspondence with the truth value

assignments to the literals containing variables $v_{h(j)}$, $v_{i(j)}$, and $v_{l(j)}$, respectively, and occurring in clause c_j . Graph C_j admits an embedding in which the outer face is the same as in Fig. 3(c) and in which no pair of edges in the same graph G_a cross if and only if at least one of the rotation schemes of $I_j^{h(j)}$, $I_j^{i(j)}$, and $I_j^{l(j)}$ is the one corresponding to a literal satisfied by a truth value assignment to the variables of F .

Variable Coherence Gadgets: For each $1 \leq i \leq n$, G^* has six edges e_1, \dots, e_6 defined as follows: If v_i appears as the same literal in $c_{x(i)}$ and in $c_{y(i)}$, then let $e_1 = (D_{x(i)}^i, A_{y(i)}^i)$, $e_2 = (E_{x(i)}^i, B_{y(i)}^i)$, and $e_3 = (F_{x(i)}^i, C_{y(i)}^i)$; otherwise, let $e_1 = (D_{x(i)}^i, C_{y(i)}^i)$, $e_2 = (E_{x(i)}^i, B_{y(i)}^i)$, and $e_3 = (F_{x(i)}^i, A_{y(i)}^i)$. Edges e_4, e_5 , and e_6 are defined analogously depending on the literals containing v_i in $c_{y(i)}$ and in $c_{z(i)}$. See Fig. 3(b). Let H^i be the subgraph of G^* induced by vertices $A_j^i, B_j^i, C_j^i, D_j^i, E_j^i$, and F_j^i , for $j = x(i), y(i), z(i)$. We assign numbers to the edges of G^* in such a way that, in any SEFE-FE of G_1, \dots, G_{14} the drawing of H^i is planar and H^i admits at most two embeddings, where one is the flip of the other. Such embeddings restricted to G_i coincide with any embedding of G_i restricted to H^i . The choice of an embedding for H^i determines the embeddings of the three variable gadgets G_j^i , with $j \in \{x(i), y(i), z(i)\}$, since H^i and G_j^i share a vertex of degree 3 and they are both triconnected.

We describe embeddings for graphs G_1, \dots, G_{14} . Each of $G_1, \dots, G_8, G_{11}, \dots, G_{14}$ is a collection of paths and isolated vertices, thus we require its unique planar embedding. We fix the embedding of G_9 and G_{10} so that the clause gadgets are one outside the other, the clause gadgets have the same outer face as in Fig. 3(c), and each vertex is in the outer face of a clause gadget C_j if and only if it is not a vertex of C_j .

If the 3-SAT instance formula F is satisfiable, then whether a literal corresponding to variable v_i is satisfied or not in a clause determines the embedding of graph H^i and of graphs G_j^i . Since each clause is satisfied, C_j can be embedded with the same outer face as in Fig. 3(c) so that no two edges of the same graph G_i cross. The vertices of graphs H^i not in C_j can be embedded lower than all the vertices of C_j and lower than all the vertices of H^l , with $l < i$, thus obtaining a SEFE-FE of G_1, G_2, \dots, G_{14} . Conversely, if a SEFE Γ of G_1, G_2, \dots, G_{14} exists, then, for each $1 \leq i \leq n$, consider variable gadget G_j^i , for each $j \in \{x(i), y(i), z(i)\}$. If G_j^i appears in Γ with the embedding of Fig. 3(a), then set the literal containing variable v_i in clause c_j to be satisfied, otherwise set it to be not satisfied. By the described correspondence between embeddings of graphs C_j , G_j^i , and H^i , this leads to a truth assignment in which every variable is assigned with a unique value and for which F is satisfied, thus concluding the proof of Theorem 5.

Concerning SGE-FE, we can show an NP-hardness proof similar to the proofs of NP-hardness for SGE of two (non-embedded) graphs [10] and for SEFE-FE of 14 embedded graphs (Theorem 5). The gadgets used to perform a reduction from 3-SAT are defined *exactly* as in the proof of Theorem 5 except for the clause gadget which is however extremely similar to the one described in the proof of Theorem 5. The fact that any two edges can cross at most once in any SGE-FE is exploited to reduce the number of graphs needed to make the correspondence between a satisfied clause and a simultaneous embedding of a clause gadget work from 14 to 13. Thus, analogously as for Theorem 5, it can be proved the following.

Theorem 6. SGE-FE is an NP-hard problem.

References

1. Angelini, P., Battista, G.D., Frati, F., Patrignani, M., Rutter, I.: Testing the Simultaneous Embeddability of Two Graphs Whose Intersection is a Biconnected Graph or a Tree. In: Iliopoulos, C.S., Smyth, W.F. (eds.) IWOCA 2010. LNCS, vol. 6460, pp. 212–225. Springer, Heidelberg (2011)
2. Angelini, P., Di Battista, G., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. In: SODA 2010, pp. 202–221 (2010)
3. Angelini, P., Geyer, M., Kaufmann, M., Neuwirth, D.: On a Tree and a Path with No Geometric Simultaneous Embedding. In: Brandes, U., Cornelsen, S. (eds.) GD 2010. LNCS, vol. 6502, pp. 38–49. Springer, Heidelberg (2011)
4. Bertolazzi, P., Battista, G.D., Liotta, G., Mannino, C.: Upward drawings of triconnected digraphs. *Algorithmica* 12(6), 476–497 (1994)
5. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. *Comput. Geom.* 36(2), 117–130 (2007)
6. Di Battista, G., Tamassia, R.: On-line planarity testing. *SIAM J. Comput.* 25, 956–997 (1996)
7. Di Giacomo, E., Liotta, G.: Simultaneous embedding of outerplanar graphs, paths, and cycles. *Int. J. Comput. Geometry Appl.* 17(2), 139–160 (2007)
8. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. *J. Graph Algorithms Appl.* 9(3), 347–364 (2005)
9. Erten, C., Kobourov, S.G., Le, V., Navabi, A.: Simultaneous graph drawing: Layout algorithms and visualization schemes. *J. Graph Algorithms Appl.* 9(1), 165–182 (2005)
10. Estrella-Balderrama, A., Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous Geometric Graph Embeddings. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 280–290. Springer, Heidelberg (2008)
11. Fowler, J.J., Gutwenger, C., Jünger, M., Mutzel, P., Schulz, M.: An SPQR-Tree Approach to Decide Special Cases of Simultaneous Embedding with Fixed Edges. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 157–168. Springer, Heidelberg (2009)
12. Fowler, J.J., Jünger, M., Kobourov, S.G., Schulz, M.: Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges. *Comput. Geom.* 44(8), 385–398 (2011)
13. Frati, F.: Embedding Graphs Simultaneously with Fixed Edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
14. Garg, A., Tamassia, R.: On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.* 31(2), 601–625 (2001)
15. Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous Graph Embeddings with Fixed Edges. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 325–335. Springer, Heidelberg (2006)
16. Geyer, M., Kaufmann, M., Vrt'o, I.: Two trees which are self-intersecting when drawn simultaneously. *Discrete Mathematics* 307(4), 1909–1916 (2009)
17. Gutwenger, C., Klein, K., Mutzel, P.: Planarity testing and optimal edge insertion with embedding constraints. *J. Graph Algorithms Appl.* 12(1), 73–95 (2008)
18. Haeupler, B., Jampani, K.R., Lubiw, A.: Testing Simultaneous Planarity When the Common Graph Is 2-Connected. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 410–421. Springer, Heidelberg (2010)
19. Jünger, M., Schulz, M.: Intersection graphs in simultaneous embedding with fixed edges. *J. Graph Alg. & Appl.* 13(2), 205–218 (2009)
20. Tamassia, R.: On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.* 16(3), 421–444 (1987)

Linear-Time Algorithms for Hole-Free Rectilinear Proportional Contact Graph Representations

Muhammad Jawaherul Alam¹, Therese Biedl², Stefan Felsner³,
Andreas Gerasch⁴, Michael Kaufmann⁴, and Stephen G. Kobourov¹

¹ Department of Computer Science, University of Arizona, Tucson, AZ, USA

² David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada

³ Institut für Mathematik, Technische Universität Berlin, Berlin, Germany

⁴ Wilhelm-Schickard-Institut für Informatik, Tübingen Universität, Tübingen, Germany

Abstract. In a proportional contact representation of a planar graph, each vertex is represented by a simple polygon with area proportional to a given weight, and edges are represented by adjacencies between the corresponding pairs of polygons. In this paper we study proportional contact representations that use rectilinear polygons without wasted areas (white space). In this setting, the best known algorithm for proportional contact representation of a maximal planar graph uses 12-sided rectilinear polygons and takes $O(n \log n)$ time. We describe a new algorithm that guarantees 10-sided rectilinear polygons and runs in $O(n)$ time. We also describe a linear-time algorithm for proportional contact representation of planar 3-trees with 8-sided rectilinear polygons and show that this is optimal, as there exist planar 3-trees that require 8-sided polygons. Finally, we show that a maximal outer-planar graph admits a proportional contact representation using rectilinear polygons with 6 sides when the outer-boundary is a rectangle and with 4 sides otherwise.

1 Introduction

Representing planar graphs as *contact graphs* has been a subject of study for many decades. In such a representation, vertices correspond to geometrical objects, such as line-segments or polygons, while edges correspond to two objects touching in some pre-specified fashion. In this paper, we consider *side contact representations* of planar graphs, where vertices are simple interior-disjoint polygons, and adjacencies are non-trivial side-contacts between the corresponding polygons. In the weighted version of the problem, the goal is to find a contact representation of G where the area of the polygon for each vertex is proportional to the weight of the vertex, which is given in advance. We call such a representation a *proportional contact representation* of G . Such representations often lead to a more compelling visualization of a planar graph than usual node-link representations [5] and have practical applications in cartography, VLSI Layout, and floor-planning. Rectilinear polygons with small number of sides (or corners) are often desirable due to aesthetic, practical, and cognitive requirements. In architectural floor-planning and VLSI design, it is also desirable to minimize the unused area in the representation. Hence we address the problem of constructing a proportional

* This research was initiated at the Dagstuhl Seminar 10461 on Schematization.

¹ Research funded in part by NSF grants CCF-0545743 and CCF-1115971.

² Research supported by NSERC.

contact representation of a planar graph with rectilinear polygons with few sides, so that the representation contains no unused area.

Related Work: Contact representations of planar graphs can be dated back to 1936 when Koebe showed that any planar graph has a representation by touching circles. While touching circles or touching triangles provide point-contact representations, side-contact representations have also been considered. For example, Gansner *et al.* [7] show that 6-sided polygons are sometimes necessary and always sufficient for side-contact representation of any planar graph with convex polygons.

Applications in VLSI or architectural layout design encourage the use of rectilinear polygons in a contact representation that fills a rectangle. In this setting it is known that 8 sides are sometimes necessary and always sufficient [8][12][22]. A characterization of the graphs admitting a more restricted rectangle-representation is given by Kozmiński and Kinnen [11] and in the dual setting by Ungar [20]. A similar characterization of graphs having representations with 6-sided rectilinear polygons is given by Sun and Sarrafzadeh [18]. Buchsbaum *et al.* [5] give an overview on the state of the art concerning rectangle contact graphs.

In the results summarized above, the vertex weights and polygonal areas are not considered. The weighted version of the problem, that of *proportional contact representations* has applications in *cartograms*, or value-by-area maps. Here, the goal is to redraw an existing geographic map so that a given weight function (e.g., population) is represented by the area of each country. Algorithms by van Kreveld and Speckmann [21] and Heilmann *et al.* [9] yield representations with rectangular polygons, but the adjacencies may be disturbed. De Berg *et al.* describe an adjacency-preserving algorithm for proportional contact representation with at most 40 sides for an internally triangulated plane graph G [6]. This was later improved to 34 sides [10].

The problem has also been studied in the dual settings, where there are weights at the internal faces of a plane graph (instead of the vertices), and the area of faces should be proportional. All planar cubic graphs admit such a drawing [19] as do all planar partial 3-trees [3], but not all planar graphs [15]. Proportional rectilinear drawings with 8-sided polygons can be found for special classes of planar graphs [14], but this approach does not extend to general planar graphs. In a recent paper, Biedl and Velázquez [4] describe the best general result, with an $O(n \log n)$ algorithm for proportional rectilinear drawings of cubic triconnected graphs with 12-sided rectilinear polygons. Translating this back to the primal setting, they show that every maximal planar graph has a proportional contact representation with 12-sided rectilinear polygons.

Our Contribution: Our main contribution is an improvement from the $O(n \log n)$ algorithm for 12-sided rectilinear polygons [4], with a new algorithm based on Schnyder realizers that runs in $O(n)$ time and provides a proportional contact representation of a maximal planar graph with 10-sided polygons.

We also describe a linear-time algorithm for proportional contact representations of planar 3-trees with 8-sided rectilinear polygons and show that this is optimal, as there exist planar 3-trees that require 8-sided polygons. Finally, we show that a maximal outer-planar graph admits a proportional contact representation using rectilinear polygons with 6 sides when the outer-boundary is a rectangle and with 4 sides otherwise. All our representations are hole-free, i.e., have no unused area inside.

2 Representations for Maximal Planar Graphs

Here we describe the algorithm for 10-sided rectilinear polygons. We construct the proportional representation of a maximally planar graph using Schnyder realizers [17], which we review briefly. A *Schnyder realizer* of a fully triangulated graph G is a partition of the interior edges of G into three sets T_1 , T_2 and T_3 of edges that can be directed such that for each interior vertex v , (1) v has out-degree exactly one in each of T_1 , T_2 and T_3 , and (2) the counterclockwise order of the edges incident to v is: entering T_1 , leaving T_2 , entering T_3 , leaving T_1 , entering T_2 , leaving T_3 .

The first condition implies that each T_i , $i = 1, 2, 3$ defines a tree spanning all the interior vertices and rooted at exactly one exterior vertex such that the edges are directed towards the root. Schnyder proved that any triangulated planar graph has such a realizer and it can be computed in $O(n)$ time [17].

Theorem 1. *Let $G = (V, E)$ be a maximal planar graph and let $w : V \rightarrow \mathbb{R}^+$ be a weight function. Then a proportional contact representation Γ with respect to w can be constructed in linear time such that each vertex of G is represented by a 10-sided rectilinear polygon in Γ , and there is no wasted area.*

We prove Theorem 1 by giving a linear-time algorithm to construct such a representation Γ of G , where each vertex of G is represented by a 10-sided rectilinear polygon with a fixed shape, illustrated in Fig. 1 (Some sides of the polygon may be degenerate.) This polygon can be decomposed into four rectangles called *foot*, *leg*, *bridge* and *body* of the polygon. The region bound by the parallel horizontal lines containing the top and the bottom of the bridge is the *bridge-strip*, and the *foot-strip* is defined analogously.

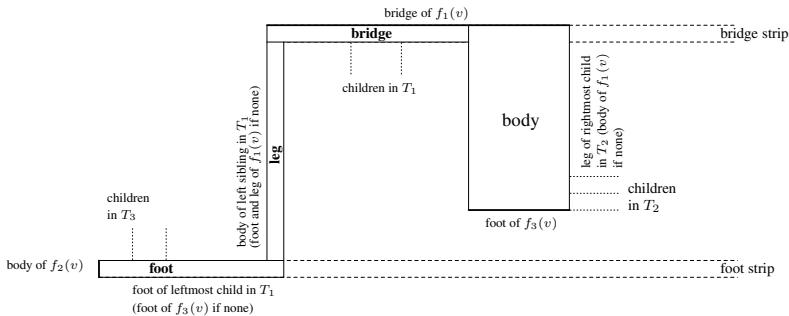


Fig. 1. A 10-sided rectilinear polygon with decomposition into foot, leg, bridge and body

Let $G = (V, E)$ be a maximal plane graph with the three outer vertices v_1 , v_2 and v_3 in counterclockwise order, and let $w : V \rightarrow \mathbb{R}^+$ be a weight function. We first find a Schnyder realizer of G that partitions the interior edges into three rooted trees T_1 , T_2 and T_3 rooted at v_1 , v_2 and v_3 , and with all their edges oriented towards the roots of the trees. We add the external edges (v_1, v_2) , (v_1, v_3) to T_1 and (v_2, v_3) to T_2 , so that all the edges of G are partitioned into the three trees. For each vertex v of G , let $f_i(v)$, $i = 1, 2, 3$ be the parent of v in T_i .

Let R be a rectangle with area equal to $\sum_{v \in V} w(v)$. We construct a proportional contact representation of G inside R . We start by cutting a rectangle $P(v_1)$ with area $w(v_1)$ for v_1 from the top of R and cutting a rectangle $P(v_2)$ with area $w(v_2)$ for v_2 from the left side of $R - P(v_1)$. In the remaining part $R' = R - P(v_1) - P(v_2)$ of the rectangle, we draw the polygons for the other vertices.

The main idea is to draw the polygons such that for each vertex v of G , the edges $(v, f_i(v))$ are realized as follows: The top of the bridge of $P(v)$ is adjacent to the bottom of the bridge of $P(f_1(v))$, the left of the foot of $P(v)$ is adjacent to the right of the body of $P(f_2(v))$ and the bottom of the body of $P(v)$ is adjacent to the top of the foot of $P(f_3(v))$. See also Fig. □. If we ensure those adjacencies, then there cannot be any other adjacencies since graph G is maximal planar, and correctness follows.

The other crucial idea is that the bridge and foot have small height and the leg has very small width, so that they together occupy less area than the weight of v . (Ensuring that their width/height is no more than $\varepsilon \leq w(v)/(W + H)$ if R' is a $W \times H$ -rectangle will do.) The bulk of the weight for v is hence in the body of v .

Our algorithm visits vertices in depth-first order in the tree T_1 and builds $P(v)$ partially before and partially after visiting the children of v (in left-to-right order according to the planar embedding.) When $P(v)$ is partially built, we reserve horizontal strips of height $\leq \varepsilon$ (called *bridge strip* and *foot strip*) for v , in which the bridge/foot will later be completed. Thus the visit at v has the following steps:

1. Fix the foot, leg and bridge-strip of $P(v)$;
2. For each child u of v in T_1 in left-to-right order, call the algorithm recursively for u ;
3. Fix the bridge and the body of $P(v)$;
4. Fix the foot-strips for the children of v in T_2 .

Details of Step 4: We explain step 4 first, since it is vital for the other steps. Any vertex u that is a child of v in T_2 can be shown to come after v in the left-to-right depth-first search order of T_1 . For all such vertices u , we reserve a foot-strip of height $\leq \varepsilon$ for $P(u)$ at the right side of the body of $P(v)$. These foot-strips are placed starting at the bottom of the body of $P(v)$, all adjacent to each other, and assigned according to the counter-clockwise order of edges around v . Choosing the heights small enough ensures that all foot-strips fit along the body. We presume that this operation has also been applied when we place $P(v_2)$, so that for every vertex $v \neq v_1, v_2$, the foot strip of v is already set when we visit v .

Details of Step 1: When we visit vertex v , the foot-strip of $P(v)$ hence has been set. The bridge-strip of $P(f_1(v))$ has also been fixed since $f_1(v)$ is visited before v . We fix the bridge strip of $P(v)$ with height $\leq \varepsilon$ just under the bridge-strip of $P(f_1(v))$ so that the two bridge strips touch each other.

To fix foot and leg of v , we have two cases. If $f_2(v)$ is not the left sibling of v in T_1 , then the foot extends (in the foot strip of v) until the body of the leftmost child of v in T_3 , or until the leg of $f_1(v)$ if there is no such child. The leg then extends upwards until the bridge strip of v . If $f_2(v)$ is the left sibling of v in T_1 , then the foot of v vanishes; we extend the leg of v from the bottom of the footstrip for v until the bottom of the bridge

strip of v . Note that in either case all parts of neighbors required for this drawing have been placed already.

Details of Step 3: To fix the bridge of $P(v)$, there are again two cases. If v has children in T_1 , then we complete the bridge of $P(v)$ so that it extends from the leg of $P(v)$ to the rightmost side of any child of v in T_1 (they all have been drawn already.) If v has no children in T_1 , then the bridge vanishes and the body of v contains the leg of v . Then we fix the body of $P(v)$ so that it extends from the bottom of the bridge-strip of $P(f_1(v))$ to the foot-strip of $P(f_3(v))$. This is possible because $f_2(f_3(v))$ always precedes v in the traversal of T_1 , and so the foot-strip of $P(f_3(v))$ has already been fixed. After knowing exactly the area consumed by foot, leg and bridge of $P(v)$, as well as the height of the body, we can choose the width so that the area of $P(v)$ is $w(v)$. (Since the foot, leg and bridge together consume little area, the width of the body is positive.)

Fig 2(b) illustrates a proportional contact representation of the maximal planar graph in Fig. 2(a) computed with our algorithm. A more detailed step-by-step correction and details of the proof of correctness is given in the full version of the paper [1].

The linear-time implementation consists of computing a Schnyder realizer of G [17], and the traversal of tree T_1 together with the local computations of the different parts of the polygons. \square

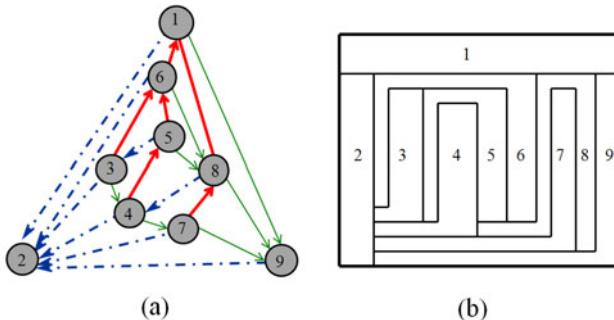


Fig. 2. (a) A maximal planar graph G , (b) a proportional contact representation of G . Only vertex 8 has 10 sides, since this only happens if a vertex has children in all of T_1 , T_2 and T_3 .

So we have now established that 10 sides are sufficient for proportional contact representation with rectilinear polygons. Yeap and Sarrafzadeh [22] gave an example of a maximal planar graph, which is also a planar 3-tree, for which at least 8-sided polygons are necessary. In very recent work [2] we managed to prove that 8-sided polygons are also sufficient. However, in contrast to the 10-gon construction given above, the proof of this result is not constructive, and the representation can be found only via numerical approximation. So while the construction with 10-gons is not theoretically best possible, it is probably of higher interest for practical settings.

3 Representations of Planar 3-trees

Here we describe proportional contact representations of planar 3-trees with fewer sides (8) in each polygon. A *3-tree* is either a 3-cycle or a graph G with a vertex v of degree

three in G such that $G - v$ is a 3-tree and the neighbors of v are adjacent. If G is planar, then it is called a *planar 3-tree*. A *plane 3-tree* is a planar embedding of a planar 3-tree. It is easy to see that starting with a 3-cycle, any planar 3-tree can be formed by recursively inserting a vertex inside a face and adding an edge between the newly added vertex and each of the three vertices on the face [313].

Using this simple construction, we can create in linear time a *representation tree* for G , which is an ordered rooted ternary tree T_G spanning all the internal vertices of G . The root of T_G is the first vertex we have to insert into the face of the three outer vertices. Adding a new vertex v in G will introduce three new faces belonging to v . The first vertex w we add in each of these faces will be a child of v in T_G . The correct order of T_G can be obtained by adding new vertices according to the counterclockwise order of the introduced faces. For any vertex v of T_G , we denote by U_v , the set of the descendants of v in T_G including v . The *predecessors* of v are the neighbors of v in G that are not in U_v . Clearly each vertex of T_G has exactly three predecessors. We now have the following lemma.

Lemma 1. *Let $G = (V, E)$ be a plane 3-tree and let $w : V \rightarrow \mathbb{R}^+$ be a weight function. Then a proportional contact representation of G can be obtained in linear time where each vertex of G is represented by an 8-sided rectilinear polygon.*

Proof. Let T_G be the representation tree of G . For any vertex v of T_G , let $W(v)$ denote the summation of the weights assigned to each of the vertices in U_v . A linear-time bottom-up traversal of T_G is sufficient to compute $W(v)$ for each vertex v of G . In the followings, we construct a proportional contact representation of G inside any rectangle R with area equal to the summation of the weights for all the vertices of G .

Let a, b, c be the three outer vertices of G in the counterclockwise order. We first draw the polygons for a, b and c . We cut a rectangle $P(a)$ with area $w(a)$ for a from the top of R , cut a rectangle $P(b)$ with area $w(b)$ from the left side of $R - P(a)$ and cut an L-shaped strip $P(c)$ of area $w(c)$ for c from the right side and the bottom of $R - P(a) - P(b)$, as illustrated in Fig. 3(a). We now draw the polygons for the vertices in T_G inside the rectangle $R - P(a) - P(b) - P(c)$ by a top-down traversal of T_G . While we traverse a vertex v of T_G , we recursively draw the polygons for the vertices of U_v inside a rectangle R_v with area $W(v)$ such that R_v shares two of its sides with the polygon for one of the predecessors of v and the other two sides with the polygons for the other two predecessors. Note that this condition holds for the rectangle $R - P(a) - P(b) - P(c)$ representing the root of T_G . Let v be a vertex of T_G with predecessors p_1, p_2, p_3 and let $pqrs$ be the rectangle with area $W(v)$ where ps, pq and qrs are part of the boundary of the polygons for p_1, p_2 and p_3 , respectively. From $pqrs$, we then cut three rectangles $R_1 = t_1t_2rs, R_2 = pt_3t_4t_5$ and $R_3 = qt_6t_7t_8$ with areas $W(u_1), W(u_2)$ and $W(u_3)$, respectively, as illustrated in Fig. 3(b), where u_1, u_2 and u_3 are the three children of v in T_G (some of them might be empty). Then the 8-sided polygon obtained by $pqrs - R_1 - R_2 - R_3$ has area $w(v)$ and has common boundary with all of the polygons representing its predecessors. Finally, we recursively fill out the rectangles R_1, R_2, R_3 by polygons representing the vertices in U_{u_1}, U_{u_2} and U_{u_3} , respectively. Since the polygon representing each vertex v of G can be computed in constant time, the time complexity for constructing the representation of G is linear. \square

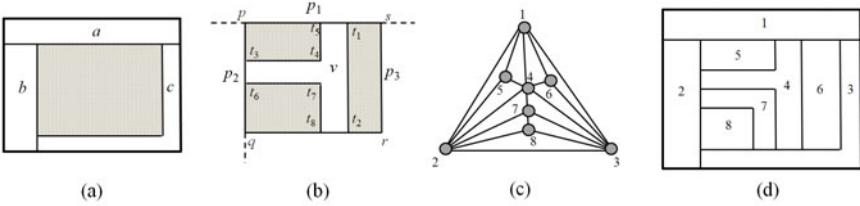


Fig. 3. (a)–(b) Illustration for the proof of Lemma 1. (c) a planar 3-tree G , and (d) a proportional contact representation of G .

Fig. 3(d) illustrates our construction for the planar 3-tree in Fig. 3(c). The algorithmic upper bound of 8 sides per polygon is also matched with the corresponding lower bound with a planar 3-tree for which at least 8-sided polygons are necessary in a contact representation with rectilinear polygons [22]. We thus have the following result.

Theorem 2. *Polygons with 8 sides are always sufficient and sometimes necessary for proportional contact representations of planar 3-trees with rectilinear polygons.*

4 Representations for Maximal Outer-Planar Graphs

Here we describe proportional contact representations of maximal outer-planar graphs with even fewer sides (6 and 4) in each polygon. An *outer-planar graph* is a graph that has an *outer-planar embedding*, i.e., a planar embedding with every vertex in the outer face. An outer-planar graph to which no edges can be added without violating outer-planarity is a *maximal outer-planar graph*. It is easy to see that each internal face in an outer-planar embedding of a maximal outer-planar graph is a triangle, and for $n \geq 3$ the outer-face is a simple cycle containing all vertices. We will give a linear-time algorithm to construct a proportional contact representation of a maximal outer-planar graph with rectangles. Before that, we need the following definitions.

Let Γ be a contact representation using rectangles for vertices (but with the outside not necessarily a rectangle). Let B be the bounding box of Γ . We say that a vertex v *occupies the top* of a representation Γ if there exists a horizontal line ℓ such that the rectangle representing v is exactly the intersection of B with the upper half-space of ℓ . In other words, the rectangle of v contains all of the top end of the bounding box of Γ . Similarly we define that a vertex v *occupies the right* of Γ .

Lemma 2. *Let G be a maximal outer-planar graph, and let (s, t) be an edge on the outer-face, with s before t in clockwise order. Then a proportional contact-representation Γ of G with rectangles can be computed in linear time such that s occupies the top of Γ and t occupies the right of $\Gamma - s$.*

Proof. We give an algorithm that recursively computes Γ . Constructing Γ is easy when G is a single edge (s, t) ; see Fig. 4. We thus assume that G has at least 3 vertices. Let x be the (unique) third vertex on the inner face that is adjacent to (s, t) . Then graph G can be split into two graphs at vertex x and edge (s, t) : $G[s, x]$ consists of the graph induced

by all vertices between s and x in counter-clockwise order around the outer-face, and $G[x, t]$ consists of the graph induced by the vertices between t and x .

Recursively draw $G[s, x]$ and remove s from it; call the result Γ_s . Recursively draw $G[x, t]$ and remove x and t from it; call the result Γ_t . Then scale the width of Γ_t until the bounding box of Γ_t is less wide than the rectangle of x in Γ_s . To maintain a proportional contact representation, scale the height of Γ_t by the inverse of the scale-factor for the width. Now Γ_t can be attached at the bottom right end of the representation of x in Γ_s . Add a rectangle for t on the right that spans the whole height (and extends below it at the bottom), and make its width such that its area is as prescribed for t . Add a rectangle for s such that it spans the whole width (and extends below it at the left), and make its height such that its area is as prescribed for s . This gives the desired representation.

We now show that the above algorithm can be implemented in linear time. In order to do this, we make sure that all coordinates in the representation are scaled at most once. Let T be the dual graph of G minus the vertex for the outer-face; it is easy to see that T is a tree with maximum degree three. Root T at the vertex that corresponds to the inner face $\{s, x, t\}$; then the subtrees of T correspond to the dual trees of the subgraphs. Rather than re-scaling Γ_t at each recursive step, we only re-scale the bounding box of Γ_t and store at the node of T that represents $G[t, x]$ the scale-factors for the width and height that must be applied to all nodes in Γ_t . At the end of the algorithm a linear-time top-down traversal finds the scaling factor for each vertex v of T by multiplying all the scaling factors stored along the path from v_x to v . Then with another linear-time top-down traversal of T we can compute the coordinates of all the points in Γ , which concludes the construction. \square

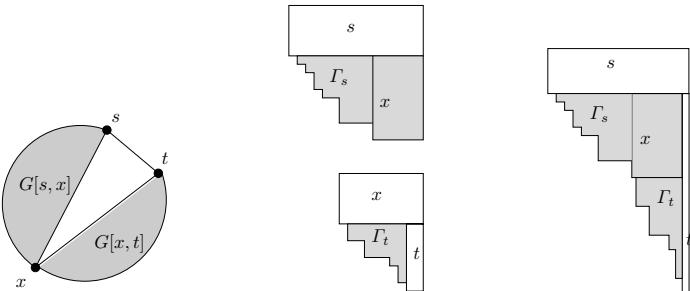


Fig. 4. Combining the drawings of two subgraphs

Since a rectangle is a rectilinear polygon with the fewest sides possible, the representation obtained by the above algorithm is also optimal. However, the outer boundary of the representation obtained by our construction has size $\Theta(n)$. It was already known that the outer-face cannot be a rectangle if the vertices are rectangles [16], but we improve this to a stronger result:

Lemma 3. *There exists a maximal outer-planar graph for which any contact representation with rectangles requires $\Omega(n)$ sides on the outer-face.*

Proof. Consider any maximal outer-planar graph G such that $\lfloor n/2 \rfloor$ vertices have degree two (any maximal outer-planar graph whose inner dual is a full binary tree suffices). Suppose Γ is a proportional contact representation of G with rectangles. Since rectangles are convex, no two of them can share two sides. Therefore any vertex v of degree 2 shares at most two of its sides with other vertices, and so at least two of its sides with the outer boundary of Γ . Furthermore, these two sides must be consecutive on $P(v)$, since otherwise v would be a cut vertex in G . The common endpoint of these two sides is then a corner of the outer boundary of Γ , so the outer-face has at least $\lfloor n/2 \rfloor$ sides. \square

Lemma 3 implies that there exist outer-planar graphs for which any contact representation with an the outer-boundary of constant size requires at least one of the polygons to have at least six sides. With the following lemma we show that this lower bound of six sides can also be matched with any given weights.

Lemma 4. *Let $G = (V, E)$ be a maximal outer-planar graph and let $w : V \rightarrow \mathbb{R}^+$ be a weight function. Then a proportional contact-representation Γ of G with 6-sided rectilinear polygons can be computed in linear time such that the outer-boundary of Γ is a rectangle.*

Proof. It is quite straightforward to prove this by analyzing the structure of an outer-planar graph, but it also follows from both previous results in this paper. We only sketch this here.

First, if G is maximal outer-planar, then we can add one vertex v_0 to it that is adjacent to all others. Then create a Schnyder realizer such that v_0 is the root of tree T_1 . Then all its incident edges are labeled with 1, which means that all other vertices are leaves in tree T_1 . Apply our construction from Section 2. One can easily verify that vertices that are leaves in T_1 are drawn with 6-gons in this construction. Omitting the added vertex v_0 (which is a rectangle that spans the top) yields the desired representation.

As a second proof, observe that $G \cup v_0$ is also a 3-tree, and moreover, any vertex v has at most two children in T_G . Apply the construction of Section 3 and split the rectangle of weight $W(v)$ in such a way that $P(v)$ has at most 6 sides; one can verify that this is always possible if v has at most two children in T_G . \square

Summing up all the results in this section, we have the following theorem.

Theorem 3. *For a rectilinear proportional contact representation of a maximal outer-planar graph, rectangles are always sufficient and necessary, and six-sided polygons are sometimes necessary (and always sufficient) when the outer-boundary has a constant number of sides.*

5 Conclusion

We gave an algorithm for a proportional contact representation of a maximal planar graph with 10-sided rectilinear polygons, which improves on the previously known upper bound of 12.

We also described algorithms for special classes of planar graphs with 8-sided rectilinear polygons and a similar approach might be extended to general planar graphs. Finally, we described algorithms for 6-sided and 4-sided rectilinear representation for outer-planar graphs.

All algorithms in this paper can be implemented in linear time, and require nothing more complicated than Schnyder realizers and other elementary planar graph manipulations. In contrast, the very recent improvement in the number of sides to 8 [2], the proof is non-constructive and requires numerical approximations to find the contact representation. Finding a constructive proof (and preferably linear-time algorithm) to construct 8-sided proportional contact representations of maximal planar graphs remains open.

References

1. Alam, M.J., Biedl, T., Felsner, S., Gerasch, A., Kaufmann, M., Kobourov, S.G.: Linear-time algorithms for proportional contact graph representations. Technical Report CS-2011-19, University of Waterloo (2011)
2. Alam, M.J., Biedl, T., Felsner, S., Kaufmann, M., Kobourov, S., Ueckert, T.: Computing cartograms with optimal complexity (2011) (submitted)
3. Biedl, T., Ruiz Velázquez, L.E.: Drawing Planar 3-Trees with Given Face-Areas. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 316–322. Springer, Heidelberg (2010)
4. Biedl, T., Ruiz Velázquez, L.E.: Orthogonal Cartograms with Few Corners Per Face. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 98–109. Springer, Heidelberg (2011)
5. Buchsbaum, A.L., Gansner, E.R., Procopiuc, C.M., Venkatasubramanian, S.: Rectangular layouts and contact graphs. ACM Transactions on Algorithms 4(1) (2008)
6. de Berg, M., Mumford, E., Speckmann, B.: On rectilinear duals for vertex-weighted plane graphs. Discrete Mathematics 309(7), 1794–1812 (2009)
7. Gansner, E.R., Hu, Y.F., Kaufmann, M., Kobourov, S.G.: Optimal Polygonal Representation of Planar Graphs. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 417–432. Springer, Heidelberg (2010)
8. He, X.: On floor-plan of plane graphs. SIAM Journal of Computing 28(6), 2150–2167 (1999)
9. Heilmann, R., Keim, D.A., Panse, C., Sips, M.: Recmap: Rectangular map approximations. In: 10th IEEE Symp. on Information Visualization (InfoVis 2004), pp. 33–40 (2004)
10. Kawaguchi, A., Nagamochi, H.: Orthogonal Drawings for Plane Graphs with Specified Face Areas. In: Cai, J.-Y., Cooper, S.B., Zhu, H. (eds.) TAMC 2007. LNCS, vol. 4484, pp. 584–594. Springer, Heidelberg (2007)
11. Koźmiński, K., Kinnen, E.: Rectangular duals of planar graphs. Networks 15, 145–157 (1985)
12. Liao, C.-C., Lu, H.-I., Yen, H.-C.: Compact floor-planning via orderly spanning trees. Journal of Algorithms 48, 441–451 (2003)
13. Mondal, D., Nishat, R.I., Rahman, M.S., Alam, M.J.: Minimum-area drawings of plane 3-trees. In: CCCG, pp. 191–194 (2010)
14. Rahman, M.S., Miura, K., Nishizeki, T.: Octagonal drawings of plane graphs with prescribed face areas. Computational Geometry 42(3), 214–230 (2009)
15. Ringel, G.: Equiareal graphs. In: Bodendiek, R. (ed.) Contemporary Methods in Graph Theory, pp. 503–505. Wissenschaftsverlag (1990)
16. Rinsma, I.: Nonexistence of a certain rectangular floorplan with specified area and adjacency. Environment and Planning B: Planning and Design 14, 163–166 (1987)

17. Schnyder, W.: Embedding planar graphs on the grid. In: SODA, pp. 138–148 (1990)
18. Sun, Y., Sarrafzadeh, M.: Floorplanning by graph dualization: *L*-shaped modules. *Algorithmica* 10(6), 429–456 (1993)
19. Thomassen, C.: Plane cubic graphs with prescribed face areas. *Combinatorics, Probability & Computing* 1, 371–381 (1992)
20. Ungar, P.: On diagrams representing graphs. *J. London Math. Soc.* 28, 336–342 (1953)
21. van Kreveld, M.J., Speckmann, B.: On rectangular cartograms. *Computational Geometry* 37(3), 175–187 (2007)
22. Yeap, K.-H., Sarrafzadeh, M.: Floor-planning by graph dualization: 2-concave rectilinear modules. *SIAM Journal on Computing* 22, 500–526 (1993)

Fully Retroactive Approximate Range and Nearest Neighbor Searching

Michael T. Goodrich and Joseph A. Simons

Department of Computer Science, University of California, Irvine, USA

Abstract. We describe fully retroactive dynamic data structures for approximate range reporting and approximate nearest neighbor reporting. We show how to maintain, for any positive constant d , a set of n points in \mathbb{R}^d indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support, for any small constant $\epsilon > 0$, $(1 + \epsilon)$ -approximate range reporting queries at any point in the timeline in $O(\log n + k)$ time, where k is the output size. We also show how to answer $(1 + \epsilon)$ -approximate nearest neighbor queries for any point in the past or present in $O(\log n)$ time.

1 Introduction

Spatiotemporal data types are intended to represent objects that have geometric characteristics that change over time.

The important feature of such objects is that their critical characteristics, such as when they appear and disappear in a data set, exist in a *timeline*. The representation of such objects has a number of important applications, for instance, in video and audio processing, geographic information systems, and historical archiving. Moreover, due to data editing or cleaning needs, spatiotemporal data sets may need to be updated in a dynamic fashion, with changes that are made with respect to the timeline. Thus, in this paper we are interested in methods for dynamically maintaining geometric objects that exist in the context of a timeline. Queries and updates happen in *real time*, but are indexed in terms of the timeline.

In this paper, we are specifically interested in the dynamic maintenance of a set of d -dimensional points that appear and disappear from a data set in terms of indices in a timeline, for a given fixed constant $d \geq 1$. Points should be allowed to have their appearance and disappearance times changed, with such changes reflected forward in the timeline. We also wish to support time-indexed approximate range reporting and nearest-neighbor queries in such data sets. That is, we are interested in the dynamic maintenance of spatiotemporal point sets with respect to these types of geometric queries.

1.1 Related Work

Approximate Searching. Arya and Mount [3] introduce the approximate nearest neighbor problem for a set of points, S , such that given a query point q , a

point of S will be reported whose distance from q is at most a factor of $(1 + \epsilon)$ from that of the true nearest neighbor of q . Arya *et al.* [5] show that such queries can be answered in $O(\log n)$ time for a fixed constant $\epsilon > 0$. Chan [9] shows how to achieve a similar bound. Arya and Mount [4] also introduce the approximate range searching problem for a set, S , where a range R (e.g. a sphere or rectangle) of diameter w is given as input and every point in S that is inside R is reported as output and no point that is more than a distance of ϵw outside of R is reported. Let k be the number of points reported. Arya and Mount show that such queries can be answered in $O(\log n + k)$ time for fixed constant $\epsilon > 0$. Eppstein *et al.* [18] describe the skip quadtree structure, which supports $O(\log n + k)$ -time approximate range searching as well as $O(\log n)$ -time point insertion and deletion.

Our approach to solving approximate range searching and approximate nearest neighbor problems are based on the quadtree structure [28]. In this structure, regions are defined by squares in the plane, which are subdivided into four equal-sized squares for any regions containing more than a single point. So each internal node in the underlying tree has up to four children and regions have optimal aspect ratios. Typically, this structure is organized in a compressed fashion [7], so that paths in the tree consisting of nodes with only one non-empty child are compressed to single edges. This structure is related to the balanced box decomposition (BBD) trees of Arya *et al.* [3, 4, 5], where regions are defined by hypercubes with smaller hypercubes subtracted away, so that the height of the decomposition tree is $O(\log n)$. Similarly, Duncan *et al.* [17] define the balanced aspect-ratio (BAR) trees, where regions are associated with convex polytopes of bounded aspect ratio, so that the height of the decomposition tree is $O(\log n)$.

Computational Geometry with respect to a Timeline. Although we are not familiar with any previous work on retroactive d -dimensional approximate range searching and nearest-neighbor searching, we nevertheless would like to call attention to the fact that incorporating a time dimension to geometric constructions and data structures is well-studied in the computational geometry literature.

- Atallah [6] studies several *dynamic computational geometry* problems, including convex hull maintenance, for points moving according to fixed trajectories.
- Subsequently, a number of researchers have studied geometric motion problems in the context of *kinetic data structures* (e.g., see [22]). In this framework, a collection of geometric objects is moving according to a fixed set of known trajectories, and changes can only happen in the present.
- Driscoll *et al.* [16] introduce the concept of *persistent data structures*, which support time-related operations where updates occur in the present and queries can be performed in the past, but updates in the past fork off new timelines rather than propagate changes forward in the same timeline.

All of this previous work differs from the approach we are taking in this paper, since in these previous approaches objects are not expected to be retroactively changed “in the past.”

Demaine *et al.* [13] introduce the concept of *retroactive data structures*, which is the framework we follow in this paper. In this approach, a set of data is maintained with respect to a timeline. Insertions and deletions are defined with respect to this timeline, so that each insertion has a time parameter, t , and so does each deletion. Likewise, queries are performed with respect to the time parameter as well. The difference between this framework and the dynamic computational geometry approaches mentioned above, however, is that updates can be done retroactively “in the past,” with the changes necessarily being propagated forward. If queries are only allowed in the current state (i.e., with the highest current time parameter), then the data structure is said to be *partially retroactive*. If queries can be done at any point in the timeline, then the structure is said to be *fully retroactive*. Demaine *et al.* [13] describe a number of results in this framework, including a structure for fully-retroactive 1-dimensional successor queries with $O(\log^2 n)$ -time performance. They also show that any data structure for a decomposable search problem can be converted into a fully retroactive structure at a cost of increasing its space and time by a logarithmic factor.

Acar *et al.* [1] introduce an alternate model of retroactivity, which they call *non-oblivious* retroactivity. In this model, one maintains the historical sequence of queries as well as insertions and deletions. When an update is made in the past, the change is not necessarily propagated all the way forward to the present. Instead, a non-oblivious data structure returns the first operation in the timeline that has become *inconsistent*, that is an operation whose return value has changed because of the retroactive update. As mentioned above, we only consider the original model of retroactivity as defined by Demaine *et al.* [13] in this paper.

Blelloch [8] and Giora and Kaplan [20] consider the problem of maintaining a fully retroactive dictionary that supports successor or predecessor queries. They both base their data structures on a structure by Mortensen [25], which answers fully retroactive one dimensional range reporting queries, although Mortensen framed the problem in terms of two dimensional orthogonal line segment intersection reporting. In this application, the x -axis is viewed as a timeline for a retroactive data structure for 1-dimensional points. The insertion of a segment $[(x_1, y), (x_2, y)]$ corresponds to the addition of an insert of y at time x_1 and a deletion of y at time x_2 . Likewise, the removal of such a segment corresponds to the removal of these two operations from the timeline. For this 1-dimensional retroactive data structuring problem, Blelloch and Giora and Kaplan give data structures that support queries and updates in $O(\log n)$ time. Dickerson *et al.* [15] describe a retroactive data structure for maintaining the lower envelope of a set of parabolic arcs and give an application of this structure to the problem of cloning a Voronoi diagram from a server that answers nearest-neighbor queries.

1.2 Our Results

In this paper, we describe fully retroactive dynamic data structures for approximate range reporting and approximate nearest neighbor searching. We show how to maintain, for any positive constant, $d \geq 1$, a set of n points in \mathbb{R}^d indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support, for any small constant $\epsilon > 0$, $(1 + \epsilon)$ -approximate range reporting queries at any point in the timeline in $O(\log n + k)$ time, where k is the output size. Note that in this paper we consider circular ranges defined by a query point q and radius r . We also show how to answer $(1 + \epsilon)$ -approximate nearest neighbor queries for any point in the past or present in $O(\log n)$ time. Our model of computation is the real RAM, as is common in computational geometry algorithms (e.g., see [29]).

The main technique that allows us to achieve these results is a novel, multidimensional version of fractional cascading, which may be of independent interest. Recall that in the (1-dimensional) *fractional cascading* paradigm of Chazelle and Guibas [11, 12], one searches a collection of sorted lists (of what are essentially numbers), which are called *catalogs*, that are stored along nodes in a search path of a catalog graph, G , for the same element, x . In *multidimensional fractional cascading*, one instead searches a collection of finite subsets of \mathbb{R}^d for the same point, p , along nodes in a search path of a catalog graph, G . In our case, rather than have each catalog represented as a one-dimensional sorted list, we instead represent each catalog as a multidimensional “sorted list,” with points ordered as they would be visited in a space-filling curve (which is strongly related to how the points would be organized in a quadtree, e.g., see [7]).

By then sampling in a fashion inspired by one-dimensional fractional cascading, we show¹ how to efficiently perform repeated searching of multidimensional catalogs stored at the nodes of a search path in a suitable catalog graph, such as a segment tree (e.g., see [29]), with each of the searches involving the same d -dimensional point or region.

Although it is well known that space-filling curves can be applied to the problem of approximate nearest neighbor searching, we are not aware of any extension of space-filling curves to approximate range reporting. Furthermore, we believe that we are the first to leverage space-filling curves in order to extend dynamic fractional cascading into a multi-dimensional problem.

2 A General Approach to Retroactivity

Recall that a query Q is *decomposable* if there is a binary operator \square (computable in constant time) such that $Q(A \cup B) = \square(Q(A), Q(B))$. Demaine *et al.* [13] showed that we can make any decomposable search problem retroactive by maintaining each element ever inserted in the structure as a line segment along a time dimension between the element’s insertion and deletion times.

¹ The details for our constructions are admittedly intricate, so some details of proofs are given in the full version of this paper [21].

Thus each point p in R^d is now represented by a line segment parallel to the time-axis in R^{d+1} dimensions. For example when extending the query to be fully-retroactive, a one-dimensional successor query becomes a two-dimensional vertical ray shooting query, and a one-dimensional range reporting query becomes a two-dimensional orthogonal segment intersection query.

Thus, we maintain a segment tree to allow searching over the segments in the time dimension, and augment each node of the segment tree with a secondary structure supporting our original query in d dimensions. Let S be the set of nodes in the segment tree on a root-to-leaf path searching down for t in the time dimension. To answer a fully-retroactive query, we perform the same d -dimensional query at each node in S . This transformation costs an extra $\log n$ factor in space, query time, and update time, which we would nevertheless like to avoid.

Recall that Mortensen [25] and Giora and Kaplan [20] both solve the fully-retroactive versions of decomposable search problems, and are both able to avoid the extra $\log n$ factor in query and update time. Therefore inspired by their techniques, we propose the following as a general strategy for optimally solving the fully-retroactive version of any decomposable search problem.

1. Suppose we have an optimal data structure D for the non-retroactive problem which supports queries in polylogarithmic time $T(n)$.
2. Represent each d -dimensional point as a line segment in $d + 1$ dimensions.
3. Build a weight-balanced segment tree with polylogarithmic branching factor over the segments as described by [20].
4. Augment the root of the segment tree with an optimal search structure D .
5. Augment each node of the segment tree with a colored dynamic fractional cascading (CDFC) data structure.
6. Perform a retroactive query at time t by performing the non-retroactive query on the non-retroactive data structure at the root of the segment tree, and for each node on the search path for t in the segment tree, perform the query in each of the CDFC structures (Figure II).

The CDFC data structure must be cleverly tuned to support a *colored* (but non-retroactive) version of the d -dimensional query in $O(T(n) \cdot \log \log n / \log n)$ time. In a colored query, each element in the structure is given a color, and the query also specifies a set of colors. We only return elements whose color is contained in the query set. The colors are required because the segment tree has high degree. Each color represents a pair of children of the current node in the segment tree. Thus we encode which segments overlap the search path via their colors. Since the segment tree has a polylogarithmic branching factor, we spend $T(n)$ time searching at the root and $O(T(n) \cdot \log \log n / \log n)$ time searching in the CDFC structures at each of the $O(\log / \log n \log n)$ nodes. Therefore, the total time required by a query is an optimal $O(T(n))$. Updates follow a similar strategy, but may require us to periodically rebuild sections of the segment tree. We can still achieve the desired (amortized) update time, and the analysis closely follows [20].

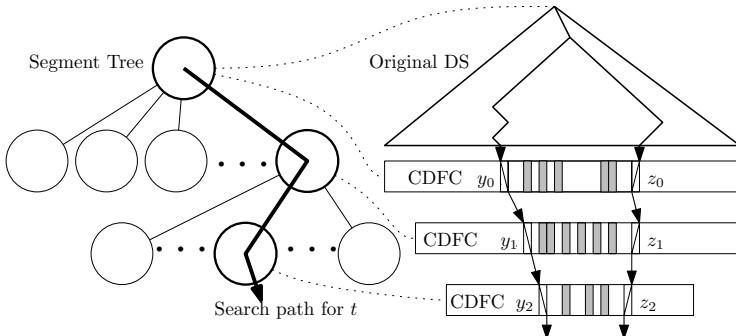


Fig. 1. An application of our strategy to a fully-retroactive one-dimensional range reporting query. Shaded elements represent elements with colors indicating they are present at time t .

One of the key difficulties in applying this strategy lies in the design of the colored dynamic fractional cascading data structure, especially in problems where the dimension $d > 1$. In fact, the authors are not aware of any previous application of dynamic fractional cascading techniques to any multidimensional search problem. However, in the following we show how techniques using space filling curves can be applied to extend the savings of fractional cascading into a multidimensional domain. First, we apply the above strategy in the simpler case when $d = 1$. Then we extend this result using space filling curves to support Fully-Retroactive range reporting queries and approximate nearest neighbor queries in \mathbb{R}^d . Note that in one dimension a nearest neighbor query reduces to a successor and predecessor query.

Lemma 1. *There exists a colored dynamic fractional cascading data structure which supports updates in $O(\log \log N)$ amortized time, colored successor and predecessor queries in $O(\log \log N)$ worst case time and colored range reporting queries in $O(\log \log N + k)$ worst case time, where N is the number of elements stored and k is the number of elements reported.*

Proof: We extend the generalized union-split-find structure of [20] to also support colored range queries. See [21] for details. \square

Space Filling Curves. The z-order, due to Morton [26], is commonly used to map multidimensional points down to one dimension. By now space filling curves are well-studied and multiple authors have applied them specifically to approximate nearest neighbor queries [14, 10, 9, 24]. However, we extend their application to approximate range searching as well. Furthermore, we believe that we are the first to leverage space-filling curves to extend dynamic fractional cascading techniques to multidimensional problems such as these.

Lemma 2. *The set of points in any quadtree cell rooted at $[0, 1]^d$ form a contiguous interval in the z-order.*

Proof: Due to Bern *et al.* [7]. Also see [21]. □

Lemma 3. *Let P be a set of points in \mathbb{R}^d . Define a constant $c = \sqrt{d}(4d+4) + 1$. Suppose that we have $d+1$ lists $P + v^j, j = 0, \dots, d$, each one sorted according to its z-order. We can find a query point q 's c -approximate nearest neighbor in P by examining the the $2(d+1)$ predecessors and successors of q in the lists.*

Proof: Due to Liao *et al.* [24]. Also see [21] for details. □

3 Main Results

In this section we give our primary results: data structures for fully-retroactive approximate range queries and fully-retroactive approximate nearest neighbor (ANN) queries. Recall that an approximate range query $\text{report}(q, r, \epsilon, t)$ defines an inner range Q^- , the region within a radius r of the query point q and an outer range Q^+ , the area within a radius of $(1+\epsilon)r$ of q . We want to return all the points inside Q^- and exclude all points outside Q^+ for a particular point in time t . Points that fall between Q^- and Q^+ at time t may or may not be reported. Points not present at time t will never be reported.

Theorem 1 (Fully-Retroactive Approximate Range Queries). *For any positive constant $d \geq 1$, we can maintain a set of n points in \mathbb{R}^d indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support for any small constant $\epsilon > 0$, $(1+\epsilon)$ -approximate range reporting queries at any point in the timeline in $O(\log n + k)$ time, where k is the output size. The space required by our data structure is $O(n \log n / \log \log n)$.*

Proof: We follow the general strategy outlined in Section 2. We augment the root of the segment tree with a skip quadtree [18], an optimal structure for approximate range and nearest neighbor queries in \mathbb{R}^d . We also augment each node of the segment tree with a specialized CDFC structure which we now describe.

We extend the CDFC structure from Lemma 1 to maintain d -dimensional points such that given a query set of colors C_q and d -dimensional quadtree cell, it returns all points contained in that cell with colors in C_q . By Lemma 2, for all points y, q, z such that $y < q < z$ in the z-order, any quadtree cell containing y and z must also contain q . Furthermore, for a given quadtree, each cell is uniquely defined by the leftmost and rightmost leaves in the cell's subtree. Therefore, the d -dimensional cell query reduces to a one-dimensional range query in the z-order on the unique points which define the quadtree cell (Figure 2). Thus, by leveraging Lemma 1 and maintaining the points according to their z-order, we support the required query in $O(\log \log n + k)$ time.

The skip quadtree contains all the points ever inserted into our data structure, irrespective of the time that they were inserted or deleted. The inner and outer range of a query partition the set of quadtree cells into three subsets: the *inner*

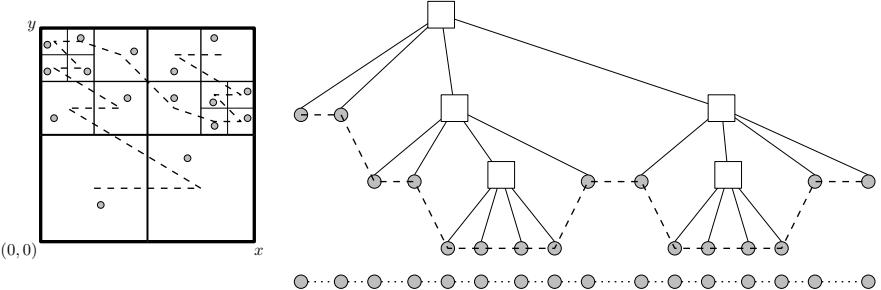


Fig. 2. The z-order curve corresponds to the order the leaves would be visited in an in-order traversal of the quadtree. We can store the points in a linked list in this order. Any element between two elements i, j in the linked list must fall in the same quadtree cell as i and j .

cells, which are completely contained in Q^+ , the *outer cells*, which are completely disjoint from Q^- , and the stabbing cells, which partially overlap Q^+ and Q^- . Eppstein *et al.* [19] showed that a skip quadtree can perform an approximate range query in $O(\log n + \epsilon^{1-d})$ time expected and with high probability, or worst case time for a deterministic skip quadtree. Using their algorithm we can find the set I of $O(\epsilon^{1-d})$ disjoint inner cells in the same time bound. Based on the correctness of their algorithm we know that the set of points we need to report are covered by these cells. We report the points present at time t as follows: For each cell $i \in I$, if i is a leaf, we report only the points in i , which are present at time t in constant time. Otherwise, we find the first and last leaves y_0 and z_0 according to an in-order traversal of i 's subtree in $O(\log n)$ time. Then we perform a fully-retroactive range query on cell i in the segment tree, using z_0 and y_0 to guide the query on cell i in the CDFC structures. Correctness follows since a point satisfies the retroactive range query if and only if it is in one of the cells we examine and is present at time t .

For each of the $O(\epsilon^{1-d})$ cells in I we spend $O(\log n + k_i)$ time and so the total running time is $O(\epsilon^{1-d} \log n + k) = O(\log n + k)$, since ϵ is a small constant. The skip quadtree is linear space, and thus the space usage is dominated by the segment tree. The total space required is $O(n \log n / \log \log n)$. \square

Corollary 1 (Fully-Retroactive ANN Queries.). *We can maintain, for any positive constant, $d \geq 1$, a set of n points in \mathbb{R}^d indexed by time such that we can perform insertions or deletions at any point in the timeline in $O(\log n)$ amortized time. We support, for any small constant $\epsilon > 0$, $(1 + \epsilon)$ -approximate nearest neighbor queries for any point in the past or present in $O(\log n)$ time. The space required by our data structure is $O(n \log n / \log \log n)$.*

Proof: By combining the data structure of [20] with Lemma 3 and storing the d -dimensional points in that structure according to their z-order, we already have a data structure for fully-retroactive c -approximate nearest neighbor queries.

However, c is polynomial function of d , and for $d = 2$, c is already greater than 15. In order to support $(1 + \epsilon)$ nearest neighbor queries for any $\epsilon > 0$, we require the data structure of the previous theorem.

We know from Lemma 3 that we can find a c -approximate nearest neighbor by using $d + 1$ different shifts of the points. Therefore, we augment our data structure so that instead of a single CDFC structure at each segment tree node, we keep an array of $d + 1$ CDFC structures corresponding to the z-order of each of the $d + 1$ sets of shifted points. Given a query point q , we can find the predecessor and successor of q at time t in each of the $d + 1$ z-orders in $O(d \log n)$ time. Out of these $2(d + 1)$ points, let p be the point that is closest to q . By Lemma 3, p is a c -approximate nearest neighbor. Let r be the distance between p and q . As observed by multiple authors [2][27][23], we can find a $(1 + \epsilon)$ -approximate nearest neighbor via a bisecting search over the interval $[r/c, r]$. This search requires $O(\log(1/\epsilon))$ fully-retroactive spherical emptiness queries. We can support a retroactive spherical emptiness query in $O(\log n)$ time with only a slight modification to our retroactive approximate range query. Instead of returning k points in the range, we just return the first point we find, or `null` if we find none. Thus the total time required is $O(d \log n + \log(1/\epsilon) \log n) = O(\log n)$ since we assume ϵ and d are constant. The space usage only increases by a factor of d when we store the $d + 1$ shifted lists, and thus the space required is still $O(n \log n / \log \log n)$. \square

References

1. Acar, U.A., Blelloch, G.E., Tangwongsan, K.: Non-oblivious retroactive data structures. Tech. Rep. CMU-CS-07-169, Carnegie Mellon University (2007)
2. Arya, S., da Fonseca, G.D., Mount, D.M.: A Unified Approach to Approximate Proximity Searching. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 374–385. Springer, Heidelberg (2010)
3. Arya, S., Mount, D.M.: Approximate nearest neighbor queries in fixed dimensions. In: Proc. 4th ACM-SIAM Sympos. Discrete Algorithms, pp. 271–280 (1993)
4. Arya, S., Mount, D.M.: Approximate range searching. Comput. Geom. Theory Appl. 17, 135–152 (2000)
5. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J. ACM 45, 891–923 (1998)
6. Atallah, M.J.: Some dynamic computational geometry problems. Computers and Mathematics with Applications 11(12), 1171–1181 (1985)
7. Bern, M., Eppstein, D., Teng, S.-H.: Parallel Construction of Quadtrees and Quality Triangulations. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1993. LNCS, vol. 709, pp. 188–199. Springer, Heidelberg (1993)
8. Blelloch, G.E.: Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In: 19th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 894–903 (2008)
9. Chan, T.M.: Approximate nearest neighbor queries revisited. Discrete and Computational Geometry 20, 359–373 (1998)

10. Chan, T.M.: A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions (2006) (manuscript)
11. Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. *Algorithmica* 1(3), 133–162 (1986)
12. Chazelle, B., Guibas, L.J.: Fractional cascading: II. Applications. *Algorithmica* 1, 163–191 (1986)
13. Demaine, E.D., Iacono, J., Langerman, S.: Retroactive data structures. *ACM Trans. Algorithms* 3 (May 2007)
14. Derryberry, J., Sheehy, D., Sleator, D.D., Woo, M.: Achieving spatial adaptivity while finding approximate nearest neighbors. In: Proceedings of the 20th Canadian Conference on Computational Geometry, CCCG 2008, pp. 163–166 (2008)
15. Dickerson, M.T., Eppstein, D., Goodrich, M.T.: Cloning Voronoi Diagrams Via Retroactive Data Structures. In: de Berg, M., Meyer, U. (eds.) *ESA 2010. LNCS*, vol. 6346, pp. 362–373. Springer, Heidelberg (2010)
16. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. *J. Comput. Syst. Sci.* 38, 86–124 (1989)
17. 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)
18. Eppstein, D., Goodrich, M.T., Sun, J.Z.: The skip quadtree: A simple dynamic data structure for multidimensional data. In: 21st ACM Symp. on Computational Geometry (SCG), pp. 296–305 (2005)
19. Eppstein, D., Goodrich, M.T., Sun, J.Z.: The skip quadtree: a simple dynamic data structure for multidimensional data. In: 21st ACM Symp. on Computational Geometry, pp. 296–305 (2005)
20. Giora, Y., Kaplan, H.: Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Trans. Algorithms* 5, 28:1–28:51 (2009)
21. Goodrich, M.T., Simons, J.A.: Fully Retroactive Approximate Range and Nearest Neighbor Searching. ArXiv e-prints (September 2011)
22. Guibas, L.J.: Kinetic data structures — a state of the art report. In: Agarwal, P.K., Kavraki, L.E., Mason, M. (eds.) *Proc. Workshop Algorithmic Found. Robot.*, pp. 191–209. A. K. Peters, Wellesley (1998)
23. Har-Peled, S.: A replacement for voronoi diagrams of near linear size. In: Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci., pp. 94–103 (2001)
24. Liao, S., Lopez, M., Leutenegger, S.: High dimensional similarity search with space filling curves. In: Proceedings of the 17th International Conference on Data Engineering, pp. 615–622 (2001)
25. Mortensen, C.W.: Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In: 14th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 618–627 (2003)
26. Morton, G.M.: A computer oriented geodetic data base; and a new technique in file sequencing, Tech. rep., IBM Ltd. (1966)
27. Mount, D.M., Park, E.: A dynamic data structure for approximate range searching. In: ACM Symp. on Computational Geometry, pp. 247–256 (2010)
28. Orenstein, J.A.: Multidimensional tries used for associative searching. *Inform. Process. Lett.* 13, 150–157 (1982)
29. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction, 3rd edn. Springer, Heidelberg (1990)

Compact Representation of Posets

Arash Farzan¹ and Johannes Fischer^{2,*}

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany
afarzan@mpi-inf.mpg.de

² Karlsruhe Institute of Technology, Karlsruhe, Germany
johannes.fischer@kit.edu

Abstract. We give a data structure for storing an n -element poset of width w in essentially minimal space. We then show how this data structure supports the most interesting queries on posets in either constant time, or in time that depends only on w and the size of the in-/output, but not on n . Our results also have direct applicability to DAGs of low width.

1 Introduction

Partially ordered sets (posets) are used to model an order relation between entities in many practical applications. For instance, in a computational geometric scenario it can be that such entities are points in space and the order relationship is dominance. Another scenario is where the entities are sets and the order relationships are containment. In graph theory, the entities are vertices of a directed acyclic graph (DAG), and the relationships are reachability. As the size of such objects grow, space-efficient representation of the corresponding poset becomes critical.

Random posets with no particular combinatorial properties are not compressible beyond $\Omega(n^2)$ bits [4]. Therefore, in this paper, we consider posets with small width. The width of posets is a very common structural parameter and is formally defined as the size of the largest antichain (set of incomparable objects).

Our representation of posets occupies the minimum amount of space in the worst case over all posets of a given width (up to lower order terms). It supports many useful operations on posets efficiently in the compressed form. The most fundamental operation is testing the existence of an order relation between two given elements. This is equivalent to the well-known problem of reachability testing in DAGs, and we support this operation in constant time. Thus, our work fits into the area of *compressed* or *succinct data structures*, where the aim is to store objects from a universe of cardinality L in $(1+o(1)) \lg L$ bits¹ of space, while still supporting elementary operations on the objects in fast time.

The problem of compact representation of posets is closely related to the problem of representing a DAG. Indeed, in this paper, the description of our

* Supported by the German Research Foundation (DFG).

¹ Throughout this paper, function \lg denotes the binary logarithm.

representation switches freely between posets and graphs. It is easy to observe that even a DAG of width as small as one requires $\Omega(n^2)$ bits for encoding (consider an initial path $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n$, where arbitrary additional links of the form $v_i \rightarrow v_j$ may be inserted for any pair $i < j - 1$.) Hence, encoding *entire* DAGs cannot use less space than encoding just the adjacency matrix of *arbitrary* graphs. To escape from this dilemma, our representation is an encoding of the *transitive reduction* of a DAG G , and not of G itself. The transitive reduction of G is the smallest subgraph of G whose transitive closure equals the transitive closure of G ; hence, our representation can also be viewed as a compact representation of the transitive closure of G .

In summary, our representation can be considered as a succinct encoding of a poset which supports operations on the poset, its corresponding DAG, and the transitive reduction of the DAG. We assume the word-RAM model, where the size of a word is $\Omega(\lg n)$ bits (n is the number of elements in the poset, or, equivalently, the number of vertices of the input DAG), and fundamental arithmetic operations on words (additions, shifts, etc.) can be computed in constant time.

1.1 Our Work in Context

Succinct data structures are widely available for trees and (un)directed graphs (e.g., [8, 5]). While these data structures allow for efficient local navigation, they do not encode long-range interactions such as reachability. For reachability in planar directed graphs, Thorup [10] gave a compact oracle. For lattices, Talamo and Vocca [9] gave an efficient representation. For general posets, we are only aware of the work by Daskalakis *et al.* [3], who show a representation for posets with n elements and width w that requires $O(nw)$ words and supports reachability in $O(1)$. We reduce the space to $O(nw)$ bits and extend the set of supported operations. Hence, this paper describes the *first* succinct data structure specifically designed for DAGs, using for the first time potentially less space than storing arbitrary graphs.

1.2 Contributions

Given a poset (\mathcal{P}, \preceq) of width w over n elements, let G be its corresponding DAG such that edge (v, w) exists iff $v \preceq w$ (hence G is transitively closed). Let G_r denote the transitive reduction of G (it is well-known that DAGs have a *unique* transitive reduction).

We present an encoding of G_r (or, equivalently, of \mathcal{P}) with $(1 + \epsilon)n \lg n + 2nw(1 + o(1))$ bits of space, which supports the following set of operations on \mathcal{P} , G , and G_r ($\epsilon > 0$ is an arbitrarily small constant):

reachability(x, y): this is to test the order relation $x \stackrel{?}{\preceq} y$ in \mathcal{P} . Equivalently, it is to determine the presence of the edge (x, y) in G , or reachability of vertex y from x in G_r .

succ _{G} (x), **pred** _{G} (x): this is to report all successors (predecessors, respectively) of x : *i.e.* $\text{succ}_G(x) = \{y | x \preceq y\}$, and $\text{pred}_G(x) = \{y | y \preceq x\}$.

Table 1. Supported operations together with their running times, where k is the size of the output of the corresponding operation

Operation	Running time
<code>reachability</code> (x, y)	$O(1)$
<code>succ</code> _{G} (x), <code>pred</code> _{G} (x)	$O(w + k)$
<code>range</code> (x, y)	$O(w + k)$
<code>adj</code> _{G_r} (x, y)	$O(w)$
<code>succ</code> _{G_r} (x), <code>pred</code> _{G_r} (x)	$O(w^2)$
<code>a-succ</code> _{G_r} (x), <code>a-pred</code> _{G_r} (x)	$O(w)$
<code>LUB</code> (x_1, \dots, x_t), <code>GLB</code> (x_1, \dots, x_t)	$O(w^2 + t)$
<code>a-LUB</code> (x_1, \dots, x_t), <code>a-GLB</code> (x_1, \dots, x_t)	$O(w \min(t, w) + t)$

`range`(x, y): this is to report all elements that are successors of x and predecessors of y .

`adj` _{G_r} (x, y): this operation is to determine whether edge (x, y) is an edge in the transitive reduction G_r .

`succ` _{G_r} (x), `pred` _{G_r} (x): these operations are defined on the transitive reduction G_r , and are to report the out-neighbors (in-neighbors, respectively) of x : i.e. to report all elements y such that (x, y) is an edge in G_r ((y, x) is an edge, respectively).

`a-succ` _{G_r} (x), `a-pred` _{G_r} (x): this is to report an *arbitrary* out-neighbor (in-neighbor, respectively) of x in G_r .

`LUB`(x_1, \dots, x_t), `GLB`(x_1, \dots, x_t): this is to report the least upper bound (LUB), and the greatest lower bound (GLB) of x_1, \dots, x_t , respectively. The set of LUBs consists of all elements z that are successors of all x_1, \dots, x_t , but there is no $z' \prec z$ which is also a successor of all x_1, \dots, x_t . The set of GLBs is defined symmetrically.

`a-LUB`(x_1, \dots, x_t), `a-GLB`(x_1, \dots, x_t): this is to report a *representative* LUB (or GLB, respectively).

The time for performing these operations by our representation is shown in Tbl. 1. Our set operations is similar to that of [9] which are defined over posets that are lattices. The only operation from [9] we do not support efficiently is a *path-query* (list an arbitrary path from x to y in G_r), although we could easily find such a path in $O(w^2 \times k)$ time for a path of length k by repeated applications of `succ` _{G_r} (\cdot) and `reachability`(\cdot, y).

2 Preliminaries

2.1 Posets

We consider n -element (labeled) posets (\mathcal{P}, \preceq) of fixed width w (the width is defined as the size of the largest antichain in the poset). Because the elements in \mathcal{P} are labeled, we can assume that $\mathcal{P} = \{1, 2, \dots, n\}$. Let $N_w(n)$ denote the number of such posets. Brightwell and Goodall [2] showed the following:

Lemma 1. *For fixed n , there are positive constants $\alpha, \beta, A(w)$, and $B(w)$ such that*

$$n!4^{n(w-1)}A(w)n^{-\alpha w^2} \leq N_w(n) \leq n!4^{n(w-1)}B(w)n^{-\beta w^2}.$$

This implies that an n -element poset \mathcal{P} of width w can be encoded in $\lg(N_w(n)) = n \lg n + 2n(w-1) - \Theta(w^2 \lg n) + O(\lg n)$ bits, and that this space is asymptotically optimal. In this paper we focus on the space efficient representation of posets of small width, which are likely to occur in many applications.

A *chain* $C \subseteq \mathcal{P}$ in a poset \mathcal{P} is a set $C = \{v_1, \dots, v_x\}$ of mutually comparable elements (for any $v_i, v_j \in C$, $v_i \preceq v_j$ or $v_j \preceq v_i$). A *chain decomposition* of \mathcal{P} is a set of disjoint chains $\{C_1, \dots, C_q\}$ such that $\bigcup_i C_i = \mathcal{P}$. The size of a chain decomposition is the number of chains in it. Dilworth's Theorem states that there is a decomposition of \mathcal{P} of size w , which is clearly optimal.

We shall focus on chains that are *sorted*, $C = (v_1, \dots, v_x)$ with $v_1 \preceq v_2 \preceq \dots \preceq v_x$. For obtaining sorted chains, we have the following result:

Lemma 2 (Thm. 3.8 in [3]). *Given a poset \mathcal{P} of width w , \mathcal{P} can be decomposed into w sorted chains C_1, \dots, C_w in $O(w^2 n \log(n/w))$ time.*

From now on, we implicitly assume that the chains are sorted.

2.2 Transitive Reductions

The *transitive reduction* G_r of a directed graph G is defined as follows [1]:

1. there is a directed path from vertex u to vertex v in G_r if and only if there is a directed path from u to v in G , and
2. there is no graph with fewer arcs than G_r satisfying condition (1).

For DAGs, this coincides with the notion of *minimal equivalent digraphs* [6]. The output of the transitive reduction of a DAG can be considered as a poset, as it is the compact representation of the transitive closure of a DAG, which in turn yields a partial order. It is well-known that transitive reductions of DAGs are unique.

Unfortunately, computing the transitive reduction is as expensive as computing the transitive closure, which, in turn, is dominated by the time for matrix multiplication ($O(n^{2.376})$ using the Coppersmith-Winograd algorithm).

2.3 Succinct Data Structures

Consider a *bit-string* $S[1, n]$ of length n . We define the fundamental *rank-* and *select-*operations on S as follows: $\text{rank}_1(S, i)$ gives the number of 1's in the prefix $S[1, i]$, and $\text{select}_1(S, i)$ gives the position of the i 'th 1 in S , reading S from left to right ($1 \leq i \leq n$). Operations $\text{rank}_0(S, i)$ and $\text{select}_0(S, i)$ are defined similarly for 0-bits. The following lemma summarizes a by-now classic result:

Lemma 3 (see, e.g., [8]). *A bit-string of length n can be represented in $n + o(n)$ bits such that rank- and select-operations are supported in $O(1)$ time.*

We also need a tool for encoding permutations succinctly. For this, we have:

Lemma 4 (Munro et al. [7]). *A permutation π of $\{1, \dots, n\}$ can be stored in $(1 + \epsilon)n \lg n + o(n)$ bits for an arbitrarily small constant $\epsilon > 0$ such that the application of π and its inverse π^{-1} are both supported in $O(1/\epsilon)$ time.*

3 The Representation

We now describe our succinct data structure for posets. Our general approach is to first store the elements in the order resulting from the chain decomposition (Sect. 3.1), and to encode the “interaction” between the chains separately (Sect. 3.2).

3.1 Storing the Chains

Assume we are given a poset \mathcal{P} of width w , decomposed into w sorted chains C_1, \dots, C_w by Lemma 2. For any element $u \in \mathcal{P}$, let $c(u)$ be defined such that u appears on chain $C_{c(u)}$ (hence $c(u)$ gives the *chain number* of $u \in \mathcal{P}$). Further, let $d(u)$ denote u ’s depth on that chain, $d(u) = |\{v \in C_{c(u)} \mid v \prec u\}|$. Our internal representation of the elements $u \in \mathcal{P}$ will consist of $(c(u), d(u))$ -tuples. In order to be able to “translate” back and forth between the external names $1, \dots, n$ and our (chain,depth)-tuples, we build the following data structure: in an array π of length n , first list all elements on chain C_1 in increasing depth, then those elements on C_2 , and so on, until reaching chain C_w . (Hence π is a permutation of $\{1, \dots, n\}$ corresponding to the lexicographic (chain,depth)-order.)

Further, in a bit-vector B of length n we mark the first element in π of any chain with a ‘1’; all other elements are ‘0’. Bit-vector B is prepared for constant-time rank- and select-queries. With these definitions, given $u \in \mathcal{P}$, $c(u) = \text{rank}_1(B, \pi^{-1}(u))$ (π^{-1} denotes the inverse permutation), and $d(u) = \pi^{-1}(u) - \text{select}_1(B, c(u))$. Conversely, given a tuple $(c(u), d(u))$, we can compute u as $u = \pi(\text{select}_1(B, c(u)) + d(u))$. Thus, we can translate back and forth from an original vertex label u to our labeling, which is tuple $(c(u), d(u))$.

We store the permutation π using Lemma 4 which uses $(1 + \epsilon)n \lg n + o(n)$ bits of space and permits the retrieval of $\pi(\cdot)$ and $\pi^{-1}(\cdot)$ in $O(1)$ time (here and in the following, we drop the dependencies of all running times on the constant ϵ). Vector B is stored using Lemma 3 it requires $n + o(n)$ bits such that both rank and select can be computed in $O(1)$ time.

3.2 Storing the Order Relation between Chains

We maintain a compact data structure $\mathcal{D}(C_i, C_j)$ for any pair of chains C_i and C_j that essentially stores the order between the elements of the two chains.

For every element $u \in C_i$ and every chain C_j , let $d(u, j)$ be defined as the “depth” on C_j of the smallest element larger than u . More formally: $d(u, j) = \min\{d(v) \mid v \in C_j \text{ and } u \preceq v\}$, and $d(u, j) = |C_j|$ if no element on C_j is larger than u . Note that the $d(\cdot, \cdot)$ -function generalizes the $d(\cdot)$ -function, in the sense that $d(u, c(u)) = d(u)$.

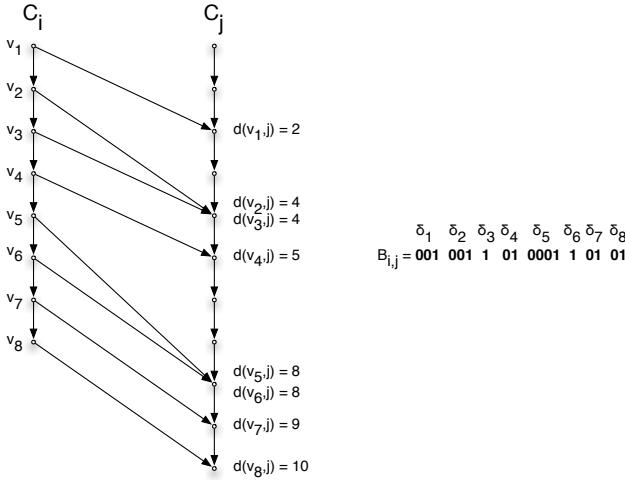


Fig. 1. Two chains and the corresponding “depths” of the elements of C_i on C_j . $B_{i,j}$ is the bit vector corresponding to chain i with respect to j

For storing the $d(\cdot, \cdot)$ -values in a space-efficient way, observe that $d(v_1, j) \leq d(v_2, j) \leq \dots \leq d(v_x, j)$ for a chain $C_i = (v_1, \dots, v_x)$ and any j . Hence, we can encode all $d(\cdot, j)$ -values for a chain C_i in a single bit-vector $B_{i,j}$, by letting $\delta_k^j = d(v_k, j) - d(v_{k-1}, j)$ denote the difference between successive $d(\cdot, j)$ -values (set $d(v_0, j) = 0$ for handling the border case), and concatenating the unary representations of the δ -values (see Fig. 1 for an example):

$$B_{i,j} = 0^{\delta_1^j} 1 0^{\delta_2^j} 1 \dots 0^{\delta_x^j} 1 .$$

We store this bit vector using Lemma 3 to be able to perform rank and select operations in constant time; the resulting data structure is called $\mathcal{D}(C_i, C_j)$.

Then a $d(\cdot, \cdot)$ -value can be retrieved as

$$d(u, j) = \text{select}_1(B_{c(u),j}, d(u) + 1) - d(u) - 1 ,$$

as the select-statement brings us to the position of the $d(u) + 1$ ’st ‘1’ in $B_{c(u),j}$; subtracting from this position the number of ‘1’s up to this point (which is precisely $d(u)$) gives the number of ‘0’s up to this point, hence the depth on chain C_j . (The ‘+1’ and ‘-1’ account for the fact that the depth-values start at 0, whereas rank and select start “counting” at one.)

Conversely, for every element v and every chain C_i with $i \neq c(v)$, let $d^{-1}(v, i)$ denote the depth on C_i of the largest element smaller than v (d^{-1} is the “inverse” of d). More formally: $d^{-1}(v, i) = \max\{d(u) \mid u \in C_i \text{ and } u \preceq v\}$, and $d^{-1}(v, i) = -1$ if no element on C_i is smaller than v . This operation can be performed, in constant time, by computing

$$d^{-1}(v, i) = \text{rank}_1(B_{i,c(v)}, \text{select}_0(B_{i,c(v)}, d(v) + 1)) - 1 .$$

Hence, we have the following:

Lemma 5. *The data structure $\mathcal{D}(C_i, C_j)$ stored for two chains C_i, C_j requires $|C_i| + |C_j| + o(|C_i| + |C_j|)$ bits and supports the application of d and d^{-1} in constant time.*

Our representation consists of storing structure $\mathcal{D}(C_i, C_j)$ for all pairs of chains. The total size of structures $\mathcal{D}(C_i, C_j)$ can be easily computed as $2n(w - 1)(1 + o(1))$ bits, as any element $v \in C_j$ contributes at most one '1' and one '0' to all of the $w - 1$ bit-vectors $B_{i,j}$ for $i \neq j$. (In order to make the total redundancies add up to $o(nw)$ bits, we concatenate all bit vectors $B_{i,j}$, and build a *global* rank/select-structure over the entire bit stream.) Further, because the structures of Sect. 3.1 allow us to translate between (chain,depth)-tuples and the elements in \mathcal{P} , we can summarize this section as follows:

Lemma 6. *A poset \mathcal{P} with n elements and width w can be represented in $(1 + \epsilon)n \lg n + 2nw(1 + o(1))$ bits for an arbitrary small constant $\epsilon > 0$. The representation supports, in constant time, the following operations: given a node v and a chain C_j , (1) $c(v)$: report the chain number of v , (2) $d(v)$: report the depth of v on its chain, (3) $d(v, j)$: report the smallest element $u \in C_j$ with $v \preceq u$, and (4) $d^{-1}(v, j)$: report the largest element $u \in C_j$ with $u \preceq v$.*

4 Operations

In this section, we show how various operations are efficiently supported by our representation. The running times of the operations depend only on the width w and the size k of the output, but *not* on the size n of the underlying poset.

4.1 Operation reachability(x, y)

The representation of Lemma 6 is well-suited to the efficient support of the reachability query: we first check if $c(u) = c(v)$, and if so, we return TRUE if $d(u) \leq d(v)$, and FALSE otherwise. If, on the other hand, $c(u) \neq c(v)$, we return TRUE if $d(u, c(v)) \leq d(v)$, and FALSE otherwise. Lemma 6 readily provides support for these steps in constant time.

4.2 Operations $\text{succ}_G(x)$ and $\text{pred}_G(x)$

We explain only the implementation of $\text{succ}_G(x)$, as the implementation of $\text{pred}_G(x)$ is symmetrical using d^{-1} instead of d . Operation $\text{succ}_G(x)$ is supported in $O(w + k)$ time, where k is the number of elements reported. We first determine the chain C_i where x belongs to ($i = c(x)$). Then we consider all w chains in turn. For each such chain C_j , we retrieve the value $d(x, j)$ from data structure $\mathcal{D}(C_i, C_j)$. It only remains to report all elements on chain C_j whose depth is at least $d(x, j)$. Because the chains are sorted, this last step takes $O(k)$ overall time.

4.3 Operation $\text{range}(x, y)$

We first determine the chains where x and y belong to (say $x \in C_i$, and $y \in C_j$). Next, we consider all chains C_k in order. Using structure $\mathcal{D}(C_i, C_k)$, we retrieve the value $d(x, k)$, and analogously, using structure $\mathcal{D}(C_k, C_j)$, we retrieve the value $d^{-1}(y, k)$.

If $d(x, k) \leq d^{-1}(y, k)$, we report all elements on chain C_k whose depth is in between the two; otherwise, there is nothing to report and we proceed with the next chain. Clearly, the operation is performed in overall $O(w + k)$ time, where k is the number of elements reported.

4.4 Operation $\text{adj}_{G_r}(x, y)$

We first check whether y is reachable from x as otherwise, edge (x, y) does not exist in G_r . However, this check, although necessary, is not sufficient, as the edge could be absent in G_r because it is implied by transitivity from other edges.

As usual, we first determine the chain C_i where x belongs to. For any chain C_j with $j \neq i$, using data structure $\mathcal{D}(C_i, C_j)$, we retrieve the smallest element v_j that is reachable from x (hence $d(v_j) = d(x, j)$). We further define v_i as the element immediately following x on C_i .

In order for (x, y) to be an edge in G_r , y must be one of v_j 's. Furthermore, it must not be reachable from any other v_j . We can easily check the former in constant time per chain, and the latter using w applications of $\text{reachability}(v_j, y)$ for all j 's in overall $O(w)$ time.

4.5 Operations $\text{succ}_{G_r}(x)$ and $\text{pred}_{G_r}(x)$

We only give the description of $\text{succ}_{G_r}(x)$, as $\text{pred}_{G_r}(x)$ is symmetrical. As in operation $\text{adj}_{G_r}(\dots)$, we determine the chain C_i where x belongs to. For any chain C_j , $j \neq i$, using data structure $\mathcal{D}(C_i, C_j)$, we retrieve the smallest element v_j that is reachable from x . We further define v_i as the element immediately following x . Clearly, the v_j 's are the only candidates for being out-neighbors of x . Hence, by checking $\text{adj}_{G_r}(x, v_j)$ for all j , in total time $O(w^2)$ we can report all successors of x .

4.6 Operations $\text{a-succ}_{G_r}(x)$ and $\text{a-pred}_{G_r}(x)$

If we want to report just *one* successor or predecessor of x , we can be faster than operations $\text{succ}_{G_r}(x)$ or $\text{pred}_{G_r}(x)$, respectively. We only give the implementation of $\text{a-succ}_{G_r}(x)$, as that of $\text{a-pred}_{G_r}(y)$ is symmetrical. As in previous operations, we first determine the chain C_i of x . For any chain C_j with $j \neq i$, using data structure $\mathcal{D}(C_i, C_j)$, we retrieve the smallest element v_j that is reachable from x . We also define v_i as the element immediately following x .

As before, the only candidates for the answer to $\text{a-succ}_{G_r}(x)$ are the v_j 's. It remains to find one *direct* successor of x . Observe that such a v_{j^*} , which is an

out-neighbor of x in G_r , must have the property that it is not reachable from any other v_j ; in other words, we must have $d(v_j, j^*) > d(v_{j^*})$ for all $j \neq j^*$.

To find j^* , we start with v_1 on C_1 as our candidate for v_{j^*} . We then visit all remaining chains $k = 2, \dots, w$ in order. When entering step k , let v_j ($j < k$) be the current candidate (hence $j = 1$ in the beginning). Now we check if $d(v_k, j) > d(v_j)$, and if so, then v_j remains a candidate for being a direct successor, and we proceed with checking the next chain. Otherwise ($d(v_k, j) \leq d(v_j)$ and hence $d(v_k, j) = d(v_j)$, as $d(v_k, j) < d(v_j)$ would imply that v_j is not the smallest element on C_j reachable from x), we take v_k as the new candidate (set $j \leftarrow k$), and proceed with the next chain as before.

This way, each chain k is visited exactly once to check if $d(v_k, j) > d(v_j)$ for the current candidate v_j . After all chains have been checked, the remaining candidate is returned as the answer to $\text{a-succ}_{G_r}(x)$. Note in particular that even when a new candidate v_k replaces an old candidate v_j , we do not have to check again the chains $\ell < k$ already checked, as $d(v_\ell, j) > d(v_j)$ implies $d(v_\ell, k) > d(v_k)$ for every processed chain C_ℓ (for otherwise $d(v_k, j) > d(v_j)$). Hence, the overall running time is $O(w)$.

4.7 Operations $\text{LUB}(x_1, \dots, x_t)$ and $\text{GLB}(x_1, \dots, x_t)$

We only describe the implementation of $\text{LUB}(x_1, \dots, x_t)$, as the implementation of $\text{GLB}(x_1, \dots, x_t)$ is symmetrical. To begin with, if $t > w$, we only retain the deepest x_i on any chain; this initial filtering costs $O(t)$ time and leaves us with at most w of the x_i 's.

First note that on each chain there can be at most *one* element that is possibly a LUB. To find these “candidates,” for every chain C_j we determine the depth of the smallest element larger than any of the x_i 's by computing $d_j = \max_{1 \leq i \leq t} \{d(x_i, j)\}$ and letting ℓ_j be the corresponding candidate (ℓ_j is the d_j 'th element on C_j). This step takes overall $O(w \min(w, t))$ time.

Now observe that ℓ_j is a true LUB iff no other $\ell_{j'}, j' \neq j$, is a predecessor of ℓ_j . Hence, we need to check if $d(\ell_{j'}, j) > d_j$ for all $j' \neq j$, and report ℓ_j if it passes all such tests. This pairwise check takes $O(w^2)$ overall time. With the initial filtering, the total running time is therefore $O(w^2 + t)$.

4.8 Operations $\text{a-LUB}(x_1, \dots, x_t)$ and $\text{a-GLB}(x_1, \dots, x_t)$

Again, we can be faster if we only want to report a representative LUB or GLB. We only describe the implementation of $\text{a-LUB}(x_1, \dots, x_t)$, as the implementation of $\text{a-GLB}(x_1, \dots, x_t)$ is symmetrical.

We first determine the ℓ_j 's and their depths d_j as in operation $\text{LUB}(x_1, \dots, x_t)$. These are again the only candidates for being a LUB. We then start with ℓ_1 as a candidate, and walk through all remaining chains $k = 2, \dots, w$ in order, as in operation a-succ_{G_r} : when entering step k , let ℓ_j ($j < k$) be the current candidate (hence $j = 1$ in the beginning). Now we check if $d(\ell_k, j) > d(\ell_j)$, and if so, then ℓ_j remains a candidate for being a direct successor, and we proceed with checking the next chain. Otherwise, we take ℓ_k as the new candidate (set $j \leftarrow k$), and

proceed. By the same reasoning as in Sect. 4.6, when a new candidate ℓ_k replaces an old candidate ℓ_j it is not necessary to repeat the check $d(\ell_{j'}, k) > d(\ell_k)$ for the chains $j' \neq j$ that have already been visited, as the opposite would imply $d(\ell_{j'}, k) \leq d(j)$, a contradiction to the choice of j . Hence, the overall running time is $O(w)$.

5 Summary and Open Questions

We opened the problem of representing posets and transitive reductions of DAGs succinctly. Our main result is that we can represent an n -element poset or DAG of width w in $(1 + \epsilon)n \lg n + 2nw(1 + o(1))$ bits of space for an arbitrary small constant $\epsilon > 0$, such that the queries of Tbl. II are supported in times mentioned in the same table. This is the first succinct data structure for DAGs, using less space than storing arbitrary graphs whenever $w = o(n)$.

Our algorithms and data structures scale with the width w of the underlying poset or DAG; in fact, our succinct representation has essentially optimal size (in the worst case over all posets of width w). We showed that our representation supports the most common queries in times that only depend on w and the size of the output, but not on the number of elements. In the future, it would be interesting to lower the running times of the more expensive operations, and to enhance the set of supported queries, for example finding arbitrary or shortest paths between two given nodes, distance oracles, etc.

References

1. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. *SIAM J. Comput.* 1(2), 131–137 (1972)
2. Brightwell, G., Goodall, S.: The number of partial orders of fixed width. *Order* 13(4), 315–337 (1996)
3. Daskalakis, C., Karp, R.M., Mossel, E., Riesenfeld, S., Verbin, E.: Sorting and selection in posets. *SIAM J. Comput.* 40(3), 597–622 (2011)
4. Dhar, D.: Entropy and phase transitions in partially ordered sets. *J. Math. Phys.* 19(8), 1711–1713 (1978)
5. Farzan, A., Munro, J.I.J.: Succinct Representation of Arbitrary Graphs. In: Halperin, D., Mehlhorn, K. (eds.) *Esa 2008. LNCS*, vol. 5193, pp. 393–404. Springer, Heidelberg (2008)
6. Moyles, D.M., Thompson, G.L.: An algorithm for finding a minimum equivalent graph of a digraph. *J. ACM* 16(3), 455–460 (1969)
7. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct Representations of Permutations. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) *ICALP 2003. LNCS*, vol. 2719, pp. 345–356. Springer, Heidelberg (2003)
8. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.* 31(3), 762–776 (2001)
9. Talamo, M., Vocca, P.: An efficient data structure for lattice operations. *SIAM J. Comput.* 28(5), 1783–1805 (1999)
10. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM* 51(6), 993–1024 (2004)

Explicit Array-Based Compact Data Structures for Triangulations*

Luca Castelli Aleardi¹ and Olivier Devillers²

¹ LIX (Ecole Polytechnique, France)

amturing@lix.polytechnique.fr

² INRIA Sophia Antipolis – Méditerranée , France

olivier.devillers@inria.fr

Abstract. We consider the problem of designing space efficient solutions for representing triangle meshes. Our main result is a new explicit data structure for compactly representing planar triangulations: if one is allowed to permute input vertices, then a triangulation with n vertices requires at most $4n$ references ($5n$ references if vertex permutations are not allowed). Our solution combines existing techniques from mesh encoding with a novel use of minimal Schnyder woods. Our approach extends to higher genus triangulations and could be applied to other families of meshes (such as quadrangular or polygonal meshes). As far as we know, our solution provides the most parsimonious data structures for triangulations, allowing constant time navigation in the worst case. Our data structures require linear construction time, and all space bounds hold in the worst case. We have implemented and tested our results, and experiments confirm the practical interest of compact data structures.

Keywords: triangulations, compact representations, mesh data structures, graph encoding, Schnyder woods.

1 Introduction

The large diffusion of geometric meshes (in application domains such as geometry modeling, computer graphics), and especially their increasing complexity has motivated a huge number of recent works in the domain of graph encoding and mesh compression. In particular, the *connectivity* information of a mesh (describing the incidence relations) represents the most expensive part (compared to the geometry information): for this reason most works try to reduce the first kind of information, involving the combinatorial structure of the underlying graph. Many works addressed the problem from the *compression* [24][25] point of view: compression schemes aim to reduce the number of bits as much as possible, possibly close to theoretical minimum bound according to information theory. For applications requiring the manipulation of input data, a number of explicit (pointer-based) data structures [7][3][4][18] have been developed for many classes of surface and volume meshes. Most geometric algorithms require

* This work is supported by ERC (agreement ERC StG 208471 - ExploreMap).

data structures which are easy to implement, allowing fast navigation between mesh elements (edges, faces and vertices), as well as efficient update primitives. Not surprisingly common mesh representations are redundant and store a huge amount of information in order to achieve the prescribed requirements. In this work we address the problem above (reducing memory requirements) from the point of view of *compact data structures*: the goal is to reduce the redundancy of common explicit representations, while still supporting efficient navigation.

1.1 Existing Mesh Data Structures

Classical data structures in most programming environments do admit *explicit pointer-based* implementations. Each pointer stores at most one reference: pointers allow to navigate in the data structure through address indirection, and storing/manipulating service bits within references is not allowed. Many popular geometric data structures (such as *Quad-edge*, *Winged-edge*, *Half-edge*) fit in this framework. In edge-based representations basic elements are edges (or half-edges): navigation is performed storing, for each edge, a number of references to incident mesh elements. For example, in the *Half-edge DS* each half-edge stores a reference to the next and opposite half-edge, together with a reference to an incident vertex (which gives 3 references for each of the $6n$ half-edges, for a triangulation having n vertices).

Compact practical solutions. Several works [9, 27, 22, 11, 10, 20, 19, 21] try to reduce the number of references stored by common mesh representations: this leads to more compact solutions, whose performances (in terms of running time) are still really of practical interest. In this case array-based implementations are sometimes preferred to pointer-based representations, depending on the flexibility of the programming environment. Many data structures (*triangle-based*, *array-based compact half-edge*, *SOT/SQUAD data structures*) make use of a slightly stronger assumption: each memory word can store a $\lg n$ bits integer reference¹, and C bits are reserved as *service bits* (C is a small constant, commonly between 1 and 4). Moreover, basic arithmetic operations are allowed on references: such as addition, multiplication, floored division, and bit shifts/masks. An interesting general approach is based on the reordering of mesh elements: for example, storing consecutively the half-edges of a face allows to save 3 references per face (as in *Directed Edge* [9], which requires $13n$ references instead of the $19n$ stored by *Half-edge*). Or still, storing edges/faces according to the vertex ordering allows to implicitly represent the map from edges/faces to vertices. This is one of the ingredients used by the *SOT* data structure [20], which represents triangulations with $6n$ references. Adopting an interesting heuristic one may even obtain a more compact solution [19], requiring about $(4 + c)$ references per vertex: as shown by experiments c is a small value (between 0.09 and 0.3 for tested meshes), but there are no theoretical guarantees in the worst case.

¹ For a mesh with n elements, $\lg n := \lceil \log_2 n \rceil$ bits are required to distinguish all the elements. The length w of each memory word is assumed to be $\Omega(\lg n)$

Theoretically optimal solutions. For completeness, we mention that *succinct representations* [15, 6, 23, 12, 11, 28] are successful in representing meshes with the minimum amount of bits, while supporting local navigation in worst case $O(1)$ time. They run under the *word-Ram model*, where basic arithmetic and bitwise operations on words of size $O(\lg n)$ are performed in $O(1)$ time. One main idea (underlying almost all solutions) is to reduce the size, and not only the number, of references: one may use graph separators or hierarchical graph decomposition techniques in order to store in a memory word an arbitrary (small) number of tiny references. Typically, one may stores up to $O(\frac{\lg n}{\lg \lg n})$ sub-words of length $O(\lg \lg n)$ each. Unfortunately, the amount of auxiliary bits needed by the encoding becomes asymptotically negligible only for very huge graphs, which makes succinct representations of mainly theoretical interest.

Finally, we observe that the parsimonious use of references may affect the navigation time: for example, the access to some mesh elements requires more than $O(1)$ time in the worst case [20, 19, 22] (as reported in Table 1).

Table 1. Comparison between existing data structures for triangle meshes. All bounds hold in the worst case, at the exception of SQUAD data structure, whose performances are interesting in practice for common meshes, but with no theoretical guarantees.

Data structure	references	navigation	vertex access	dynamic
Edge-based data structures [18, 3, 4] triangle based [4]/Corner Table	$18n + n$ $12n + n$	$O(1)$ $O(1)$	$O(1)$ $O(1)$	<i>yes</i> <i>yes</i>
Directed edge [9]	$12n + n$	$O(1)$	$O(1)$	<i>yes</i>
2D catalogs [10]	$7.67n$	$O(1)$	$O(1)$	<i>yes</i>
Star vertices [22]	$7n$	$O(d)$	$O(1)$	<i>no</i>
TRIPOD [27] + reordering / Thm 1	$6n$	$O(1)$	$O(d)$	<i>no</i>
SOT data structure [20]	$6n$	$O(1)$	$O(d)$	<i>no</i>
SQUAD data structure [19]	$(4 + c)n$	$O(1)$	$O(d)$	<i>no</i>
(no vertex reordering) Thm 2	$5n$	$O(1)$	$O(d)$	<i>no</i>
(with vertex reordering) Thm 3	$4n$	$O(1)$	$O(d)$	<i>no</i>
(with vertex reordering) Cor 3	$6n$	$O(1)$	$O(1)$	<i>no</i>

1.2 Preliminaries

Combinatorial aspects of triangulations. In this work we exploit a deep and strong combinatorial characterization of planar triangulations. A planar triangulation is a simple planar map where every face (including the infinite face) has degree 3. Triangulations are *rooted* if there is one distinguished *root face*, denoted by (v_0, v_1, v_2) , with a distinguished incident *root edge* $\{v_0, v_1\}$. *Inner edges* (and *inner vertices*) are those not belonging to the root face (v_0, v_1, v_2) ².

² We will denote by the ordered pair (u, v) an edge oriented toward v , while $\{u, v\}$ will denote an edge regardless of its direction. In our drawings the root face coincide with the infinite exterior face.

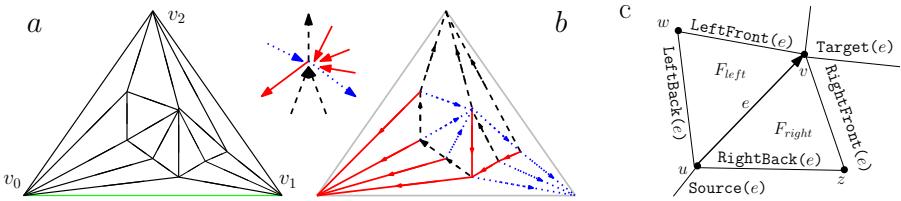


Fig. 1. A planar triangulation with 9 vertices (a), endowed with a (minimal) Schnyder wood (b) (the local Schnyder condition around inner vertices is also shown). Picture (c) illustrates the navigational operations supported by our representations.

As pointed out by Schnyder [26], the inner edges of a planar triangulation can be partitioned into three sets T_0 , T_1 , T_2 , which are plane trees spanning all inner nodes, and rooted at v_0 , v_1 and v_2 respectively. This spanning condition can be derived from a local condition: the inner edges can be oriented in such a way that every inner node is incident to exactly 3 outgoing edges, and the orientation/coloration of edges must satisfy a special local rule (see Fig. II).

Definition 1 ([26]). *Let \mathcal{G} be a planar triangulation with root face (v_0, v_1, v_2) . A **Schnyder wood** of \mathcal{G} is an orientation and labeling, with label in $\{0, 1, 2\}$ of the inner edges such that the edges incident to the vertices v_0 , v_1 , v_2 are all ingoing and are respectively of color 0, 1, and 2. Moreover, each inner vertex v has exactly three outgoing incident edges, one for each color, and the edges incident to v in counter clockwise (ccw) order are: one outgoing edge colored 0, zero or more incoming edges colored 2, one outgoing edge colored 1, zero or more incoming edges colored 0, one outgoing edge colored 2, and zero or more incoming edges colored 1 (this is referred to as **local Schnyder condition**).*

Navigational operators. Here are the operators supported by our representations. Let $e = (u, v)$ be an edge oriented toward v , which is incident to (u, v, w) (its left triangle) and to (u, v, z) (its right triangle), as depicted in Fig. II(c).

- $\text{LeftBack}(e)$, returns the edge $\{u, w\}$.
- $\text{LeftFront}(e)$, returns the edge $\{v, w\}$.
- $\text{RightBack}(e)$, returns the edge $\{u, z\}$.
- $\text{RightFront}(e)$, returns the edge $\{v, z\}$.
- $\text{Source}(e)$ (resp. $\text{Target}(e)$), returns the origin (resp. destination) of e ;
- $\text{Edge}(u)$, returns an edge incident to vertex u ;
- $\text{Point}(u)$, returns the geometric coordinates of vertex u .

The operators above are supported by most mesh representations [93], and allow full navigation in the mesh as required in geometric processing algorithms: their combination allows to iterate on the edges incident to a given node, or to walk around the edges incident to a given face.

Overview of our solution. In order to design new compact array-based data structures, we make use of many ingredients: some of them concerning the

combinatorics of graphs, and some of them pertaining the design of compact (explicit) data structures. The main steps of our approach are the following:

- as done in [9,20,19], we perform a reordering of cells (edges), to implicitly represent the map from vertices to edges, and the map from edges to vertices;
- as done in [27], we exploit the existence of 3-orientations (edges orientations where every inner vertex has outgoing degree 3) for planar triangulations [26]. This allows to store only two references per edge.

Combining these two ideas one can easily obtain an array-based representation using $6n$ references, allowing $O(1)$ time navigation between edges and $O(d)$ time for the access to a vertex of degree d : for the sake of completeness, this simple solution will be detailed in Theorem 1. Our main contribution is to show how to get further improvements and generalizations³, using the following ideas:

- we exploit the existence of *minimal Schnyder woods*, without cycles of directed edges oriented in ccw direction. And also the fact that, given the partition (T_0, T_1, T_2) , the two trees T_0 and T_1 , are sufficient to retrieve the triangulation. With these ideas we to store only $5n$ references (Theorem 2);
- we further push the limits of the previous reordering approach: by arranging the input points according to a given permutation and using a special kind of order on plane trees (the so-called *DFUDS* order [5]), we are able to use only $4n$ references (Theorem 3);
- in the full version of this paper, we also show how to reformulate a recent generalization of Schnyder woods [14], in order to deal with genus g triangulations: our representation requires at most $5(n + 4g)$ references;

To our knowledge, these are the best (worst case and guaranteed) upper bounds obtained so far, which improve previous existing results.

2 Compactly Representing Triangulations

2.1 The First Data Structure: Simple and Still Redundant

We first design a simple data structure requiring $6n$ references, which allows to perform all navigational operators in worst case $O(1)$ time, and **Target** operator in $O(d)$ time (when retrieving a degree d vertex). This is a preliminary step in describing a more compact solution. Observe that our first scheme achieves the same space bounds as the *Tripod data structure* by Snoeyink and Speckmann [27]. Although both solutions are based on the properties of Schnyder woods, the use of references (between edges) is different: this is one of the features which allow to make our scheme even more compact in the sequel.

³ Detailed proofs of the results presented here can be found in [13].

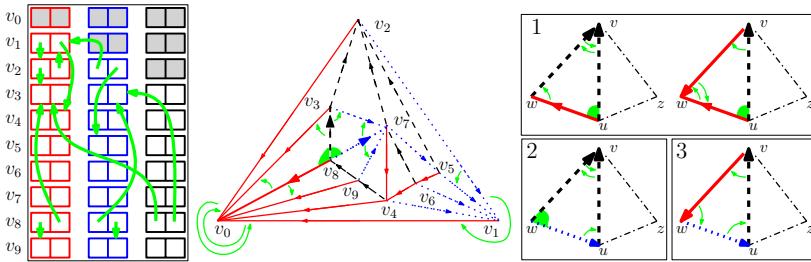


Fig. 2. Our first solution. For each vertex we store 6 references, corresponding to the its 3 outgoing edges. Table T is drawn as a bi-dimensional array of size $n \times 6$. The case analysis of Theorem 1 is illustrated on the right (where edge (u, v) has color 2).

Scheme description. We firstly compute a Schnyder wood of the input triangulation \mathcal{G} (in linear time, as shown in [26]). We define the tree $\overline{\mathcal{T}}_0$ by adding edges (v_1, v_0) and (v_2, v_0) to \mathcal{T}_0 ; we add the edge (v_2, v_1) to the tree \mathcal{T}_1 , as depicted in Fig. 2. Each edge gets a color and an orientation: edges in $\overline{\mathcal{T}}_0$ are called red edges, those in \mathcal{T}_1 blue edges and those in \mathcal{T}_2 black edges. For each vertex, we store 6 integers representing the 3 incident outgoing edges.

Vertices will be identified by integers $0 \leq i < n$ and edges by integers $3 \leq j < 3n$ (some indices between 0 and 8 are omitted, as it will be shown in the sequel). Our data structure consists of an array T of size $6n$, two arrays of bits S_a, S_b of size $3n$, and an array P of size n storing the geometric coordinates of the points. The entries of T and P are sorted according to the order of input points, which facilitates the implementation of the **Point** operator. By convention, the three edges having vertex i as source are indexed $3i, 3i + 1$ and $3i + 2$, where edge having index $3i + c$ has color c . For each oriented edge we store two references to 2 neighboring edges. References are arranged in table T , in such a way that for each inner node u of \mathcal{G} , the outgoing edges associated with u are stored consecutively in T (refer to Fig. 2). The adjacency relations of an inner edge j are stored in entries $2j$ and $2j + 1$ of table T , as follows:

- $T[2j] = \text{LeftFront}(j)$, and $T[2j + 1] = \text{RightFront}(j)$;

Arrays S_a and S_b have an entry for each edge and are defined as follows:

- $S_a[j] = 1$ if edge j and $\text{LeftBack}(j)$ have the same source, 0
- $S_b[j] = 1$ if edge j and $\text{RightBack}(j)$ have the same source, 0.

Edges belonging to (v_0, v_1, v_2) are stored in a similar manner⁴: some edge indices are not used, since vertices on the outer face do not have outdegree 3 .

References encoding. As T is an array of integers $< 3n$ and S_a, S_b are arrays of bits, in practice these three arrays can be stored in a single array. Just encode the service bits within the references stored in T , where first k bits of an integer

⁴ For the sake of simplicity, in our examples the first 3 vertices belong to the root face. But there is no such a restriction in our implementations.

represent the index of an edge. Since we have at most $3n$ edges, we can set $k = \lceil \log 3n \rceil$. In this section we use only 2 service bits per edge: having 2 references per edge, we just store 1 service bit in each reference. Assuming 32 bits integers, we can encode triangulations having up to 2^{31} edges.

Theorem 1. *Let \mathcal{G} be a triangulation with n vertices. The representation described above requires $6n$ references, while allowing to support Target operator in $O(d)$ time (when dealing with degree d vertices) and all other operators in $O(1)$ worst case time (a detailed proof can be found in [13]).*

2.2 More Compact Solutions, via Minimal Schnyder Woods

In order to reduce the space requirements, we exploit the existence of a special kind of Schnyder wood, called *minimal*, not containing ccw oriented triangles:

Lemma 1 ([8]). *Let \mathcal{G} be a planar triangulation. Then it is possible to compute in linear time a Schnyder wood without ccw oriented cycles of directed edges.*

New scheme. We modify the representation described in previous section: the first step is to endow \mathcal{G} with its minimal Schnyder wood (no ccw oriented triangles). Outgoing edges of color 0 and 1 will be still represented with two references each, while we will store only one reference for each outgoing edge of color 2 (different cases are illustrated by Fig. 3, top pictures). More precisely, let $e = (u, v)$ be an edge having face (u, v, w) at its left and face (u, v, z) at its right, and let q be the vertex defining the triangle (v, z, q) . For a vertex u we store 5 entries $T[5u] \dots T[5u + 4]$ as follows (let $e = (u, v)$ be of color c)

- for $c = 1$ (as in Theorem 1), we store in $T[5u + 2]$ and $T[5u + 3]$ two references, respectively to $\{v, w\}$ and $\{v, z\}$; (edges cw and ccw around v)
- for $c = 2$, we store in $T[5u + 4]$ a reference to:
 - edge $\{v, z\}$, if $\{z, u\}$ is directed toward u , (edge ccw around v)
 - edge $\{v, w\}$ otherwise; (edge cw around v)
- for $c = 0$, we store one reference in $T[5u]$ to $\{v, w\}$ (edge cw around v) and one reference in $T[5u + 1]$ to:
 - edge $\{q, v\}$ if $\{q, v\}$ is of color 1 oriented toward v (and thus (v, z) must be of color 2), (second edge ccw around v)
 - edge $\{v, z\}$ otherwise (edge ccw around v), as in Theorem 1

As before, the values of $S_a[e]$ and $S_b[e]$ describe the orientations of edges `LeftBack`(e) and `RightBack`(e): service bits and modulo 3 computations suffice to retrieve the orientation of edges and to distinguish all cases (since we need 2 per edge, and we have 5 references, we have to use 2 service bits per reference).

Theorem 2. *Let \mathcal{G} be a triangulation with n vertices. There exists a representation requiring $5n$ references, allowing efficient navigation, as in Theorem 1.*

A detailed proof is in [13], the case analysis is illustrated by pictures in Fig. 3.

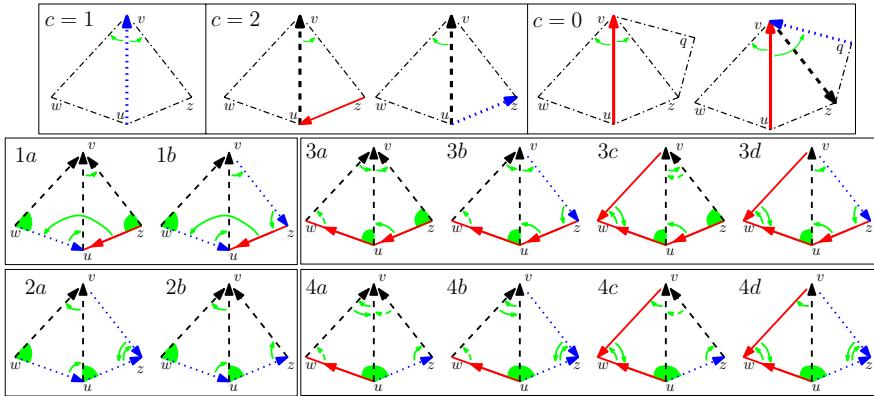


Fig. 3. More compact scheme. Neighboring relations between edges are represented by tiny oriented (green) arcs corresponding to stored references, and by filled (green) corners which implicitly describe adjacency relations between outgoing edges incident to a same vertex: because of local Schnyder rule, we do not need to store references between neighboring outgoing edges.

2.3 Further Reducing the Space Requirement

Allowing to exploit a permutation of the input vertices (reordering the vertices according to a given permutation), we are able to save one more reference per vertex. Let us first recall a result concerning the traversal of plane trees, which has been already applied to the encoding of trees [5] and labeled graphs [2]:

Lemma 2 ([5]). *Let \mathcal{T} be a plane tree whose nodes are labeled according to the DFUDS (Depth First Unary Degree Sequence) traversal of \mathcal{T} . Then the children of a given node $v \in \mathcal{T}$ have all consecutive labels.*

Scheme description. We first compute a minimal Schnyder wood of \mathcal{G} , and perform a DFUDS traversal of $\overline{\mathcal{T}}_0$ starting from its root v_0 : as $\overline{\mathcal{T}}_0$ is a spanning tree of all vertices of \mathcal{G} , we obtain a vertex labeling such that, for every vertex $v \in \mathcal{G}$, the children of v in $\overline{\mathcal{T}}_0$ have consecutive labels (as illustrated in Fig. 4). We then reorder all vertices (their associated data) according to their DFUDS label, and we store entries in table T accordingly. This allows us to save one reference per vertex: roughly speaking, we do not store a reference to **LeftFront** for edges in $\overline{\mathcal{T}}_0$, which leads to store for each vertex 4 references in table T . Let $e = (u, v)$ be of color c (incident to faces (u, v, w) and (u, v, z)), and let (q, v, z) the triangle sharing edge $\{v, z\}$ (as illustrated in Fig. 4). For edges of color $c = 1$, we store in $T[4u + 1]$ and $T[4u + 2]$ two references, to edges $\{v, w\}$ and $\{v, z\}$ respectively. For edges of color $c = 0$, we store in $T[4u]$ a reference to edge $\{q, v\}$, if $\{v, z\}$ is oriented toward z of color 2 and $\{q, v\}$ is oriented to v of color 1. We store a reference to edge $\{v, z\}$ otherwise. For edges of color $c = 2$, we store in $T[4u + 3]$ a reference to edge $\{v, z\}$, if $\{z, u\}$ is oriented toward u ; and a reference to edge $\{v, w\}$ otherwise. Service bits are stored in arrays S_a , S_b as in Theorem 2. We

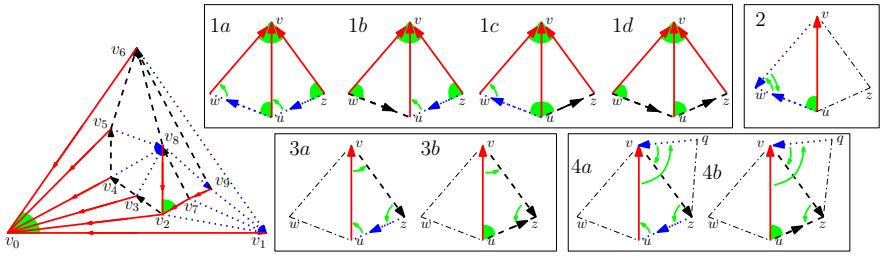


Fig. 4. A planar triangulation whose vertices are labeled according to a DFUDS traversal of tree \overline{T}_0 (left). On the right are shown all cases involved in the proof of Theorem 3: we now store only one reference for edges of color 0 (red edges), since most adjacency relations are implicitly described by the DFUDS labels.

can state the following result (the case analysis is partially illustrated by pictures in Fig. 4, see [13] for more details):

Theorem 3. *Let \mathcal{G} be a triangulation with n vertices. If one is allowed to permute the input vertices (their associated geometric data) then \mathcal{G} can be represented using $4n$ references, supporting navigation as in previous representations.*

Corollary 1. *If one is allowed to permute input points, then there exists a compact representation requiring $6n$ references which supports all navigation operators (including Target) in worst case $O(1)$ time.*

For dealing with the *higher genus case*, the key ingredient are g -Schnyder woods, a generalization of Schnyder woods for genus g triangulated surfaces [14]. We get a compact representation requiring about 5 references per vertex, applying to a g -Schnyder woods the approach relying on DFUDS order:

Theorem 4. *Let \mathcal{G} be a triangulation of genus g with n vertices. If one is allowed to permute input points, then there exists a representation requiring at most $5(n + 4g)$ references, supporting efficient navigation as in Theorem 1.*

Our approach is quite general and could be applied to other important classes of meshes, such as polygonal or quadrangular meshes homeomorphic to the sphere: just observe that nice (minimal) orientations (with bounded outgoing degree) also exist for planar quadrangulations and 3-connected graphs [17][16].

3 Experimental Results

We have written Java array-based implementations of mesh data structures and performed tests⁵ on various kinds of data (3D models and random triangulations generated with an uniform random sampler [24]). As in previous works [9][20] we consider two geometric processing procedures: computing *vertex degrees* (involving edge navigation) and *vertex normals* (involving vertex access operators and

⁵ We tested on a Dell XT2, with Core 2 Duo 1.6GHz, 32bit Windows 7, Java 1.6

Table 2. Comparison of mesh data structures. Runtime performances are expressed in nanoseconds per vertex. Vertex ordering of input points is the same for all tested data structures (vertices are accessed sequentially according to their original order).

(A) Computing vertex degree

Mesh type	Halfedge (19n)	Winged edge (19n)	Compact 6n Thm 1 (Basic)	Compact 6n Thm 1 (Fast)	Compact 5n Thm 2
Bunny	77	90	138	104	244
Iphigenia	73	91	145	108	255
Eros	58	66	133	98	233
Pierre’s hand	40	48	119	91	223
Random 500K vert.	68	84	163	126	278
Random 1M vert.	67	81	157	120	277

(B) Computing vertex normals (with simple floating precision)

Mesh type	vertices	faces	Winged-edge	Thm1 (Basic)	Thm 1 (Fast)	Thm 2
Bunny	26002	52K	607	797	724	1117
Iphigenia	49922	99K	549	820	726	1165
Eros	476596	953K	548	756	684	1061
Pierre’s hand	773465	1.54M	477	724	646	980

geometric calculations). Table 2 reports comparisons with existing data structures: *Half-edge* and *Winged-edge*. We have a compact representation using $6n$ references (*Compact 6n Basic*), encoded following the scheme described in Theorem 1. We have also a faster version (referred to as *Compact 6n Fast*), where division/modulo computations are replaced by bit shifts/masks (using 2 service bits per reference). The use of minimal Schnyder woods further speeds up the data structure (reducing the number of cases to consider): as shown in Table 2(A)-(B) the obtained speed up is not negligible. As one could expect, non-compact mesh representations are faster (*Half-edge* being slightly faster than *Winged-edge*). Our data structures achieves good trade-offs between space usage and runtime performances. While being 3 or 4 times more compact for connectivity, our structures are slightly slower, loosing a factor between 1.16 and 1.90 (comparing *Compact 6n Fast* to *Winged-edge*) on tested data for topological navigation (see Table 2). Our representations are even more competitive when considering geometric calculations: as shown in Table 2, our *Compact 6n Fast* is just slightly slower than *Winged-edge* (between 1.19 and 1.52 times slower).

References

1. Alumbaugh, T.J., Jiao, X.: Compact array-based mesh data structures. In: Proc. of the 14th Intern. Meshing Roundtable (IMR), pp. 485–503 (2005)
2. Barbay, J., Castelli-Aleardi, L., He, M., Munro, J.I.: Succinct representation of labeled graphs. Algorithmica (to appear, 2011); preliminary version in ISAAC 2007
3. Baumgart, B.G.: Winged-edge polyhedron representation. Technical report, Stanford (1972)
4. Baumgart, B.G.: A polyhedron representation for computer vision. In: AFIPS National Computer Conference, pp. 589–596 (1975)

5. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. *Algorithmica* 43(4), 275–292 (2005)
6. Blanford, D., Blelloch, G., Kash, I.: Compact representations of separable graphs. In: *SODA*, pp. 342–351 (2003)
7. Boissonnat, J.-D., Devillers, O., Pion, S., Teillaud, M., Yvinec, M.: Triangulations in CGAL. *Comp. Geometry* 22, 5–19 (2002)
8. Brehm, E.: 3-orientations and Schnyder-three tree decompositions. Master's thesis, Freie Universitaet Berlin (2000)
9. Campagna, S., Kobbelt, L., Seidel, H.P.: Direct edges - a scalable representation for triangle meshes. *Journal of Graphics Tools* 3(4), 1–12 (1999)
10. Castelli-Aleardi, L., Devillers, O., Mebarki, A.: Catalog Based Representation of 2D triangulations. *Internat. J. Comput. Geom. Appl.* 21(4), 393–402 (2011)
11. Castelli-Aleardi, L., Devillers, O., Schaeffer, G.: Succinct Representation of Triangulations with a Boundary. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) *WADS 2005*. LNCS, vol. 3608, pp. 134–145. Springer, Heidelberg (2005)
12. Castelli-Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representations of planar maps. *Theor. Comput. Sci.* 408(2–3), 174–187 (2008)
13. Castelli-Aleardi, L., Devillers, O.: Explicit array-based compact data structures for triangulations. INRIA research report 7736 (2011), <http://hal.archives-ouvertes.fr/inria-00623762/>
14. Castelli-Aleardi, L., Fusy, E., Lewiner, T.: Schnyder woods for higher genus triangulated surfaces, with applications to encoding. *Discr. & Comp. Geom.* 42(3), 489–516 (2009)
15. Chuang, R.C.-N., Garg, A., He, X., Kao, M.-Y., Lu, H.-I.: Compact Encodings of Planar Graphs via Canonical Orderings and Multiple Parentheses. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) *ICALP 1998*. LNCS, vol. 1443, pp. 118–129. Springer, Heidelberg (1998)
16. de Fraysseix, H., Ossona de Mendez, P.: On topological aspects of orientations. *Disc. Math.* 229, 57–72 (2001)
17. Felsner, S.: Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order* 18, 19–37 (2001)
18. Guibas, L.J., Stolfi, J.: Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM Trans. Graph.* 4(2), 74–123 (1985)
19. Gurung, T., Laney, D., Lindstrom, P., Rossignac, J.: SQUAD: Compact representation for triangle meshes. *Comput. Graph. Forum* 30(2), 355–364 (2011)
20. Gurung, T., Rossignac, J.: SOT: compact representation for tetrahedral meshes. In: *Proc. of the ACM Symp. on Solid and Physical Modeling*, pp. 79–88 (2009)
21. Gurung, T., Luffel, M., Lindstrom, P., Rossignac, J.: LR: compact connectivity representation for triangle meshes. *ACM Trans. Graph.* 30(4), 67 (2011)
22. Kallmann, M., Thalmann, D.: Star-vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools* 6, 7–18 (2002)
23. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. *SIAM J. on Computing* 31(3), 762–776 (2001)
24. Poulalhon, D., Schaeffer, G.: Optimal coding and sampling of triangulations. *Algorithmica* 46, 505–527 (2006)
25. Rossignac, J.: Edgebreaker: Connectivity compression for triangle meshes. *Transactions on Visualization and Computer Graphics* 5, 47–61 (1999)
26. Schnyder, W.: Embedding planar graphs on the grid. In: *SODA*, pp. 138–148 (1990)
27. Snoeyink, J., Speckmann, B.: Tripod: a minimalist data structure for embedded triangulations. In: *Workshop on Comput. Graph Theory and Combinatorics* (1999)
28. Yamanaka, K., Nakano, S.: A compact encoding of plane triangulations with efficient query supports. *Inf. Process. Lett.* 110, 803–809 (2010)

Space-Efficient Data-Analysis Queries on Grids

Gonzalo Navarro^{1,*} and Luís M. S. Russo^{2,**}

¹ Dept. of Computer Science, University of Chile
`gnavarro@dcc.uchile.cl`

² INESC-ID / IST, Tech. Univ. of Lisbon, Portugal
`luis.russo@ist.utl.pt`

Abstract. We consider various data-analysis queries on two-dimensional points. We give new space/time tradeoffs over previous work on semigroup and group queries such as sum, average, variance, minimum and maximum. We also introduce new solutions to queries rarely considered in the literature such as two-dimensional quantiles, majorities, successor/predecessor and mode queries. We face static and dynamic scenarios.

1 Introduction

Multidimensional grids arise as natural representations to support conjunctive queries in databases [2]. Typical queries such as “find all the employees with age between x_0 and x_1 and salary between y_0 and y_1 ” translate into a two-dimensional range reporting query on coordinates age and salary.

Counting the points in a two-dimensional range $Q = [x_0, x_1] \times [y_0, y_1]$, *i.e.*, computing $\text{COUNT}(Q)$, is arguably the most primitive operation in *data analysis*. Given n points, one can compute COUNT in time $O(\log n / \log \log n)$ using “linear” space, $O(n)$ integers [6]. This time is optimal within space $O(n \text{polylog}(n))$ [9]. Bose *et al.* [3] achieve the same time using asymptotically minimum space, $n + o(n)$ integers.

In this paper we focus on other data-analysis queries. Our results build on the *wavelet tree* [1], a succinct-space variant of a classical structure by Chazelle [6]. The wavelet tree has been used to handle various geometric problems, *e.g.* [13, 3]. We adapt this structure for queries we call “statistical”: The points have an associated value in $[0, W] = [0, W - 1]$. Then, given a rectangle Q , we consider the following queries:

SUM/AVG/VAR: The sum/average/variance of the values in Q .

MIN/MAX: The minimum/maximum value in Q .

QUANTILE: The k -th smallest value in Q .

MAJORITY(α): The values appearing with relative frequency $> \alpha$ in Q .

SUCC/PRED: The successor/predecessor of a value w in Q .

* Supported in part by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

** Funded by FCT proj. TAGS PTDC/EIA-EIA/112283/2009,ISTRION PTDC/EIA-EIA/114521/2009 and PIDDAC Program funds (INESC-ID multiannual funding).

These operations enable data-analysis queries such as “the average salary of employees whose annual production is between x_0 and x_1 and whose age is between y_0 and y_1 ”. The minimum operation can be used to determine “the employee with the lowest salary”, in the previous conditions. The α -majority operation can be used to compute “which are frequent ($\geq 20\%$) salaries”. Quantile queries are useful to determine “which are the 10% highest salaries”. Successor queries allow one determining “which are the salaries above \$100,000”.

Other applications for such queries are frequently found in Geographic Information Systems (GIS), where the points have a geometric interpretation and the values can be city sizes, industrial production, topographic heights, and so on. Yet another application comes from Bioinformatics, where two-dimensional points with intensities are obtained from DNA microarrays, and various kinds of data-analysis activities are carried out on them. See Rahul *et al.* [20] for an ample discussion on some of these applications and several others.

Willard [21] solved two-dimensional range-sum queries on finite groups within $O(n \log n)$ -integers space and $O(\log n)$ time. This is easily extended to SUM, AVG, and VAR. Alstrup *et al.* [1] obtained the same complexities for the semigroup model, which includes MIN/MAX. The latter can also be solved in constant time using $O(n^2)$ -integers space [4]. Rahul *et al.* [20] considered a variant of QUANTILE where one reports the *top-k smallest/largest values* in a range. They obtain $O(n \log^2 n)$ -integers space and $O(\log n + k \log \log n)$ time. Durocher and Morrison [8] consider the *mode* (most repeated value) in a two-dimensional range. Their times are sublinear but super-polylogarithmic by far.

Section 3 gives our new results for statistical queries. Our spaces range from $O(n)$ to $O(n \log n)$ integers, offering novel space/time tradeoffs for partial sums on groups (including SUM, AVG, and VAR) and for MIN/MAX queries. We enrich wavelet trees with data that speeds up the computation of these queries. Space is then reduced by sparsifying this extra data. This gives in particular a space/time tradeoff to the recent top- k smallest/largest solution [20].

The solutions to quantile, majority and successor/predecessor queries use a single data structure, a wavelet tree built on the universe of the point values. A sub-grid at each node stores the points whose values are within a value range. With this structure we also solve mode and *top-k most-frequent* queries.

While we rarely improve the best current query times, we offer significant space reductions. This can be critical to maintain large datasets in main memory.

2 Wavelet Trees

Wavelet trees [11] are defined on top of the basic RANK and SELECT functions. Let B denote a bitmap, *i.e.*, a sequence of 0’s and 1’s. $\text{RANK}(B, b, i)$ counts the number of times bit $b \in \{0, 1\}$ appears in $B[0, i]$, assuming $\text{RANK}(B, b, -1) = 0$. The dual operation, $\text{SELECT}(B, b, i)$, returns the position of the i -th occurrence of b , assuming $\text{SELECT}(B, b, 0) = -1$.

The wavelet tree represents a sequence $S[0, n[$ over alphabet $\Sigma = [0, \sigma[,$ and supports access to any $S[i]$, as well as RANK and SELECT on S , by reducing them

to bitmaps. It is a complete binary tree where each node v may have a left child labeled 0 (called the 0-child of v) and a right child labeled 1 (called the 1-child). The sequence of labels obtained when traversing the tree from the ROOT down to a node v is the *binary label* of v and is denoted $L(v)$. Likewise we denote $V(L)$ the node that is obtained by following the sequence of bits L , thus $V(L(v)) = v$. The binary labels of the leaves correspond to the binary representation of the symbols of Σ . Given $c \in \Sigma$ we denote by $V(c)$ the leaf that corresponds to symbol c . By $c\{..d\}$ we represent the sequence of the first d bits in c . Therefore, for increasing values of d , the $V(c\{..d\})$ nodes represent the path to $V(c)$.

Each node v represents (but does not store) the subsequence $S(v)$ of S formed by the symbols whose binary code starts with $L(v)$. At each node v we only store a (possibly empty) bitmap, denoted $B(v)$, of length $|S(v)|$, so that $B(v)[i] = 0$ iff $S(v)[i]\{..d\} = L(v) \cdot 0$, where $d = |L(v)| + 1$, that is, if $S(v)[i]$ belongs to the 0-child. A bit position i in $B(v)$ can be mapped to a position in each of its child nodes: we map i to position $R(v, b, i) = \text{RANK}(B(v), b, i) - 1$ of the b -child. We refer to this procedure as the *reduction* of i , and use the same notation to represent a sequence of steps, where b is replaced by a sequence of bits. Thus $R(\text{ROOT}, c, i)$, for a symbol $c \in \Sigma$, represents the reduction of i from the ROOT using the bits in the binary representation of c . With this notation we describe the way in which the wavelet tree computes RANK, which is summarized by the equation $\text{RANK}(S, c, i) = R(\text{ROOT}, c, i) + 1$. We use a similar notation $R(v, v', i)$, to represent descending from node v towards a given node v' , instead of explicitly describing the sequence of bits b such that $L(v') = L(v) \cdot b$ and writing $R(v, b, i)$.

An important path in the tree is the one obtained by choosing $R(v, B(v)[i], i)$ at each node, *i.e.*, at each node we decide to go left or right depending on the bit we are currently tracking. The resulting leaf is $V(S[i])$, therefore this process provides a way to obtain the elements of S . The resulting position is $R(\text{ROOT}, S[i], i) = \text{RANK}(S, S[i], i) - 1$.

It is also possible to move upwards on the tree, reverting the process computed by R . Let node v be the b -child of v' . Then, if i is a bit position in $B(v)$, we define the position $Z(v, v', i)$, in $B(v')$, as $\text{SELECT}(B(v'), b, i + 1)$. In general when v' is an ancestor of v the notation $Z(v, v', i)$ represents the iteration of this process. For a general sequence, SELECT can be computed by this process, as summarized by the equation $\text{SELECT}(S, c, i) = Z(V(c), \text{ROOT}, i - 1)$.

Lemma 1 ([11,13,18]). *The wavelet tree for a sequence $S[0, n]$ over alphabet $\Sigma = [0, \sigma[$ requires at most $n \log \sigma + o(n)$ bits of space¹. It solves RANK, SELECT, and access to any $S[i]$ in time $O(\log \sigma)$.*

Proof. The structure proposed by Grossi *et al.* [11] used $n \log \sigma + O\left(\frac{n \log \sigma \log \log n}{\log n}\right) + O(\sigma \log n)$ bits. Mäkinen and Navarro showed how to use only one pointer per level, reducing the last term to $O(\log \sigma \log n) = O(\log^2 n) = o(n)$. Finally, Pătrașcu [18] showed how to support binary RANK and SELECT in constant time, while reducing the redundancy of the bitmaps to $O(n / \log^2 n)$, which added over the $n \log \sigma$ bits gives $o(n)$ as well. \square

¹ From now on the space will be measured in bits and log will be to the base 2.

2.1 Representation of Grids

Consider a set \mathbf{P} of n distinct two-dimensional points (x, y) . We map coordinates to rank space using a standard method [61]. We store two sorted arrays X and Y with all the (possibly repeated) x and y coordinates, respectively. Then we convert any point (x, y) into rank space $[0, n] \times [0, n]$ in time $O(\log n)$ using two binary searches. The space of X and Y corresponds to the bare point data and will not be further mentioned. Range queries are also mapped to rank space via binary searches (in an inclusive manner in case of repeated values). This mapping time will be dominated by other query times.

Therefore we store the points of \mathbf{P} on a $[0, n] \times [0, n]$ grid, with exactly one point per row and one per column. We regard this set as a sequence $S[0, n]$ and the grid is formed by the points $(i, S[i])$. We represent S using a wavelet tree.

The information relative to a point $p_0 = (x_0, y_0)$ is usually tracked from the ROOT and denoted $R(\text{ROOT}, y_0\{..d\}, x_0)$. A pair of points $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$, where $x_0 \leq x_1$ and $y_0 \leq y_1$, defines a rectangle; this is the typical query range we consider in this paper. Rectangles have an implicit representation in wavelet trees, spanning $O(\log n)$ nodes [13]. The binary representation of y_0 and y_1 share a (possibly empty) common prefix. Therefore the paths $V(y_0\{..d\})$ and $V(y_1\{..d\})$ have a common initial path and then split at some node of depth k , i.e., $V(y_0\{..d\}) = V(y_1\{..d\})$ for $d \leq k$ and $V(y_0\{..d'\}) \neq V(y_1\{..d'\})$ for $d' > k$. Geometrically, $V(y_0\{..k\}) = V(y_1\{..k\})$ corresponds to the smallest horizontal band of the form $[j \cdot n/2^k, (j+1) \cdot n/2^k]$ that contains the query rectangle Q , for an integer j . For $d' > k$ the nodes $V(y_0\{..d'\})$ and $V(y_1\{..d'\})$ correspond respectively to successively thinner, non-overlapping bands that contain the coordinates y_0 and y_1 .

Given a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ we consider the nodes $V(y_0\{..d\} \cdot 1)$ such that $y_0\{..d\} \cdot 1 \neq y_0\{..d+1\}$, and the nodes $V(y_1\{..d\} \cdot 0)$ such that $y_1\{..d\} \cdot 0 \neq y_1\{..d+1\}$. These nodes, together with $V(y_0)$ and $V(y_1)$, form the implicit representation of $[y_0, y_1]$, denoted $\text{IMP}(y_0, y_1)$. The size of this set is $O(\log n)$. Let us recall a well-known application of this decomposition.

Lemma 2. *Given n two-dimensional points, the number of points inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\text{COUNT}(Q)$, can be computed in time $O(\log n)$ with a structure that requires $n \log n + o(n)$ bits.*

Proof. The result is $\sum_{v \in \text{IMP}(y_0, y_1)} R(\text{ROOT}, v, x_1) - R(\text{ROOT}, v, x_0 - 1)$. Notice that all the values $R(\text{ROOT}, y_0\{..d\}, x)$ and $R(\text{ROOT}, y_1\{..d\}, x)$ can be computed sequentially, in total time $O(\log n)$, for $x = x_1$ and $x = x_0 - 1$. For a node $v \in \text{IMP}(y_0, y_1)$ the desired difference can be computed from one of these values in time $O(1)$. Then the lemma follows. \square

2.2 Dynamism

We can support point insertions and deletions on a fixed $n \times n$ grid. Dynamic variants of the bitmaps stored at each wavelet tree node raise the extra space to $o(\log n)$ per point and multiply the times by $O(\log n / \log \log n)$ [12, 15].

Lemma 3. *Given s points on an $n \times n$ grid, there is a structure using $s \log n + o(s \log n)$ bits, answering queries in time $O(t \log n / \log \log n)$, where t is the time complexity of the query using static wavelet trees. It handles point insertions and deletions in time $O(\log^2 n / \log \log n)$.*

Proof. We use the same data structure and query algorithms of the static wavelet trees described in Section 2.1 yet representing their bitmaps with the dynamic variants [12][15]. We also maintain vector X , but not Y ; we use the y -coordinates directly instead since the wavelet tree handles repetitions in y .

Instead of an array, we use for X a multiary tree with arity $O(\sqrt{\log n})$ and with leaves holding $\Theta(\log n / \log \log n)$ elements [15]. This retains the same time penalty factor and poses an extra space (on top of the bare coordinates) of $o(s \log n)$ bits. When inserting a new point (x, y) , apart from inserting x into X , we track the point downwards in the wavelet tree, doing the insertion at each of the $\log n$ bitmaps. Deletion is analogous. \square

3 Statistical Queries

We now consider points weighted by an integer-valued function $w : \mathbf{P} \rightarrow [0, W[$. Let us define the sequence of weights $W(v)$ associated to each wavelet tree node v . If $S(v) = p_0, p_1, \dots, p_{|S(v)|}$, then $W(v) = w(p_0), w(p_1), \dots, w(p_{|S(v)|})$.

3.1 Range Sum, Average, and Variance

We start with a solution to several range sum problems on groups.

Theorem 1. *Given n two-dimensional points with associated values in $[0, W[$, the sum of the point values inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\text{SUM}(Q)$, can be computed in time $O(\ell \log_\ell n)$, with a structure that requires $n \log n + n \log_\ell n (\log W + O(1))$ bits, for any $\ell \in [2, n]$. It can also compute the average of the values, $\text{AVG}(Q)$. A similar structure requiring 3 times the space computes the variance of the values, $\text{VAR}(Q)$.*

Proof. We enrich the bitmaps of the wavelet tree for \mathbf{P} . For each node v we represent its vector $W(v) = w(p_0), w(p_1), \dots, w(p_{|S(v)|})$ as a bitmap $A(v)$, where we concatenate the unary representation of the $w(p_i)$'s, i.e., $w(p_i)$ 0's followed by a 1. These bitmaps $A(v)$ are represented in a compressed format [17] which requires at most $|S(v)| \log W + O(|S(v)|)$ bits. With this structure we can determine the sum $w(p_0) + w(p_1) + \dots + w(p_i)$, i.e., the partial sums, in constant time by means of $\text{SELECT}(A(v), 1, i)$ queries², $\text{WSUM}(v, i) = \text{SELECT}(A(v), 1, i+1) - i$ is the sum of the first $i+1$ values. In order to compute $\text{SUM}(Q)$ we use a formula similar to the one of Lemma 2.

$$\sum_{v \in \text{IMP}(y_0, y_1)} \text{WSUM}(v, R(\text{ROOT}, v, x_1)) - \text{WSUM}(v, R(\text{ROOT}, v, x_0 - 1)). \quad (1)$$

² Using constant-time SELECT structures on their internal bitmap H [17].

To obtain the tradeoff related to ℓ , we store the A bitmaps only for nodes v whose height $h(v)$ is a multiple of $h = \lceil \log \ell \rceil$, including the leaves. If a node $v \in \text{IMP}(y_0, y_1)$ occurs at a level that does not have an associated bitmap $A(v)$, it is necessary to branch recursively to both children until we reach a level that has such a bitmap. In this case we use Eq. (1) to sum over all the nodes in the levels that have $A(v)$ bitmaps. The time raises by up to $2^h = O(\ell)$ factor. However, if a node v must access $2^h > 1$ bitmaps $A(\cdot)$, a node v' with height $h(v') = h(v) + 1$ must access only 2^{h-1} . As in $\text{IMP}(y_0, y_1)$ there are at most two nodes of each height, the total extra cost over all the nodes of heights $[h \cdot i, h \cdot (i+1)[$, for any $i \in [0, \log(n)/h[$ is not $O(h\ell)$ but amortizes to $2^h + 2^{h-1} + \dots = 2 \cdot 2^h$, and hence the overall extra cost factor is $O(2^h/h) = O(\ell / \log \ell)$.

The average weight inside Q is computed as $\text{AVG}(Q) = \text{SUM}(Q)/\text{COUNT}(Q)$, where the latter is computed with the same structure by just adding up the interval lengths in $\text{IMP}(y_0, y_1)$. To compute variance we use an additional instance of the same data structure, with weights $w'(p) = w^2(p)$, and then $\text{VAR}(Q) = \text{SUM}'(Q) - \text{SUM}(Q)^2/\text{COUNT}(Q)$, where SUM' uses the weights w' . The space triples because the weights $w'(p)$ require $\lceil 2 \log W \rceil$ bits in bitmaps $A(v)$. \square

The solution applies to finite groups $(G, \oplus, ^{-1}, 0)$. We store $\text{WSUM}(v, i) = w(p_0) \oplus w(p_1) \oplus \dots \oplus w(p_i)$, directly using $\lceil \log |G| \rceil$ bits per entry. The terms $\text{WSUM}(v, i) - \text{WSUM}(v, j)$ of Eq. (1) are replaced by $\text{WSUM}(v, j)^{-1} \oplus \text{WSUM}(v, i)$.

A dynamic variant is obtained by using the dynamic wavelet trees of Lemma 3, and a dynamic partial sums data structure [14] instead of $A(v)$.

Theorem 2. *Given s points on an $n \times n$ grid, with associated values in $[0, W[$, there is a structure using $s \log n + s \log_\ell n \log W(1 + o(1))$ bits, for any $\ell \in [2, W]$, that answers the queries in Thm. 1 in time $O(\ell \log^2 n)$, and supports point insertions/deletions, and value updates, in time $O(\log^2 n / \log \log n + \log n \log_\ell n)$.*

Proof. The dynamic searchable partial sums [14] take $n \log W + o(n \log W)$ bits to store an array of n values, and support partial sums, as well as insertions and deletions of values, in time $O(\log n)$. This multiplies the time complexities of Thm. 1. For insertions we must insert the new bits at all the levels as in Lemma 3 and also insert the new values at $\log_\ell n$ levels. Deletions are analogous. To update a value we just delete and reinsert the point. \square

3.2 Range Minima and Maxima

For the one-dimensional problem there exists a data structure using just $2n + o(n)$ bits, which answers queries in constant time without accessing the values [9]. This structure allows for a better space/time tradeoff compared to range sums.

For the queries that follow we do not need the exact $w(p)$ values, but just their order. So, if $W > n$, we store an array with the (at most) n different values of $w(p)$ in the point set, and map all the weights to rank space in time $O(\log n)$, which will be negligible. The extra space that this requires is $n \lceil \log W \rceil$ bits. In exchange, many complexities will be expressed in terms of $u = \min(n, W)$.

Theorem 3. *Given n two-dimensional points with associated values in $[0, W[$, the minimum of the point values inside a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\text{MIN}(Q)$, can be found in time $O(\log \ell \log n)$, with a structure using $n \log_\ell n \log u + O(n \log n) + n \log W$ bits, for any $\ell \in [2, n]$ and $u = \min(n, W)$. A similar structure answers $\text{MAX}(Q)$ within the same space and time.*

Proof. We associate to each node v the one-dimensional data structure [9] corresponding to $W(v)$, which takes $2|W(v)| + o(|W(v)|)$ bits. This adds up to $O(n \log n)$ bits overall. Let us call $\text{WMIN}(v, i, j) = \arg \min_{i \leq m \leq j} W(v)[m]$ the one-dimensional operation. Then we can find in constant time the position of the minimum value inside each $v \in \text{IMP}(y_0, y_1)$ (without the need to store the values in the node), and the range minimum is

$$\min_{v \in \text{IMP}(y_0, y_1)} W(v)[\text{WMIN}(v, R(\text{ROOT}, v, x_0), R(v, \text{ROOT}, v, x_1 + 1) - 1)].$$

To complete the comparison we need to compute the $O(\log n)$ values $W(v)[m]$ of different nodes v . By storing the $A(v)$ bitmaps of Thm. [1] every $\log \ell$ levels, the time is just $O(\log \ell \log n)$ because we have to track down just one point for each $v \in \text{IMP}(y_0, y_1)$. The case of $\text{MAX}(Q)$ is analogous. \square

We can now solve the top- k query of Rahul *et al.* [20] by iterating over Thm. [3]. Once we identify that the overall minimum is some $W(v)[m]$ from the range $W(v)[i, j]$, we can find the second minimum among the other candidate ranges plus the ranges $W(v)[i, m - 1]$ and $W(v)[m + 1, j]$. A priority queue handling the ranges will perform k minimum extractions and $O(k + \log n)$ insertions, and its size will be limited to k . So the overall time is $O(\log \ell \log n + k(\log \ell + \log k))$ using a priority queue with constant insertion time [5]. This is comparable with previous time complexities [20] within much less space (even for $\ell = 1$).

3.3 Range Median and Quantiles

We compute the median element, or more generally, the k -th smallest value $w(p)$ in an area $Q = [x_0, x_1] \times [y_0, y_1]$ (the median corresponds to $k = \text{COUNT}(Q)/2$).

From now on we use a different wavelet tree decomposition, on the universe $[0, u[$ of $w(\cdot)$ values rather than on y coordinates. This can be seen as a wavelet tree on grids rather than on sequences: the node v of height $h(v)$ stores a grid $G(v)$ with the points $p \in \mathbf{P}$ such that $\lfloor w(p)/2^{h(v)} \rfloor = L(v\{\dots\log u\} - h(v))$. Note that each leaf c stores the points p with weight $w(p) = c$.

Theorem 4. *Given n two-dimensional points with associated values in $[0, W[$, the k -th smallest value of points within a query rectangle $Q = [x_0, x_1] \times [y_0, y_1]$, $\text{QUANTILE}(k, Q)$, can be found in time $O(\ell \log n \log_\ell u)$, with a structure using $n \log n \log_\ell u + O(n \log u) + n \log W$ bits, for any $\ell \in [2, u]$ and $u = \min(n, W)$. The time drops to $O(\ell \log n \log_\ell u / \log \log n)$ by using $n \log n \log_\ell u(1 + o(1)) + O(n \log u) + n \log W$ bits of space.*

Proof. We use the wavelet tree on grids just described, representing each grid $G(v)$ with the structure of Lemma 2. To solve this query we start at root of the wavelet tree of grids and consider its left child, v . If $t = \text{COUNT}(Q) \geq k$ on grid $G(v)$, we continue the search on v . Otherwise we continue the search on the right child of the root, with parameter $k - t$. When we arrive at a leaf corresponding to value c , then c is the k -th smallest value in $\mathbf{P} \cap Q$.

Notice that we need to reduce the query rectangle to each of the grids $G(v)$ found in the way. We store the X and Y arrays only for the root grid, which contains the whole \mathbf{P} . For this and each other grid $G(v)$, we store a bitmap $X(v)$ so that $X(v)[i] = b$ iff the i -th point in x -order is stored at the b -child of v . Similarly, we store a bitmap $Y(v)$ with the same bits in y -order. Therefore, when we descend to the b -child of v , for $b \in \{0, 1\}$, we remap x_0 to $\text{RANK}(X(v), b, x_0)$ and x_1 to $\text{RANK}(X(v), b, x_1 + 1) - 1$, and analogously for y_0 and y_1 with $Y(v)$.

The bitmaps $X(v)$ and $Y(v)$ add up to $O(n \log u)$ bits of space. For the grids, consider that each point in each grid contributes at most $\log n + o(1)$ bits, and each $p \in \mathbf{P}$ appears in $\lceil \log u \rceil - 1$ grids (as the root grid is not really necessary).

To reduce space, we store the grids $G(v)$ only every $\lceil \log \ell \rceil$ levels (the bitmaps $X(v)$ and $Y(v)$ are still stored for all the levels). This gives the promised space. For the time, the first decision on the root requires computing up to ℓ operations $\text{COUNT}(Q)$, but the decision on its child requires half of them. Just as in Thm. II, the total time adds up to $O(\ell \log n \log_\ell u)$.

To achieve the final alternative space/time, replace our wavelet trees of Lemma 2 by Bose et al.'s counting structure [3]. \square

Note that the basic construction allows us to count the number of points $p \in Q$ whose values $w(p)$ fall in a given range $[w_0, w_1]$, within time $O(\ell \log n \log_\ell u)$ or $O(\ell \log n \log_\ell u / \log \log n)$. This is another useful operation for data analysis, and can be obtained with the formula $\sum_{v \in \text{IMP}(w_0, w_1)} \text{COUNT}(Q)$.

3.4 Range Majority

An α -majority of a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ is a weight that occurs more than $\alpha \cdot \text{COUNT}(Q)$ times inside Q , for some $\alpha \in [0, 1]$. We can solve this problem with the same structure used for Thm. 4.

Theorem 5. *The structures of Thm. 4 can compute all the α -majorities of the point values inside Q , $\text{MAJORITY}(\alpha, Q)$, in time $O(\frac{1}{\alpha} \ell \log n \log_\ell u)$ or $O(\frac{1}{\alpha} \ell \log n \log_\ell u / \log \log n)$, respectively, where α can be chosen at query time.*

Proof. For $\alpha \geq \frac{1}{2}$ we find the median c of Q and then use the leaf c to count its frequency in Q . If this is more than $\alpha \cdot \text{COUNT}(Q)$, then c is the answer, else there is no α -majority. For $\alpha < \frac{1}{2}$, we solve the query by probing all the $(i \cdot \alpha) \text{COUNT}(Q)$ -th elements in Q . \square

Culpepper et al. [7] show how to find the mode, and in general the k most repeated values inside Q , using successively more refined QUANTILE queries. Let the k -th most repeated value occur $\alpha \cdot \text{COUNT}(Q)$ times in Q , then we require at most $4/\alpha$ quantile queries [7]. The same result can be obtained by probing successive values $\alpha = 1/2^i$ with $\text{MAJORITY}(\alpha)$ queries.

3.5 Range Successor and Predecessor

The successor (predecessor) of a value w in a rectangle $Q = [x_0, x_1] \times [y_0, y_1]$ is the smallest (largest) weight larger (smaller) than, or equal to, w in Q . We also have an efficient solution using our wavelet trees on grids.

Theorem 6. *The structures of Thm. 4 can compute the successor and predecessor of a value w within the values of the points inside Q , $\text{SUCC}(w, Q)$ and $\text{PRED}(w, Q)$, in time $O(\ell \log n \log_\ell u)$ or $O(\ell \log n \log_\ell u / \log \log n)$, respectively.*

Proof. We consider the nodes $v \in \text{IMP}(w, +\infty)$ from left to right, tracking rectangle Q in the process. The condition for continuing the search below a node v that is in $\text{IMP}(w, +\infty)$, or is a descendant of one such node, is that $\text{COUNT}(Q) > 0$ on $G(v)$. $\text{SUCC}(w, Q)$ is the value associated with the first leaf found by this process. Likewise, $\text{PRED}(w, Q)$ is computed by searching $\text{IMP}(-\infty, w)$ from right to left. To reduce space we store the grids only every $\lceil \log \ell \rceil$ levels, and thus determining whether a child has a point in Q may cost up to $O(\ell \log n)$. Yet, as for Thm. 11, the total time amortizes to $O(\ell \log n \log_\ell u)$. \square

3.6 Dynamism

Our dynamic wavelet tree of Lemma 3 supports range counting and point insertions/deletions on a fixed grid in time $O(\log^2 n / \log \log n)$ (other tradeoffs exist [16]). If we likewise assume that our grid is fixed in Thms. 4, 5 and 6, we can also support point insertions and deletions (and thus changing the value of a point).

Theorem 7. *Given s points on an $n \times n$ grid, with associated values in $[0, W[$, there is a structure using $s \log n \log_\ell W(1+o(1))$ bits, for any $\ell \in [2, W]$, that answers queries QUANTILE, SUCC and PRED in time $O(\ell \log^2 n \log_\ell W / \log \log n)$, and query MAJORITY(α) in time $O(\frac{1}{\alpha} \ell \log^2 n \log_\ell W / \log \log n)$. It supports point insertions and deletions, and value updates, in time $O(\log^2 n \log_\ell W / \log \log n)$.*

Proof. We use the data structure of Thms. 4, 5 and 6, modified as follows. We build the wavelet tree on the universe $[0, W[$ and thus do not map the universe values to rank space. The grids $G(v)$ use the dynamic structure of Lemma 3, on global y -coordinates $[0, n]$. We maintain the global array X of Lemma 3 plus the vectors $X(v)$ of Thm. 4, the latter using dynamic bitmaps [12, 15]. The time for the queries follows immediately. For updates we track down the point to insert/delete across the wavelet tree, inserting or deleting it in each grid $G(v)$ found in the way, and also in the corresponding vector $X(v)$. \square

4 Final Remarks

Many other data-analysis queries may be of interest. A prominent one lacking good solutions is to find the mode, that is, the most frequent value, in a rectangle, and its generalization to the top- k most frequent values. There has been some recent progress on the one-dimensional version [10] and even in two dimensions [8], but the results are far from satisfactory.

References

1. Alstrup, S., Brodal, G., Rauhe, T.: New data structures for orthogonal range searching. In: FOCS, pp. 198–207 (2000)
2. Berg, M., Cheong, O., Kreveld, M., Overmars, M.: Orthogonal range searching: Querying a database. In: Computational Geometry, pp. 95–120. Springer, Heidelberg (2008)
3. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct Orthogonal Range Search Structures on a Grid with Applications to Text Indexing. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 98–109. Springer, Heidelberg (2009)
4. Brodal, G., Davoodi, P., Rao, S.: On space efficient two dimensional range minimum data structures. Algorithmica (2011), doi:10.1007/s00453-011-9499-0
5. Carlsson, S., Munro, J.I., Poblete, P.V.: An Implicit Binomial Queue with Constant Insertion Time. In: Karlsson, R., Lingas, A. (eds.) SWAT 1988. LNCS, vol. 318, pp. 1–13. Springer, Heidelberg (1988)
6. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Comp. 17(3), 427–462 (1988)
7. Culpepper, J.S., Navarro, G., Puglisi, S.J., Turpin, A.: Top- k Ranked Document Search in General Text Databases. In: de Berg, M., Meyer, U. (eds.) ESA 2010, part II. LNCS, vol. 6347, pp. 194–205. Springer, Heidelberg (2010)
8. Durocher, S., Morrison, J.: Linear-space data structures for range mode query in arrays. CoRR, 1101.4068 (2011)
9. Fischer, J.: Optimal Succinctness for Range Minimum Queries. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 158–169. Springer, Heidelberg (2010)
10. Gagie, T., Navarro, G., Puglisi, S.J.: Colored Range Queries and Document Retrieval. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 67–81. Springer, Heidelberg (2010)
11. Grossi, R., Gupta, A., Vitter, J.: High-order entropy-compressed text indexes. In: SODA, pp. 841–850 (2003)
12. He, M., Munro, J.I.: Succinct Representations of Dynamic Strings. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 334–346. Springer, Heidelberg (2010)
13. Mäkinen, V., Navarro, G.: Rank and select revisited and extended. Theo. Comp. Sci. 387(3), 332–347 (2007)
14. Mäkinen, V., Navarro, G.: Dynamic entropy-compressed sequences and full-text indexes. ACM Trans. Alg., 4(3), art. 32 (2008)
15. Navarro, G., Sadakane, K.: Fully-functional static and dynamic succinct trees. CoRR, 0905.0768v5 (2010)
16. Nekrich, Y.: Orthogonal range searching in linear and almost-linear space. Comp. Geom. Theo. and App. 42(4), 342–351 (2009)
17. Okanohara, D., Sadakane, K.: Practical entropy-compressed rank/select dictionary. In: ALENEX (2007)
18. Pătrașcu, M.: Succincter. In: FOCS, pp. 305–313 (2008)
19. Pătrașcu, M.: Lower bounds for 2-dimensional range counting. In: STOC, pp. 40–46 (2007)
20. Rahul, S., Gupta, P., Janardan, R., Rajan, K.S.: Efficient Top- k Queries for Orthogonal Ranges. In: Katoh, N., Kumar, A. (eds.) WALCOM 2011. LNCS, vol. 6552, pp. 110–121. Springer, Heidelberg (2011)
21. Willard, D.: New data structures for orthogonal range queries. SIAM J. Comp. 14(232–253) (1985)

A Polynomial Kernel for FEEDBACK ARC SET on Bipartite Tournaments

Pranabendu Misra¹, Venkatesh Raman²,
M. S. Ramanujan², and Saket Saurabh²

¹ Chennai Mathematical Institute, Chennai, India
pranabendu@cmi.ac.in

² The Institute of Mathematical Sciences, Chennai, India
{vraman,msramanujan,saket}@imsc.res.in

Abstract. In the k -FEEDBACK ARC/VERTEX SET problem we are given a directed graph D and a positive integer k and the objective is to check whether it is possible to delete at most k arcs/vertices from D to make it acyclic. Dom et al.[CIAC, 2006] initiated a study of the FEEDBACK ARC SET problem on bipartite tournaments (k -FASBT) in the realm of parameterized complexity. They showed that k -FASBT can be solved in time $O(3.373^k n^6)$ on bipartite tournaments having n vertices. However, until now there was no known polynomial sized problem kernel for k -FASBT. In this paper we obtain a cubic vertex kernel for k -FASBT. This completes the kernelization picture for the FEEDBACK ARC/VERTEX SET problem on tournaments and bipartite tournaments, as for all other problems polynomial kernels were known before. We obtain our kernel using a non-trivial application of “independent modules” which could be of independent interest.

1 Introduction

In the k -FEEDBACK ARC/VERTEX SET problem we are given a directed graph D and a positive integer k and the objective is to check whether it is possible to delete at most k arcs/vertices from D to make it acyclic. These are classical NP-complete problems and have been extensively studied in every paradigm that deals with coping with NP-hardness including approximation and parameterized algorithms. These problems have several applications and in many of these applications we can restrict our inputs to more structured directed graphs like *tournaments* (orientation of a complete graph) and *bipartite tournaments* (orientation of a complete bipartite graph).

FEEDBACK ARC SET on tournaments is useful in *rank aggregation*. In *rank aggregation* we are given several rankings of a set of objects, and we wish to produce a single ranking that on average is as consistent as possible with the given ones, according to some chosen measure of consistency. This problem has been studied in the context of voting [9][13], machine learning [12], and search engine ranking [17]. A natural consistency measure for rank aggregation is the number of pairs that occur in a different order in the two rankings. This leads

to Kemeny rank aggregation [24][25], a special case of a weighted version of k -FEEDBACK ARC SET on tournaments (k -FAST). Similarly, FEEDBACK ARC SET on bipartite tournaments finds its usefulness in applications that require establishment of mappings between “ontologies”, we refer to [28] for further details.

In this paper, we consider k -FEEDBACK ARC SET problem on bipartite tournaments. More precisely the problem we study is the following:

k -FEEDBACK ARC SET IN BIPARTITE TOURNAMENTS (k -FASBT)

Input: A bipartite tournament $H = (X \cup Y, E)$ and a positive integer k .

Parameter: k

Question: Does there exist a subset $F \subseteq E$ of at most k arcs whose removal makes H acyclic?

In the last few years several algorithmic results have appeared around the k -FEEDBACK ARC SET problem on tournaments and bipartite tournaments. Speckenmeyer [29] showed that FEEDBACK VERTEX SET is NP-complete on tournaments. k -FAST was conjectured to be NP-complete for a long time [5], and only recently was this proven. Ailon et al. [2] gave a randomized reduction from FEEDBACK ARC SET on general directed graphs, which was independently derandomized by Alon [3] and Charbit et al. [11]. From approximation perspective, initially a constant factor approximation was obtained for k -FAST in [32] and later it was shown in [26] that it admits a polynomial time approximation scheme. Now we turn to problems on bipartite tournaments. Cai et al. [10] showed that FEEDBACK VERTEX SET on bipartite tournaments is NP-complete. They have also established a min-max theorem for feedback vertex set on bipartite tournaments. However, only recently Guo et al. [20] showed that k -FASBT is NP-complete. k -FASBT is also known to admit constant factor approximation algorithms [21][33].

These problems are also well studied in parameterized complexity. In this area, a problem with input size n and a parameter k is said to be fixed parameter tractable (FPT) if there exists an algorithm to solve this problem in time $f(k) \cdot n^{O(1)}$, where f is an arbitrary function of k . Raman and Saurabh [27] showed that k -FAST is FPT by obtaining an algorithm running in time $O(2.415^k \cdot k^{4.752} + n^{O(1)})$. Recently, Alon et al. [4] have improved this result by giving an algorithm for k -FAST running in time $O(2^{O(\sqrt{k} \log^2 k)} + n^{O(1)})$. Moreover, a new algorithm due to Karpinsky and Schudy [23] with running time $O(2^{O(\sqrt{k})} + n^{O(1)})$ improves again the complexity of k -FAST. Dom et al. [16] obtained an algorithm with running time $O(3.373^k n^6)$ for k -FASBT based on a new forbidden subgraph characterization. In this paper we investigate k -FASBT from the view point of kernelization, currently one of the most active subfields of parameterized algorithms.

A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm, called a *kernelization* algorithm, that reduces the input instance to an instance whose size is bounded by a polynomial $p(k)$ in k , while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the

problem. Kernelization has been at the forefront of research in parameterized complexity in the last couple of years, leading to various new polynomial kernels as well as tools to show that several problems do not have a polynomial kernel under some complexity-theoretic assumptions [6,7,8,14,15,18,31]. In this paper we continue the current theme of research on kernelization and obtain a *cubic vertex* kernel for k -FASBT. That is, we give a polynomial time algorithm which given an input instance (H, k) to k -FASBT obtains an equivalent instance (H', k') on $O(k^3)$ vertices. This completes the kernelization picture for the FEEDBACK ARC/VERTEX SET problem on tournaments and bipartite tournaments, as for all other problems polynomial kernels were known before. FEEDBACK VERTEX SET admits a kernel with $O(k^2)$ and $O(k^3)$ vertices on tournaments and bipartite tournaments, respectively [1,4,16]. However, only recently an $O(k)$ vertex kernel for k -FAST was obtained [6]. We obtain our kernel for k -FASBT using a data reduction rule that applies independent modules in a non trivial way. Previously, clique and transitive modules were used in obtaining kernels for CLUSTER EDITING and k -FAST [6,19].

2 Preliminaries

A bipartite tournament $H = (X \cup Y, E)$ is an orientation of a complete bipartite graph, meaning its vertex set is the union of two independent disjoint sets X and Y and for every pair of vertices $u \in X$ and $v \in Y$ there is exactly one arc between them. Some times we will use $V(H)$ to denote $X \cup Y$. For a vertex $v \in H$, we define $N^+(v) = \{w \in H | (v, w) \in E\}$ and $N^-(v) = \{u \in H | (u, v) \in E\}$. Essentially, the sets $N^+(v)$ and $N^-(v)$ are the set of out-neighbors and in-neighbors of v respectively. By C_4 we denote a directed cycle of length 4. Given a digraph $D = (V, E)$ and a subset W of either V or E , by $D \setminus W$ we denote the digraph obtained by deleting W from either V or E .

Given a directed graph $D = (V, E)$, a subset $F \subseteq E$ of arcs is called a feedback arc set *fas* if $D \setminus F$ is a directed acyclic graph. A feedback arc set F is called *minimal fas* if none of the proper subsets of F is a fas. Given a directed graph $D = (V, E)$ and a set F of arcs in E define $D\{F\}$ to be the directed graph obtained from D by reversing all arcs of F . In our arguments we will need the following folklore characterization of minimal feedback arc sets in directed graphs.

Proposition 1. *Let $D = (V, A)$ be a directed graph and F be a subset of A . Then F is a minimal feedback arc set of D if and only if $D\{F\}$ is a directed acyclic graph.*

3 Modular Partitions

A *Module* of a directed graph $D = (V, E)$ is a set $S \subseteq V$ such that $\forall u, v \in S$, $N^+(u) \setminus S = N^+(v) \setminus S$, and $N^-(u) \setminus S = N^-(v) \setminus S$. Essentially, every vertex in S has the same set of in-neighbors and out-neighbors outside S . The empty set

and the whole of the vertex set are called *trivial modules*. We always mean non-trivial modules unless otherwise stated. Every vertex forms a singleton module. A *maximal module* is a module such that we cannot extend it by adding any vertex. The *modular partition*, \mathcal{P} , of a directed graph D , is a partition of the vertex set V into $(V_1, V_2, \dots, V_\ell)$, such that every V_i is a maximal module. Now we look at some simple properties of the modules of a bipartite tournament.

Lemma 1. *Let $H = (X \cup Y, E)$ be a bipartite tournament, and S be any non-trivial module of H . Then S is an independent set. Thus $S \subset X$ or $S \subset Y$.*

Proof. If $|S| = 1$ then it is obvious. So we assume that $|S| \geq 2$. But S cannot contain two vertices x and y such that $x \in X$ and $y \in Y$ because their neighborhoods excluding S are different. Hence, S contains vertices from only one of the partitions. Hence, S is an independent set. Now because H is a bipartite tournament we have that $S \subseteq X$ or $S \subseteq Y$. \square

Consider the modular partition $\mathcal{P} = \mathbb{A} \cup \mathbb{B}$ of $X \cup Y$, where $\mathbb{A} = \{A_1, A_2, \dots\}$ is a partition of X and $\mathbb{B} = \{B_1, B_2, \dots\}$ is a partition of Y . Let $A \in \mathbb{A}$ and $B \in \mathbb{B}$ be two modules. Since A and B are modules, all the arcs between them are either directed from A to B (denoted by $A \rightarrow B$) or B to A (denoted by $B \rightarrow A$).

Lemma 2 ([30]). *For a bipartite tournament $H = (X \cup Y, E)$, a modular partition is unique and it can be computed in $O(|X \cup Y| + |E|)$.*

Now we define the well known notion of quotient graph of a modular partition.

Definition 1. *To a modular partition $\mathcal{P} = \{V_1, \dots, V_\ell\}$ of a directed graph D , we associate a quotient directed graph $\mathcal{D}_{\mathcal{P}}$, whose vertices $\{v_1, \dots, v_\ell\}$ are in one to one correspondence with the parts of \mathcal{P} . There is an arc (v_i, v_j) from vertex v_i to vertex v_j of $\mathcal{D}_{\mathcal{P}}$ if and only if $V_i \rightarrow V_j$. We denote its vertex set by $V(\mathcal{D}_{\mathcal{P}})$ and the edge set by $E(\mathcal{D}_{\mathcal{P}})$.*

We conclude this section with the observation that for a bipartite tournament H , the quotient graph $\mathcal{H}_{\mathcal{P}}$ corresponding to the modular partition \mathcal{P} , is a bipartite tournament. We refer to the recent survey of Habib and Paul [22] for further details and other algorithmic applications of modular decomposition.

4 Reductions and Kernel

In this section we show that k -FASBT admits a polynomial kernel with $O(k^3)$ vertices. We provide a set of reduction rules and assume that at each step we use the first possible applicable rule. After each reduction rule we discuss its soundness, that is, we prove that the input and output instances are equivalent. If no reduction rule can be used on an instance (H, k) , we claim that $|V(H)|$ is bounded by $O(k^3)$. Through out this paper, whenever we say cycle, we mean directed cycle; whenever we speak of a fas, we mean a *minimal fas*, unless otherwise stated.

4.1 Data Reduction Rules

We start with some simple observations regarding the structure of directed cycles in bipartite tournaments.

Lemma 3 ([16]). *A bipartite tournament H has a cycle C if and only if it has a C_4 .*

Lemma 4. \star^1 *Let $H = (V, E)$ be a bipartite tournament. If $v \in V$ is part of some cycle C , then it is also part of some C_4 .*

We now have the following simple rule.

Reduction Rule 1. *If $v \in V$ is not part of any cycle in H then remove v , that is, return an instance $(H \setminus v, k)$.*

Lemma 5. \star *Reduction Rule 1 is sound and can be applied in polynomial time.*

Reduction Rule 2. *If there is an arc $e \in E$, such that there are at least $k + 1$ C_4 's, which pairwise have only e in common, then reverse e and reduce k by 1. That is, return the instance $(H \setminus \{e\}, k - 1)$.*

Lemma 6. *Reduction Rule 2 is sound and can be applied in polynomial time.*

Proof. Such an arc e must be in any fas of size $\leq k$. Otherwise, we have to pick at least one distinct arc for each of the cycles, and there are $\geq k + 1$ of them.

To find such arcs, we use the idea of an *opposite arc*. Two vertex disjoint arcs $e_1 = (a, b)$ and $e_2 = (c, d)$ are called opposite arcs if (a, b, c, d) forms a C_4 . Consider the set of opposite arcs of an arc e ; this can be easily computed in polynomial time. It is easy to see that an arc e has $\geq k + 1$ vertex disjoint opposite arcs if and only if there are $\geq k + 1$ C_4 which pairwise have only e in common. Hence, we only need to check if the size of the opposite set, for any arc e , is larger than k and reverse e if that is the case. \square

Let $H = (X \cup Y, E)$ be a bipartite tournament. From now onwards we fix the unique modular partition $\mathcal{P} = \mathbb{A} \cup \mathbb{B}$ of $X \cup Y$, where $\mathbb{A} = \{A_1, A_2, \dots\}$ is a partition of X and $\mathbb{B} = \{B_1, B_2, \dots\}$ is a partition of Y . Next we show how a C_4 interacts with the modular partition.

Lemma 7. *Let H be a bipartite tournament, then any C_4 in H has each of its vertices in different modules of \mathcal{P} .*

Proof. Let u and v be any two vertices of a C_4 in H . If u and v are from different partitions of H then by Lemma 1, they cannot be in the same module. And if they are from the same partition, then there is some vertex w from the other partition that comes between them in the cycle as $u \rightarrow w \rightarrow v$, and hence in the outgoing neighbourhood of one but the incoming neighbourhood of the other. So they cannot be in the same module. \square

¹ Proofs of results labelled with \star have been omitted due to lack of space.

The main reduction rule that enables us to obtain an $O(k^3)$ kernel is based on the following crucial lemma. It states that there exists an optimum solution where all arcs between two modules are either a part of the solution or none of them are.

Lemma 8. *Let X_1 and Y_1 be two modules of a bipartite tournament $H(X \cup Y, E)$ such that $X_1 \subseteq X$ and $Y_1 \subseteq Y$ and let $E(X_1, Y_1)$ be the set of arcs between these two modules. Let F be any minimal fas of H . Then there exists an fas F^* such that $|F^*| \leq |F|$ and $E(X_1, Y_1) \subseteq F^*$ or $E(X_1, Y_1) \cap F^* = \emptyset$.*

Proof. Let $X_1 = \{x_1, x_2, \dots, x_r\}$ and $Y_1 = \{y_1, y_2, \dots, y_s\}$. Let $e = (x_1, y_1)$ be an arc of $E(X_1, Y_1)$. We define, what we call a mirroring operation with respect to the arc e , that produces a solution F' that contains all the arcs of the module if e was in F and does not contain any arc of the module if e was not in F ; i.e. the mirroring operation mirrors the intersection of the solution F with the arc e , and the arcs incident on the end points of e , to all arcs of the module. To obtain F^* we will mirror that arc of the module that has the smallest intersection with F among the arcs incident on its end points. We define this operation formally below.

The operation $\text{mirror}(F, e, E(X_1, Y_1))$ returns a subset $F'(e)$ of arcs obtained as follows. Let F_{XY} is the set of all arcs in F which have at least one end point in $X_1 \cup Y_1$. Let $e \in E(X_1, Y_1)$ and define $\text{Ext}(e) = \{f \in F_{XY} \mid f \cap e \neq \emptyset, f \notin E(X_1, Y_1)\}$, i.e. $\text{Ext}(e)$ is the set of all external edges in F_{XY} which are incident on endpoints of e . Let $\text{Ext} = \cup_{i=1}^{rs} \text{Ext}(e_i)$.

$$\begin{aligned} F_1(e) &= \{(x_i, y) \mid (x_1, y) \in \text{Ext}(e)\} \cup \{(y_i, x) \mid (y_1, x) \in \text{Ext}(e)\} \\ &\quad \cup \{(y, x_i) \mid (y, x_1) \in \text{Ext}(e)\} \cup \{(x, y_i) \mid (x, y_1) \in \text{Ext}(e)\}, \\ F_2(e) &= \begin{cases} E(X_1, Y_1) & e \in F \\ \emptyset & e \notin F \end{cases} \end{aligned}$$

For an edge e define, $F'(e) = F - F_{XY} \cup F_1(e) \cup F_2(e)$. We claim that $F'(e)$ for any arc $e \in E(X_1, Y_1)$ is a feedback arc set, and that there exists an $e \in E(X_1, Y_1)$ such that $|F'(e)| \leq |F|$, and we will output F^* as $F'(e)$ for that e .

Claim 1: $F'(e)$ is a feedback arc set for any arc $e \in E(X_1, Y_1)$.

Proof. Suppose not, and let C be a cycle in the graph $H\{F'(e)\}$, and let $e = (x_s, y_t)$ and replace any vertex $x_j, j \neq s$ by x_s and any vertex $y_j, j \neq t$ by y_t in the cycle C to get C' (See Fig. 1). We claim that C' is a closed walk (from which a cycle can be obtained), in $H\{F\}$ contradicting the fact that F was a fas for H .

That C' is a closed walk is clear because for any arc (u, x_j) or (x_j, u) incident on $x_j, j \neq s$, the corresponding arcs (u, x_s) or (x_s, u) exists as X_1 is a module. Similarly for any arc (u, y_j) or (y_j, u) incident on $y_j, j \neq t$, the corresponding arcs (u, y_t) or (y_t, u) exists as Y_1 is a module.

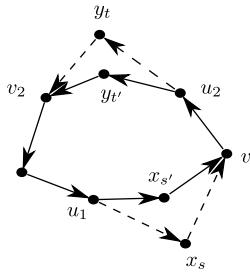


Fig. 1. Constructing a closed walk from the cycle C

Note that C' does not contain arcs incident on $x_j, j \neq s$ or $y_j, j \neq t$ and the only arcs affected by the mirroring operation to obtain F' are the arcs incident on $x_j, j \neq s$ or $y_j, j \neq t$. Hence C' is a walk in $H\{F\}$. This proves Claim 1. \square

Claim 2: There exists an arc $e \in E(X_1, Y_1)$ such that $|F'(e)| \leq |F|$.

Proof. Let e_1, e_2, \dots, e_{rs} be the arcs in $E(X_1, Y_1)$. By the definition of F' , we have

$$\sum_{i=1}^{rs} |F'(e_i)| = (rs)|F| - (rs)|F_{XY}| + \sum_{i=1}^{rs} |F_1(e_i)| + \sum_{i=1}^{rs} |F_2(e_i)|.$$

We argue that the last three terms cancel out. For $x \in X_1$, define $Ext(x) = \{f \in Ext \mid f \cap x \neq \emptyset\}$. Whenever an edge incident on x is mirrored by other arcs, $Ext(x)$ is repeated on each of the r vertices in X_1 and there are s edges in $E(X_1, Y_1)$ which are incident on x . Therefore $Ext(x)$ is repeated (rs) times. Similarly we can define $Ext(y)$ and show that it is repeated (rs) times. Therefore the third term $\sum_{i=1}^{rs} |F_1(e_i)| = rs|Ext|$. And whenever e is in F , it is repeated (rs) times, therefore the last term $\sum_{i=1}^{rs} |F_2(e_i)| = rs|F_{XY} - Ext|$. Therefore the sum of the last two terms is nothing but (rs) times $|F_{XY}|$.

So we have $\sum_{i=1}^{rs} |F'(e_i)| = (rs)|F|$. Thus there is an arc $e \in E(X_1, Y_1)$ such that $|F'(e)| \leq |F|$. Note that to find such an i , we can simply find that i that has the smallest intersection between F and the arcs incident on its end points. \square

This completes the proof of Lemma 8. \square

From this point, we will only consider fas which either contains all the arcs between two modules or contains none of them. An easy consequence of this lemma is that if S is a module of size at least $k+1$, then a fas of size at most k contains no arc incident on S . This is because between any other module R and S there are at least $k+1$ arcs and they all cannot be in the fas.

Reduction Rule 3. In $\mathbb{A} \cup \mathbb{B}$ truncate all modules of size greater than $k+1$ to $k+1$. In other words if S is a module of size more than $k+1$ then delete all but $k+1$ of its vertices arbitrarily.

Lemma 9. *Reduction Rule 3 is sound.*

Proof. Once we have a modular partition, we can see which modules have size $\geq k+1$ and arbitrarily delete vertices from them to reduce their size to $k+1$.

Next we show that the rule is correct. Suppose H is a bipartite tournament and H' is obtained from H by deleting a single vertex v from some large module S which has $\geq k+2$ vertices. Consider a fas F' of H' of size $\leq k$ and let $S' = S \setminus \{v\}$, which is a module in H' . Since $|S'| \geq k+1$, no arc incident on S' is in F' . Now reverse the arcs of F' in H . If F' is not a fas of H then there is a cycle C of length 4 in H . If C does not contain v then it will be present in H' with F' reversed, which is a contradiction. Otherwise the C contains v . Let $u \in S'$ and consider the cycle C' of length 4 obtained from C by replacing v with u . C' is present in H' with F' reversed, which is again a contradiction.

Now we can use induction on the number of vertices deleted from a module, to show that truncating a single module does not change the solution. We can then use induction on the number of truncated modules to show that the rule is correct. \square

4.2 Analysis of Kernel Size

We apply the above Reduction Rules 1, 2 and 3 exhaustively (until no longer possible) and obtain a reduced instance (H', k') of k -FASBT from the initial instance (H, k) . Observe that we will not keep applying these rules indefinitely to an instance because either the size of the graph or the parameter k drops. For brevity we abuse notation and denote the reduced instance also by (H, k) . As before we fix the unique modular partition $\mathcal{P} = \mathbb{A} \cup \mathbb{B}$ of $X \cup Y$, where $\mathbb{A} = \{A_1, A_2, \dots\}$ is a partition of X and $\mathbb{B} = \{B_1, B_2, \dots\}$ is a partition of Y . Let $\mathcal{H}_{\mathcal{P}}$ be the corresponding quotient graph.

The following lemma lists some properties of the quotient graph.

Lemma 10. [★] *Let $\mathcal{H}_{\mathcal{P}}$ be the quotient graph of H . Then the following hold:*

- (a) *Every vertex in $\mathcal{H}_{\mathcal{P}}$ is part of some C_4 .*
- (b) *For each arc $e \in E(\mathcal{H}_{\mathcal{P}})$ there are $\leq k$ C_4 in $\mathcal{H}_{\mathcal{P}}$ which pairwise have e as the only common arc.*
- (c) *If H has a fas of size $\leq k$ then $\mathcal{H}_{\mathcal{P}}$ has a fas of size $\leq k$.*
- (d) *For all $u, v \in V(\mathcal{H}_{\mathcal{P}})$, $N^+(u) \neq N^+(v)$ or equivalently $N^-(u) \neq N^-(v)$.*

Let $\mathcal{H}_{\mathcal{P}}$ be the quotient graph of H . Let F be an fas of $\mathcal{H}_{\mathcal{P}}$ of size at most k . Let T be the topological sort of $\mathcal{H}_{\mathcal{P}}$ obtained after reversing the arcs of F . If we arrange the vertices of $\mathcal{H}_{\mathcal{P}}$ from left to right in order of T , then only the arcs of F go from right to left. For an arc $e \in F$, the span of e is the set of vertices which lie between the endpoints of e in T . We call a vertex v an *affected vertex* if there is some arc in F which is incident on v . Otherwise we call v an *unaffected vertex*.

Lemma 11. *If v is an unaffected vertex, then there exists an arc $e \in F$ such that v is in the span of e .*

Proof. Let v be any unaffected vertex in \mathcal{H}_P . By lemma 10(a) there is a C_4 u, v, w, t which contains v . Since v is unaffected the order $u \rightarrow v \rightarrow w$ is fixed in T . Therefore the arc $t \rightarrow u$ goes from right to left in T . Hence, v is contained in the span of (t, u) . \square

Lemma 12. *Let u and v be two unaffected vertices from the same vertex partition of the bipartite tournament \mathcal{H}_P such that u occurs before v in T . Then there exists w from the other partition which lies between u and v in T .*

Proof. Since \mathcal{H}_P is the quotient graph of H and u and v are different modules, therefore there is some vertex w in the other partition such that $(u, w), (w, v) \in E(\mathcal{H}_P)$. And these two arcs are not in the fas F as u and v are unaffected. Hence, w comes between u and v in T . \square

Lemma 13. *There are at most $2k + 2$ unaffected vertices in the span of any arc $e \in F$.*

Proof. Let the span of e contain $2k + 3$ unaffected vertices. Then without loss of generality assume that $k + 2$ of these vertices come from the partition A . Then there are at least $k + 1$ vertices of partition B which lie between each pair of consecutive vertices of A by Lemma 12. This gives us $k + 1$ C_4 's which pairwise have only e in common. But then the graph H contains an arc e' for which there are $k + 1$ C_4 's which pairwise have only e' in common. This contradicts the fact that H was reduced with respect to Rule 2. \square

Reduction Rule 4. *If \mathcal{H}_P contains more than $2k^2 + 2k$ vertices then return NO.*

Lemma 14. *Reduction rule 4 is sound.*

Proof. By Lemma 10(c), if H has a fas of size $\leq k$, then \mathcal{H}_P has a fas of size $\leq k$. If there are more than $2k^2 + 2k$ vertices in \mathcal{H}_P , then there is some arc e whose span contains more than $2k + 2$ unaffected vertices, this contradicts Lemma 13. \square

Hence, we may assume that \mathcal{H}_P contains $O(k^2)$ vertices.

Lemma 15. *H contains $O(k^3)$ vertices.*

Proof. Each vertex in \mathcal{H}_P corresponds to a module in H and any module in H has size at most $k + 1$ (due to Reduction Rule 3). Hence, there are $O(k^3)$ vertices in H . \square

Lemma 15 implies the following theorem.

Theorem 1. *k -FASBT admits a polynomial kernel with $O(k^3)$ vertices.*

5 Conclusion

In this paper we give a polynomial kernel for k -FASBT with $O(k^3)$ vertices. It is an interesting open problem to obtain a kernel with $O(k)$ or even $O(k^2)$ vertices. Our result adds to a small list of problems for which modular based approach has turned out to be useful. It will be interesting to find more applications of graph modules for kernelization. We also remark that the kernel obtained in the paper generalizes to multi-partite tournaments.

References

1. Abu-Khzam, F.N.: A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences* 76(7), 524–531 (2010)
2. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: ranking and clustering. In: ACM Symposium on Theory of Computing (STOC), pp. 684–693 (2005)
3. Alon, N.: Ranking tournaments. *SIAM J. Discrete Math.* 20(1), 137–142 (2006)
4. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 49–58. Springer, Heidelberg (2009)
5. Bang-Jensen, J., Thomassen, C.: A polynomial algorithm for the 2-path problem for semicomplete digraphs. *SIAM J. Discrete Math.* 5(3), 366–376 (1992)
6. Bessy, S., Fomin, F.V., Gaspers, S., Paul, C., Perez, A., Saurabh, S., Thomassé, S.: Kernels for feedback arc set in tournaments. In: FSTTCS, pp. 37–47 (2009)
7. 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)
8. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) Kernelization. In: FOCS, pp. 629–638 (2009)
9. Borda, J.: Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences* (1781)
10. Cheng Cai, M., Deng, X., Zang, W.: A min-max theorem on feedback vertex sets. *Math. Oper. Res.* 27(2), 361–371 (2002)
11. Charbit, P., Thomassé, S., Yeo, A.: The minimum feedback arc set problem is NP-hard for tournaments. *Combin. Probab. Comput.* 16(1), 1–4 (2007)
12. Cohen, W.W., Schapire, R.E., Singer, Y.: Learning to order things. In: Advances in Neural Information Processing Systems (NIPS), pp. 451–457 (1997)
13. Condorcet, M.: *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix* (1785)
14. Dell, H., van Melkebeek, D.: Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In: STOC, pp. 251–260. ACM (2010)
15. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through Colors and IDs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
16. 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), 76–86 (2010)
17. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: World Wide Web Conference, WWW (2001)

18. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: SODA, pp. 503–510 (2010)
19. Guo, J.: A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.* 410(8-10), 718–726 (2009)
20. Guo, J., Hüffner, F., Moser, H.: Feedback arc set in bipartite tournaments is NP-Complete. *Inf. Process. Lett.* 102(2-3), 62–65 (2007)
21. Gupta, S.: Feedback arc set problem in bipartite tournaments. *Inf. Process. Lett.* 105(4), 150–154 (2008)
22. Habib, M., Paul, C.: A survey of the algorithmic aspects of modular decomposition. *Computer Science Review* 4(1), 41–59 (2010)
23. Karpinski, M., Schudy, W.: Faster algorithms for feedback arc set tournament, kemeny rank aggregation and betweenness tournament. CoRR abs/1006.4396 (2010)
24. Kemeny, J.: Mathematics without numbers. *Daedalus* 88, 571–591 (1959)
25. Kemeny, J., Snell, J.: Mathematical models in the social sciences. Blaisdell (1962)
26. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: ACM Symposium on Theory of Computing (STOC), pp. 95–103 (2007)
27. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. *Theor. Comput. Sci.* 351(3), 446–458 (2006)
28. Sanghvi, B., Koul, N., Honavar, V.: Identifying and Eliminating Inconsistencies in Mappings Across Hierarchical Ontologies. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6427, pp. 999–1008. Springer, Heidelberg (2010)
29. Speckenmeyer, E.: On Feedback Problems in Digraphs. In: Nagl, M. (ed.) WG 1989. LNCS, vol. 411, pp. 218–231. Springer, Heidelberg (1990)
30. Tedder, M., Corneil, D.G., Habib, M., Paul, C.: Simpler Linear-Time Modular Decomposition Via Recursive Factorizing Permutations. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 634–645. Springer, Heidelberg (2008)
31. Thomassé, S.: A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms* 6(2) (2010)
32. van Zuylen, A., Hegde, R., Jain, K., Williamson, D.P.: Deterministic pivoting algorithms for constrained ranking and clustering problems. In: ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 405–414 (2007)
33. van Zuylen, A.: Linear programming based approximation algorithms for feedback set problems in bipartite tournaments. *Theor. Comput. Sci.* 412(23), 2556–2561 (2011)

Fixed-Parameter Complexity of Feedback Vertex Set in Bipartite Tournaments

Sheng-Ying Hsiao

Department of Computer Science and Information Engineering,
National Taiwan University
d97033@csie.ntu.edu.tw

Abstract. Let G be an n -node bipartite tournament, i.e., a complete bipartite graph, each of whose edges has an orientation. We address the fixed-parameter complexity of the NP-complete problem of determining, for any given parameter k , whether G admits a k -node subset whose removal from G yields an acyclic graph. The best previously known upper bound, due to Sasatte, is $O(3^k \cdot \text{poly}(n))$. In this paper, we show that the fixed-parameter complexity is $O(2^k \cdot \text{poly}(n))$.

1 Introduction

Let G be an n -node directed graph. A node subset F of G is a *feedback (vertex) set* for G if $G \setminus F$ is acyclic. Figure 1(a) illustrates a bipartite tournament G and a subset F of G . The node subset F is a feedback set for G because $G \setminus F$ is acyclic as shown in Figure 1(b). We address the fixed-parameter complexity of the classic NP-complete problem [13] of determining, for any given parameter k , whether G admits a k -node feedback set. Chen, Liu, Lu, O’Sullivan, and Razgon [5] settled the long-standing open question regarding the fixed-parameter tractability of the problem by giving an $O(4^k k! \cdot k^3 n^4)$ -time algorithm, which remains the state-of-the-art for general G . The problem is NP-complete even if G is a *tournament* [18] (respectively, *bipartite tournament* [3]), that is, G is obtained by assigning a direction to each edge of an undirected complete (respectively, complete bipartite) graph. For the case that G is a tournament, it can be reduced to the 3-hitting set problem, which can be solved in $O(2.076^k \cdot \text{poly}(n))$ time by Wahlström [22]. For the case that G is a bipartite tournament, Sasatte [17] showed that the problem can be solved in $O(3^k \cdot n^2 + n^3)$ time. In this paper, we prove a lower fixed-parameter complexity of the problem for bipartite tournaments in terms of k , as summarized in the following theorem.

Theorem 1. *Given an n -node bipartite tournament G and a positive integer k , it takes $O(2^k \cdot n^6)$ time to either output a k -node feedback set of G or report that G does not admit any k -node feedback set.*

For the undirected version of the problem for general G , there has been a series of improvements on its fixed-parameter complexity. Cao, Chen, and Liu [4] gave the best deterministic algorithm running in $O(3.83^k \cdot kn^2)$ time. The best

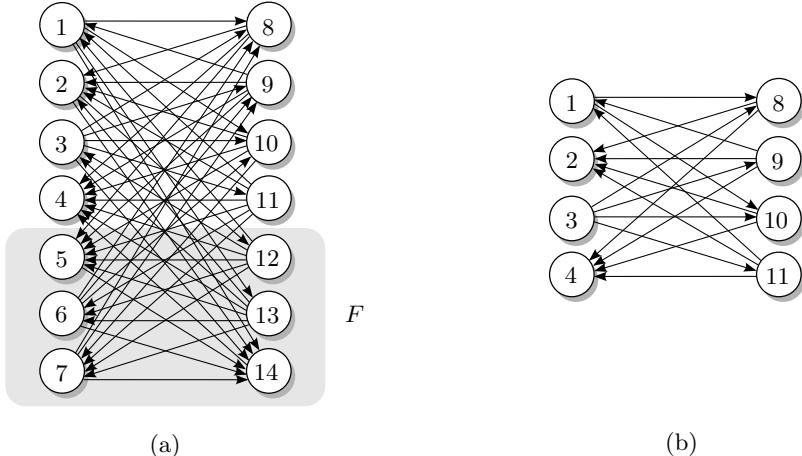


Fig. 1. (a) A bipartite tournament G . (b) $G \setminus F$ is acyclic.

currently known upper bound by the randomized algorithm, due to Cygan, Nederlof, Marcin Pilipczuk, Michał Pilipczuk, van Rooij, and Wojtaszczyk [6], is $O(3^k \cdot \text{poly}(n))$. If G has both directed and undirected edges, Bonsma and Lokshtanov [1] showed that the fixed-parameter complexity of the problem is $O(47.5^k k! \cdot O(n^4))$. For the weighted version of the problem for tournaments, Raman and Saurabh [15] gave a fixed-parameter algorithm running in $O(2.415^k \cdot \text{poly}(n))$ time. The complexity was reduced to $O(2^k \cdot n^2(\log \log n + k))$ by Dom, Guo, Hüffner, Niedermeier, and Truß [7]. The result of Sasatte [17] on bipartite tournaments also allows integral node weights. For the optimization version of the problem, the best currently known approximation ratios are 2 for bipartite tournaments due to van Zuylen [19], 2.5 for tournaments due to Cai, Deng, and Zang [2], and $O(\log k^* \log \log k^*)$ for general G due to Even, Naor, Schieber, and Sudan [8], where k^* is the smallest cardinality of feedback sets of G . See [20, 21, 9, 14] for related work on tournaments and [16, 11, 10] for related work on bipartite tournaments.

The rest of the paper is organized as follows. Section 2 gives the preliminaries, which includes a reduction from the theorem to the main lemma via iterative compression. Section 3 shows the framework of our proof of the main lemma. The details of the framework will appear in the full version of this paper.

2 Preliminaries

2.1 Canonical Sequence for Acyclic Bipartite Tournament

For any sequence T , let $|T|$ denote the length of T . For each $i = 1, 2, \dots, |T|$, let T_i be the i -th element of T . Let G be an n -node acyclic bipartite tournament. The *canonical sequence* for G is the sequence T of node sets that can be obtained from

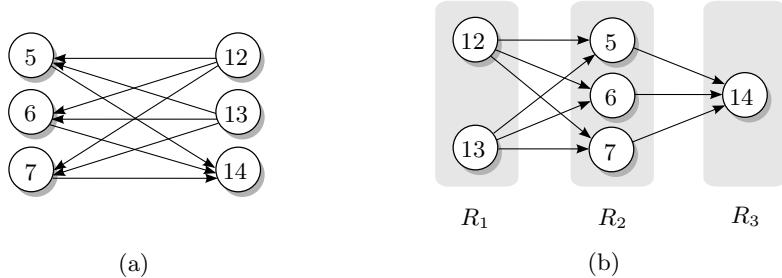


Fig. 2. (a) An acyclic bipartite tournament H . (b) The canonical sequence (R_1, R_2, R_3) for H .

G in $O(n^2)$ time as follows. For each $i \geq 1$, let T_i consist of the nodes without incoming edges in $G \setminus \bigcup_{j=1}^{i-1} T_j$. Let $|T|$ be the smallest integer i with $\bigcup_{j=1}^i T_j = V(G)$. For instance, given an acyclic bipartite tournament H in Figure 2(a), the sequence (R_1, R_2, R_3) of node sets in Figure 2(b) is the canonical sequence for H .

Lemma 1. *Let G be an n -node acyclic bipartite tournament. Let T be the canonical sequence for G . The following statements hold.*

- (1) $T_1, T_2, \dots, T_{|T|}$ form a partition of $V(G)$.
- (2) For each directed edge (u, v) of G , the node set T_i containing u precedes the node set T_j containing v in the sequence (i.e., $i < j$).
- (3) $\bigcup_{i \text{ is odd}} T_i$ and $\bigcup_{i \text{ is even}} T_i$ are the partite sets of G .

Proof. The first two statements are straightforward. Observe that for each $i = 1, 2, \dots, |T| - 1$, there is an arc (u, v) of G with $u \in T_i$ and $v \in T_{i+1}$. Thus, the third statement can be proved by verifying that any two distinct nodes u and v of G are in the same partite set if and only if neither (u, v) nor (v, u) is an arc of G .

2.2 Allying Sequence of Node Sets and Increasing Subsequence of Node Sets

Let X and X' be two sequences of node sets. We say that X' allies with X if there exists a function $i : \{0, 1, \dots, |X'|\} \rightarrow \{0, 1, \dots, |X|\}$ with

- $0 = i(0) < i(1) < \dots < i(|X'|) = |X|$ and
- $i(j')$ is even if and only if j' is even for all $0 \leq j' \leq |X'|$

such that $X'_j = X_{i(j-1)+1} \cup X_{i(j-1)+3} \cup \dots \cup X_{i(j)-2} \cup X_{i(j)}$ holds for each $j = 1, 2, \dots, |X'|$. For instance, if X and X' are as shown in Figures 3(a) and 3(c), then X' allies with X , where $i(0) = 0$, $i(1) = 3$, $i(2) = 4$, and $i(3) = 7$.

Let Y be a sequence of node sets. We say that \preceq is a *quasi-order* for Y if \preceq is a reflexive and transitive binary relation defined for the elements of Y .

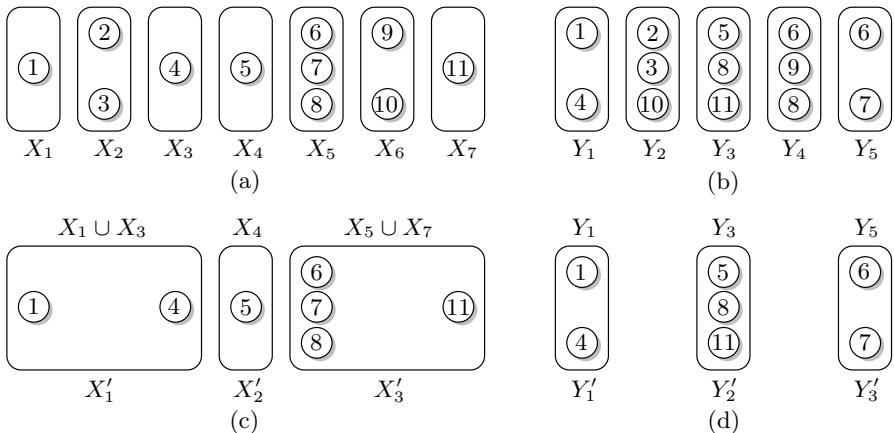


Fig. 3. (a) A sequence X of node sets. (b) A sequence Y of node sets. (c) A sequence X' of node sets that allies with X . (d) A subsequence Y' of Y .

Let \preceq be a quasi-order for Y . A subsequence Y' of Y *increases* with respect to \preceq if $Y'_1 \preceq Y'_2 \preceq \cdots \preceq Y'_{|Y'|}$. For instance, if Y and Y' are as shown in Figures 3(b) and 3(d) and $Y_1 \preceq Y_3 \preceq Y_5$, then Y' increases with respect to \preceq .

2.3 Consistent Node Sequence and Compatible Node Sequence

A node sequence x is *consistent* with a sequence X of node sets if

- each pair of nodes in x are distinct, and
 - there exists a function $i : \{1, 2, \dots, |x|\} \rightarrow \{1, 2, \dots, |X|\}$ with $i(1) \leq i(2) \leq \dots \leq i(|x|)$ such that $x_j \in X_{i(j)}$ holds for each $j = 1, 2, \dots, |x|$.

A node sequence x is *compatible* with two sequences X and Y of node sets with respect to a quasi-order \preceq for Y if there exist

- a sequence X' of node sets that allies with X , and
 - a subsequence Y' of Y that increases with respect to \preceq

such that x is consistent with both X' and Y' . For instance, $(1, 4, 5, 8, 11, 6, 7)$ is a node sequence compatible with X and Y in Figures 3(a) and 3(b) with respect to any quasi-order \preceq for Y with $Y_1 \preceq Y_3 \preceq Y_5$.

2.4 A Reduction

Theorem 1 can be proved via the following main lemma using iterative compression [5,7]. Let $\|S\|$ denote the cardinality of a node set S .

Lemma 2. Let H be an r -node bipartite tournament. Let ℓ be an integer. Given a feedback set F of H , it takes $O(r^5)$ time to either output an ℓ -node feedback set F^* of H with $F \cap F^* = \emptyset$ or report that H does not admit such an F^* .

For any node subset V of G , let $G[V]$ denote the subgraph of G induced by V . For instance, F is a node subset of G in Figure 4(a). The subgraph $G[F]$ of G which is induced by F is illustrated in Figure 4(a).

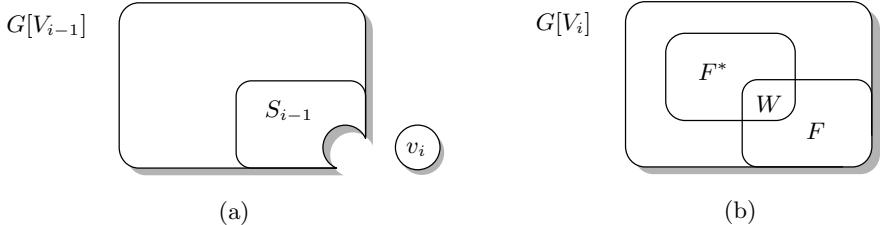


Fig. 4. (a) A subgraph $G[V_{i-1}]$ of a bipartite tournament G , a node subset S_{i-1} of V_{i-1} , and a node v_i of $G \setminus V_{i-1}$. (b) A subgraph $G[V_i]$ and three node subsets F^* , W , and F of V_i , where $V_i = V_{i-1} \cup \{v_i\}$ and $W \cup F = S_{i-1} \cup \{v_i\}$.

Proof (of Theorem 1). For each $i = 1, 2, \dots, n$, let v_i be the i -th node of G . Let $V_i = \{v_1, \dots, v_i\}$. Let $S_k = V_k$. For each index i with $k+1 \leq i \leq n$, if S_{i-1} is a k -node feedback set of $G[V_{i-1}]$ (see Figure 4(a)), we show that it takes $O(2^k \cdot n^5)$ time to either output a k -node feedback set S_i of $G[V_i]$ or report that $G[V_i]$ does not admit any k -node feedback set: $S = S_{i-1} \cup \{v_i\}$ is a $(k+1)$ -node feedback set of $G[V_i]$ (see Figure 4(a),(b)). For any subset S_i of V_i , if $W = S \cap S_i$, $F^* = S_i \setminus W$, and $F = S \setminus W$, then $F^* \cap F = \emptyset$. Moreover, S_i is a k -node feedback set of $G[V_i]$ if and only if F^* is a $(k - \|W\|)$ -node feedback set of $G[V_i \setminus W]$. For each subset W of S , $F = S \setminus W$ is a feedback set of $H = G[V_i \setminus W]$ (see Figure 4(b)). Let $\ell = k - \|W\|$. By Lemma 2, it takes $O(n^5)$ time to determine whether H admits an ℓ -node feedback set F^* with $F \cap F^* = \emptyset$. Since S has 2^{k+1} subsets, it takes $O(2^k \cdot n^5)$ time to determine whether there exists a subset W of S such that $G[V_i \setminus W]$ admits a $(k - \|W\|)$ -node feedback set F^* disjoint from $S \setminus W$. If such an W exists, then $S_i = F^* \cup W$ is a k -node feedback set of $G[V_i]$; otherwise, $G[V_i]$ does not admit any k -node feedback set.

If there is an index i with $k+1 \leq i \leq n$ such that $G[V_i]$ does not admit any k -node feedback set, we report that G does not admit any k -node feedback set; otherwise, we output S_n . The overall running time is $O(2^k \cdot n^6)$.

The rest of the paper proves Lemma 2

3 Framework of Our Proof for the Main Lemma

3.1 Good Sequence

Let G be a bipartite tournament. Let F be a subset of $V(G)$. We say that node v is *reachable* from node u via F if there is a sequence x of nodes in F with $|x| \geq 1$

such that $(u, x_1, x_2, \dots, x_{|x|}, v)$ is a directed path of G . Let Y be a sequence of node sets such that $\bigcup_{i=1}^{|Y|} Y_i = V(G) \setminus F$. Let \preceq be a quasi-order for Y . We say that Y is *good* for F with respect to \preceq if the following statements hold for any two distinct nodes u and v in $G \setminus F$.

Case 1: u and v are reachable from each other via F .

There exist exactly one index i (respectively, j) such that $u \in Y_i$ (respectively, $v \in Y_j$) with $i \neq j$. Moreover, $i < j$ implies $Y_i \not\preceq Y_j$ and $j < i$ implies $Y_j \not\preceq Y_i$.

Case 2: v is not reachable from u via F and u is not reachable from v via F .

$\{u, v\} \subseteq Y_i$ holds for some $1 \leq i \leq |Y|$.

Case 3: v is reachable from u via F and u is not reachable from v via F .

$i < j$ and $Y_i \preceq Y_j$ hold for any Y_i containing u and any Y_j containing v .

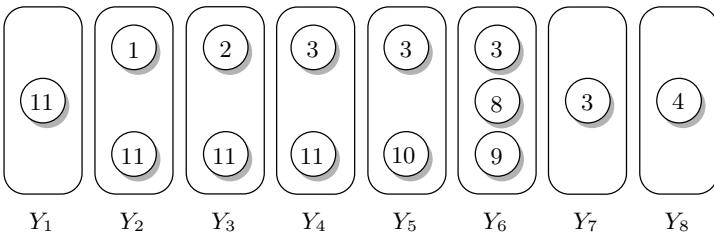


Fig. 5. A sequence Y of node sets whose nodes are in $V(G) \setminus F$, where G is a bipartite tournament and F is a feedback set for G as in Figure 1(a)

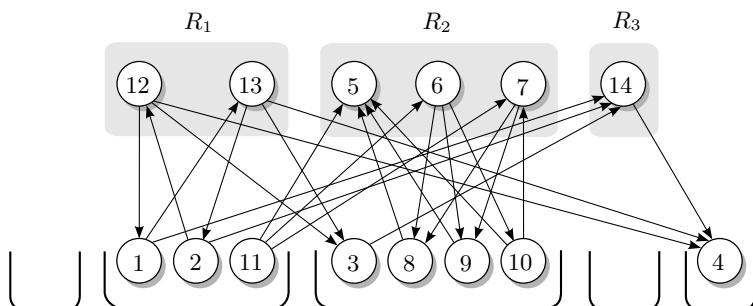


Fig. 6. All nodes of G and all the edges which connect the nodes of F and the nodes of $V(G) \setminus F$, where G is a bipartite tournament and F is a feedback set for G as in Figure 1(a). (R_1, R_2, R_3) is the canonical sequence for $G[F]$ as shown in Figure 2(b).

For instance, let G be a bipartite tournament and F be a subset of $V(G)$ as shown in Figure 1(a). Let Y be a sequence of node sets such that $\bigcup_{i=1}^{|Y|} Y_i = V(G) \setminus F$ as shown in Figure 5. If \preceq is a quasi-order for Y with $Y_1 \preceq Y_2 \preceq Y_4 \preceq Y_5 \preceq Y_6 \preceq Y_7 \preceq Y_8$, $Y_1 \preceq Y_3 \preceq Y_4$, and $Y_2 \not\preceq Y_3$, then Y is good for F with respect to \preceq . We verify the three cases for some pairs of nodes as follows. The bipartite tournament H in Figure 2(a) is $G[F]$. Figure 2(b) arranges $G[F]$ with arcs starting at left and ending at right. Figure 6 illustrates the edges connection between F and $V(G) \setminus F$. By observing Figure 6, nodes 1, 2 are reachable from each other via F . Since $Y_2 \not\preceq Y_3$, case 1 holds for nodes 1, 2. By observing Figure 2(b) and Figure 6, node 3 is not reachable from node 11 via F and node 11 is not reachable from node 3 via F . Since $\{3, 11\} \subseteq Y_4$, case 2 holds for nodes 3, 11. By observing Figure 2(b) and Figure 6, node 2 is not reachable from node 8 via F and node 8 is reachable from node 2 via F because of the directed path $(2, 12, 6, 8)$. Since $Y_3 \preceq Y_6$, case 3 holds for nodes 2, 8.

Let Y be a sequence of node sets. We say that Y is *successive* if $v \in \bigcap_{i=j_1}^{j_2} Y_i$ holds for each $v \in Y_{j_1} \cap Y_{j_2}$ with $j_1 \leq j_2$. For instance, the sequence Y of node sets in Figure 5 is successive. One can observe that $3 \in Y_4 \cap Y_5 \cap Y_6 \cap Y_7$ and $11 \in Y_1 \cap Y_2 \cap Y_3 \cap Y_4$.

3.2 Framework for Proving the Main Lemma

The framework of our proof for proving Lemma 2 is inspired by the algorithm of computing longest common subsequences for solving the feedback set problem in tournaments [7].

Lemma 3. *Let G be an n -node bipartite tournament. Let F be a feedback set for G such that each cycle of G contains at least two nodes of $G \setminus F$. It takes $O(n^3)$ time to construct a successive sequence Y of node sets with a quasi-order \preceq for Y such that Y is good for F with respect to \preceq and $|Y| = O(n)$.*

Lemma 4. *Let X be the canonical sequence for an acyclic bipartite tournament and Y be a successive sequence of node sets. Let \preceq be a quasi-order for Y . If $|X| = O(n)$, $|Y| = O(n)$, $\|\bigcup_{i=1}^{|X|} X_i\| = O(n)$, and $\|\bigcup_{i=1}^{|Y|} Y_i\| = O(n)$, then it takes $O(n^5)$ time to compute a longest node sequence that is compatible with X and Y with respect to \preceq for Y .*

Lemma 5. *Let G be a bipartite tournament. Let F be a feedback set for G such that each cycle of G contains at least two nodes of $G \setminus F$. Let X be the canonical sequence for $G \setminus F$. Let Y be a successive sequence of node sets and \preceq be a quasi-order for Y such that Y is good for F with respect to \preceq . If x is a longest node sequence that is compatible with X and Y with respect to \preceq for Y , then $V(G) \setminus (F \cup x)$ is a minimum feedback set for G that is disjoint from F .*

The proofs of Lemmas 3 and 5 will be given in our full paper. To prove Lemma 2, we give an algorithm to compute a maximum acyclic subgraph of H which includes F by computing a longest topological sort. This topological sort is actually a linear ordering of the node set $F \cup x$ as in Lemma 5.

Proof (of Lemma 2). It takes $O(r^2)$ time to determine whether $H[F]$ is cyclic. If $H[F]$ is cyclic, then we report that H does not admit the required F^* . The rest of the proof assumes that $H[F]$ is acyclic. Let S consist of the nodes u in $V(H) \setminus F$ such that there exists a cycle C of H with $u \in V(C)$ and $V(C) \setminus \{u\} \subseteq F$. Computing S spends $O(r^3)$ time.

Let $G = H \setminus S$. F is a feedback set for G such that each cycle of G contains at least two nodes of $G \setminus F$. It takes $O(r^2)$ time to obtain the canonical sequence X for $G \setminus F$. By Lemma 3, it takes $O(r^3)$ time to obtain a successive sequence Y of node sets with a quasi-order \preceq for Y such that Y is good for F with respect to \preceq and $|Y| = O(r)$. Since Y is good for F with respect to \preceq , $\|\bigcup_{i=1}^{|Y|} Y_i\| = \|V(G) \setminus F\| = O(r)$. By Lemma 10, $|X| = O(r)$ and $\|\bigcup_{i=1}^{|X|} X_i\| = O(r)$. By Lemma 4, we can compute a longest compatible node sequence x of X and Y with respect to \preceq for Y in $O(r^5)$ time. By Lemma 5, $F' = V(G) \setminus (F \cup x)$ is a minimum feedback set for G disjoint from F . Therefore, $F' \cup S$ is a minimum feedback set for H disjoint from F .

Let $F^* = F' \cup S$. If $\|F^*\| \leq \ell$ and there are sufficient nodes of $V(H) \setminus (F \cup F^*)$ to add them into F^* such that $\|F^*\| = \ell$, then F^* is the desired feedback set. Otherwise, we report that the required F^* does not exist. The overall running time is $O(r^5)$.

3.3 Dynamic Programming for Proving Lemma 4

We give a dynamic programming algorithm for computing the longest compatible sequence. The idea is inspired by the algorithm for the set-set LCS problem [12].

Let ε denote the empty node sequence. Let S be a node set and x be a node sequence. We define $S - x = S \setminus \{v \mid v \text{ is a node of } x\}$. We say x is a *permutation* of S if x consists of all the nodes of S and each node appears exactly once in x . We define $per(S)$ to be the node sequence that is the first permutation of S in alphabetical order. Let Q be a set of node sequences. We define $\max(Q)$ to be the longest node sequence in Q . If there is more than one longest node sequence, we select the first longest node sequence in alphabetical order.

Given the canonical sequence X for an acyclic bipartite tournament and a successive sequence Y of node sets with a quasi-order \preceq for Y , we let $X_0 = Y_0 = \emptyset$, and let $Y_0 \preceq Y_i$ for each $i = 0, 1, \dots, |Y|$. We define the function $comp$ as follows.

- $comp(0, j, 0) = \varepsilon$, for all $1 \leq j \leq |Y|$.
- $comp(i, 0, h) = \varepsilon$, for all $1 \leq i \leq |X|$ and $0 \leq h \leq \lfloor \frac{i-1}{2} \rfloor$.
- $comp(i, j, h) = \max(Q_X \cup Q_Y)$, for all $1 \leq i \leq |X|$, $1 \leq j \leq |Y|$, and $0 \leq h \leq \lfloor \frac{i-1}{2} \rfloor$, where

$$\begin{aligned} Q_X &= \{x \bullet per(X^* \cap (Y_j - x)) \mid \\ &x = comp(i - 2h - 1, j, t), 0 \leq t \leq \left\lfloor \frac{i-2h-2}{2} \right\rfloor\}, \end{aligned}$$

$$\begin{aligned} Q_Y = \{x \bullet per((X^* - x) \cap Y_j) \mid \\ x = comp(i, t, h), 0 \leq t \leq j-1, \text{ and } Y_t \preceq Y_j\}, \\ \text{and } X^* = X_{i-2h} \cup X_{i-2h+2} \cup \dots \cup X_i. \end{aligned}$$

Lemma 6. Let X be the canonical sequence for an acyclic bipartite tournament and Y be a successive sequence of node sets. Let \preceq be a quasi-order for Y . Given integers i, j, h such that $comp(i, j, h)$ is defined, $comp(i, j, h)$ is the longest node sequence among all node sequences that are consistent with some sequence X' allying with (X_1, X_2, \dots, X_i) and some subsequence Y' of (Y_1, Y_2, \dots, Y_j) increasing with respect to \preceq where $X'_{|X'|} = X_{i-2h} \cup X_{i-2h+2} \cup \dots \cup X_i$ and $Y'_{|Y'|} = Y_j$.

The proof of Lemma 6 will be given in our full paper.

Proof (of Lemma 6). We compute the function $comp$ with respect to X and Y with \preceq for Y . Given $1 \leq j \leq |Y|$ and $0 \leq h \leq \lfloor \frac{|X|-1}{2} \rfloor$, by Lemma 6, the node sequence $comp(|X|, j, h)$ is the longest sequence among all sequences that are consistent with some sequence X' allying with X and some subsequence Y' of (Y_1, Y_2, \dots, Y_j) increasing with respect to \preceq where $X'_{|X'|} = X_{|X|-2h} \cup X_{|X|-2h+2} \cup \dots \cup X_{|X|}$ and $Y'_{|Y'|} = Y_j$. Thus, the desired longest compatible sequence of X and Y with respect to \preceq for Y is the longest sequence among all $comp(|X|, j, h)$ with $1 \leq j \leq |Y|$ and $0 \leq h \leq \lfloor \frac{|X|-1}{2} \rfloor$.

The function $comp$ can be computed by dynamic programming. It takes $O(n^2)$ time to set all $comp(i, 0, h)$ with $1 \leq i \leq |X|$, $0 \leq h \leq \lfloor \frac{i-1}{2} \rfloor$ and all $comp(0, j, 0)$ with $1 \leq j \leq |Y|$ to be empty node sequence. For any indices i, j, h with $1 \leq i \leq |X|$, $1 \leq j \leq |Y|$, and $0 \leq h \leq \lfloor \frac{i-1}{2} \rfloor$, computing a node sequence of $Q_X \cup Q_Y$ spends $O(\|\bigcup_{i=1}^{|X|} X_i\| + \|\bigcup_{i=1}^{|Y|} Y_i\|)$ time. Hence, computing $comp(i, j, h)$ takes $O(n^2)$ time. Thus, the overall running time for computing $comp$ is $O(n^2) + O(n^2) \cdot O(|X|^2 \cdot |Y|) = O(n^5)$.

References

1. Bonsma, P., Lokshtanov, D.: Feedback Vertex Set in Mixed Graphs. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 122–133. Springer, Heidelberg (2011)
2. Cai, M.-C., Deng, X., Zang, W.: An approximation algorithm for feedback vertex sets in tournaments. SIAM Journal on Computing 30(6), 1993–2007 (2001)
3. Cai, M.-C., Deng, X., Zang, W.: A min-max theorem on feedback vertex sets. Mathematics of Operations Research 27(2), 361–371 (2002)
4. Cao, Y., Chen, J., Liu, Y.: On Feedback Vertex Set New Measure and New Structures. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 93–104. Springer, Heidelberg (2010)
5. Chen, J., Liu, Y., Lu, S., O’Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. Journal of the ACM 55(5), 1–19 (2008)

6. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (to appear, 2011)
7. Dom, M., Guo, J., Hüffner, F., Niedermeier, R., Truss, A.: Fixed-parameter tractability results for feedback set problems in tournaments. *Journal of Discrete Algorithms* 8(1), 76–86 (2010)
8. Even, G., Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica* 20(2), 151–174 (1998)
9. Gaspers, S., Mnich, M.: Feedback vertex sets in tournaments. In: de Berg, M., Meyer, U. (eds.) *Proceedings of the 18th Annual European Symposium on Algorithms*, pp. 267–277 (2010)
10. Guo, J., Hüffner, F., Moser, H.: Feedback arc set in bipartite tournaments is NP-complete. *Information Processing Letters* 102(2-3), 62–65 (2007)
11. Gupta, S.: Feedback arc set problem in bipartite tournaments. *Information Processing Letters* 105(4), 150–154 (2008)
12. Hirschberg, D.S., Larmore, L.L.: The set-set LCS problem. *Algorithmica* 4(4), 503–510 (1989)
13. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) *Complexity of Computer Computations*, pp. 85–103. Plenum Press (1972)
14. Karpinski, M., Schudy, W.: Faster Algorithms for Feedback Arc Set Tournament, Kemeny Rank Aggregation and Betweenness Tournament. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) *ISAAC 2010. LNCS*, vol. 6506, pp. 3–14. Springer, Heidelberg (2010)
15. Raman, V., Saurabh, S.: Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science* 351(3), 446–458 (2006)
16. Sasatte, P.: Improved approximation algorithm for the feedback set problem in a bipartite tournament. *Operations Research Letters* 36(5), 602–604 (2008)
17. Sasatte, P.: Improved FPT algorithm for feedback vertex set problem in bipartite tournament. *Information Processing Letters* 105(3), 79–82 (2008)
18. Speckenmeyer, E.: On Feedback Problems in Digraphs. In: Nagl, M. (ed.) *WG 1989. LNCS*, vol. 411, pp. 218–231. Springer, Heidelberg (1990)
19. van Zuylen, A.: Linear programming based approximation algorithms for feedback set problems in bipartite tournaments. *Theoretical Computer Science* 412(23), 2556–2561 (2011)
20. van Zuylen, A., Hegde, R., Jain, K., Williamson, D.P.: Deterministic pivoting algorithms for constrained ranking and clustering problems. In: *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 405–414 (2007)
21. van Zuylen, A., Williamson, D.P.: Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research* 34(3), 594–620 (2009)
22. Wahlström, M.: Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems. PhD thesis, Department of Computer and Information Science, Linköpings universitet (2007)

Parameterized Algorithms for Inclusion of Linear Matchings

Sylvain Guillemot

Department of Computer Science, Iowa State University, Ames, IA 50011, USA
`guillemo@free.fr`

Abstract. A *linear matching* consists of $2n$ vertices ordered linearly, together with n vertex-disjoint edges. In this article, we study the LINEAR MATCHING INCLUSION problem, which takes two linear matchings, called the *pattern* and the *target*, and asks if there is an order-preserving mapping of the pattern into the target. We consider several parameterizations of this problem, for which we obtain parameterized algorithms and hardness results. In addition, we settle the parameterized complexity of the related NESTING-FREE 2-INTERVAL PATTERN problem.

1 Introduction

A *linear matching* consists of $2n$ vertices ordered linearly, together with n vertex-disjoint edges. In this article, we study the LINEAR MATCHING INCLUSION problem, where we have to decide if a given linear matching (the *pattern*) is a substructure of another linear matching (the *target*). This problem is a special case of the ARC-PRESERVING SUBSEQUENCE problem [3], for sequences defined over a unary alphabet. The introduction of the ARC-PRESERVING SUBSEQUENCE problem was motivated by applications in computational biology, where arc-annotated sequences are used to represent the secondary structure of RNA. While the subsequence problem for strings is polynomial, the ARC-PRESERVING SUBSEQUENCE problem is NP-hard [3], and it appears that its hardness stems from the arc structures rather than the sequences themselves. This leads us to consider the LINEAR MATCHING INCLUSION problem, as it retains the hardness of ARC-PRESERVING SUBSEQUENCE, and since we can expect that algorithms for the former will extend to the latter. We point out that LINEAR MATCHING INCLUSION is NP-hard, even when the pattern has no nested arcs [8].

This article investigates the parameterized complexity of the LINEAR MATCHING INCLUSION problem. Parameterized complexity theory [2,5] aims at obtaining algorithms for NP-hard problems which will be practical for small values of a problem-dependent parameter. An algorithm is *fixed-parameter tractable* (FPT) if its running time is bounded by $f(k)n^c$, where n is the input size and k is the *parameter* associated with the input. We study various parameterizations of the LINEAR MATCHING INCLUSION problem, obtaining FPT algorithms as well as hardness results. Two natural parameters are the lengths of the pattern and the target, which we denote by k, n respectively. In addition, we consider the nesting depth of the

target, denoted by d , as well as the difference of lengths between the target and the pattern, denoted by p . Our results are as follows. We obtain an FPT algorithm for the joint parameters (d, k) (Section 3.1) and an FPT algorithm for the parameter p (Section 3.2). We also obtain dynamic-programming algorithms when the pattern is nesting-free (Section 3.3). Furthermore, we show that the problem is unlikely to admit an FPT algorithm for the single parameter k , by proving its hardness for the class W[1] (Section 4.1). Finally, we settle the parameterized complexity of the related NESTING-FREE 2-INTERVAL PATTERN problem introduced by [10], by proving its hardness for the class W[1] (Section 4.2).

2 Problem Definition

A *linear matching* (of length n) is a graph $M = (V_M, E_M)$, where $V_M = [2n]$ and E_M is a partition of V_M in classes of size two. If $e \in E$ has the form $\{i, j\}$ with $i < j$, we denote it as $\langle i, j \rangle$. We introduce the precedence / crossing / nesting relations on edges. Given $e = \langle i, j \rangle, e' = \langle i', j' \rangle \in E$, we say that:

- e precedes e' (denoted $e \prec e'$) iff $i < j < i' < j'$;
- e crosses e' (denoted $e \between e'$) iff $i < i' < j < j'$;
- e nests in e' (denoted $e \sqsubset e'$) iff $i' < i < j < j'$.

A set $S \subseteq E$ is called *nesting-free* iff there are no $e, e' \in S$ s.t. $e \sqsubset e'$. The *nesting depth* of M , denoted by $d(M)$, is the size of a longest chain in (E_M, \sqsubset) .

Let $P = (V_P, E_P)$ and $T = (V_T, E_T)$ be two linear matchings. An *embedding* of P into T is a strictly increasing function $\phi : V_P \rightarrow V_T$ such that: if $\langle i, j \rangle \in E_P$ then $\langle \phi(i), \phi(j) \rangle \in E_T$. When ϕ is an embedding of P into T , we extend it to a function $\phi : E_P \rightarrow E_T$ such that: if $e = \langle i, j \rangle$, then $\phi(e) := \langle \phi(i), \phi(j) \rangle$. Also, given a function $\phi : E_P \rightarrow E_T$, it induces a function $\phi : V_P \rightarrow V_T$ such that whenever $\phi(\langle a, b \rangle) = \langle c, d \rangle$, we have $\phi(a) = c$ and $\phi(b) = d$.

We say that P occurs in T iff there is an embedding of P into T . We consider the following problem:

Name: LINEAR MATCHING INCLUSION

Input: Two linear matchings $P = (V_P, E_P)$ and $T = (V_T, E_T)$.

Question: Does P occur in T ?

P is called the *pattern* and T is called the *target*. We define the following parameters for the LINEAR MATCHING INCLUSION problem: the length k of the pattern, the length n of the target, the nesting depth d of the target, the size excess $p = |V_T| - |V_P|$.

3 Algorithms for Linear Matching Inclusion

3.1 An FPT Algorithm for Parameters d and k

In this section, we demonstrate the fixed-parameter tractability of LINEAR MATCHING INCLUSION for parameters d and k . We show the following theorem.

Theorem 1. LINEAR MATCHING INCLUSION can be solved in $O(d^k kn)$ time.

To prove Theorem 1, we introduce the following constrained version of LINEAR MATCHING INCLUSION.

Name: CONSTRAINED LINEAR MATCHING INCLUSION

Input: A linear matching $P = (V_P, E_P)$, a linear matching $T = (V_T, E_T)$, a family S of sets $S(x)$ ($x \in E_P$) such that each set is a nesting-free subset of E_T .

Question: Is there an embedding ϕ of P into T such that: for every $x \in E_P$, $\phi(x) \in S(x)$?

We first establish the polynomial-time solvability of CONSTRAINED LINEAR MATCHING INCLUSION.

Lemma 1. CONSTRAINED LINEAR MATCHING INCLUSION can be solved in $O(kn)$ time.

Proof. Consider a mapping $\phi : E_P \rightarrow E_T$. We say that $i \in V_P$ is a *bad position* w.r.t. ϕ if $\phi(i - 1) \geq \phi(i)$, otherwise we say that i is a *good position* w.r.t. ϕ . The algorithm maintains the following elements:

- a mapping $\phi : E_P \rightarrow E_T$,
- a doubly linked list Bad containing the bad positions w.r.t. ϕ ,
- for each $i \in V_P$, a pointer p_i to the cell of Bad containing i , or \perp if there is no such cell.

We use a subroutine $UPDATE(i)$ which proceeds as follows. Suppose that i is an endpoint of $e \in E_P$. If $\phi(e)$ is the last edge of $S(e)$, then we return “error”. Otherwise, let f be the first edge of $S(e)$ following $\phi(e)$, then we do $\phi(e) \leftarrow f$. In addition, if $e = \langle a, b \rangle$, we update Bad and the p_i ’s by examining the positions $a, a + 1, b, b + 1$. For each such position j , we check whether it becomes a good position or a bad position; if it was bad and becomes good, we remove j from Bad thanks to p_j , and we update p_j to \perp ; if it was good and becomes bad, we add a new cell c to Bad storing j , and we update p_j to c . These operations can be performed in $O(1)$ time.

We now describe the algorithm for LINEAR MATCHING INCLUSION. At the beginning of the algorithm, we let $\phi(e)$ be the first edge of $S(e)$, and we initialize Bad and the p_i ’s in $O(n)$ time. While $Bad \neq \emptyset$, we choose an element $i \in Bad$ and we call $UPDATE(i)$. If the call fails, we return “no”. If we reach a point where $Bad = \emptyset$, we conclude that ϕ is an embedding of P into T , and we return “yes”.

The algorithm constructs a sequence of mappings $\phi_1 < \phi_2 < \dots < \phi_r$, with ϕ_i the mapping obtained at the i th step of the loop. Since each step of the loop takes $O(1)$ time, the algorithm takes time $O(r) = O(kn)$. Let us argue for the correctness. If the algorithm accepts, then ϕ_r is an embedding of P into T , since $Bad = \emptyset$ ensures that ϕ_r is strictly increasing. Conversely, suppose that ψ is an embedding of P into T . At each step, we have (i) for every $j \in V_P$, $\phi_i(j) \leq \psi(j)$, (ii) for every $j \in Bad$, $\phi_i(j) < \psi(j)$. Point (ii) ensures that an operation $UPDATE$ can never fail, thus the algorithm eventually finds $\phi_r \leq \psi$ embedding of P into T . \square

We are now ready to prove our theorem.

Proof of Theorem 2. For every $e \in E_T$, let $d(e)$ denote the length of a longest chain of inclusions in T , ending at e . For every $1 \leq i \leq d$, let $E_i = \{e \in E_T : d(e) = i\}$. Then the E_i are antichains of (E_T, \sqsubset) partitioning E_T . It follows that each E_i is nesting-free. The algorithm for LINEAR MATCHING INCLUSION examines each function $\psi : E_P \rightarrow [d]$. For a given ψ , it solves an instance (P, T, S_ψ) of CONSTRAINED LINEAR MATCHING INCLUSION with S_ψ mapping every $x \in E_P$ to the set $E_{\psi(x)}$. The correctness of the algorithm is easy to see, and the running time is $O(d^k \times kn)$. \square

3.2 An FPT Algorithm for Parameter p

In this section, we consider the complexity of the LINEAR MATCHING INCLUSION problem parameterized by $p = |V_T| - |V_P|$. Since LINEAR MATCHING INCLUSION is a special case of the ARC-PRESERVING SUBSEQUENCE problem, it follows from [9] that the problem is FPT in p . However, this gives an algorithm with running time $O(p^{O(p^3)}n)$. The following theorem gives a faster algorithm.

Theorem 2. LINEAR MATCHING INCLUSION can be solved in $O((3p)^p n)$ time.

Our algorithm borrows some ideas from [9]. In particular, we use the idea of computing a “partially specified” mapping from P to T , refining it at each step. This is formalized by the notion of fragmentation. We call a *fragmentation* of (P, T) a tuple $\mathcal{F} = (\mathcal{P}_P, \mathcal{P}_T)$, where \mathcal{P}_P is a partition of V_P into intervals I_1, \dots, I_r , and \mathcal{P}_T is a partition of V_T into intervals J_1, \dots, J_r such that: for every $1 \leq i \leq r$, $|J_i| \geq |I_i|$. A pair (I_i, J_i) is a *fragment* of \mathcal{F} ; the fragment is *trivial* if $|J_i| = |I_i|$.

Fix an embedding ϕ of P into T . We say that ϕ is *compatible* with \mathcal{F} iff $\phi(I_i) \subseteq J_i$ for every $1 \leq i \leq r$. We say that ϕ is *strongly compatible* with \mathcal{F} iff for every $1 \leq i \leq r$, there is a partition of I_i into intervals I'_i, I''_i , such that $\phi(I'_i)$ is an initial segment of J_i and $\phi(I''_i)$ is a final segment of J_i . Suppose that $I_i = [p, q]$ and $J_i = [r, s]$. Following [9], we introduce the following definitions. Given $x \in I_i$, we denote $x_{left} = x + r - p$ and $x_{right} = x + s - q$. We say that x is *skew* for ϕ if $x_{left} < \phi(x) < x_{right}$. A set $S \subseteq V_P$ is a *skew-set* for \mathcal{F} iff for every embedding ϕ of P into T compatible with \mathcal{F} , there is some element of S skew for ϕ .

Lemma 2. Let \mathcal{F} be a fragmentation of (P, T) . There is an algorithm FINDEMBEDDINGORSKEWSET(\mathcal{F}) running in $O(n)$ time such that:

1. if there is an embedding of P into T strongly compatible with \mathcal{F} , then it returns one;
2. otherwise, it returns a skew-set S for \mathcal{F} of size at most three.

Proof. We construct a digraph G as follows. It contains vertices x_{left}, x_{right} for $x \in V_P$. By convention, let $\neg x_{left} = x_{right}$ and $\neg x_{right} = x_{left}$. We say that a vertex x_a is *nice* if P contains the arc $\{x, y\}$ and T contains the arc $\{x_a, y_b\}$ with $b \in \{\text{left}, \text{right}\}$. Then G contains the following arcs:

1. if x_a is not nice, an arc $x_a \rightarrow \neg x_a$;
2. if $x \in I_i$ and $y = x + 1 \in I_i$, arcs $y_{left} \rightarrow x_{left}$ and $x_{right} \rightarrow y_{right}$;
3. if $\{x, y\} \in E_P$ and $\{x_a, y_b\} \in E_T$, an arc $x_a \rightarrow y_b$.

Let us color red the arcs of type 1, and blue the arcs of type 2 or 3. Note that if ϕ is an embedding of P into T compatible with \mathcal{F} , then: for a blue arc $x_a \rightarrow y_b$, we have $\phi(x) = x_a \Rightarrow \phi(y) = y_b$. We rely on the following claim, whose proof is omitted.

Claim 1. The following are equivalent: (i) there is an embedding of P into T strongly compatible with \mathcal{F} , (ii) there is no strongly connected component of G containing two vertices x_{left}, x_{right} .

We compute in $O(|G|) = O(n)$ time an enumeration C_1, \dots, C_m of the strongly connected components of G , such that for $i < j$, there is no arc from C_j to C_i . If (ii) holds, we can obtain an embedding of P into T strongly compatible with \mathcal{F} , and return it. If (ii) is violated for some element x , we obtain a cycle C containing x_{left}, x_{right} . There are three cases.

- Case 1: if C contains only blue arcs. We then have $\phi(x) = x_{left} \Leftrightarrow \phi(x) = x_{right}$, and thus x is skew for any embedding of P into T . We let $S = \{x\}$.
- Case 2: if C contains one red arc $x_a \rightarrow \neg x_a$. Then $\phi(x) = \neg x_a \Rightarrow \phi(x) = x_a$, and thus $\phi(x) \neq \neg x_a$. Let y be the vertex adjacent to x in P , since x_a is not nice $\phi(x) = x_a$ implies that y is skew. Thus, if y is not skew then x is skew. We let $S = \{x, y\}$.
- Case 3: if C contains at least two red arcs $x_a \rightarrow \neg x_a$ and $y_b \rightarrow \neg y_b$. Suppose w.l.o.g. that the path from $\neg x_a$ to y_b consists of blue arcs. Let u be the vertex adjacent to x in P , and let v be the vertex adjacent to y in P . Since x_a is not nice $\phi(x) = x_a$ implies that u is skew. Also, $\phi(x) = \neg x_a$ implies that $\phi(y) = y_b$, and since y_b is not nice this implies that v is skew. Thus, if u, v are not skew then x is skew. We let $S = \{x, u, v\}$.

This concludes the proof of the Lemma. \square

Proof of Theorem 2. We use a recursive procedure $\text{TESTFRAGMENTATION}(\mathcal{F})$ which proceeds as follows:

- we call $\text{FINDEMBEDDINGORSKEWSET}(\mathcal{F})$;
- if the algorithm returns an embedding of P into T strongly compatible with \mathcal{F} , we answer “yes”;
- if the algorithm finds a set $S \subseteq V_P$, we do the following for every $x \in S$. Suppose that $x \in I_i$, $I_i = [a, b]$ and $J_i = [c, d]$. For every $x_{left} < y < x_{right}$, let $\mathcal{F}_{x,y}$ be the fragmentation obtained from \mathcal{F} by replacing I_i with $[a, x[, [x, b]$ and J_i with $[c, y[, [y, d]$. We call $\text{TESTFRAGMENTATION}(\mathcal{F}_{x,y})$ for every $x \in S$, $x_{left} < y < x_{right}$, and we answer “yes” if one of the recursive calls does.

The algorithm for LINEAR MATCHING INCLUSION calls $\text{TESTFRAGMENTATION}(\mathcal{F})$ starting with the fragmentation \mathcal{F} consisting of $I_1 = V_P$ and $J_1 = V_T$.

The correctness of the algorithm is straightforward. Let us argue for the running time. Observe that at each recursive call $\text{TESTFRAGMENTATION}(\mathcal{F})$, the

fragmentation \mathcal{F} consists of only nontrivial fragments. This holds initially, and when calling recursively on $\mathcal{F}_{x,y}$ this follows from the fact that $x_{\text{left}} < y < x_{\text{right}}$. Since \mathcal{F} can have at most p nontrivial fragments, it follows that the height of the search tree is at most p . Since each call to the algorithm FINDEMBEDDINGORSKEWSET takes $O(n)$ time, and since the degree of the search tree is at most $3p$, we obtain that the running time is $O((3p)^p n)$. \square

3.3 An Algorithm for Nesting-Free Patterns

To obtain our results, we rely on a dynamic-programming algorithm for the LINEAR MATCHING INCLUSION problem (Lemma 3), obtained by adapting an algorithm of [1] for the PERMUTATION PATTERN problem. Let $M = (V, E)$ be a linear matching. Given $e = \langle i, j \rangle$, we let $l(e) = i, r(e) = j$. Given $x \in V, y \in E$, we say that x is adjacent to y if one of $x - 1, x + 1$ is an endpoint of y . Given $x, y \in E$, we say that x is adjacent to y if $l(x)$ or $r(x)$ is adjacent to y .

Definition 1. Let $M = (V, E)$ be a linear matching, and let $X \subseteq E$. The border of X , denoted by $\partial(X)$, is the set of elements $x \in X$ adjacent to an element of $E - X$.

Definition 2. Let $M = (V, E)$ be a linear matching, and let $\sigma = e_1, \dots, e_n$ be an enumeration of E . Given $1 \leq i \leq n$, let $E_i = \{e_1, \dots, e_i\}$. We define the width of σ as $w(\sigma) = \max_i |\partial(E_i)|$.

We give an algorithm whose complexity is bounded in terms of the width of the pattern.

Lemma 3. Let P, T be linear matchings of respective size k, n , and let σ be an enumeration of P of width w . We can decide in $O(kn^{w+1})$ time if P occurs in T .

Let $P = (V, E)$ be a nesting-free linear matching of length k . We can partition V into intervals V_1, \dots, V_r such that each arc of E join some V_i to V_{i+1} . We let S_i ($1 \leq i < r$) denote the set of arcs joining V_i to V_{i+1} . We let $s = \max_i |S_i|$. We give two bounds for the width of a nesting-free linear matching.

Lemma 4. We can construct in polynomial time an enumeration σ of P such that $w(\sigma) \leq 2s$.

Proof. Let $\sigma = e_1, \dots, e_k$ enumerate the arcs of P from left to right. Fix $1 \leq i \leq k$, and let $E_i = \{e_1, \dots, e_i\}$, we show that $|\partial(E_i)| \leq 2s$. Suppose that $e_i \in S_j$. Observe that a vertex preceding $l(e_i)$ is incident to an arc of E_i , for otherwise we would have an arc $e' < e_i$ not in E_i . Suppose that $e \in E_i$ is adjacent to an element of $E - E_i$. By the above observation, we have $l(e) \leq l(e_i) < r(e) \leq r(e_i)$. It follows that $e = e_i$ or $e \not\parallel e_i$, and thus $e \in S_{j-1} \cup S_j$. Thus, $|\partial(E_i)| \leq 2s$. \square

Lemma 5. We can construct in polynomial time an enumeration σ of P such that $w(\sigma) \leq 2r - 1$.

Proof. A *left-set* is a subset $X \subseteq V$ s.t. (i) for every $1 \leq i \leq r$, $X \cap V_i$ is an initial interval of V_i , (ii) there is no edge of P joining X to $V - X$. We construct inductively an enumeration $\sigma = e_1, \dots, e_k$ such that for each i , the set of endpoints of $E_i = \{e_1, \dots, e_i\}$ forms a left-set X_i . Suppose that we have constructed σ up to $i < k$. Let S be the set of indices j such that $V_j - X_i \neq \emptyset$, and given $j \in S$ let x_j be the first element of $V_j - X_i$.

Claim. P contains an edge of the form $\langle x_j, x_{j+1} \rangle$ with $j, j+1 \in S$.

Proof. Suppose not. For every $j \in S$, let f_j denote the arc incident to x_j . Let j be the first index of S . Then f_j is incident to a vertex $u \in V_{j-1} \cup V_{j+1} - X_i$. We cannot have $u \in V_{j-1}$ by choice of j , hence $u \in V_{j+1}$ and $u > x_{j+1}$. Then f_{j+1} is incident to a vertex $u' \in V_j \cup V_{j+2} - X_i$. If we had $u' \in V_j$, since $f_j \not\parallel f_{j+1}$ we would obtain that $u' < x_j$ and $u' \notin X_i$, contradiction; hence $u' \in V_{j+2}$ and $u' > x_{j+2}$. By iterating this argument, we reach a point where f_l must be incident to an element of V_{l+1} , but $l \notin S$, contradiction. \square

According to the claim let us choose for e_{i+1} an edge of the form $\langle x_j, x_{j+1} \rangle$ with $j, j+1 \in S$. Then the set of endpoints of E_{i+1} forms a left-set. Observe that this implies that $|\partial(E_{i+1})| \leq 2r - 1$, since only the first and last element of an interval $X_{i+1} \cap V_j$ can be adjacent to an element of $E - E_{i+1}$. \square

As a consequence of Lemmas 3, 4 and 5, we obtain the following theorem.

Theorem 3. *Let $w = \min(2s, 2r - 1)$. Given T of length n , we can decide in $O(kn^{w+1})$ time if P occurs in T .*

4 Hardness Results

4.1 Hardness of Linear Matching Inclusion

We obtain our results by introducing an intermediate problem called MULTICOLORED SEQUENCE (Mcs), and by giving a reduction of MCS to the LINEAR MATCHING INCLUSION problem.

Let us first define the MULTICOLORED SEQUENCE problem. Given $p \in \mathbb{N}$, a p -*multiorder* \mathcal{O} is a p -tuple $(V_1, \leq_1), \dots, (V_p, \leq_p)$ where each (V_i, \leq_i) is a total order, and where the sets V_i are pairwise disjoint. The *ground set* of \mathcal{O} is $V := V_1 \cup \dots \cup V_p$. An *element* of \mathcal{O} is a set $X = \{x_1, \dots, x_p\}$ where $x_i \in V_i$ for each i . Given two elements of \mathcal{O} , $X = \{x_1, \dots, x_p\}$ and $Y = \{y_1, \dots, y_p\}$, we denote $X \preceq_{\mathcal{O}} Y$ if and only if $x_i \leq_i y_i$ for each i . With these notions, we are now ready to define the MULTICOLORED SEQUENCE problem.

Name: MULTICOLORED SEQUENCE

Input: integers p, q , a p -multiorder \mathcal{O} with ground set V , a sequence $\mathcal{F} = F_1, \dots, F_q$ where each F_i is a family of subsets of V .

Question: find a chain $X_1 \preceq_{\mathcal{O}} X_2 \preceq_{\mathcal{O}} \dots \preceq_{\mathcal{O}} X_q$ such that for each $i \in [q]$, X_i is included in an element of F_i .

We will need the following hardness results for MCS. Given two integers r, s , let $(r, s) - \text{MCS}$ denote the restriction of the MCS problem to instances such that: (i) for every $i \in [p]$, the set V_i has size at most r , (ii) for every $i \in [q]$, the set F_i has size at most s .

Proposition 1. $(2, 3) - \text{MCS}$ is NP-hard.

Proposition 2. MCS is W[1]-hard for parameters (p, q) .

We now describe our main reduction from MCS to LINEAR MATCHING INCLUSION.

Proposition 3. There is a polynomial-time reduction which maps an instance $(p, q, \mathcal{O}, \mathcal{F})$ of $(r, s) - \text{MCS}$ to an instance (P, T) of LINEAR MATCHING INCLUSION, with P a linear matching of size $2(q + 1) + pq$, with $d(P) \leq 2$ and $d(T) \leq 2rs$.

Proof. Consider an instance $I = (p, q, \mathcal{O}, \mathcal{F})$ of $(r, s) - \text{MCS}$, where \mathcal{O} consists of sets $(V_1, \leq_1), \dots, (V_p, \leq_p)$, and where $\mathcal{F} = F_1, \dots, F_q$. Let us add to each V_i a maximal element \top_i , and let us add a set $F_{q+1} = \{S_{q+1}\}$ with $S_{q+1} = \{\top_1, \dots, \top_p\}$. We describe the construction of an instance $I' = (P, T)$ of LINEAR MATCHING INCLUSION.

Construction of P. The pattern P is constructed as follows. It consists of intervals I_1, \dots, I_{q+1} appearing in this order. Each interval I_i contains points $a_i, b_i, c_{i,1}, \dots, c_{i,2p}, d_i, e_i$. For every $1 \leq i \leq q$, P contains an arc $\alpha_i = \langle a_i, e_i \rangle$ and an arc $\beta_i = \langle b_i, d_i \rangle$. For every $1 \leq i < q$, P contains p arcs $\gamma_{i,1}, \dots, \gamma_{i,p}$ joining I_i to I_{i+1} . These arcs are defined in the following way: (a) if i is even, then $\gamma_{i,j}$ joins $c_{i,2j-1}$ to $c_{i+1,2j-1}$; (b) if i is odd, then $\gamma_{i,j}$ joins $c_{i,2j}$ to $c_{i+1,2j}$. Finally, we discard the points of I_1 and I_{q+1} not incident to an arc.

Construction of T. The target T is constructed as follows. It consists of $q + 1$ intervals I'_1, \dots, I'_{q+1} appearing in this order. Let S_1, \dots, S_m be an arbitrary enumeration of F_i , then I'_i contains a point a'_i , the intervals $I'_{i,S_1}, \dots, I'_{i,S_m}$, and a point e'_i . An interval $I'_{i,S}$ contains a point $b'_{i,S}$, intervals $I'_{i,S,1}, \dots, I'_{i,S,p}$, and a point $d'_{i,S}$. An interval $I'_{i,S,j}$ is further subdivided into intervals $I'_{i,S,j,x}, I''_{i,S,j,x}$ for $x \in V_j$. If $V_j = \{x_1, \dots, x_m\}$ with $x_1 <_j \dots <_j x_m$, then: (a) if i is odd, these intervals appear in the order $I''_{i,S,j,x_1}, I'_{i,S,j,x_1}, \dots, I''_{i,S,j,x_m}, I'_{i,S,j,x_m}$; (b) if i is even, these intervals appear in the order $I'_{i,S,j,x_m}, I''_{i,S,j,x_m}, \dots, I'_{i,S,j,x_1}, I''_{i,S,j,x_1}$.

The linear matching T is the union of three sets T_α, T_β and T_γ . The set T_α contains, for every $1 \leq i \leq q$, an arc $\alpha'_i = \langle a'_i, e'_i \rangle$. The set T_β contains, for every $1 \leq i \leq q$ and $S \in F_i$, an arc $\beta'_{i,S} = \langle b'_{i,S}, d'_{i,S} \rangle$. The set T_γ contains, for every $1 \leq i < q$, $S \in F_i, S' \in F_{i+1}, 1 \leq j \leq p$, a set $T_{i,S,S',j}$ of arcs joining $I'_{i,S,j}$ to $I'_{i+1,S',j}$. The set $T_{i,S,S',j}$ is constructed in the following way: for each $x \in S \cap V_j$, we add to $T_{i,S,S',j}$ an arc $\gamma'_{i,S,S',j,x}$ joining $I'_{i,S,j,x}$ to $I''_{i+1,S',j,x}$. Finally, the endpoints of arcs incident to a set $I'_{i,S,j,x}$ or $I''_{i,S,j,x}$ are ordered arbitrarily.

The construction is clearly doable in polynomial time. We omit the correctness proof due to space limitations. \square

As a consequence of Propositions 1, 2 and 3, we obtain:

Theorem 4. 1. LINEAR MATCHING INCLUSION is NP-hard, even when $d(P) \leq 2$ and $d(T) \leq 12$.
 2. LINEAR MATCHING INCLUSION is W[1]-hard for parameter k , even when $d(P) \leq 2$.

4.2 Hardness of Nesting-Free 2-Interval Pattern

Given an interval $I = [a, b]$, we let $l(I) = a$ and $r(I) = b$. Given two intervals I, I' , we denote $I < I'$ iff $r(I) < l(I')$. A *2-interval* is a set $I \cup J$, where I and J are intervals such that $I < J$. We introduce precedence / crossing / nesting relations on 2-intervals, similarly to the relations defined in Section 2. Given two 2-intervals $S = I \cup J, S' = I' \cup J'$, we say that:

- S precedes S' (denoted $S \prec S'$) iff $I < J < I' < J'$;
- S crosses S' (denoted $S \between S'$) iff $I < I' < J < J'$;
- S nests in S' (denoted $S \sqsubset S'$) iff $I' < I < J < J'$.

Let $\mathcal{R} \subseteq \{\prec, \between, \sqsubset\}$. We say that a set \mathcal{F} of 2-intervals is \mathcal{R} -structured iff every pair of 2-intervals of \mathcal{F} is comparable by some $R \in \mathcal{R}$. The \mathcal{R} -STRUCTURED 2-INTERVAL PATTERN [10] is as follows.

Name: \mathcal{R} -STRUCTURED 2-INTERVAL PATTERN

Input: A set \mathcal{F} of 2-intervals, an integer k .

Question: Does there exist a set $\mathcal{I} \subseteq \mathcal{F}$ of size k which is \mathcal{R} -structured?

The problem is polynomial when $\mathcal{R} = \{\prec, \sqsubset\}$ or when $|\mathcal{R}| = 1$ [10]. When $\mathcal{R} = \{\prec, \between, \sqsubset\}$, the problem is equivalent to the MAXIMUM INDEPENDENT SET for 2-interval graphs, which was shown W[1]-hard in [4]. When $\mathcal{R} = \{\between, \sqsubset\}$, the problem was shown W[1]-hard in [7]. We settle the last case, by showing the W[1]-hardness when $\mathcal{R} = \{\prec, \between\}$. For convenience, we call this problem NESTING-FREE 2-INTERVAL PATTERN.

Theorem 5. NESTING-FREE 2-INTERVAL PATTERN is W[1]-hard for parameter k .

Proof. We reduce from MCS parameterized by (p, q) . Let $I = (p, q, \mathcal{O}, \mathcal{F})$ be an instance of MCS. Suppose that \mathcal{O} consists of orders (V_i, \leq_i) and that $\mathcal{F} = F_1, \dots, F_q$. Let \perp_i, \top_i denote the minimum and maximum element of (V_i, \leq_i) , and given $x \in V_i$ let $\text{rank}_i(x)$ denote the rank of x in (V_i, \leq_i) , i.e. $\text{rank}_i(x) = |\{y \in V_i : y \leq_i x\}|$. We add a set F_0 which contains a single set $S_\perp = \{\perp_1, \dots, \perp_p\}$, and we add a set F_{q+1} which contains a single set $S_\top = \{\top_1, \dots, \top_p\}$. Now, for each $0 \leq i \leq q+1$ let $s_i = |F_i|$, and let $S_{i,1}, \dots, S_{i,s_i}$ be an enumeration of F_i . Let $r = 1 + \max_i |V_i|$ and $s = \max_i s_i$. We construct an instance $I' = (\mathcal{F}, k)$ of NESTING-FREE 2-INTERVAL PATTERN, where $k = (q+2) + p(q+1)$, and where \mathcal{F} is the following family of 2-intervals.

Let $L = 2rs$ and $M = (2L + r)(p + 2)$. The support of \mathcal{F} is the interval $I = [1, (q + 2)M]$, and it is subdivided into the intervals I_0, \dots, I_{q+1} where $I_i = [a_i, b_i]$ with $a_i = iM + 1$ and $b_i = (i + 1)M$. Fix $0 \leq i \leq q + 1$, then I_i contains the following set \mathcal{S}_i of subintervals.

For $1 \leq t \leq s_i$, we let $c_{i,t} = a_i + 2r(t - 1) + 1$, $d_{i,t} = a_i + 2r(t - 1) + (2L + r)p + 1$, then \mathcal{S}_i is the following set of length L -intervals: (i) an interval $L_{i,t}$ starting at $c_{i,t}$, and an interval $R_{i,t}$ starting at $d_{i,t}$; (ii) for each $1 \leq j \leq p$, for each $x \in V_j$ with $\text{rank}_j(x) =: r_x$: an interval $I_{i,t,j,x}$ starting at $c_{i,t} + L + (2L + r)(j - 1) + r_x - 1$; an interval $I'_{i,t,j,x}$ starting at $c_{i,t} + 2L + (2L + r)(j - 1) + r_x - 1$.

The family \mathcal{F} then contains the following 2-intervals: (i) the family \mathcal{F}_a of *anchor components*: for $0 \leq i \leq q + 1$, $1 \leq t \leq s_i$, the 2-interval $A_{i,t} = L_{i,t} \cup R_{i,t}$; (ii) the family \mathcal{F}_b of *propagation components*: for $0 \leq i \leq q$, $1 \leq t \leq s_i$ and $1 \leq t' \leq s_{i+1}$, for $1 \leq j \leq p$, for $x \in S_{i,t} \cap V_j$, \mathcal{F}_b contains a 2-interval $B_{i,t,t',j,x} = I'_{i,t,j,x} \cup I_{i+1,t',j,x}$.

Observe that the construction can be carried out in polynomial time. The correctness proof is omitted due to space limitations. \square

We point out that our W[1]-hardness result implies that the optimization version of NESTING-FREE 2-INTERVAL PATTERN has no EPTAS. A PTAS with running time $O(n^{2/\epsilon+3})$ was described in [6].

References

1. Ahal, S., Rabinovich, Y.: On Complexity of the Subpattern Problem. SIAM J. Discrete Math. 22(2), 629–649 (2008)
2. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
3. Evans, P.: Algorithms and Complexity fo Annotated Sequence Analysis. PhD thesis, University of Victoria, Canada (1999)
4. Fellows, M.R., Hermelin, D., Rosamond, F.A., Vialette, S.: On the parameterized complexity of multiple-interval graph problems. Theoretical Computer Science 410(1), 53–61 (2009)
5. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, New York (2006)
6. Jiang, M.: A PTAS for the Weighted 2-Interval Pattern Problem over the Preceding-and-Crossing Model. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA. LNCS, vol. 4616, pp. 378–387. Springer, Heidelberg (2007)
7. Jiang, M.: On the Parameterized Complexity of Some Optimization Problems Related to Multiple-Interval Graphs. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 125–137. Springer, Heidelberg (2010)
8. Li, S.C., Li, M.: On two open problems of 2-interval patterns. Theoretical Computer Science 410(24–25), 2410–2423 (2009)
9. Marx, D., Schlotter, I.: Parameterized Complexity of the Arc-Preserving Subsequence Problem. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 244–255. Springer, Heidelberg (2010)
10. Vialette, S.: On the computational complexity of 2-interval pattern matching problems. Theoretical Computer Science 312(2–3), 223–249 (2004)

Computational Study on Bidimensionality Theory Based Algorithm for Longest Path Problem

Chunhao Wang and Qian-Ping Gu

School of Computing Science, Simon Fraser University,
Burnaby BC Canada, V5A 1S6
{cwa39,qgu}@cs.sfu.ca

Abstract. Bidimensionality theory provides a general framework for developing subexponential fixed parameter algorithms for NP-hard problems. In this framework, to solve an optimization problem in a graph G , the branchwidth $\text{bw}(G)$ is first computed or estimated. If $\text{bw}(G)$ is small then the problem is solved by a branch-decomposition based algorithm which typically runs in polynomial time in the size of G but in exponential time in $\text{bw}(G)$. Otherwise, a large $\text{bw}(G)$ implies a large grid minor of G and the problem is computed or estimated based on the grid minor. A representative example of such algorithms is the one for the longest path problem in planar graphs. Although many subexponential fixed parameter algorithms have been developed based on bidimensionality theory, little is known on the practical performance of these algorithms. We report a computational study on the practical performance of a bidimensionality theory based algorithm for the longest path problem in planar graphs. The results show that the algorithm is practical for computing/estimating the longest path in a planar graph. The tools developed and data obtained in this study may be useful in other bidimensional algorithm studies.

Keywords: Experimental algorithms, bidimensional algorithms, branch-decomposition, grid minor, longest path problem.

1 Introduction

For NP-hard problems, fixed parameter exact algorithms have been extensively studied. Bidimensionality theory introduced in [6][7][9] provides a general framework for designing subexponential fixed parameter algorithms for NP-hard problems in a graph G . A major basis for bidimensionality theory is the relationship between branchwidth and the largest grid minor size of G .

The notions of *branchwidth* and *branch-decomposition* are introduced by Robertson and Seymour [9] in their graph minor theory. A branch-decomposition of a graph G is a system of vertex-cut sets represented as links of a tree whose leaves are edges of G . The *width* of a branch-decomposition is the maximum size of a cut set in the system and the *branchwidth* of G , denoted by $\text{bw}(G)$, is the

minimum width of all possible branch-decompositions of G . A graph H is called a *minor* of a graph G if H can be obtained from a subgraph of G through a sequence (maybe empty) of edge contractions. A $k \times h$ grid is a graph on vertex set $\{(i, j) | 0 \leq i < k, 0 \leq j < h, i, j : \text{integer}\}$ such that vertices (i, j) and (i', j') are adjacent if and only if $|i - i'| + |j - j'| = 1$. We denote by $\text{gm}(G)$ the largest integer g such that G contains a $g \times g$ grid as a minor. Robertson, Seymour and Thomas show that $\text{gm}(G) \leq \text{bw}(G) \leq 2^{0.2\text{gm}(G)(\text{gm}(G)+1)^4}$ for general graphs and $\text{gm}(G) \leq \text{bw}(G) \leq 4\text{gm}(G)$ for planar graphs [18]. The linear bound of the form $\text{gm}(G) \leq \text{bw}(G) \leq \text{cgm}(G)$ has been extended to bounded genus graphs [7] and to graphs excluding a fixed minor [8].

Informally, a problem in a graph G with solution value $f(G)$ is *bidimensional* if the value f for the $g \times g$ grid grows as g increases and $f(H) \leq f(G)$ if H is a minor of G . A representative example of bidimensional problems is the longest path problem. Given a graph G and an integer l , the problem is to determine if there is a path of length at least l in G . The optimization version of the problem is to find a path with the largest length in G . The longest path problem is NP-complete for general graphs [12] and remains NP-complete for planar graphs [11]. Because of its importance, the longest path problem has been extensively studied and bidimensionality theory based algorithms have been developed for the problem in planar graphs [10]. Let $f(G)$ denote the length of the longest path of G . To decide if $f(G) \geq l$, $\text{bw}(G)$ is first computed. If $\text{bw}(G) \geq c\sqrt{l+1}$ then from $\text{bw}(G) \leq \text{cgm}(G)$, graph G has a $\sqrt{l+1} \times \sqrt{l+1}$ grid minor which has a path of length $(\sqrt{l+1})^2 - 1 = l$, and thus $f(G) \geq l$ is concluded. Otherwise, $f(G)$ is computed by a branch-decomposition based algorithm in $O(2^{c\alpha\text{bw}(G)}n^{O(1)})$ time, where α is a constant. Generally, in the framework of bidimensionality theory, to decide if $f(G) \geq l$ for a given l , one first computes or estimates $\text{bw}(G)$. If $\text{bw}(G)$ is large then by the linear bound $\text{bw}(G) \leq \text{cgm}(G)$, $f(G) \geq l$ may be concluded. Otherwise, $f(G)$ is computed exactly by a branch-decomposition based algorithm which typically runs in polynomial time in the size of G but in exponential time in $\text{bw}(G)$ (i.e., $O(2^{c\alpha\text{bw}(G)}n^{O(1)})$). With a small $\text{bw}(G)$, $f(G)$ can often be computed efficiently. Since the coefficient c in the linear bound $\text{bw}(G) \leq \text{cgm}(G)$ appears in the exponent of the running time, it is critical to reduce this coefficient.

A $k \times h$ cylinder is a graph on vertex set $\{(i, j) | 0 \leq i < k, 0 \leq j < h, i, j : \text{integer}\}$ such that vertices (i, j) and (i', j') are adjacent if and only if $i' \equiv (i \pm 1) \pmod{k}$ and $j' = j$ or $i' = i$ and $|j - j'| = 1$. Notice that a $k \times h$ cylinder contains a $k \times h$ grid as a minor. Recently, Gu and Tamaki give a algorithm (the GT Algorithm) which, given a planar graph G and integers $k \geq 3$ and $h \geq 1$, computes either a branch-decomposition of width at most $k + 2h - 3$ or a $k \times h$ cylinder minor in $O(n^2)$ time [13]. Taking $k = h$, the GT Algorithm computes a cylinder minor with $k \geq \text{bw}(G)/3$, reducing c from 4 to 3 (i.e., $\text{bw}(G) \leq 3\text{gm}(G)$). Let $\text{cm}(G)$ be the largest integer g such that G has a $g \times \lceil g/2 \rceil$ cylinder minor. The GT Algorithm implies $\text{bw}(G) \leq 2\text{cm}(G)$. Since a $k \times h$ grid/cylinder has a path of length $k \times h - 1$, the bound $\text{bw}(G) \leq 2\text{cm}(G)$ implies that G has a path

of length at least $\frac{(\text{bw}(G))^2}{8} - 1$ which is a better lower bound for the length of a longest path than $\frac{(\text{bw}(G))^2}{9} - 1$ guaranteed by $\text{bw}(G) \leq 3\text{gm}(G)$.

Although bidimensionality theory based algorithms have been proposed for many NP-hard problems, little is known about the practical performance of these algorithms. Hurdles for computational studies of these algorithms may include the implementations of algorithms for computing $\text{bw}(G)$, the branch-decomposition based algorithms for exactly solving the problems and algorithms for finding the grid minors of size guaranteed by the linear bound $\text{bw}(G) \leq \text{cgm}(G)$. None of the implementations is trivial. Recently, there are progresses in computing $\text{bw}(G)$ and the optimal branch-decomposition for planar graphs. Let G be a planar graph and β an integer. It is known that $\text{bw}(G) \leq \beta$ can be decided in $O(n^2)$ time [20] and an optimal branch-decomposition can be computed in $O(n^3)$ time [14][20]. Computational studies for computing $\text{bw}(G)$ and optimal branch-decompositions of G are reported [1][2].

In this paper, we report a computational study on a bidimensional algorithm for the longest path problem in planar graphs. For a planar graph G , we first compute $\text{bw}(G)$. If $\text{bw}(G) < 2^{3/2}\sqrt{l+1}$ we compute an optimal branch-decomposition of G and use this decomposition and the algorithm by Dorn et al. [10] (the DPBF Algorithm) to compute a longest path of G . If $\text{bw}(G) \geq 2^{3/2}\sqrt{l+1}$, we conclude that G has a path of length at least l based on the bound $\text{bw}(G) \leq 2\text{cm}(G)$. We test the DPBF Algorithm on a wide range of planar graphs with up to 15,000 edges. The computational results show that the DPBF Algorithm is practical for large planar graphs when the branchwidth of the graphs is no larger than 10. For graphs with large branchwidth, the bound $\text{bw}(G) \leq 2\text{cm}(G)$ guarantees that G has a path of length at least $\frac{(\text{bw}(G))^2}{8} - 1$.

Based on the bound $\text{bw}(G) \leq 4\text{gm}(G)$ [18], Bodlaender, Gregoriev and Koster give an efficient algorithm (the BGK Algorithm) which computes a $k \times k$ grid minor with $k \geq \text{bw}(G)/4$ and report a computational study on this algorithm [3]. We implement the GT Algorithm which, by taking $k = h$, finds a $k \times k$ cylinder minor of G with $k \geq \text{bw}(G)/3$. In practice, both the BGK Algorithm and the GT Algorithm are time efficient and find $k \times k$ grid minors with $k \geq \text{bw}(G)/2$ for most test instances. Further, a better lower bound can be obtained from the $k \times h$ cylinder minor by choosing different combinations of k and h in the GT Algorithm.

In practice, computing a large grid/cylinder minor often gives a better lower bound than $\frac{(\text{bw}(G))^2}{8} - 1$ for the longest path. In many applications, a path of length guaranteed by the minor may need to be explicitly computed as well. For graphs with large branchwidth, our results show that cylinder minors computed by the GT Algorithm give a better estimation on the length of the longest path than using the formulas $\text{bw}(G) \leq 3\text{gm}(G)$ and $\text{bw}(G) \leq 2\text{cm}(G)$.

To evaluate how close the cylinder minors found by the GT Algorithm are to the largest ones, we design an exact algorithm for computing the largest cylinder minor of a planar graph G . Because the running time of the exact algorithm is not bounded by a polynomial in the size of G , the algorithm can find the largest cylinder minor for small G (up to 300 edges) with 2 GByte memory in a practical

time. The computational results on small graphs show that the GT Algorithm finds cylinder minors very close to optimal in practice. We may expect that for large instances of planar graphs, the GT Algorithm also finds near-optimal cylinder minors.

In Sec. 2, we introduce the preliminaries of our work. We briefly review the DPBF Algorithm, and introduce the exact algorithm for the largest cylinder minor in Sec. 3. The computational results are reported in Sec. 4. The final section concludes the paper.

2 Preliminaries

In this paper, graphs are undirected. For a graph G , we denote by $V(G)$ the vertex set and $E(G)$ the edge set of G . For each element $e \in E(G)$, we view e as a subset of $V(G)$ with two elements. For a subset $A \subseteq E(G)$ of edges we denote by $V(A) = \bigcup_{e \in A} e$ the set of vertices in edges of A . A *vertex-cut* of G is a set of vertices such that the removal of these vertices disconnects G into at least two connected components.

For a subset $A \subseteq E(G)$ the *subgraph* $G[A]$ of G induced by A is the graph with edge set A and vertex set $V(A)$. For a subset $A \subseteq E(G)$ of edges, we denote $E(G) \setminus A$ by \overline{A} . A *separation* of a graph G is a pair (A, \overline{A}) of subsets of $E(G)$. For each $A \subseteq E(G)$, we denote by $\partial(A)$ the vertex set $V(A) \cap V(\overline{A})$. The *order* of a separation (A, \overline{A}) is $|\partial(A)| = |\partial(\overline{A})|$.

A *branch-decomposition* of G is a tree T such that the set of leaves of T is $E(G)$ and each internal node of T has node degree 3. For each link e of T , removing e separates T into two subtrees. Let A_e and \overline{A}_e be the sets of leaves of the subtrees. The *width* of e is $|\partial(A_e)| = |\partial(\overline{A}_e)|$. The *width* of T is the maximum width of all links of T . The *branchwidth* of G , denoted by $\text{bw}(G)$, is the minimum width of all branch-decompositions of G .

A graph is *planar* if it can be drawn on a sphere Σ without crossing edges. We assume a planar graph G has such a drawing on Σ and call the drawing a *plane graph*. For a plane graph G , we call each connected component of Σ after removing the vertices and edges of G from Σ a *face* of G . We denote by $F(G)$ the set of faces of G .

3 Algorithms Implemented

To perform the computational study, we implement the DPBF Algorithm and the GT Algorithm. We also design and implement an exact algorithm for the largest cylinder minor in planar graphs. Due to the space limitation, we only briefly review the DPBF Algorithm, and introduce the exact algorithm for cylinder minors.

The DPBF Algorithm computes a longest path in a planar graph in two major steps (readers may refer to [10] for more details). (1) An optimal branch-decomposition T of a plane graph G is found. (2) A longest path of G is computed by dynamic programming based on T . Step (1) can be done in $O(n^3)$ time [14, 20].

In Step (2), the branch-decomposition T is first converted to a rooted binary tree. Removing a link e from T disconnects T into two subtrees: one contains the root of T and other does not. We denote by A_e the set of edges of G associated with the leaves of the subtree which does not contain the root of T , and denote by \overline{A}_e the set of edges associated with the leaves of the other subtree. Then e induces the separation (A_e, \overline{A}_e) of G . The dynamic programming step finds the partial solutions in the subgraph $G[A_e]$ for every link e of T from leaves to the root in a bottom-up way. It is shown in [10] that the DPBF Algorithm solves the longest path problem in planar graphs in $O(2^{3.404\text{bw}(G)}n^{5/2} + n^3)$ time [1].

Given a planar graph G and integers $k \geq 3$ and $h \geq 1$, the GT Algorithm computes in $O(n^2)$ time either a $k \times h$ cylinder minor of G or a branch-decomposition of G with width at most $k + 2h - 3$. Readers may refer to [13] for the description of the algorithm.

A cylinder consists of rows and columns. Columns are vertex-disjoint paths, while rows are vertex-disjoint cycles. For a $k \times h$ cylinder embedded on Σ , we call the cycle with the vertex set $\{(i, h-1) | 0 \leq i < k\}$ ($\{(i, 0) | 0 \leq i < k\}$, resp.) the *top cycle* (*bottom cycle*, resp.). We call the face incident only to the vertices of the top cycle (*bottom cycle*, resp.) the *top face* (*bottom face*, resp.). Each row cycle induces a vertex-cut separating the top face from the bottom face. The intuition of this exact algorithm is as follows. Assume that a plane graph G has a $k \times h$ cylinder as a minor. We first find the vertex-cut induced by the top cycle by enumerating all vertex-cuts of a certain size by the method from [4], then we identify the vertex-cuts induced by the rest of row cycles by computing contours by the method from [13], and finally we find a cylinder minor from the h vertex-cuts and k vertex-disjoint paths. The running time of our algorithm is not bounded by a polynomial in n . Readers may refer to [21] for the details of this algorithm.

4 Computational Results

4.1 Results for Longest Paths

We implement the DPBF Algorithm and test its performance on a wide range of planar graphs. Due to the space limitation, we only report the results on a set of random maximal planar graphs generated by LEDA [16] (Class I), a set of graphs taken from TSPLIB [17] (Class II) and a set of random planar graphs generated by PIGALE library [5] (Class III). These classes of graphs are representatives of planar graphs and are widely used in research on related problems. We use two computers in this paper. *Computer A* has an AMD Athlon 64 X2 Dual Core 4600+ (2.4 GHz) CPU and 3 GByte RAM, and the operating system is SUSE Linux 10.2. *Computer B* has an Intel Pentium 4 3.00 GHz CPU and 2 GByte RAM, and the operating system is Cent OS 5.5. We use Computer A to test the DPBF Algorithm. The algorithm is implemented with C++.

¹ No explicit analysis on the constant $O(1)$ in $2^{3.404\text{bw}(G)}n^{O(1)} + n^3$ is given for the longest path problem in [10]. Readers may refer to [21] for the analysis of $O(2^{3.404\text{bw}(G)}n^{5/2} + n^3)$.

Table 1. Computational results for planar longest paths. The column “bw” indicates the branchwidth of the graphs and “bw-time” is the time for computing the optimal branch-decomposition in Step (1) of the DPBF Algorithm. The tool of [1] is used for this purpose. The column “lp” indicates the length of the longest path, “lp-time” is the time of Step (2) of the DPBF Algorithm for computing the longest path, “total-time” is the total running time of the DPBF Algorithm, and “max-mem” is the memory space used by the algorithm. The time is in seconds and memory is in MByte.

class	graphs	# of edges	bw	bw-time	lp	lp-time	total-time	max-mem
I	max1000	2994	4	68.05	731	14.14	82.19	15.32
	max2000	5994	4	301.68	1436	50.7	352.38	28.8
	max3000	8994	4	1095.91	1807	111.84	1207.75	45.92
	max4000	11994	4	1856.6	2557	196.67	2053.27	72.65
	max5000	14994	4	3138.08	3164	303.7	3441.78	94.12
II	eil51	140	8	0.11	50	145.3	145.41	38.23
	lin105	292	8	2.48	104	654.44	656.92	90.47
	pr144	393	9	0.52	143	2742.86	2743.38	306.11
	kroB150	436	10	0.83	149	91475.42	91476.25	2014.36
	pr226	586	7	1.56	225	96.53	98.09	45
	ch130	377	10	0.54	129	38787.61	38788.15	998.6
III	p153	248	4	0.97	131	0.33	1.3	5.13
	p257	433	5	1.73	226	1.94	3.67	8.66
	p495	852	5	11.66	426	4.97	16.63	14.74
	p855	1434	6	21.06	726	28.05	49.11	59.03
	p1277	2128	9	61.98	1087	10074.09	10135.07	2308.92
	p2099	3369	7	232.45	1670	639.93	639.93	309.85
	p3586	6080	8	2583.52	2982	8224.52	8224.52	2311.43

The computational results are shown in Table 1. The DPBF Algorithm runs in $O(2^{3.404\text{bw}(G)}n^{5/2} + n^3)$ time. Graphs in Class I have small branchwidth (at most 4 [15]). Therefore, the DPBF Algorithm can handle large graphs of more than ten thousand edges in a few minutes. For graphs in Class II, the branchwidth grows quickly as the size of graphs increases. The DPBF Algorithm can handle the graphs of size up to a few hundred edges in this class in a practical time and memory space. For the graphs in Class III, the branchwidth grows slowly as the size of graphs increases. The DPBF Algorithm can handle the graphs of size up to a few thousand edges in a practical time and memory space. The DPBF Algorithm uses exponential memory space in the branchwidth. This is a bottleneck of the algorithm. Our results show that for graphs of branchwidth up to 10, the DPBF Algorithm is able to compute the longest path in few hours but may not be able to handle graphs with branchwidth greater than 10 with 3 GByte memory.

When the branchwidth of a planar graph G is large, the DPBF Algorithm may not be able to compute a longest path of G in a practical time and memory space. For such graphs, we can compute a grid/cylinder minor and a path of length guaranteed by the minor. We use the GT Algorithm to compute the cylinder minor. We test the algorithm on a wide range of planar graphs. Because of the space limitation, we only show the results for the Class II graphs. We use Computer B to test the GT Algorithm. The algorithm is implemented with C++. The results are tabulated in Table 2. A $k \times h$ cylinder implies a path of

Table 2. Computational results of the GT Algorithm and the comparison between the computation based lower bounds and the formula based ones. The column “cm” indicates the size of the cylinder minors found by the GT algorithm, “cm-time” is the time (in seconds) for finding it. The column “cmpt. LB” is the computation based lower bound implied by the cylinder minor, and “fmla. LB” is the formula based lower bound.

graphs	# of vertices	# of edges	bw	cm	cm-time	cmpt. LB	fmla. LB
lin105	105	292	8	8×4	0.09	31	7
kroB150	150	436	10	11×5	0.16	54	11
pr226	226	586	7	8×3	0.66	23	5
ch130	130	377	10	9×5	0.27	44	11
pr1002	1002	2972	21	20×10	8.57	199	54
d1655	1655	4890	29	24×12	17.73	287	104
u2152	2152	6312	31	24×13	31.96	311	119
pr2392	2392	7125	29	28×13	45.21	363	104
pcb3038	3038	9101	40	35×18	250.06	629	199
fl3795	3795	11326	25	21×12	324.524	251	77
fnl4461	4461	13359	48	41×23	302.87	942	287

length $k \times h - 1$, which is a (computation based) lower bound on the length of a longest path of this planar graph. From $\text{bw}(G) \leq 2\text{cm}(G)$, we have a (formula based) lower bound $\frac{(\text{bw}(G))^2}{8} - 1$.

Given a planar graph G and integers $k \geq 3$ and $h \geq 1$, the GT Algorithm either finds a $k \times h$ cylinder minor or asserts $\text{bw}(G) \leq k + 2h - 3$. Therefore, we try different combinations of k and h in order to maximize the lower bound (with $k \times h$ maximized) on the length of a longest path. For graphs in Table 2, the best lower bound is obtained when $k \approx 2h$. The computational results confirm that for all tested graphs, the computation based lower bounds obtained by the GT Algorithm are much better than the formula based ones. A path in a graph of n vertices has length at most $n - 1$. The computation based lower bound is a constant fraction of the length of the longest path. Moreover, the GT Algorithm is practical. For large planar graphs with thousands of edges, a lower bound on the longest path problem can be obtained by this algorithm within a few minutes.

4.2 Results for Grid/Cylinder Minors

The exact algorithm for computing the largest cylinder minor provides a tool for evaluating how close the cylinder minors found by approximation algorithms are to the largest ones. We implement this algorithm and test its performance on Class II graphs. Since the running time of this algorithm is not polynomial, the implementation can handle only small instances of graphs in a practical time and memory space. Computer B is used for testing the exact algorithm. The algorithm is implemented with C++. The results are tabulated in Table 3.

The computational results show that this exact algorithm can handle planar graphs of size up to about 300 edges in a practical time with 2 GByte memory.

Table 3 also shows the comparison between the size of cylinder minors found by the GT Algorithm and the ones found by the exact algorithm. For some

Table 3. Computational results of the optimal largest cylinder minor algorithm, and the closeness between the size of the optimal cylinder minors and the size of the cylinder minors found by the GT Algorithm. The column “cm-optimal” (“cm-GT”, resp.) is the cylinder minor found by this exact algorithm (the GT Algorithm, resp.), “time” indicates the time (in seconds) for computing the largest cylinder minor, and “time1” and “time2” are the time for answering “NO” for computing $k \times (h+1)$ cylinder minor and $(k+1) \times h$ cylinder minor, respectively. For “time1” and “time2”, the entries “–” mean that the algorithm cannot finish with 2 GByte memory. The entries of “cm” marked with “*” mean that the cylinder minor computed by the exact algorithm is not asserted to be the largest, since either “time1” or “time2” is not available.

graphs	# of edges	cm-optimal	time	time1	time2	cm-GT
eil51	140	7×4	1.21	0.04	0.03	7×4
eil76	215	8×5	0.07	0.04	0.03	8×5
pr76	218	8×6	4.4	0.05	0.03	8×5
rat99	279	7×6	0.78	0.11	553.61	7×6
rd100	286	$7 \times 6^*$	1.15	0.14	–	7×6
kroB100	284	$8 \times 6^*$	23.55	275.55	–	8×5
lin105	292	7×7	1.63	0.15	718.11	7×6
ch130	377	$9 \times 6^*$	271.22	0.14	–	9×5
pr144	393	$9 \times 4^*$	80.92	–	–	9×4

instances (pr76, kroB100, lin105 and ch130), the cylinder minors computed by the GT Algorithm are smaller than but very close to those computed by the exact algorithm. For other instances, the GT Algorithm computes the cylinder minors as large as the exact solutions. For small planar graphs, the computational results suggest that the quality of cylinder minors found by the GT Algorithm is very close to optimal. Therefore, we expect that for large planar graphs, the GT Algorithm still produces near-optimal solutions.

The computational study of the BGK Algorithm for computing large grid minors is reported for Class II graphs [3]. We compare the size of the grid minors found by the BGK Algorithm with that found by the GT Algorithm (taking $k = h$). Computer B is used for testing the GT Algorithm. The results are tabulated in Table 4. For the size of $k \times k$ grid minors, Table 4 shows that, among the 20 test instances, the GT Algorithm finds a larger grid minor for seven instances, a smaller grid minor for only two instances, and a grid minor of the same size for 11 instances, compared with the BGK Algorithm. Due to the differences of implementation details and computers used, we do not compare the running times of the two algorithms. From Table 4, it can be shown that the GT Algorithm is practical even for large planar graphs with thousands of edges. It is worth mentioning that the GT Algorithm can compute $k \times h$ grid minor without the constraint that $k = h$. Since a $k \times h$ cylinder/grid minor without the constraint $k = h$ gives a better lower bound on many bidimensional parameters, we conclude that the GT Algorithm is a practical tool for the computational study of bidimensionality theory based algorithms.

Table 4. Comparison of the computational results between the BGK Algorithm and the GT Algorithm. The column “bw” indicates the branchwidth of the graph, “gm-BGK” is the size of $k \times k$ grid minors found by the BGK algorithm, “cm-GT” is the size of $k \times h$ grid minors found by the GT Algorithm, and “cm-GT time” indicates the time for finding it (in seconds). If we restrict $k = h$, the size of $k \times k$ grid minors found by the GT Algorithm is shown in column “gm-GT”, and “gm-GT time” indicates the time for finding it (in seconds).

graphs	# of edges	bw	gm-BGK	gm-GT	gm-GT time	cm-GT	cm-GT time
eil51	140	8	4	4	0.06	7×4	0.05
lin105	292	8	5	6	0.1	7×6	0.1
ch130	377	10	5	6	0.3	9×5	0.27
pr144	393	9	5	5	0.25	9×4	0.28
kroB150	436	10	7	7	0.42	11×5	0.16
tsp225	622	12	9	7	0.22	14×5	0.21
pr226	586	7	5	5	0.2	11×3	0.62
a280	788	13	6	6	0.31	15×5	0.31
pr299	864	11	6	7	0.49	11×6	0.23
rd400	1183	17	10	10	1.3	17×8	0.89
pcb442	1286	17	9	9	0.86	15×8	1.16
u574	1708	17	11	13	2.04	15×12	2.65
p654	1806	10	7	7	0.82	7×7	0.82
d657	1958	22	11	12	4.9	24×8	2.04
pr1002	2972	21	12	13	15.39	20×10	8.57
rl1323	3950	22	13	15	17.09	17×14	14.28
d1655	4890	29	19	18	15.43	22×16	15.79
rl1889	5631	22	14	16	23.37	18×15	25.09
u2152	6312	31	22	22	33.98	22×23	36.57
pr2392	7125	29	16	18	21.76	18×19	30.14

5 Concluding Remarks

Bidimensionality theory provides a general framework for developing subexponential fixed parameter algorithms for solving many NP-hard problems. We conduct a computational study on a bidimensional algorithm for the longest path problem in planar graphs. For this purpose, we implement the DPBF Algorithm for exactly computing longest paths and the GT Algorithm for computing large cylinder minors. Our results show that the DPBF Algorithm can solve the problem for graphs with branchwidth up to 10 in a practical time. For graphs with large branchwidth, the cylinder minors computed by the GT Algorithm give a good estimation on the length of longest paths. We also design and implement an algorithm which computes exactly a largest cylinder minor for a planar graph. The computational results of this algorithm show that the cylinder minors found by the GT Algorithm are very close to optimal. Only a subset of the computational results of our work is presented in this paper. Readers may refer to [21] for more details of this study. To our best knowledge, this work is the first computational study on bidimensionality theory based algorithms. The tools developed and data obtained in this work may give a base for other studies on bidimensional algorithms. It is worth to perform computational studies on other bidimensional problems.

References

1. Bian, Z., Gu, Q.P.: Computing Branch Decomposition of Large Planar Graphs. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 87–100. Springer, Heidelberg (2008)
2. Bian, Z., Gu, Q.P., Marzban, M., Tamaki, H., Yoshitake, Y.: Empirical study on branchwidth and branch decomposition of planar graphs. In: Proc. of the 9th SIAM Workshop on Algorithm Engineering and Experiments (ALENEX 2008), pp. 152–165 (2008)
3. Bodlaender, H.L., Grigoriev, A., Koster, A.M.C.A.: Treewidth lower bounds with brambles. *Algorithmica* 51(1), 81–89 (2008)
4. Byers, T.H., Waterman, M.S.: Determining all optimal and near-optimal solutions when solving shortest path problems by dynamic programming. *Operations Research* 32(6), 1381–1384 (1984)
5. de Fraysseix, H., de Mendez, P.O.: PIGALE-Public Implementation of a Graph Algorithm Library and Editor. SourceForge project page, <http://sourceforge.net/projects/pigale>
6. Demaine, E., Fomin, F., Hajiaghayi, M., Thilikos, D.: Bidimensional parameters and local treewidth. *SIAM J. Discret. Math.* 18(3), 501–511 (2005)
7. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM (JACM)* 52(6), 866–893 (2005)
8. Demaine, E.D., Hajiaghayi, M.T.: Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica* 28(1), 19–36 (2008)
9. Demaine, E.D., Hajiaghayi, M.T., Thilikos, D.M.: The bidimensional theory of bounded-genus graphs. *SIAM Journal on Discrete Mathematics* 20(2), 357–371 (2007)
10. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. *Algorithmica* 58(3), 790–810 (2010)
11. Garey, M.R., Johnson, D.S., Tarjan, R.E.: The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.* 5(4), 704–714 (1976)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability, a Guide to the Theory of NP-Completeness. Freeman, New York (1979)
13. Gu, Q.P., Tamaki, H.: Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica* (to appear, 2011)
14. Gu, Q.P., Tamaki, H.: Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Transactions on Algorithms (TALG)* 4(3), 1–13 (2008)
15. Marzban, M., Gu, Q.P., Jia, X.: Computational study on planar dominating set problem. *Theoretical Computer Science* 410(52), 5455–5466 (2009)
16. Mehlhorn, K., Näher, S.: LEDA: A Platform for Combinatorial and Geometric Computing. Cambridge University Press (1999)
17. Reinelt, G.: TSPLIB—A traveling salesman problem library. *INFORMS Journal on Computing* 3(4), 376 (1991)
18. Robertson, N., Seymour, P., Thomas, R.: Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B* 62(2), 323–348 (1994)
19. Robertson, N., Seymour, P.D.: Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B* 52(2), 153–190 (1991)
20. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. *Combinatorica* 14(2), 217–241 (1994)
21. Wang, C.: Computational study on bidimensionality theory based algorithms. MSc Thesis, Simon Fraser University (August 2011)

Sorting, Searching, and Simulation in the MapReduce Framework

Michael T. Goodrich¹, Nodari Sitchinava², and Qin Zhang²

¹ Department of Computer Science, University of California, Irvine, USA
`goodrich@ics.uci.edu`

² MADALGO*, Department of Computer Science, University of Aarhus, Denmark
`{nodari, qinzhang}@madalgo.au.dk`

Abstract. We study the MapReduce framework from an algorithmic standpoint, providing a generalization of the previous algorithmic models for MapReduce. We present optimal solutions for the fundamental problems of all-prefix-sums, sorting and multi-searching. Additionally, we design optimal simulations of the well-established PRAM and BSP models in MapReduce, immediately resulting in optimal solutions to the problems of computing fixed-dimensional linear programming and 2-D and 3-D convex hulls.

1 Introduction

MapReduce [2][3] is a programming paradigm for designing parallel and distributed algorithms. Building on pioneering work by Feldman *et al.* [6] and Karloff *et al.* [12], our interest in this paper is in studying the MapReduce framework from an algorithmic standpoint. In the MapReduce framework, a computation is specified as a sequence of map, shuffle, and reduce steps that operate on a set $X = \{x_1, x_2, \dots, x_N\}$ of values:

- A *map step* applies a function, μ , to each value, x_i , to produce a finite set of key-value pairs (k, v) . To allow for parallel execution, the computation of the function $\mu(x_i)$ must depend only on x_i .
- A *shuffle step* collects all the key-value pairs produced in the previous map step, and produces a set of lists, $L_k = (k; v_1, v_2, \dots)$, where each such list consists of all the values, v_j , such that $k_j = k$ for a key k assigned in the map step.
- A *reduce step* applies a function, ρ , to each list $L_k = (k; v_1, v_2, \dots)$, formed in the shuffle step, to produce a set of values, y_1, y_2, \dots . The reduction function, ρ , is allowed to be defined sequentially on L_k , but should be independent of other lists $L_{k'}$ where $k' \neq k$.

The outputs from a reduce step can, in general, be used as inputs to another round of map-shuffle-reduce steps. Thus, a typical MapReduce computation is described as a sequence of map-shuffle-reduce steps that perform a desired action in a series of *rounds*.

* MADALGO is the Center for Massive Data Algorithmics, a center of the Danish National Research Foundation.

Evaluating MapReduce Algorithms. Since the shuffle step requires communication among computers over the network, we desire to minimize the number of rounds in a MapReduce algorithm (ideally, keeping it to a constant). However, there are several metrics that one can use to measure the efficiency of a MapReduce algorithm over the course of its execution, including the following.

- We can consider R , the *number of rounds* that the algorithm uses.
- If we let $n_{r,1}, n_{r,2}, \dots$ denote the mapper and reducer I/O sizes for round r , so that $n_{r,i}$ is the size of the inputs and outputs for mapper/reducer i in round r , then we define the *communication complexity of round r* as $C_r = \sum_i n_{r,i}$. We can also define $C = \sum_{r=0}^{R-1} C_r$, the *communication complexity* for the entire algorithm.
- We can let t_r denote the *internal running time* for round r , which is the maximum internal running time taken by a mapper or reducer in round r , where we assume $t_r \geq \max_i \{n_{r,i}\}$. We can also define *total internal running time*, $t = \sum_{r=0}^{R-1} t_r$, for the entire algorithm.

We can make a crude calibration of a MapReduce algorithm using the following:

- L : the latency L of the shuffle network: the number of steps that a mapper or reducer has to wait until it receives its first input in a given round.
- B : the bandwidth of the shuffle network: the number of elements in a MapReduce computation that can be delivered by the shuffle network in any time unit.

Given these parameters, a lower bound for the total running time, T , of an implementation of a MapReduce algorithm can be characterized as follows:

$$T = \Omega \left(\sum_{r=0}^{R-1} (t_r + L + C_r/B) \right) = \Omega(t + RL + C/B).$$

Memory-Bound and I/O-Bound MapReduce Algorithms. Note, that MapReduce allows design of trivial one-round algorithms that are actually sequential by mapping inputs to a single key and then implementing a sequential algorithm on the reducer with that key. To steer the programmers away from such sequential implementations, recent algorithmic formalizations of the MapReduce paradigm have focused primarily on optimizing the round complexity, R , while restricting the memory size or input/output size for reducers. Karloff *et al.* [12] define their MapReduce model, MRC, so that each reducer's I/O size is restricted to be $\mathcal{O}(N^{1-\epsilon})$ for some constant $0 < \epsilon < 1$, and Feldman *et al.* [6] define their model, MUD, so that reducer memory size is restricted to be $\mathcal{O}(\log^c N)$, for some constant $c \geq 0$, and reducers are further required to process their inputs in a single pass.

In this paper, we follow the I/O-bound approach, as it seems to correspond better to the way reducer computations are specified, but we take a somewhat more general characterization than Karloff *et al.* [12], in that we do not bound the I/O size for reducers explicitly to be $\mathcal{O}(N^{1-\epsilon})$, but instead allow it to be an arbitrary parameter:

- We define M to be an upper bound on the *I/O-buffer memory size* for all reducers used in a given MapReduce algorithm. That is, we predefine M to be a parameter and require that $\forall r, i : n_{r,i} \leq M$.

We can use M in the design and/or analysis of MapReduce algorithms. For instance, if each round of an algorithm has a reducer with an I/O size of at most M , then we say that this algorithm is an *I/O-memory-bound MapReduce algorithm* with parameter M and we can give a simplified lower bound on the time, T , for such an algorithm as

$$T = \Omega(R(M + L) + C/B).$$

Our Contributions. In Section 2 we present a BSP-like [13] computational framework which we prove to be equivalent to the I/O-memory-bound MapReduce model. This formulation is more familiar in the distributed algorithms community, making the design and analysis of algorithms more intuitive. The new formulation allows a simple simulation result of the BSP algorithms in the MapReduce model with no slowdown in the number of rounds, resulting in straightforward MapReduce implementations of a large number of existing algorithms for the BSP model and its variants.

In Section 3 we present simulation of CRCW PRAM algorithms in our generalized MapReduce model, extending the EREW PRAM simulation results of Karloff et al. [12]¹ (which also holds in our generalized model). Our simulation achieves only $\Theta(\log_M P)$ slowdown in the round complexity, which is asymptotically optimal for a generic simulation. Our CRCW PRAM simulation results achieve their efficiency through the use of an implicit data structure we call *invisible funnel trees*. It can be viewed as placing virtual multi-way trees rooted at the input items, which funnel concurrent read and write requests to the data items, but are never explicitly constructed.

For problems with no known constant time CRCW PRAM solutions we show that we can design efficient algorithms directly in our generic MapReduce framework. Specifically, in Section 4 using the idea of invisible funnel trees we develop solutions to the fundamental problems of *prefix sums* and randomized *indexing* of the input.

Finally, what is perhaps most unusual about the MapReduce framework is that there is no explicit notion of “place” for where data is stored nor for where computations are performed. This property is perhaps what led DeWitt and Stonebraker [4] to say that it does not support indexed searches. Nevertheless, in Section 5 we show that the MapReduce framework does in fact support efficient *multi-searching* – the problem of searching for a number of keys in a search tree of roughly equal size.

For ease of exposition let $\lambda = \log_M N$. All our algorithms exhibit $\mathcal{O}(\lambda)$ round and $\mathcal{O}(\lambda N)$ communication complexities. Note, that in practice it is reasonable to assume that $M = \Omega(N^\epsilon)$ for some constant $\epsilon > 0$, resulting in $\lambda = \mathcal{O}(1)$, i.e. constant round and linear communication complexities for all our algorithms.

2 Algorithmic Framework for I/O-memory-bound MapReduce

In this section we define a graph-based framework that can be implemented in the I/O-memory-bound MapReduce model with the same round and communication complexities and makes algorithm design simpler and more intuitive.

¹ Their original proof was identified for the CREW PRAM model, but there was a flaw in that version, which could violate the I/O-buffer-memory size constraint during a CREW PRAM simulation. Based on a personal communication, we have learned that the subsequent version of their paper will identify their proof as being for the EREW PRAM.

Consider a set V of computing nodes. Let $A_v(r)$ be a set of items, each of constant size (in words), associated with each node $v \in V$ in round r . $A_v(r)$ defines the state of v . Let f be a sequential function defined the same for all nodes. Function f takes as input the state $A_v(r)$ of a node v and returns a new set $B_v(r)$, in the process destroying $A_v(r)$ ². Each item of $B_v(r)$ is of the form (w, a) , where $w \in V$ and a is a new item. We define the following computation which proceeds in R rounds.

At the beginning of the computation only the input nodes v have non-empty states $A_v(0)$. The state of an input node consists of a single input item.

In round r , each node v with non-empty state $A_v(r) \neq \emptyset$ performs the following. First, v applies function f on $A_v(r)$. This results in the new set $B_v(r)$ and deletion of $A_v(r)$. Then, for each element $b = (w, a) \in B_v(r)$, node v sends item a to node w . Note that if $w = v$, then v sends a back to itself. As a result of this process, each node may receive a set of items from others. Finally, the set of received items at each node v defines the new state $A_v(r+1)$ for the next round. The items comprising the non-empty states $A_v(r)$ after R rounds define the outputs of the entire computation at which point the computation halts.

The number of rounds, R , defines the *round complexity* of the computation. The total number of all the items sent (or, equivalently, received) by the nodes in each round r defines the *communication complexity* C_r of round r (in words), that is, $C_r = \sum_v |B_v(r)|$. Finally, the communication complexity C of the entire computation is defined as $C = \sum_{r=0}^{R-1} C_r = \sum_{r=0}^{R-1} \sum_v |B_v(r)|$. Note that this definition implies that nodes v whose states $A_v(r)$ are empty at the beginning of round r do not contribute to the communication complexity. Thus, the set V of nodes can be much larger than the size of the input. But, as long as only a small number of them has non-empty $A_v(r)$ at the beginning of each round, the communication complexity of the computation is bounded. Let K_{max} be the upper bound on such nodes in any round.

Observe that during the computation, in order for node v to send items to node w in round r , v should know the label of the destination w , which can be obtained by v in the following possible ways (or any combination thereof): 1) the link (v, w) can be encoded in f as a function of the label of v and round r , 2) some node might send the label of w to v in the previous round, or 3) node v might keep the label of w as part of its state by constantly sending it to itself.

Thus, the above computation can be viewed as a computation on a *dynamic* directed graph $G = (V, E)$, where an edge $(v, w) \in E$ in round r represents a possible communication link between v and w during that round. The encoding of edges (v, w) as part of function f is equivalent to defining an *implicit* graph ; keeping all edges within a node throughout the computation is equivalent to defining a *static* graph. For ease of exposition, we define the following primitive operations that can be used within f at each node v :

- create an item; delete an item; modify an item; keep item x (that is, the item x will be sent to v itself by creating an item $(v, x) \in B_v(r)$); send an item x to node w (create an item $(w, x) \in B_v(r)$).

² Note that while f is defined the same for all nodes, since it takes $A_v(r)$ and, consequently, v as input, the output of f may vary at different nodes.

- create an edge; delete an edge. This is essentially the same as creating an item and deleting an item, since explicit edges are just maintained as items at nodes. These operations will simplify exposition when dealing with explicitly defined graphs G on which computation is performed.

The following theorem shows that the above framework captures the essence of computation in the MapReduce framework.

Theorem 1. *Let $G = (V, E)$ and f be defined as above such that in each round each node $v \in V$ sends, keeps and receives at most M items. Then computation on G with round complexity R and communication complexity C can be simulated in the I/O-memory-bound MapReduce model with the same round and communication complexities using K_{\max} mappers/reducers.*

Proof. We implement round $r = 0$ of computation on G in the I/O-memory-bound MapReduce framework using only the Map and Shuffle steps and every round $r > 0$ using the Reduce step of round $r - 1$ and a Map and Shuffle step of round r .

1. Round $r = 0$: (a) Computing $B_v(r) = f(A_v(r))$: Initially, only the input nodes have non-empty sets $A_v(r)$, each of which contains only a single item. Thus, the output $B_v(r)$ only depends on a single item, fulfilling the requirement of Map. We define Map to be the same as f , i.e., it outputs a set of key-value tuples (w, x) , each of which corresponds to an item (w, x) in $B_v(r)$. (b) Sending items to destinations: The Shuffle step on the output of the Map step ensures that all tuples with key w will be sent to the same reducer, which corresponds to the node w in G .
2. Round $r > 0$: First, each reducer v that receives a tuple $(v; x_1, x_2, \dots, x_k)$ (as a result of the Shuffle step of the previous round) simulates the computation at node v in G . That is, it simulates the function f and outputs a set of tuples (w, x) , each of which corresponds to an item in $B_v(r)$. We then define Map to be the identity map: On input (w, x) , output key-value pair (w, x) . Finally, the Shuffle step of round r completes the simulation of the round r of computation on graph G by sending all tuples with key w to the same reducer that will simulate node w in G in round $r + 1$.

Keeping an item is equivalent to sending it to itself. Thus, each node in G sends and receives at most M items implying that the above is a correct I/O-memory-bound MapReduce algorithm. \square

The above theorem gives an abstract way of designing MapReduce algorithms. More precisely, to design a MapReduce algorithm, we define graph G and a sequential function f to be performed at each node $v \in V$. This is akin to designing BSP algorithms and is a more intuitive way than defining Map and Reduce functions.

Note that in the above framework we can easily implement a global loop primitive spanning multiple rounds: each item maintains a counter that is updated at each round. We can also implement *parallel tail recursion* by defining the labels of nodes to include the recursive call stack identifiers.

3 Simulation Results

BSP simulation. The reader may observe that the generic MapReduce model of the previous section is very similar to the BSP model of Valiant [13], leading to the following conclusion.³

Theorem 2. *Given a BSP algorithm \mathcal{A} that runs in R super-steps with a total memory size N using $P \leq N$ processors, we can simulate \mathcal{A} using R rounds and $C = \mathcal{O}(RN)$ communication in the I/O-memory-bound MapReduce framework with reducer memory size bounded by $M = \lceil N/P \rceil$ using $\mathcal{O}(P)$ mappers/reducers.*

CRCW PRAM simulation. In this section we present a simulation of f -CRCW PRAM model, the strongest variant of the PRAM model, where concurrent writes to the same memory location are resolved by applying a commutative semigroup operator f , such as *Sum*, *Min*, *Max*, etc., to all values being written to the same memory address.

The input to the simulation of a PRAM algorithm \mathcal{A} is specified by an indexed set of P processor items, p_1, \dots, p_P , and an indexed set of initialized PRAM memory cells, m_1, \dots, m_N , where N is the total memory size used by \mathcal{A} . For ease of exposition we assume that $P = N^{\mathcal{O}(1)}$, i.e. $\log_M P = \mathcal{O}(\log_M N) = \mathcal{O}(\lambda)$.

The main challenge in simulating the algorithm \mathcal{A} in the MapReduce model is that there may be as many as P reads and writes to the same memory cell in any given step and P can be significantly larger than M , the memory size of reducers. Thus, we need to have a way to “fan in” these reads and writes. We accomplish this by using *invisible funnel trees*, where we imagine that there is a different implicit $\mathcal{O}(M)$ -ary tree that has the set of processors as its leaves and is rooted at each memory cell. Intuitively, our simulation algorithm involves routing reads and writes up and down these N trees. We view them as “invisible”, because we do not actually maintain them explicitly, since that would require $\Theta(PN)$ additional memory cells.

Each invisible funnel tree is an undirected⁴ rooted tree \mathcal{T} with branching factor $d = M/2$ and height $L = \lceil \log_d P \rceil = \mathcal{O}(\lambda)$. The root of the tree is defined to be at level 0 and leaves at level $L - 1$. We label the nodes in \mathcal{T} such that the k -th node (counting from the left and starting with index 0) on level l is defined as $v = (l, k)$. Then, we can identify the parent of a non-root node $v = (l, k)$ as $p(v) = (l - 1, \lfloor k/d \rfloor)$ and the q -th child of v as $w_q = (l + 1, k \cdot d + q)$. Thus, given a node $v = (j, (l, k))$, i.e., the k -th node on level l of the j -th tree, we can uniquely identify the label of its parent $p(v)$ and each of its d children and without maintaining the edges explicitly.

At the initialization step, we send m_j to the root node of the j -th tree, i.e., m_j is sent to node $(j, \text{root}) = (j, (0, 0))$. For each processor p_i ($1 \leq i \leq P$), we send π_i , the state of processor p_i to node u_i . Again, throughout the algorithm, each node keeps the items that it has received in previous rounds until they are explicitly deleted.

Each step of the PRAM algorithm \mathcal{A} is specified as a read sub-step, followed by a constant-time internal computation, followed by a write sub-step performed by each of P processors. We show how to simulate each of these sub-steps.

³ Due to space constraints, all omitted proofs can be found in the full version of the paper [9].

⁴ Each undirected edge is represented by two directed edges.

- 1a. **Bottom-up read phase.** For each processor p_i that attempts to read memory location m_j , node u_i sends an item encoding a read request (in the following we simply say a read request) to the i -th leaf node of the j -th tree, i.e. to node $(j, L - 1, i)$, indicating that it would like to read the contents of the j -th memory cell.

For $l = L - 1$ downto 1 do:

- For each node v at level l , if it received read request(s) in the previous round, then it sends a read request to its parent $p(v)$.

- 1b. **Top-down read phase.** The root node in the j -th tree sends the value m_j to child (j, w_k) if child w_k has sent a read request at the end of the bottom-up read phase.

For $l = 1$ to $L - 2$ do:

- For each node v at level l , if it received m_j from its parent in the previous round, then it sends m_j to all those children who have sent v read requests during the bottom-up read phase. After that v deletes all of its items.

Each leaf v sends m_j to the node u_i ($1 \leq i \leq P$) if u_i has sent v a read request at the beginning of the bottom-up read phase. After that v deletes all of its items.

2. **Internal computation phase.** At the end of the top-down phase, each node u_i receives its requested memory item m_j , performs the internal computation, updates the state π_i , and sends an item z encoding a write request to the node $(j, L - 1, i)$ if processor p_i wants to write z to the memory cell m_j .

3. **Bottom-up write phase.** For $l = L - 1$ downto 0 do:

- For each node v at level l , if it received write request(s) in the previous round, let z_1, \dots, z_k ($k \leq d$) be the items encoding those write requests. If v is not a root, it applies the semigroup function on input z_1, \dots, z_k , sends the result z' to its parent, and then deletes all of its items. Otherwise, if v is a root, it modifies its current memory item to z' .

When we have completed the bottom-up write phase, we are inductively ready for simulating the next step in the PRAM algorithm. In each round at most $\mathcal{O}(N + P)$ nodes are non-empty, thus, we have the following:

Theorem 3. *Given a CRCW PRAM algorithm \mathcal{A} with write conflicts resolved according to a commutative semigroup operator such that \mathcal{A} runs in T steps using P processors and N memory cells, we can simulate \mathcal{A} in the I/O-memory-bound MapReduce framework in the optimal $R = \Theta(\lambda T)$ rounds and with $C = \mathcal{O}(\lambda T(N + P))$ communication complexity using $\mathcal{O}(N + P)$ mappers/reducers.*

Applications. Theorem 2 immediately implies $\mathcal{O}(\lambda)$ round and $\mathcal{O}(\lambda N)$ communication complexity MapReduce solutions for problems of sorting and computing 2-dimensional convex hull via simulation of the BSP solutions [8, 7]. In the full version of the paper [9] we present an alternative randomized algorithm for sorting with the same complexity but which might be simpler to implement in practice than the simulation of the complicated BSP algorithm in [8].

By Theorem 3, we can simulate any CRCW (thus, also CREW) PRAM algorithm. For example, simulation of the PRAM algorithm of Alon and Megiddo [1] for linear programming in fixed dimensions produces a MapReduce algorithm with $\mathcal{O}(\lambda)$ round and $\mathcal{O}(\lambda N)$ communication complexities.

4 Prefix Sums and Random Indexing

The best known PRAM algorithm for prefix sums runs in $\mathcal{O}(\log^* N)$ time on Sum-CRCW model [5], resulting in a $\mathcal{O}(\lambda \log^* N)$ MapReduce algorithm (by Theorem [3]). In this section, we show how we can improve this result to $\mathcal{O}(\lambda)$ rounds.

The all-prefix-sum problem is usually defined on an array of integers. Since there is no notion of arrays in the MapReduce framework, but rather a collection of items, we define the all-prefix-sum problem as follows: given a collection of items x_i , where x_i holds an integer a_i and an index value $0 \leq i \leq N - 1$, compute for each item x_i a new value $b_i = \sum_{j=0}^i a_j$.

Lemma 1. *Given an indexed collection of N numbers, we can compute all prefix sums in the I/O-memory-bound MapReduce framework in $\mathcal{O}(\lambda)$ round and $\mathcal{O}(\lambda N)$ communication complexities.*

Proof (Sketch). The classic PRAM algorithm for computing prefix sums [10] can be viewed as a computation along a virtual binary tree on top of the inputs. To compute the prefix sums in MapReduce we replace the binary tree with the invisible funnel tree and perform similar 2-pass computation. The details of the algorithm are straightforward and are presented in the full version of the paper [9]. The depth of the invisible funnel tree is λ , resulting in the stated round and communication complexities. \square

Quite often, the input to the MapReduce computation is a collection of items with no particular ordering or indexing. If each input element is annotated with an estimate $N \leq \hat{N} \leq N^c$ of the size of the input, for some constant $c \geq 1$, then we can modify the all-prefix-sum algorithm to generate a random indexing for the input with high probability as follows.⁵

We define the invisible funnel tree \mathcal{T} on \hat{N}^3 leaves, thus, the height of the tree is $L = \lceil 3 \log_d \hat{N} \rceil = \mathcal{O}(\lambda)$. In the initialization step, each input node picks a random index i in the range $[0, \hat{N}^3 - 1]$ and sends $a_i = 1$ to the leaf node $v = (L - 1, i)$ of \mathcal{T} , and performing prefix sums computation on these values. Note that some leaf nodes might receive more than one item, so we make straightforward modifications to the prefix sums algorithm to address this complication (see [9] for details).

Lemma 2. *A random indexing of the input can be performed on a collection of data in the I/O-memory-bound MapReduce framework in $\mathcal{O}(\lambda)$ round and $\mathcal{O}(\lambda N)$ communication complexities with high probability.*

5 Multi-searching and Sorting

Let \mathcal{T} be a balanced binary search tree and Q be a set of queries. Let $N = |\mathcal{T}| + |Q|$. The problem of multi-search asks to annotate each query $q \in Q$ with a leaf $v \in \mathcal{T}$, such that the root-to-leaf search path for q in \mathcal{T} terminates at v .

⁵ Throughout the paper, when we say an event holds with high probability we mean the probability is at least $1 - 1/N$.

Goodrich [7] provides a solution to the multi-search problem in the BSP model. His solution first converts the binary search tree into a B-tree with the branching parameter $M = \lceil N/P \rceil$ (P is the number of BSP processors) in a natural way, i.e. each node of the B-tree corresponds to a subtree of the original binary tree of height $\Theta(\log_2 M)$. The height of the B-tree is $\Theta(\lambda) = \Theta(\log_M N)$. Then it replicates each node to relieve congestion during query routing by estimating the query load of each node by routing a small sample of the queries down the B-tree. The replicated nodes are connected to others in such a way that the set of nodes reachable from each replicated root node comprises the “skeleton” of the original B-tree (see Goodrich [7] for details). Call the resulting graph G . Finally, all the queries are distributed randomly across all the copies of the root nodes and propagated down in G to the leaf nodes (and their copies).

The depth of G is $\Theta(\lambda)$ with each level consisting of $\mathcal{O}(|Q|/M)$ B-tree nodes each containing $\Theta(M)$ routing elements. Thus, the size of G is $\mathcal{O}(|\mathcal{T}| + \lambda|Q|)$. And by Theorem 2 we obtain a MapReduce solution to multi-search with $\mathcal{O}(\lambda)$ round and $\mathcal{O}(\lambda|\mathcal{T}| + \lambda^2|Q|) = \mathcal{O}(\lambda^2 N)$ communication complexities.

In this section we present a solution that improves the communication complexity to $\mathcal{O}(\lambda N)$, while still achieving $\mathcal{O}(\lambda)$ round complexity; with high probability.

Multi-searching. To solve the multi-search problem in MapReduce with optimal $\mathcal{O}(\lambda N)$ communication complexity, consider a random partition of Q into λ subsets $Q_1, Q_2, \dots, Q_\lambda$ each containing $\mathcal{O}(N/\lambda)$ queries. By the above discussion, we clearly can construct a search structure G based on the query set Q_1 , consisting of $\Theta(\lambda)$ levels each containing $\mathcal{O}(N/\lambda)$ routing elements, i.e. $|G| = \mathcal{O}(N)$. We can also implement a MapReduce algorithm \mathcal{A} which propagates any query set Q' of size $|Q'| = \mathcal{O}(N/\lambda)$ down this search structure G .

To answer the multi-search queries for all queries Q , we proceed in $\Theta(\lambda)$ rounds. In round i , $1 \leq i \leq \lambda$, we feed new subset Q_i of queries to the $\mathcal{O}(N/\lambda)$ root nodes of G and propagate the queries down to the leaves using algorithm \mathcal{A} . This approach can be viewed as a pipelined execution of λ multi-searches on G .

Finally, to implement the random partitioning of Q into λ subsets, we perform a random indexing for Q (Lemma 2) and assign query with index j to subset $Q_{\lceil j/\lambda \rceil}$. A node v containing a query $q \in Q_i$ keeps q (by sending it to itself) until round i , at which point it sends q to the appropriate source node of G .

Theorem 4. *Given a binary search tree \mathcal{T} of size N , we can perform a multi-search of N queries over \mathcal{T} in the I/O-memory-bound MapReduce model in $\mathcal{O}(\lambda)$ rounds with $\mathcal{O}(\lambda N)$ communication with high probability.*

Proof (Sketch). Let L_1, \dots, L_λ be the λ levels of nodes of G . First, all query items in the first query batch Q_1 passes (i.e., routed down) L_j ($1 \leq j \leq \lambda$) in one round with probability at least $1 - \mathcal{O}(N/\lambda) \cdot N^{-c}$. This is because for each node v in L_j , at most M query items of Q_1 will be routed to v with probability at least $1 - N^{-c}$ for any constant c (by similar analysis as in [7]). Thus v can send all those queries to the next level in the next round without violating the output constraint. The statement follows by taking the union of all the nodes in L_j . Similarly, we can prove that any Q_i ($1 \leq i \leq \lambda$) can pass L_j ($1 \leq j \leq \lambda$) in one round with the same probability since all sets Q_i have equal distributions. Since there are λ batches of queries and they are fed

into G in a pipelined fashion, by union bound it follows that with probability at least $1 - \lambda^2 \cdot \mathcal{O}(N/\lambda) \cdot N^{-c} \geq 1 - 1/N$ (by choosing a sufficiently large constant c) the whole process completes within $\mathcal{O}(\lambda)$ rounds. The communication complexity follows directly because we only send $\mathcal{O}(|G| + |Q|) = \mathcal{O}(N)$ items in each round. \square

Applications and discussion. The solution to the multi-search problem combined with Theorem 2 immediately implies a solution to the problem of 3-dimensional convex hull via simulation of the BSP solution [7]. In the full version of the paper [9] we also present a simple sorting algorithm which uses the multi-searching solution and is easier to implement in practice than the direct simulation of the BSP sorting algorithm [8] using Theorem 2.

In [9] we also describe a queuing strategy that reduces the failure probability of Theorem 4 from $1/N$ to $N^{-\Omega(M)}$. The queuing algorithm may be of independent interest because it removes some of the constraints of the framework of Section 2.

Acknowledgments. We would like to thank Riko Jakob for pointing out the lower bound for our CRCW PRAM simulation.

References

1. Alon, N., Megiddo, N.: Parallel linear programming in fixed dimension almost surely in constant time. J. ACM 41(2), 422–434 (1994)
2. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
3. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. Commun. ACM 53(1), 72–77 (2010)
4. DeWitt, D.J., Stonebraker, M.: MapReduce: A major step backwards. Database Column (2008), <http://databasecolumn.vertical.com/database-innovation/mapreduce-a-major-step-backwards/>
5. Eisenstat, S.C.: $O(\log^* n)$ algorithms on a Sum-CRCW PRAM. Computing 79(1), 93–97 (2007)
6. Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., Svitkina, Z.: On distributing symmetric streaming computations. In: Teng, S.H. (ed.) SODA, pp. 710–719. SIAM (2008)
7. Goodrich, M.T.: Randomized fully-scalable BSP techniques for multi-searching and convex hull construction. In: SODA, pp. 767–776 (1997)
8. Goodrich, M.T.: Communication-efficient parallel sorting. SIAM Journal on Computing 29(2), 416–432 (1999)
9. Goodrich, M.T., Sitchinava, N., Zhang, Q.: Sorting, searching, and simulation in the mapreduce framework (2011), <http://arxiv.org/abs/1101.1902>
10. JáJá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Reading (1992)
11. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. In: 20th Annual ACM Symposium on Theory of Computing (STOC), pp. 334–343 (1988)
12. Karloff, H., Suri, S., Vassilvitskii, S.: A model of computation for MapReduce. In: Proc. ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 938–948 (2010)
13. Valiant, L.G.: A bridging model for parallel computation. Comm. ACM 33, 103–111 (1990)

External-Memory Multimaps

Elaine Angelino^{2,*}, Michael T. Goodrich^{1, **},
Michael Mitzenmacher^{2,***}, and Justin Thaler^{2,†}

¹ Dept. of Computer Science, University of California, Irvine
`goodrich@ics.uci.edu`

² School of Engineering and Applied Sciences, Harvard University,
`{michaelm,jthaler,elaine}@eecs.harvard.edu`

Abstract. Many data structures support dictionaries, also known as maps or associative arrays, which store and manage a set of key-value pairs. A *mymap* is a generalization that allows multiple values to be associated with the same key. For example, the inverted file data structure commonly used in search engines is a type of mymap, with words as keys and document pointers as values. We study the mymap abstract data type and how it can be implemented efficiently online in external memory frameworks, with constant expected I/O performance. The key technique used to achieve our results is a combination of cuckoo hashing using buckets that hold multiple items with a multiqueue implementation to cope with varying numbers of values per key. Our results are provably optimal up to constant factors.

1 Introduction

A *mymap* is a simple abstract data type (ADT) that generalizes the map ADT to support key-value associations in a way that allows multiple values to be associated with the same key. Specifically, it is a dynamic container, C , of key-value pairs, which we call *items*, supporting (at least) the following operations:

- $\text{insert}(k, v)$: insert the key-value pair, (k, v) . This operation allows for there to be existing key-value pairs having the same key k , but we assume w.l.o.g. that the particular key-value pair (k, v) is itself not already present in C .
- $\text{isMember}(k, v)$: return true if the key-value pair (k, v) is present in C .
- $\text{remove}(k, v)$: remove the key-value pair (k, v) from C . This operation returns an error condition if (k, v) is not currently in C .
- $\text{findAll}(k)$: return the set of all key-value pairs in C having key equal to k .
- $\text{removeAll}(k)$: remove from C all key-value pairs having key equal to k .
- $\text{count}(k)$: Return the number of values associated with key k .

* A preliminary version of this work appears as a Brief Announcement at SPAA 2011.

** Supported in part by the NSF under grants 0724806, 0713046, and 0847968, and by the ONR under MURI grant N00014-08-1-1015.

*** Supported in part by the NSF grants 0915922 and 0964473.

† Supported by a DoD NDSEG Fellowship, and partially by NSF grant CNS-0721491.

Surprisingly, we are not familiar with any previous discussion of this abstract data type in the theoretical algorithms and data structures literature. Nevertheless, abstract data types equivalent to the above ADT, as well as multimap implementations, are included in the C++ Standard Template Library (STL), Guava—the Google Java Collections Library, and the Apache Commons Collection 3.2.1 API. The existence of these implementations provides empirical evidence for the usefulness of this abstract data type. In this work we describe efficient external-memory implementations of the multimap ADT. Due to space constraints, this paper is abbreviated. Many more details and proofs are available in the full version [1].

Motivation: A primary motivation for studying multimaps is that associative data in the real world often exhibits extreme non-uniformities. For example, many real-world data sets follow a power law with respect to data frequencies indexed by rank. A standard example is the frequency of words in a corpus of documents. We could use a multimap data structure to allow us to retrieve all instances of a query word w in such a corpus, but would require one that could handle large skews in the number of values per key. In this case, the multimap could be viewed as providing dynamic functionality for a classic static data structure, known as an *inverted file* or *inverted index* (e.g., see Knuth [6]). Given a collection Γ of documents, an inverted file allows one to list, for any word w , all the places in Γ where w appears.

Another powerful motivation for studying multimaps is graphical data [2]. A multimap can represent a graph: keys correspond to nodes, values correspond to neighbors, findAll operations list all neighbors of a node, and removeAll operations delete a node from the graph. The degree distribution of many real-life graphs follow a power law, motivating efficient handling of non-uniformity.

Related Work: Inverted files have standard applications in text indexing (e.g., see Knuth [6]), and are important data structures for modern search engines. Several works expand inverted files to support incremental and batched insertions, based on hash tables or B-trees, but many do not support fast deletions. Büttcher and Clarke [3] consider the trade-offs for allowing for both insertions and deletions in an inverted file, and Guo *et al.* [5] describe a solution for performing such operations by using a type of B-tree. Blandford and Blelloch [2] consider dictionaries on variable-length strings, but not in an external memory model. Recent work by Pagh *et al.* [9] studies cache-oblivious hashing, and achieves extremely fast (expected average) query time under realistic assumptions on the cache, asymptotically matching known cache-aware solutions. Like all work on hashing that assumes each key has a single value, [9] does not support the fast findAll and removeAll operations as we do in this work.

Our work utilizes a variation on cuckoo hash tables. We assume the reader has some familiarity with such hash tables, as originally presented by Pagh and Rodler [8] (or as described on Wikipedia).

Finally, recent work by Verbin and Zhang [12] shows that in the external memory model, for any dynamic dictionary data structure with query cost $O(1)$,

the expected amortized cost of updates must be at least 1. As explained below, this implies our data structure is *optimal* up to constant factors.

Our Results: Our algorithms are for the standard two-level I/O model, which captures the memory hierarchy of modern computer architectures. In this model, there is a cache of size M connected to a disk of unbounded size, and the cache and disk are divided into blocks, where each block can store up to B items. Any algorithm can only operate on cached data, and algorithms must therefore make memory transfer operations, which read a block from disk into cache or vice versa. The cost of an algorithm is the number of I/Os required, with all other operations considered free. All of our time bounds hold even when $M = O(B)$, and we therefore omit reference to M throughout.

We provide an online implementation of the multimap abstract data type, where each operation must completely finish executing prior to our beginning execution of any subsequent operations. In describing our performance bounds, we use $\bar{O}(\cdot)$ to denote an expected bound, B to denote the block size, N to denote the number of key-value pairs, and n_k to denote the number of key-value pairs with key equal to k . All bounds are *unamortized*. Our bounds are $O(1)$ for `isMember(k, v)`, `remove(k, v)`, `removeAll(k)`, and `count(k)`; $\bar{O}(1)$ for `insert(k, v)`; and $O(1 + n_k/B)$ for `findAll(k)`. Our constructions are based on the combination of two external-memory data structures—external-memory cuckoo hash tables and multiqueues—which may be of independent interest.

Since we achieve query cost $O(1)$, the lower bound of [12] implies our $O(1)$ update cost is optimal up to constant factors. That we in addition achieve efficient `removeAll` and `findAll` operations demonstrates the strength of our results.

Finally, we simulate our suggested implementation to test our performance guarantees. While our implementation is not especially space efficient, yielding a memory utilization of between .32 and .39, it uses very few I/O operations, and we believe the ideas we present will yield more effective future implementations.

2 External-Memory Cuckoo Hashing

In this section, we describe external-memory versions of cuckoo hash tables with multiple items per bucket, which can be used to implement the map ADT when all key-value pairs are distinct. Later we use this approach in concert with multiqueues to support multiple values for the same key for the multimap ADT.

Cuckoo hash tables that store multiple items per bucket have been studied previously, having been introduced in [4]. Generally the analysis has been limited to buckets of a constant size (number of items) d . For our external-memory cuckoo hash table, each bucket can store B items, where B is a parameter defining our block size and is not necessarily a constant.

Formally, let $\mathcal{T} = (T_0, T_1)$ be a cuckoo hash table such that each T_i consists of $\gamma n/2$ buckets, where each bucket stores a block of size B , with $n = N/B$. One setting of particular interest is when $\gamma = 1 + \epsilon$ for some (small) $\epsilon > 0$, so that space overhead of the hash table is only an ϵ factor over the minimum possible. The items in \mathcal{T} are indexed by keys and stored in one of two locations, $T_0[h_0(k)]$

or $T_1[h_1(k)]$, where h_0 and h_1 are hash functions, and we assume for simplicity throughout the paper that all hash functions are completely random.

It should be clarified that, in some settings, the use of a cuckoo hash function may be unnecessary or even unwarranted. Indeed, if $B > c \log N$ for a suitable constant c and $\gamma = 1 + \epsilon$, we can instead use simple hash tables, with just one choice for each item; simple tail bounds suffice to show that with high probability all buckets will fit all the items that hash to it. Cuckoo hashing here allows us to avoid such “wide block assumptions,” giving a more general approach.

The important feature of the cuckoo hashing implementation is the way it may reallocate items in \mathcal{T} during an insertion. Standard cuckoo hashing, with one item per bucket, immediately evicts the previous (and only) item in a bucket when a new item is to be inserted in an occupied bucket. With multiple items per bucket, there is a choice available. We describe what is known in this setting, and how we modify it for our use here.

Let G be the *cuckoo graph*, where each bucket in \mathcal{T} is a vertex and, for each item x currently in C , we connect $T_0[h_0(x)]$ and $T_1[h_1(x)]$ as a directed edge, with the edge pointing toward the bucket it is not currently stored in. Suppose we wish to insert an item x into bucket X . If X contains fewer than B items, then we simply add x to X . Otherwise, we need to make room for the new item.

One approach for doing an insertion is to use a breadth first search on the cuckoo graph. The results of Dietzfelbinger and Weidling [4] show that for sufficiently large *constant* B , the expected insertion time is constant. Specifically, when $\gamma = 1 + \epsilon$ and $B \geq 16 \ln(1/\epsilon)$, the expected time to insert a new key is $(1/\epsilon)^{O(\log \log(1/\epsilon))}$, which is a constant. (This may require re-hashing all items in very rare cases when an item cannot be placed; the expected time remains constant.) Notice that if B grows in a fashion that is $\Omega(1)$, then breadth first search does not naturally take constant expected time, as even the time to look if items currently in the bucket can be moved will take $\Omega(B)$ time. (It might still take constant expected time, but this does not appear to follow from [4].)

For non-constant B , we can apply the following mechanism: we can use our buckets to mimic having B/c distinct subtables for some large constant c , where the i th subtable uses the ci/B th fraction of the bucket space, and each item is hashed into a specific subtable. For $B = O(N^\delta)$ for $\delta < 1$, each subtable will contain close to its expected number of items with high probability. Further, by choosing c suitably large one can ensure that each subtable is within a $1+\epsilon$ factor of its total space while maintaining an expected $(1/\epsilon)^{O(\log \log(1/\epsilon))}$ insertion time. Specifically, we have the following theorem:

Theorem 1. *Suppose that for a cuckoo hash table \mathcal{T} with at least $(1 + \epsilon)N/B$ blocks the block size satisfies $B = \Omega(1)$ and $B = O(N^\delta)$ for $\delta < 1$. Let $0 < \epsilon \leq 0.1$ be arbitrary and let C be a collection of N items. Suppose further we have B/c subtables, with $c = 16 \ln(1/\epsilon)$ with each item hashed to a subtable by a fully random hash function, and the hash functions for each subtable are fully random. Finally, suppose the items of C have been stored in \mathcal{T} by an algorithm using the partitioning process described above and the cuckoo hashing process. Then the expected time for the insertion of a new item x using a BFS is $(1/\epsilon)^{O(\log \log(1/\epsilon))}$.*

As noted in [4], a more practical approach is to use *random walk cuckoo hashing* in place of breadth first search cuckoo hashing, and we use random walk cuckoo hashing in the simulations in Section 4. Finally, item lookups and removals use a worst-case constant number of I/Os.

3 External-Memory Multimaps

We now build on the result of Section 2 to describe how to maintain a multimap that allows fast dynamic access in external memory.

The Primary Structure: To implement the multimap ADT, we begin with a primary structure that is an external-memory cuckoo hash table storing just the set of keys. In particular, each record, $R(k)$, in \mathcal{T} , is associated with a specific key, k , and holds the following fields:

- the key, k , itself
- the number, n_k , of key-value pairs in C with key equal to k
- a pointer, p_k , to a block X in a secondary table, \mathcal{S} , that stores items in C with key equal to k . If $n_k < B$, then X stores all the items with key equal to k (plus possibly some items with keys not equal to k). Otherwise, if $n_k \geq B$, then p_k points to a *first* block of items with key equal to k , with the other blocks of such items being stored elsewhere in \mathcal{S} .

This secondary storage is an external-memory data structure we are calling a *multiqueue*.

An External-Memory Location-Aware Multiqueue: The secondary storage that we need in our construction is a way to maintain a set \mathcal{Q} of queues in external memory. We assume the *header* pointers for these queues are stored in an array, \mathcal{T} , which in our external-memory multimap construction is the external-memory cuckoo hash table described above.

For any queue, Q , we wish to support the following operations:

- enqueue(x, H): add the element x to Q , given a pointer to its header, H .
- remove(x): remove x from Q . We assume in this case that each x is unique.
- isMember(x): determine whether x is in some queue, Q .

In addition, we wish to maintain all these queues in a space-efficient manner, so that the total storage is proportional to their total size. To enable this, we store all the blocks used for queue elements in a secondary table, \mathcal{S} , of blocks of size B each. Thus, each header record, H in \mathcal{T} , points to a block in \mathcal{S} .

Our intent is to store each queue Q as a doubly-linked list of blocks from \mathcal{S} . Unfortunately, some queues in \mathcal{Q} are too small to deserve an entire block in \mathcal{S} dedicated to storing their elements. So small queues must share their first block of storage with other small queues until they are large enough to deserve an entire block dedicated to their elements. Initially, all queues are assumed to be empty; hence, we initially mark each queue as *light*. In addition, the blocks in \mathcal{S} are initially empty; hence, we link the blocks of \mathcal{S} in a consecutive fashion as a doubly-linked list and identify this list as being the *free list*, F , for \mathcal{S} .

We set a heavy-size threshold at $B/3$ elements. When a queue Q stored in a block X reaches this size, we allocate a block from \mathcal{S} (taking a block off the free list F) exclusively to store elements of Q and we mark Q as **heavy**. Likewise, to avoid wasting space as elements are removed from a queue, we require any heavy queue Q to have at least $B/4$ elements. If a heavy queue's size falls below this threshold, then we mark Q as being light again and we force Q to go back to sharing space with other small queues. This may involve returning a block to the free list F . In this way, each block X in \mathcal{S} will either be empty or will have all its elements belonging to a single heavy queue or as many as $O(B)$ light queues. In addition, these rules imply that $O(B)$ element insertions are required to take a queue from the light state to the heavy state and $O(B)$ element removals are required to take a queue from the heavy state to the light state.

If a block X in \mathcal{S} is being used for light queues, then we order the elements in X according to their respective queues. Each block for a heavy queue Q stores previous and next pointers to the neighboring blocks in the linked list of blocks for Q , with the first such block pointing back to the header record for Q . As we show, this organization allows us to maintain our size and label invariants during execution of enqueue and remove operations.

One additional challenge is that we want to support the $\text{remove}(x)$ operation to have a constant I/O complexity. Thus, we cannot afford to search through a list of blocks of a queue looking for an element x we wish to remove. So, in addition to the table \mathcal{S} and its free list, F , and the headers for each queue in \mathcal{Q} , we also maintain an external-memory cuckoo hash table, \mathcal{D} , to be a dictionary that maps each queue element x to the block in \mathcal{S} that stores x . This allows our multiqueue to be **location-aware**, that is, to support fast searches to locate the block in \mathcal{S} that is holding any element x that belongs to some queue, Q .

We call any block in \mathcal{S} containing fewer than $B/4$ items **deficient**. To ensure that our multiqueue uses total storage proportional to its total size, we enforce the following two rules. Together, these rules guarantee that there are $O(N/B)$ deficient blocks in \mathcal{S} , and hence our multiqueue uses $O(N/B)$ blocks of memory.

1. Each block Y in \mathcal{T} stores a pointer d , called the deficient pointer, to a block $d(Y)$; the identity of this block is allowed to vary over time. We ensure that at all times, $d(Y)$ is the only (possibly) deficient block associated with Y that stores light queues.
2. Each heavy queue Q also stores in its header block a deficient pointer d to a block $d(Q)$. At all times, $d(Q)$ is the only (possibly) deficient block devoted to storing values for Q .

For the remainder of this subsection, we describe how to implement all multiqueue operations to obtain constant **amortized** expected or worst-case runtime. We show how to deamortize these operations in the full version of the paper [1].

The Split Action. As we perform enqueue operations, a block X may overflow its size bound, B . In this case, we need to split X in two, which we do by allocating a new block X' from \mathcal{S} (using its free list). We call X the **source** of the split, and X' the **sink** of the split. We then proceed depending on whether X contains elements from light queues or a single heavy queue.

1. X contains elements from light queues. We greedily copy elements from X into X' until X' has size at least $B/3$, keeping the elements from the same light queue together. Note that each light queue has less than $B/3$ elements, so this split will result in at least a $1/3\text{--}2/3$ balance.

To maintain our invariants, we must change the header records from X to X' for any queues that we just moved to X' . We achieve this by performing a look-up in \mathcal{T} for each key corresponding to a queue that was moved from X to X' , and modifying its header record, which requires $O(B)$ I/Os. Similarly, to support location awareness, we must also update the dictionary \mathcal{D} . So, for each element x that is moved to X' , we look up x in \mathcal{D} and update its pointer to now point to X' . In total this costs $O(B)$ I/Os.

2. X contains elements from a single heavy queue Q . In this case, we move ***no*** elements, and simply take a block X' from the free list and insert it as the head of the list of blocks devoted to Q , changing the header record H in \mathcal{T} to point to X' . We also change the deficient pointer d for Q to point to X' , and insert into X' the element that caused the split. This takes $O(1)$ I/O operations in total.

The Enqueue Operation. Given the above components, let us describe how we perform the enqueue and remove operations. We begin with the $\text{enqueue}(x, H)$ operation. We consider how this operation acts, depending on a few cases.

1. The queue for H is empty (hence, H is a null pointer and its queue is light). We examine the block Y from \mathcal{T} to which H belongs. If $d(Y)$ is null, we first take a block X of the free list and set $d(Y)$ to X before continuing. We follow the deficient pointer for Y , $d(Y)$, and add x to $d(Y)$. If this causes the size of $d(Y)$ to reach B , then we split $d(Y)$ as described above.
2. The queue Q for H is not empty. We proceed according to two cases.
 - (a) If Q is a light queue, we follow H to its block X in \mathcal{S} and add x to X . If this brings the size of Q above $B/3$, we perform a ***light-to-heavy transition***, taking a block X' off the free list, moving all elements in Q to X' , and marking Q as heavy. If this brings the size of X below $B/4$, we process X as in the remove operation below.
 - (b) If Q is heavy, we add x to $X = d(Q)$, the (possibly) deficient block for Q . If this brings the size of X to B , then we split X as described above.

Once the element x is added to a block X in \mathcal{S} , we then add x to the dictionary \mathcal{D} , and have its record point to X .

The Remove and isMember Operations. In both of these operations, we look up x in \mathcal{D} to find the block X in \mathcal{S} that contains x . In the $\text{isMember}(x)$ case, we complete the operation by simply looking for x in X . In the $\text{remove}(x)$ operation, we do this look up and then remove x from X if we find x . If this causes Q to become empty, then we update its header, H , to be null. In addition, if this operation causes the size of X to go below $B/4$, then we need to do some additional work, based on the following cases:

1. Q is a heavy queue.
 - (a) If X is the only block for Q , then Q should now be considered a light queue; hence, we continue processing it according to the case listed below where X contains only light queues. We refer to the entirety of this action as a ***heavy-to-light*** queue transition.
 - (b) Otherwise, if $X = d(Q)$, then we are done because $d(Q)$ is allowed to be deficient. If $X \neq d(Q)$, we proceed based on the following two cases:
 - i. ***d-alteration action:*** If the size of $d(Q)$ is at least $2B/3$, we simply update Q 's deficient pointer, d , to point to X instead of $d(Q)$.
 - ii. ***Merge action:*** If the size of $d(Q)$ is less than $2B/3$, then we move all of the elements of X into $d(Q)$ and we update the pointer in \mathcal{D} for each moved element. X is returned to the free list. We call X the source of the merge, and $d(Q)$ the sink. (Note that in this case, the size of $d(Q)$ becomes at most $11B/12$.)
2. X contains light queues (hence, no heavy queue elements). In this case, we visit the header H for Q . Let Y denote the block containing H .
 - (a) If $X = d(Y)$ we are done, since $d(Y)$ is allowed to be deficient.
 - (b) If $X \neq d(Y)$, let Z be the size of $d(Y)$.
 - i. ***d-alteration action:*** If $Z \geq 2B/3$ then we simply update d to point to X instead of $d(Y)$.
 - ii. ***Merge action:*** If $Z < 2B/3$, then we merge the elements in X into $d(Y)$, which now has size at most $11B/12$, and update pointers in \mathcal{D} and \mathcal{T} for the elements that are moved. We return X to the free list. We call X the source of the merge and $d(Y)$ the sink.

If a block X' is pointed to by any deficient pointer d , it is helpful to think of this as “protection” for X' from being the source of a merge. Once X' is afforded this protection, it will not lose it until its size is at least $2B/3$ (see the d -alteration action). At a high level, this will allow us to argue that if X and X' are respectively the source and sink of a merge action, neither X nor X' will be the source of a subsequent merge or split operation until it is the target of $\Omega(B)$ enqueue or remove operations, even though X' may have size very close to the deficiency threshold $B/4$. This will allow us to charge the $O(B)$ I/Os performed in any split or merge action to the $\Omega(B)$ operations that caused one of these blocks to shrink to size $B/4$ or grow to size B . We can deamortize these operations to obtain the following theorem.

Theorem 2. *We can implement a location-aware multiqueue so that the $\text{remove}(x)$ and $\text{isMember}(x)$ operations each use $O(1)$ I/Os, and the $\text{enqueue}(x, H)$ operation uses $O(1 + t(N))$ expected I/Os, where $t(N)$ is the expected number of I/Os needed to perform an insertion in an external-memory cuckoo table of size N .*

It should be clear from our description that, except for trivial cases (such as having only a constant number of elements), the space usage of our multiqueue implementation is within a constant factor of the optimal. We have not attempted to optimize this factor, though there is some flexibility in the multiqueue operations (such as when to do a split) that allow some optimization.

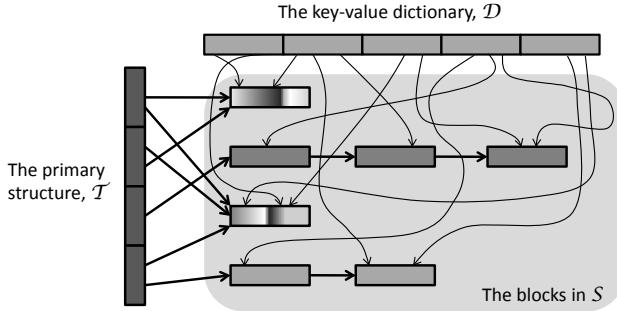


Fig. 1. The external-memory multimap

Combining Cuckoo Hashing and Location-Aware Multiqueues: We now describe how to construct an efficient external-memory multimap implementation by combining the data structures described above.

We store an external-memory cuckoo hash table, as described above, as our primary structure, \mathcal{T} , with each record pointing to a block in a multiqueue, \mathcal{S} , having an auxiliary dictionary, \mathcal{D} , implemented as yet another external-memory cuckoo hash table. The multimap ADT then functions as follows.

- $\text{insert}(k, v)$: To insert the key-value pair (k, v) we first perform a look up for k in \mathcal{T} . If there is already a record for k in \mathcal{T} , we increment its count. We then follow its pointer to the appropriate block X in \mathcal{S} , and add the pair (k, v) to \mathcal{S} , as in the enqueue multiqueue method. Otherwise we insert k into \mathcal{T} with a null header record and count 1 and then add the pair (k, v) to \mathcal{S} as in the enqueue multiqueue method.
- $\text{isMember}(k, v)$: This is identical to the $\text{isMember}(k, v)$ multiqueue operation.
- $\text{remove}(k, v)$: To remove the key-value pair (k, v) from \mathcal{C} , we perform a look up for (k, v) in \mathcal{D} . If there is no record for (k, v) in \mathcal{D} , we return an error condition. Otherwise, we follow this pointer to the appropriate block X of \mathcal{S} holding the pair (k, v) . We remove the pair (k, v) from \mathcal{S} and \mathcal{D} as in the remove multiqueue method, and decrement its count.
- $\text{findAll}(k)$: To return all key-value pairs in C having key k , we perform a look up for k in \mathcal{T} , and follow its pointer to the appropriate block of \mathcal{S} . If this is a light queue, then we return the items with key k . Otherwise, we return the entire block and all the other blocks of this queue as well.
- $\text{removeAll}(k)$: We give here a constant amortized time implementation, and provide a deamortization in [1]. To remove from C all key-value pairs having key k , we perform a look up for k in \mathcal{T} , and follow its pointer to the appropriate block X of \mathcal{S} . If this is a light queue, then we remove from X all items with key k and remove all affected pointers from \mathcal{D} ; if this causes X to become deficient, we perform a merge action or d -alteration action as in the remove multiqueue method. If this is a heavy queue, we walk through all blocks of this queue and remove all items from these blocks and return each block to the free list. We also remove all affected pointers from \mathcal{D} . Finally,

we remove the header record for k from \mathcal{T} , which implicitly sets the count of k to zero as well. We charge, in an amortized sense, the work for all the I/Os to the insertions that added these key-value pairs to C originally.

- $\text{count}(k)$: Return n_k , which we track explicitly for all keys k in \mathcal{T} .

Theorem 3. *One can implement the multimap ADT in external memory using $O(N/B)$ blocks of memory with the following: I/O performance: $O(1)$ for $\text{isMember}(k, v)$, $\text{remove}(k, v)$, $\text{removeAll}(k)$, and $\text{count}(k)$; $O(1)$ for $\text{insert}(k, v)$; and $O(1 + n_k/B)$ for $\text{findAll}(k)$.*

4 Experiments

We performed extensive simulations in order to explore how various settings of the design parameters affect I/O complexity and space usage, for both our basic algorithm and our deamortized algorithm. To summarize, in both versions our memory utilization was between .32 and .39 for all parameter settings tested, and the average I/O cost over all insert and remove operations is extremely low: never more than about 3.5 I/Os per operation. As expected, the cost distribution in the basic algorithm is bimodal – the vast majority (over 99.9%) of operations require about 4 I/Os, but a small fraction of operations require several hundred, with the maximum number of I/Os ranging between 400 and 650. In stark contrast, the deamortized implementation never requires more than a few dozen I/Os for any given operation; it also used slightly fewer I/Os on average. We have tested our performance against a B-tree variant, and preliminary tests show our performance is competitive.

References

1. Angelino, E., Goodrich, M.T., Mitzenmacher, M., Thaler, J.: External-Memory Multimaps. CoRR abs/1104.5533 (2011)
2. Blandford, D., Blelloch, G.: Compact dictionaries for variable-length keys and data with applications. ACM Trans. Alg. 4(2), 1–25 (2008)
3. Büttcher, S., Clarke, C.L.A.: Indexing time vs. query time: trade-offs in dynamic information retrieval systems. In: Proc. of 14th ACM Conf. on Information and Knowledge Management (CIKM), pp. 317–318 (2005)
4. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. Theoretical Computer Science 380, 47–68 (2007)
5. Guo, R., Cheng, X., Xu, H., Wang, B.: Efficient on-line index maintenance for dynamic text collections by using dynamic balancing tree. In: Proc. of 16th ACM Conf. on Information and Knowledge Management (CIKM), pp. 751–760 (2007)
6. Knuth, D.E.: Sorting and Searching. The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1973)
7. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, New York (2005)
8. Pagh, R., Rodler, F.: Cuckoo hashing. Journal of Algorithms 52, 122–144 (2004)

9. Pagh, R., Wei, Z., Yi, K., Zhang, Q.: Cache-Oblivious Hashing. In: Proc. of PODS, pp. 297–304 (2010)
10. Panigrahy, R.: Efficient hashing with lookups in two memory accesses. In: Proc. of the 16th Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 830–839 (2005)
11. Panigrahy, R.: Hashing, Searching, Sketching. Ph.D. thesis, Dept. of Computer Science, Stanford University (2006)
12. Verbin, E., Zhang, Q.: The limits of buffering: a tight lower bound for dynamic membership in the external memory model. In: Proc. STOC, pp. 447–456 (2010)

External Memory Orthogonal Range Reporting with Fast Updates

Yakov Nekrich*

Department of Computer Science, University of Chile, Chile
yakov.nekrich@googlemail.com

Abstract. In this paper we describe data structures for orthogonal range reporting in external memory that support fast update operations. The query costs either match the query costs of the best previously known data structures or differ by a small multiplicative factor.

1 Introduction

In the orthogonal range reporting problem a set of points is stored in a data structure so that for any d -dimensional query range $Q = [a_1, b_1] \times \dots \times [a_d, b_d]$ all points that belong to Q can be reported. Due to its fundamental nature and its applications, the orthogonal range reporting problem was studied extensively; we refer to e.g. [10, 9, 20, 4, 8] for a small selection of important publications. In this paper we address the issue of constructing dynamic data structures that support fast update operations in the external memory model.

External memory data structures for orthogonal range reporting also received significant attention, see e.g., [17, 18, 20, 6, 1, 14, 2, 15]. We refer to [19] for the definition of the external memory model and a survey of previous results. In particular, dynamic data structures for $d = 2$ dimensions are described in [17, 18, 6]. The best previously known data structure of Arge, Samoladas, and Vitter [6] uses $O((N/B) \log_2 N / \log_2 \log_B N)$ blocks of space, answers queries in $O(\log_B N + K/B)$ I/Os and supports updates in $O(\log_B N (\log_2 N / \log_2 \log_B N))$ I/Os; in [6], the authors also show that the space usage of their data structure is optimal. Recently, the first dynamic data structure that supports queries in $O(\log_B^2 N + K/B)$ I/Os in $d = 3$ dimensions was described [15].

All previously described external memory data structures with optimal or almost-optimal query cost need $\Omega((\log_B N \log_2 N) / \log_2 \log_B N)$ I/Os to support an insertion or a deletion of a point; see Table 1. This compares unfavorably with significantly lower update costs that can be achieved by internal memory data structures. For instance, the two-dimensional data structure of Mortensen [12] supports updates in $O(\log_2^f N)$ time for any constant $f > 7/8$. Moreover, the update costs of previously described external structures contain an $O(\log_2 N)$ factor. Since block size B can be large, achieving update cost that only depends on $\log_B N$ would be desirable. High cost of updates is also a drawback of the

* Supported in part by Millennium Institute for Cell Dynamics and Biotechnology (ICDB), Grant ICM P05-001-F, Mideplan, Chile.

Table 1. New data structures and some previous results for $d = 2$ dimensions. Our results are marked with an asterisk; \dagger denotes randomized results. The result in the first row of the table can be obtained from the result in [17] using a standard technique. The function $\text{ilog}(x)$ is the iterated \log^* function: $\text{ilog}(x)$ denotes the number of times we must apply the \log^* function to x before the result becomes ≤ 2 , where $\log^*(x) = \min\{t \mid \log_2^{(t)}(x) < 2\}$, and $\log_2^{(t)}(x)$ denotes the \log_2 function repeated t times.

Source	Query Cost	Update Cost	Space Usage
[17]	$O(\log_B N + \frac{K}{B})$	$O(\log_B N \log_2 N \log_2^2 B)$	$O((N/B) \log_2 N \log_2 B \log_2 \log_2 B)$
[18]	$O(\log_B N + \frac{K}{B} + \text{ilog}(B))$	$O(\log_2 N (\log_B N + (\log_2^2 N)/B))$	$O((N/B) \log_2 N)$
[6]	$O(\log_B N + \frac{K}{B})$	$O(\log_B N \log_2 N / \log_2 \log_B N)$	$O((N/B) \log_2 N / \log_2 \log_B N)$
*	$O(\log_B N + \frac{K}{B})$	$O(\log_B^{1+\varepsilon} N) \dagger$	$O((N/B) \log_2 N)$
*	$O(\log_B N + \frac{K}{B})$	$O(\log_2^2 N)$	$O((N/B) \log_2 N / \log_2 \log_B N)$
*	$O(\log_B N (\log_2 \log_B N)^2 + \frac{K}{B})$	$O(\log_B N (\log_2 \log_B N)^2)$	$O((N/B) \log_2 N)$

three-dimensional data structure described in [15]. Reducing the cost of update operations can be important in the dynamic scenario when the data structure must be updated frequently.

Our Results. We describe several data structures for orthogonal range reporting queries in $d = 2$ dimensions that achieve lower update costs. We describe two data structures that support queries in $O(\log_B N + K/B)$ I/Os. These data structures support updates in $O(\log_B^{1+\varepsilon} N)$ I/Os with high probability and in $O(\log_2^2 N)$ deterministic I/Os respectively. Henceforth ε denotes an arbitrarily small positive constant. We also describe a data structure that uses $O((N/B) \log_2 N)$ blocks of space, answers queries in $O(\log_B N (\log_2 \log_B N)^2)$ I/Os, and supports updates in $O(\log_B N (\log_2 \log_B N)^2)$ I/Os. All our results are listed in Table II.

Overview. The situations when the block size B is small and when B is not so small are handled separately. If the block size is sufficiently large, $B = \Omega(\log_2^4 N)$ for an appropriate choice of constant, our construction is based on the bufferization technique. We show that a batch of $O(B^{1/4})$ queries can be processed with $O(\log_B N)$ I/Os. Hence, we can achieve constant amortized update cost for sufficiently large B . In the case when B is small, $B = O(\log_2^4 N)$, we construct the base tree with fan-out $\log_2^\varepsilon N$ or the base tree with constant fan-out. Since $B = \text{polylog}_2(N)$, the height of the base tree is bounded by $O(\log_B N)$ or $O(\log_B N \log_2 \log_B N)$. Hence, we can reduce a two-dimensional query to a small number of simpler queries.

In section 2 we describe a data structure that supports three-sided reporting queries in $O(\log_B N + \frac{K}{B})$ I/Os and updates in $O(\frac{1}{B^\delta})$ I/Os if $B = \Omega(\log_2^4 N)$. Henceforth δ denotes an arbitrary positive constant, such that $\delta \leq 1/4$. In Appendix A, we generalize this result and obtain a data structure that supports updates in $O(1)$ I/Os and orthogonal range reporting queries in $O(\log_B N + \frac{K}{B})$ I/Os if $B = \Omega(\log_2^4 N)$. Thus if a block size is sufficiently large, there exists a data structure with optimal query cost and $O(1)$ amortized update cost. We believe that this result is of independent interest. Data structures for $B = O(\log_2^4 N)$ are described in section 4.

2 Three-Sided Range Reporting for $B = \Omega(\log_B^4 N)$

Three-sided queries are a special case of two-dimensional orthogonal range queries. The range of a three-sided query is the product of a closed interval and a half-open interval. In this section we assume that the block size $B \geq 4h \log_B^4 N$ for a constant h that will be defined later in this section. Our data structure answers three-sided queries with $O(\log_B N + K/B)$ I/Os and updates are supported in $O(1/B^\delta)$ amortized I/Os.

Our approach is based on a combination of external priority tree [6] with buffering technique [5]. Buffering was previously used to answer searching and reporting problems in one dimension. In this section we show that buffering can be applied to three-sided range reporting problem in the case when $B = \Omega(\log_B^4 N)$. At the beginning, we describe the external priority tree [6] data structure. Then, we show how this data structure can be modified so that a batch of B^δ updates can be processed in constant amortized time. Finally, we describe the procedure for reporting all points in a three-sided range $Q = [a, b] \times [c, +\infty)$.

The following Lemma is important for our construction.

Lemma 1. *A set S of $O(B^{1+\delta})$ points can be stored in a data structure that supports three-sided reporting queries in $O(K/B)$ I/Os, where K is the number of points in the answer; this data structure can be constructed with $O(B^\delta)$ I/Os.*

Proof: We can use the data structure of Lemma 1 from [6]. □

External Priority Tree. Leaves of the external priority tree contain the x -coordinates of points in sorted order. Every leaf contains $\Theta(B)$ points and each internal node has $\Theta(B^\delta)$ children. We assume throughout this section that the height of an external priority tree is bounded by $h \log_B N$. The *range* $rng(v)$ of a node v is the interval bounded by the minimal and the maximal coordinates stored in its leaves; we say that a point p belongs to (the range of) a node v if its x -coordinate belongs to the range of v . Each node is associated with a set $S(v)$, $|S(v)| = \Theta(B)$, defined as follows. Let $L(v)$ denote the set of all points that belong to the range of v . The set $S(v)$ contains B points with largest y -coordinates among all points in $L(v)$ that do not belong to any set $S(w)$, where w is an ancestor of v . Thus external priority tree is a modification of the priority tree with node degree $B^{O(1)}$, such that each node contains $\Theta(B)$ points.

The data structure $F(v)$ contains points from $\cup S(v_i)$ for all children v_i of v . By Lemma 1, $F(v)$ supports three-sided queries in $O(1)$ I/O operations. Using $F(v)$, we can answer three-sided queries in $O(\log_B N + K/B)$ I/Os; the search procedure is described in [6].

Supporting Insertions and Deletions. Now we describe a data structure that supports both insertions and deletions. We will show below how a batch of inserted or deleted points can be processed efficiently. The main idea is to maintain buffers with inserted and deleted points in all internal nodes. The buffer $D(v)$, $v \in T$, contains points that are stored in descendants of v and must be deleted. The buffer $I(v)$, $v \in T$, contains points that must be inserted into

sets $S(u)$ for a descendant u of v . A buffer can contain up to $B^{3\delta}$ elements. When a buffer $I(v)$ or $D(v)$ is full, we flush it into the children v_j of v ; all sets $I(v_j)$, $D(v_j)$, and $S(v_j)$ are updated accordingly.

Definitions of $S(v)$ and $F(v)$ are slightly modified for the dynamic structure. Every set $S(v)$ contains at most $2B$ points. If $S(v)$ contains less than $B/2$ points than $S(v_i) = \emptyset$ for each child v_i of v . The data structure $F(v)$ contains all points from $S(v_j) \cup I(v_j)$ for all children v_j of v . We store an additional data structure $R(v)$ in each internal node. $R(v)$ contains all points from $\cup S(v_i)$ for all children v_i of v . $R(v)$ can be constructed in $O(B^\delta)$ I/Os; we can obtain $B^{3\delta}$ points with highest y -coordinates stored in $R(v)$ in $O(1)$ I/Os. Implementation of $R(v)$ is very similar to implementation of $F(v)$.

Suppose that all points from the set \mathcal{D} , $|\mathcal{D}| = O(B^\delta)$, must be deleted. We remove all points from $\mathcal{D} \cap S(v_r)$ and $\mathcal{D} \cap I(v_r)$ from $S(v_r)$ and $I(v_r)$ respectively. We set $D(v_r) = D(v_r) \cup (\mathcal{D} \setminus S(v_r))$. When $D(v)$ for an internal node v is full, $|D(v)| = B^{3\delta}$, we distribute the points of $D(v)$ among the children v_j of v . Let $D_j(v)$ be the points of $D(v)$ that belong to the range of v_j and update $S(v_j)$, $D(v_j)$ as described above: We remove all points from $D_j(v) \cap S(v_j)$ and $D_j(v) \cap I(v_j)$ from $S(v_j)$ and $I(v_j)$ respectively. All points of $D_j(v) \setminus S(v_j)$ are inserted into $D(v_j)$. Finally, we update $F(v)$ and $R(v)$.

We can insert a batch \mathcal{I} of B^δ points using a similar procedure; see [16] for a description.

Answering Queries. Consider a query $Q = [a, b] \times [c, +\infty)$. Let π denote the set of all nodes that lie on the path from the root to l_a or on the path from the root to l_b , where l_a and l_b are the leaves that contain a and b respectively. Then all points inside the range Q are stored in sets $S(v)$ or $I(v)$, where the node v belongs to π or v is a descendant of a node that belongs to π . Two following facts play crucial role in the reporting procedure.

Fact 1 *Let w be an ancestor of a node v . For any $p \in S(v)$ and $p' \in S(w)$, $p.y < p'.y$. For any $p \in I(v)$ and $p' \in S(w)$, $p.y < p'.y$.*

Fact 2 *Suppose that a point $p \in S(v)$ is deleted from S (but p is not deleted from $S(v)$ yet). Then, p belongs to a set $D(w)$ for an ancestor w of v .*

We set the value of the constant h so that the height of T does not exceed $h \log_B N$. As follows from the Fact 2, the total number of deleted points in $S(v)$ is bounded by $h \cdot B^{3\delta} \log_B N \leq B/4$. Let $DEL(v) = \cup_{w=\text{anc}(v)} (D(w) \setminus \cup_{w'=\text{anc}(w)} I(w'))$, where $\text{anc}(u)$ denotes an ancestor of a node u . To wit, $DEL(v)$ is the set of all points p , such that p belongs to some set $D(w)$ for an ancestor w of v , but p does not belong to any $I(w')$ for an ancestor w' of w . By Fact 2 all points in $S(v) \cup I(v)$ that are already deleted from the data structure belong to $DEL(v)$. If the set $DEL(v)$ is known, then $DEL(v_i)$ for a child v_i of v can be constructed in $O(1)$ I/Os. Therefore we can construct $DEL(v)$ for all $v \in \pi$ in $O(\log_B N)$ I/Os.

The procedure that visits nodes $v \in \pi$ and some of its descendants and reports all points in $Q \cap S$ is similar to the procedure used in [6]. We provide a detailed description in the full version of this paper [16].

Our result is summed up in the following Lemma.

Lemma 2. *Suppose that $B^\delta \geq 4h \log_B N$ for a constant h defined above and some $\delta \leq 1/4$. Then there exists a data structure that uses $O(N/B)$ blocks of space and answers three-sided reporting queries in $O(\log_B N + K/B)$ I/Os. The amortized cost of inserting or deleting a point is $O(1/B^\delta)$.*

3 Two-Dimensional Range Reporting for $B = \Omega(\log^4 N)$

We maintain a constant fan-out tree T on the set of x -coordinates of all points. An internal node of T has at most eight children. A point p belongs to an internal node v , if its x -coordinate is stored in a leaf descendant of v . We assume that the height of T is bounded by $h_1 \log_2 N$. Each node v contains two secondary data structures that support three-sided queries of the form $[a, +\infty) \times [c, d]$ and $(-\infty, b] \times [c, d]$ respectively; both data structures contain all points that belong to v . We also store all points that belong to v in a B-tree sorted by their y -coordinates, so that all points with y -coordinates in an interval $[c, d]$ can be reported. The data structures for three-sided queries are implemented as described in Lemma 2. We implement the B-tree using the result of [5], so that updates are supported in $O(1/B^\delta)$ I/Os. Hence, updates on the secondary data structures are supported in $O(1/B^\delta)$ I/Os.

We say that a node v of T is *special* if the depth of v is divisible by $\lceil \delta \log_2 B / 3 \rceil$. To facilitate the query processing, buffers with inserted and deleted elements will be stored in the special nodes only. A node u is a *direct special descendant* of v if u is a special node, u is a descendant of v , and there is no other special node u' on the path from v to u . We denote by $\text{desc}(v)$ the set of direct special descendants of a node v . The set of nodes $\text{subset}(v)$ consists of the node v and all nodes w , such that w is a descendant of v and w is an ancestor of some node $u \in \text{desc}(v)$. In other words, every node w on a path from v to one of its direct special descendants belongs to $\text{subset}(v)$; the node v also belongs to $\text{subset}(v)$.

Let $\bar{I}(v)$ and $\bar{D}(v)$ denote the buffers of inserted and deleted points stored in a node $v \in T$. When a point is inserted, we add it to the buffer $\bar{I}(v_R)$, where v_R is the root of T . When a buffer $I(v)$ contains at least $B^{2\delta}$ elements, we visit every node $w \in \text{subset}(v)$ and insert all points $p \in I(v) \cap \text{rng}(w)$ into the secondary data structures of a node w . Then, we examine all nodes $u \in \text{desc}(v)$. For every $u \in \text{desc}(v)$, we insert all points $p \in \bar{I}(v) \cap \text{rng}(u)$ into $\bar{I}(u)$ and remove all points $p \in (\bar{I}(v) \cap \text{rng}(u)) \cap \bar{D}(u)$ from $\bar{D}(u)$. Finally, we set $I(v) = \emptyset$.

The total number of nodes in $\text{subset}(v)$ and $\text{desc}(v)$ is $O(B^\delta)$. Since each point is inserted into $O(\log_2 B)$ data structures and the total number of points is $O(B^{2\delta})$, all data structures in $\text{subset}(v)$ can be updated in $O(B^{2\delta} \log_2 B / B^\delta) = O(B^\delta \log_2 B)$ I/Os. We can also update the buffers $\bar{I}(u)$ and $\bar{D}(u)$ for each $u \in \text{desc}(v)$ in $O(1)$ I/Os. Hence, a buffer $I(v)$ can be emptied in $O(B^\delta \log_2 B)$ I/Os. Since a buffer $I(v)$ is emptied once after $\Theta(B^{2\delta})$ points were inserted into $I(v)$, the amortized cost of an insertion into $I(v)$ is $O(\log_2 B / B^\delta)$. An insertion of a point p into the data structure leads to insertions of p into $O(\log_B N)$ buffers

$I(v)$. Hence, the amortized cost of inserting a point p is $O(\log_2 N/B^\delta) = O(1)$. Deletions can be processed with a symmetric procedure.

Consider a query $Q = [a, b] \times [c, d]$. We identify the node v of T such that $[a, b] \subset rng(v)$, but $[a, b] \not\subset rng(v_i)$ for any child v_i of v . Suppose that $[a, b]$ intersects with $rng(v_l), \dots, rng(v_r)$ where $1 \leq l \leq r \leq 4$. All points $p \in S \cap Q$ are stored in the secondary structures of nodes v_l, \dots, v_r or in buffers of the special ancestors of v (possibly including the node v itself). We start by constructing sets $INS(v)$ and $DEL(v)$. The set $INS(v)$ contains all points p such that $p \in \overline{I}(w)$ for an ancestor u of v , but $p \notin \overline{D}(u')$ for an ancestor u' of u . The set $DEL(v)$ contains all points p such that $p \in \overline{D}(w)$ for an ancestor u of v , but $p \notin \overline{I}(u')$ for an ancestor u' of u . Only $O(\log_B N)$ ancestors of v are special nodes and every buffer stored in a special node contains at most $B^{2\delta}$ points. Hence, both $INS(v)$ and $DEL(v)$ can be constructed in $O(\log_B N)$ I/Os and contain $h_1 \cdot B^{2\delta} \log_B N \leq B/4$ points. We output all points of $p \in INS(v) \cap Q$ in $O(1)$ I/Os. Let \mathcal{V} be the list of all points p , such that p belongs to Q and p is stored in a child of v . The list \mathcal{V} can be generated as follows. First, we answer three-sided queries $[a, +\infty) \times [c, d]$ and $(-\infty, b] \times [c, d]$ on data structures for nodes v_l and v_r respectively. Then, we identify all points p stored in a node v_j , $l < j < r$, such that $c \leq p.y \leq d$. When the list \mathcal{V} is constructed, we traverse \mathcal{V} and output all points of \mathcal{V} that do not belong to the set DEL . The list \mathcal{V} can be generated and traversed in $O(\log_B N + \frac{|\mathcal{V}|}{B})$ I/Os. Since the total number of points in the answer is $K \geq |\mathcal{V}| - B/4$, all points of $\mathcal{V} \setminus DEL$ can be identified and reported in $O(\log_B N + \frac{|K|}{B})$ I/Os.

Our result is summed up in the following lemma

Lemma 3. *Suppose that $B^\delta \geq 4 \log_2 N$ for a constant $\delta \leq 1/4$. Then there exists a data structure that uses $O((N/B) \log_2 N)$ blocks of space and answers two-dimensional orthogonal range reporting queries in $O(\log_B N + K/B)$ I/Os. The amortized cost of inserting or deleting a point is $O(1)$.*

We can slightly improve the space usage by increasing the fan-out of the base tree. Our construction is the same as above, but every internal node has $\Theta(\log_B N)$ children. We also store an additional data structure $H(v)$ in every internal node v of T . For any $l < r$ and any $c \leq d$, $H(v)$ enables us to efficiently report all points p , such that $p.y \in [c, d]$ and p is also stored in a child v_j of v for $l < j < r$. The data structure $H(v)$ is described below.

Let $L(v)$ denote the list of all points that belong to v . Let $Y(v)$ be the set that contains y -coordinates of all points in $L(v)$. For every point $p \in L(v_l)$ and for all children v_l of v , $H(v)$ contains a “point” $\tau(p) = (p.y, \text{succ}(p.y, Y(v_l)))$. For a query c , $H(v)$ returns all points $p \in L(v)$ such that $\tau(p) \in (-\infty, c] \times [c, +\infty)$. In other words, we can report all points $p \in L(v)$ such that $p.y \leq c$ and $\text{succ}(p.y, Y(v_l)) \geq c$. An answer to query contains $O(\log_B N)$ points; at most one point for each child v_l . Using Lemma 2, $H(v)$ supports queries and updates in $O(\log_B N + K) = O(\log_B N)$ I/Os and updates in $O(1/B^\delta)$ amortized I/Os respectively.

We can report all points $p \in L(v_j)$ such that $p.y \in [c, d]$ and $l < j < r$ as follows. Using $H(v)$, we search for all points p , such that $p.y \leq c$ and

$\text{succ}(p.y, Y(v_l)) \geq c$. For every found p , $l < j < r$, we traverse the list $L(v_j)$ and report all points that follow p until a point p' , $p'.y > d$, is found. The total query cost is $O(\log_B N + K/B)$.

The global data structure supports insertions and deletions of points in the same way as shown in Lemma 3. The query answering procedure is also very similar to the procedure in the proof of Lemma 3. We identify the node v of T such that $[a, b] \subset \text{rng}(v)$, but $[a, b] \not\subset \text{rng}(v_i)$ for any child v_i of v . We also find the children v_l, \dots, v_r of v such that $[a, b]$ intersects with $\text{rng}(v_l), \dots, \text{rng}(v_r)$. The sets $\text{INS}(v)$, $\text{DEL}(v)$, and the list \mathcal{V} can be generated as described above. The only difference is that we identify all points p stored in nodes v_j , $l < j < r$, such that $c \leq p.y \leq d$ using the data structure $H(v)$.

Lemma 4. *Suppose that $B^\delta \geq 4h_1 \log_2 N$ for a constant h_1 defined in Appendix A and some $\delta \leq 1/4$. Then there is a data structure that uses $O((N/B) \log_2 N / \log_2 \log_B N)$ blocks of space and answers two-dimensional orthogonal range reporting queries in $O(\log_B N + K/B)$ I/Os. The amortized cost of inserting or deleting a point is $O(1)$.*

4 Two-Dimensional Range Reporting for small B

It remains to consider the case when the block size B is small. In this section we assume that $B = O(\log_2^4 N)$ and describe several data structures for this case.

Our first data structure is similar to structures in [18, 6]. It supports updates in $O(\log_B^2 N)$ I/Os, answers queries in $O(\log_B N + K/B)$ I/Os, and uses $O((N/B) \log_2 N / \log_2 \log_B N)$ blocks. We provide a description of this data structure in [16]. Combining this result with Lemma 4, we obtain the following Theorem

Theorem 1. *There is a data structure that uses $O((N/B) \log_2 N / \log_2 \log_B N)$ blocks of space and answers orthogonal range reporting queries in two dimensions in $O(\log_B N + \frac{K}{B})$ I/Os. Updates are supported in $O(\log_B^2 N)$ amortized I/Os.*

Constructing a standard range tree with constant fan-out we obtain a $O((N/B) \log_2 N)$ space data structure that supports queries and updates in $O(\log_B N (\log_2 \log_B N)^2 + K/B)$ and $O(\log_B N (\log_2 \log_B N)^2)$ I/Os respectively. A more extensive description of this data structure can also be found in [16].

Theorem 2. *There exists a data structure that uses $O((N/B) \log_2 N)$ blocks of space and answers orthogonal range reporting queries in two dimensions in $O(\log_B N (\log_2 \log_B N)^2 + \frac{K}{B})$ I/Os. Updates are supported in $O(\log_B N (\log_2 \log_B N)^2)$ amortized I/Os.*

Range Trees with $B^{O(1)}$ Fan-Out. Let $\varepsilon' = \varepsilon/10$. If points have integer coordinates, we can reduce the query cost by constructing a range tree with fan-out $B^{\varepsilon'}$. For every node v and every pair of indexes $i \leq j$, where v_i, v_j are the children of v , all points that belong to the children v_i, \dots, v_j of v belong to a list $L_{ij}(v)$. A data structure $E_{ij}(v)$ supports one-dimensional

one-reporting queries on a set of integers. That is, $E_{ij}(v)$ enables us to find for any interval $[c, d]$ some point $p \in L_{ij}(v)$ such that $p.y \in [c, d]$, if such p exists and if all points have integer coordinates. As described in [13], we can implement $E_{ij}(v)$ so that queries are supported in $O(1)$ time and updates are supported in $O(\log^{\varepsilon'})$ randomized time. Using $E_{ij}(v)$, it is straightforward to report all $p \in L_{ij}(v)$ with $p.y \in [c, d]$ in $O(K/B)$ I/Os. Consider a query $Q = [a, b] \times [c, d]$. We can find in $O(\log_B N)$ I/O operations $O(\log_B N)$ nodes u^t and ranges $[i_t, j_t]$, so that the x -coordinate of a point p belongs to $[a, b]$ if and only if p is stored in some list $L_{i_t j_t}(u^t)$. Hence, all points in Q can be reported by reporting all points in $L_{i_t j_t}(u^t)$ whose y -coordinates belong to $[c, d]$. The total query cost is $O(\log N / \log \log N) = O(\log_B N)$. However, the space usage is $O((N/B) \log_2^{1+8\varepsilon'} N)$ because each point is stored in $O(B^{2\varepsilon'} \log_B N) = O(\log_2^{1+8\varepsilon'} N)$ lists $L_{ij}(v)$. We can reduce the space usage if only parts of lists $L_{ij}(v)$ stored explicitly.

Let $L(v)$ denote the list of all points that belong to a node v sorted by their y -coordinates. We divide $L(v)$ into groups of points $G_s(v)$, $s = O(|L(v)|/B^{1+2\varepsilon'})$, so that each $G_s(v)$ contains at least $B^{1+2\varepsilon'}/2$ and at most $2B^{1+2\varepsilon'}$ points. Instead of $L_{ij}(v)$, we store the list $\tilde{L}_{ij}(v)$. The main idea of our space saving method is that we need to store points of $G_s(v)$ in the list $\tilde{L}_{ij}(v)$ only in the case when $G_s(v)$ contains a few points from $L_{ij}(v)$. Otherwise all relevant points can be found by querying the set $G_s(v)$ provided that $\tilde{L}_{ij}(v)$ contains a pointer to $G_s(v)$. Points and pointers are stored in each list $\tilde{L}_{ij}(v)$ according to the following rules. If $|L_{ij}(v) \cap G_s(v)| \leq B/2$, the list $\tilde{L}_{ij}(v)$ contains all points from $L_{ij}(v) \cap G_s(v)$. If $|L_{ij}(v) \cap G_s(v)| \geq 2B$, the list $\tilde{L}_{ij}(v)$ contains a pointer ptr_s to $G_s(v)$. We also store the minimal and maximal y -coordinates of points in $L_{ij}(v) \cap G_s(v)$ with each pointer to $G_s(v)$ from $\tilde{L}_{ij}(v)$. If $B/2 < |L_{ij}(v) \cap G_s(v)| < 2B$, $\tilde{L}_{ij}(v)$ contains either a pointer to $G_s(v)$ or all points from $L_{ij}(v) \cap G_s(v)$.

Instead of $E_{ij}(v)$, we will use several other auxiliary data structures. A data structure $\tilde{E}_{ij}(v)$ contains information about elements of $\tilde{L}_{ij}(v)$. For each point $p \in \tilde{L}_{ij}(v)$ we store $p.y$ in $\tilde{E}_{ij}(v)$; for every pointer ptr_s , $\tilde{E}_{ij}(v)$ contains both the minimal and the maximal y -coordinate associated with ptr_s . A data structure $E(v)$ contains the y -coordinates of all points in $L(v)$. Both $E(v)$ and all $E_{ij}(v)$ support one-reporting queries as described above. A data structure $H_s(v)$ supports orthogonal range reporting queries on $G_s(v)$. Using the data structure described in Lemma 1 of [6], we can answer three-sided reporting queries in $O(K/B)$ I/Os using $O(|G_s(v)|/B)$ blocks of space. Using the standard approach, we can extend this result to a data structure that uses $O((|G_s(v)| \log_2 B)/B)$ blocks and answers queries in $O(K/B)$ I/Os.

Now we show how we can report all points $p \in L_{ij}(v)$ with $p.y \in [c, d]$ without storing $L_{ij}(v)$. We can find an element e of $\tilde{L}_{ij}(v)$ with y -coordinate in $[c, d]$. Suppose that such e is found. Then, we traverse the list $\tilde{L}_{ij}(v)$ in $+y$ direction starting at e until a point p with $p.y > d$ or a pointer to $G_s(v)$ with the minimal y -coordinate larger than d is found. We also traverse $\tilde{L}_{ij}(v)$ in $-y$ direction until

a point p with $p.y < c$ or a pointer to $G_s(v)$ with the maximal y -coordinate smaller than c is found. For every pointer in the traversed portion of $\tilde{L}_{ij}(v)$, we visit the corresponding group $G_s(v)$ and report all points $p \in G_s(v) \cap L_{ij}(v)$ with $p.y \in [c, d]$. All relevant points in $G_s(v)$ can be reported in $O(K_s/B)$ I/Os using the data structure $H_s(v)$; here K_s denotes the number of points reported by $H_s(v)$. By definition of $\tilde{L}_{ij}(v)$, a set $G_s(v) \cap L_{ij}(v)$ contains at least $B/2$ points if there is a pointer ptr from $\tilde{L}_{ij}(v)$ to $G_s(v)$. Unless ptr is the first or the last element in the traversed portion of $\tilde{L}_{ij}(v)$, $G_s(v)$ contains B points from $[a, b] \times [c, d]$. Since B consecutive elements of the list $\tilde{L}_{ij}(v)$ contain either B points or at least one pointer to a group $G_s(v)$, the total cost of reporting all points in $L_{ij}(v)$ with $p.y \in [c, d]$ is $O(1 + K/B)$.

Now we consider the situation when there is no $e \in \tilde{E}_j(v)$, such that $e \in [c, d]$. In this case $L_{ij}(v)$ may contain some points from the range Q only if all points $p \in L(v)$ with $p.y \in [c, d]$ belong to one group $G_s(v)$. Using $E(v)$, we search for a point $p_s \in L(v)$ such that $p_s.y \in [c, d]$. If there is no such p_s , then $L(v) \cap Q = \emptyset$. Otherwise $p_s \in G_s(v)$ and we can report all points in $Q \cap G_s(v)$ in $O(1 + K/B)$ I/Os using $H_s(v)$. We need to visit $O(\log_B N)$ nodes of the range tree to answer the query; hence, the total query cost is $O(\log_B N + K/B)$ I/Os.

Since the lists $L_{ij}(v)$ are not stored, the space usage is reduced to $O((N/B) \log_2 N)$: Each list $\tilde{L}_{ij}(v)$ contains less than B points and at most one pointer for each group $G_s(v)$. Since $L(v)$ is divided into $O(|L(v)|/B^{1+2\varepsilon'})$ groups, the total size of all $L_{ij}(v)$ is $O(|L(v)|)$. All data structures $H_s(v)$ for all groups $G_s(v)$ use $O((|L(v)| \log_2 B)/B)$ blocks of space. Each point belongs to $O(\log_B N)$ nodes; therefore the total space usage is $O((N/B) \log_2 N)$.

When a new point p is inserted, we must insert it into $O(\log_B N)$ lists $L(v)$. Suppose that p is inserted into $G_s(v)$ in a node v . We insert p into $H_s(v)$ in $O(\log_2 B)$ I/Os; p is also inserted into up to $B^{2\varepsilon'} = O(\log_2^{8\varepsilon'} N)$ lists $\tilde{L}_{ij}(v)$. The one-dimensional reporting data structure for $\tilde{L}_{ij}(v)$ supports updates in $O(\log_2^{\varepsilon'} N)$ I/Os; hence, the total cost of inserting a point is $O(\log_2^{9\varepsilon'} N)$. For each pair $i \leq j$, we check whether the number of points in $L_{ij}(v) \cap G_s(v)$ equals to $2B$. Although the list $L_{ij}(v)$ is not stored, we can estimate the number of points in $L_{ij}(v) \cap G_s(v)$ by a query to the data structure $H_s(v)$. If $|L_{ij}(v) \cap G_s(v)| = 2B$, we remove all points of $\tilde{L}_{ij}(v) \cap G_s(v)$ from $L_{ij}(v)$ and insert a pointer to $G_s(v)$ into $\tilde{L}_{ij}(v)$. Points in a list $\tilde{L}_{ij}(v)$ are replaced with a pointer to a group $G_s(v)$ at most once for a sequence of $\Theta(B)$ insertions into $G_s(v)$. Hence, the amortized cost of updating $\tilde{L}_{ij}(v)$ because the number of points from $L_{ij}(v)$ in a group exceeds $2B$ is $O(\log_2^\varepsilon N)$ I/Os. Each insertion affects $O(\log_2^{8\varepsilon'} N)$ lists $L_{ij}(v)$. If the number of points in $G_s(v)$ equals $2B \log_2^{1+2\varepsilon'} N$, we split the group $G_s(v)$ into $G_1(v)$ and $G_2(v)$ of $B \log_2^{1+2\varepsilon'} N$ points each. Since up to B elements can be inserted and deleted into every list $\tilde{L}_{ij}(v)$, the amortized cost incurred by splitting a group is $O(\log_2^{9\varepsilon'} N)$. Thus the total cost of inserting a point into data structures associated with a node v is $O(\log_2^{9\varepsilon'} N)$ I/Os. Since a new point is inserted into $O(\log_2 N / \log_2 \log_2 N)$ nodes of the range tree, the total cost of an insertion is

$O(\log_2^{1+9\varepsilon'} N / \log_2 B) = O(\log_B^{1+\varepsilon} N)$. Deletions are processed in a symmetric way. Combining this result with Lemma 3, we obtain the following Theorem

Theorem 3. *Suppose that point coordinates are integers. There exists a data structure that uses $O((N/B) \log_2 N)$ blocks of space and answers orthogonal range reporting queries in two dimensions in $O(\log_B N + \frac{K}{B})$ I/Os. Updates are supported in $O(\log_B^{1+\varepsilon} N)$ amortized I/Os w.h.p. for any $\varepsilon > 0$.*

References

1. Afshani, P.: On Dominance Reporting in 3D. In: Halperin, D., Mehlhorn, K. (eds.) Esa 2008. LNCS, vol. 5193, pp. 41–51. Springer, Heidelberg (2008)
2. Afshani, P., Arge, L., Larsen, K.D.: Orthogonal Range Reporting in Three and Higher Dimensions. In: Proc. FOCS, pp. 149–158 (2009)
3. Aggarwal, A., Vitter, J.S.: The Input/Output Complexity of Sorting and Related Problems. Communications of the ACM 31(9), 1116–1127 (1988)
4. Alstrup, S., Brodal, G.S., Rauhe, T.: New Data Structures for Orthogonal Range Searching. In: Proc. FOCS 2000, pp. 198–207 (2000)
5. Arge, L.: The Buffer Tree: A Technique for Designing Batched External Data Structures. Algorithmica 37, 1–24 (2003)
6. Arge, L., Samoladas, V., Vitter, J.S.: On Two-Dimensional Indexability and Optimal Range Search Indexing. In: Proc. PODS 1999, pp. 346–357 (1999)
7. Arge, L., Vitter, J.S.: Optimal External Memory Interval Management. SIAM J. Comput. 32(6), 1488–1508 (2003)
8. Chan, T.: Persistent Predecessor Search and Orthogonal Point Location on the Word RAM. In: Proc. SODA 2011 (2011)
9. Chazelle, B.: A Functional Approach to Data Structures and its Use in Multidimensional Searching. SIAM J. on Computing 17, 427–462 (1988)
10. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and Related Techniques for Geometry Problems. In: Proc. STOC 1984, pp. 135–143 (1984)
11. Mehlhorn, K., Näher, S.: Dynamic Fractional Cascading. Algorithmica 5, 215–241 (1990)
12. Mortensen, C.W.: Fully Dynamic Orthogonal Range Reporting on RAM. SIAM J. Computing 35(6), 1494–1525 (2006)
13. Mortensen, C.W., Pagh, R., Patrascu, M.: On Dynamic Range Reporting in One Dimension. In: Proc. STOC 2005, pp. 104–111 (2005)
14. Nekrich, Y.: I/O-Efficient Point Location in a Set of Rectangles. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 687–698. Springer, Heidelberg (2008)
15. Nekrich, Y.: Dynamic Range Reporting in External Memory. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010, Part II. LNCS, vol. 6507, pp. 25–36. Springer, Heidelberg (2010)
16. Nekrich, Y.: External Memory Orthogonal Range Reporting with Fast Updates. CoRR abs/1106.6261 (2011)
17. Ramaswamy, S., Subramanian, S.: Path Caching: A Technique for Optimal External Searching. In: Proc. PODS 1994, pp. 25–35 (1994)
18. Subramanian, S., Ramaswamy, S.: The P-range Tree: A New Data Structure for Range Searching in Secondary Memory. In: Proc. SODA 1995, pp. 378–387 (1995)
19. Vitter, J.S.: External Memory Algorithms and Data Structures: Dealing with Massive Data. ACM Computing Surveys 33(2), 209–271 (2001)
20. Vengroff, D.E., Vitter, J.S.: Efficient 3-D Range Searching in External Memory. In: Proc. STOC 1996, pp. 192–201 (1996)

Analysis of Speedups in Parallel Evolutionary Algorithms for Combinatorial Optimization

(Extended Abstract)*

Jörg Lässig¹ and Dirk Sudholt^{2**}

¹ EAD Group, University of Applied Sciences Zittau/Görlitz,
02826 Görlitz, Germany

² CERCIA, University of Birmingham, Birmingham B15 2TT, UK

Abstract. Evolutionary algorithms are popular heuristics for solving various combinatorial problems as they are easy to apply and often produce good results. Island models parallelize evolution by using different populations, called islands, which are connected by a graph structure as communication topology. Each island periodically communicates copies of good solutions to neighboring islands in a process called migration. We consider the speedup gained by island models in terms of the parallel running time for problems from combinatorial optimization: sorting (as maximization of sortedness), shortest paths, and Eulerian cycles. Different search operators are considered. The results show in which settings and up to what degree evolutionary algorithms can be parallelized efficiently. Along the way, we also investigate how island models deal with plateaus. In particular, we show that natural settings lead to exponential vs. logarithmic speedups, depending on the frequency of migration.

1 Introduction

Evolutionary algorithms (EAs) are popular heuristics for various combinatorial problems as they often perform better than problem-specific algorithms with proven performance guarantees. They are easy to apply, even in cases where the problem is not well understood or when there is not enough time or expertise to design a problem-specific algorithm. Another advantage is that EAs can be parallelized easily [17]. This is becoming more and more important, given the development in computer architecture and the rising number of CPU cores. Developing efficient parallel metaheuristics is a very active research area [116].

A simple way of using parallelization is to use so-called offspring populations: new solutions (offspring) are created and evaluated simultaneously on different processors. Island models use parallelization on a higher level. Subpopulations, called islands, which are connected by a graph structure, evolve independently for some time and periodically communicate copies of good solutions to neighboring islands in a process called migration. Migration is typically performed

* The full version of the paper can be found at <http://arxiv.org/abs/1109.1766>.

** Dirk Sudholt was supported by EPSRC grant EP/D052785/1.

every τ iterations, the parameter τ being known as *migration interval*. A slow spread of information typically yields a larger diversity in the system, which can help for optimizing multimodal problems. For other problems a rapid spread of information (like setting $\tau = 1$ and migrating in every iteration) is beneficial, assuming low communication costs [9].

Despite wide-spread applications and a long history of parallel EAs, the theory of these algorithms is lagging far behind. Present theoretical work only concerns the study of the spread of information, or *takeover time*, in isolated and strongly simplified models (see, e.g., [13]) as well as investigations of island models on constructed test functions [7, 8, 10]. It is agreed that more fundamental research is needed to understand when and how parallel EAs are effective [15].

In this work we consider the speedup gained by parallelization in island models on illustrative problems from combinatorial optimization. The question is in how far using μ islands (each running an EA synchronously and in parallel) can decrease the number of iterations until a global optimum is found, compared to a single island. The number of iterations for such a parallel process is called *parallel optimization time* [9]. If the expected parallel optimization time is by a factor of $\Theta(\mu)$ smaller than the expected time for a single island, we speak of an (asymptotic) linear speedup. A linear speedup implies that a parallel and a sequential algorithm have the same total computational effort, but the parallel time for the former is smaller. We are particularly interested in the range of μ for which a linear speedup can be guaranteed. This degree of parallelizability depends on the problem, the EA running on the islands, and the parameters of the island model. Our investigation gives answers to the question how many islands should be used in order to achieve a reasonable speedup. Furthermore, it sheds light on the impact of design choices such as the communication topology and the migration interval τ as the speedup may depend heavily on these aspects.

Following previous research on non-parallel EAs [12], we consider various well-understood problems from combinatorial optimization: sorting as an optimization problem [14] (Section 4), the single-source shortest path problem [25] (Section 5), and the Eulerian cycle problem [3, 4, 11] (Section 6). As in previous studies, the purpose is not to design more efficient algorithms for well-known problems. Instead, the goal is to understand how general-purpose heuristics perform when being applied to a broad range of problems. The chosen problems contain problem features that are also present in more difficult, NP-hard problems. In particular, the Eulerian cycle problem contains so-called *plateaus*, that is, regions of the search space with equal objective function values. Our investigations pave the way for further studies that may include NP-hard problems.

2 Preliminaries

Island models evolve separate subpopulations—islands—individually for some time. Every τ generations at the end of an iteration, or *generation* using common language of EAs, copies of selected search points or *individuals* are sent as migrants to neighbored islands. Depending on their objective value f , or *fitness*,

Algorithm 1. Island model

Let $t := 0$. For all $1 \leq i \leq \mu$ initialize population P_0^i uniformly at random.

repeat

For all $1 \leq i \leq \mu$ do in parallel

Choose $x^i \in P_t^i$ uniformly at random.

Create y^i by mutation of x^i .

Choose $z^i \in P_t^i$ with minimum fitness in P_t^i .

if $f(y^i) \geq f(z^i)$ **then** $P_{t+1}^i = P_t^i \setminus \{z^i\} \cup \{y^i\}$ **else** $P_{t+1}^i = P_t^i$.

if $t \bmod \tau = 0$ and $t > 0$ **then**

Migrate copies of an individual with maximum fitness in P_{t+1}^i to all neighbored islands.

Let y^i be of maximum fitness among immigrants.

Choose $z^i \in P_{t+1}^i$ with minimum fitness in P_{t+1}^i .

if $f(y^i) > f(z^i)$ **then** $P_{t+1}^i = P_{t+1}^i \setminus \{z^i\} \cup \{y^i\}$.

Let $t = t + 1$.

migrants are in the target island's population after selection. The neighborhood of the islands is defined by a topology, a directed graph with the islands as nodes.

Algorithm 1 presents a general island model, formulated for maximization. Like in many previous studies for combinatorial optimization [12], we consider islands of population size only 1, running variants of the (1+1) EA or randomized local search (RLS). Both maintain a single search point and create a new search point in each generation by applying a mutation operator. This offspring replaces the current solution if its fitness is not worse. RLS uses local operators for mutation, while the (1+1) EA uses a stochastic neighborhood [12]. For the μ -vertex complete topology K_μ , the island model then basically equals what is known as (1+ μ) EA or (1+ μ) RLS, respectively, if we migrate in every generation ($\tau = 1$): the best of μ offspring competes with the parent as in the (1+1) EA.

We consider different topologies to account for different physical architectures and assume that the communication costs on the physical topology are so low that it allows us to focus on the parallel optimization time only. Unless specified otherwise, we assume $\tau = 1$.

3 Previous Work

The authors [9,10] presented general bounds for parallel EAs by generalizing the *fitness-level method* or *method of f -based partitions* (see Wegener [18]). The idea of the method is to divide the search space into sets A_1, \dots, A_m strictly ordered w.r.t. fitness: $A_1 <_f A_2 <_f \dots <_f A_m$ where $A <_f B$ iff $f(a) < f(b)$ for every $a \in A, b \in B$. In addition, A_m contains only global optima.

We say that a population-based algorithm (including populations of size 1) is in A_i or on level i if the current best individual in the population is in A_i . Elitist algorithms (defined as algorithms where the best solution in the population never worsens) can only increase the current level. The goal is to reach A_m . If s_i is a lower bound on the probability of leaving A_i towards any higher fitness level in one generation, the expected waiting time is at most $1/s_i$. As every level has to be left at most once, the expected optimization time is at most $\sum_{i=1}^{m-1} 1/s_i$.

The authors [9] generalized this method for island models that run elitist islands, for commonly used topologies. If migration is used in every generation, information about the current best fitness level is propagated to neighbored islands. This increases the number of islands searching for better fitness levels in parallel. The following theorem summarizes (a refinement of) our results.

Theorem 1. *Consider an island model with μ islands where each island runs an elitist EA. In every iteration each island sends copies of its best individual to all neighbored islands (i. e. $\tau = 1$). Each island incorporates the best out of its own individuals and its immigrants.*

For every partition $A_1 <_f \dots <_f A_m$ if s_i is a lower bound for the probability that in one generation an island in A_i finds a search point in $A_{i+1} \cup \dots \cup A_m$ then the expected parallel optimization time is bounded by

1. $2 \sum_{i=1}^{m-1} \frac{1}{s_i^{1/2}} + \frac{1}{\mu} \sum_{i=1}^{m-1} \frac{1}{s_i}$ for every unidirectional ring (a ring with edges in one direction) or any other strongly connected topology,
2. $3 \sum_{i=1}^{m-1} \frac{1}{s_i^{1/3}} + \frac{1}{\mu} \sum_{i=1}^{m-1} \frac{1}{s_i}$ for every undirected grid or torus graph with side lengths at least $\sqrt{\mu} \times \sqrt{\mu}$,
3. $m + \frac{1}{\mu} \sum_{i=1}^{m-1} \frac{1}{s_i}$ for the complete topology K_μ .

Assuming the fitness-level bound for the time $\sum_{i=1}^{m-1} \frac{1}{s_i}$ of a single island is asymptotically tight, all three bounds yield an asymptotic linear speedup in case the first summands are each of at most the same order as the second summand.

Apart from the different constants 2 and 3, denser topologies yield better upper bounds than sparse ones. This makes sense as with the fitness level argumentation a rapid spread of information gives the best estimates for the time an improvement is found. The motivation for studying sparse topologies is that they have lower communication cost and they yield a larger diversity. An example where this diversity is beneficial will be given in Section 6.

4 Sorting

We start our investigations with the first combinatorial problem for which EAs have been analyzed. Scharnow, Tinnefeld, and Wegener [14] considered the classical sorting problem as an optimization problem: given a sequence of n distinct elements from a totally ordered set, sorting is the problem of maximizing sortedness. W. l. o. g. the elements are $1, \dots, n$, then the aim is to find the permutation π_{opt} such that $(\pi_{\text{opt}}(1), \dots, \pi_{\text{opt}}(n))$ is the sorted sequence.

The search space is the set of all permutations π on $1, \dots, n$. Two different operators are used for mutation. An exchange chooses two indices $i \neq j$ uniformly at random from $\{1, \dots, n\}$ and exchanges the entries at positions i and j . A jump chooses two indices in the same fashion. The entry at i is put at position j and all entries in between are shifted accordingly. For instance, a jump with $i = 2$ and $j = 5$ would turn $(1, 2, 3, 4, 5, 6)$ into $(1, 3, 4, 5, 2, 6)$.

The (1+1) EA draws S according to a Poisson distribution with parameter $\lambda = 1$ and then performs $S + 1$ elementary operations. Each operation is either

Table 1. Upper bounds for expected parallel optimization times for the (1+1) EA and the corresponding island model with μ islands for sorting n objects

Algorithm	INV	HAM, LAS, EXC
(1+1) EA	$O(n^2 \log n)$ [14]	$O(n^2 \log n)$ [14]
island model on ring	$O\left(n^2 + \frac{n^2 \log n}{\mu}\right)$	$O\left(n^{3/2} + \frac{n^2 \log n}{\mu}\right)$
island model on torus	$O\left(n^2 + \frac{n^2 \log n}{\mu}\right)$	$O\left(n^{4/3} + \frac{n^2 \log n}{\mu}\right)$
island model on $K_\mu/(1+\mu)$ EA	$O\left(n^2 + \frac{n^2 \log n}{\mu}\right)$	$O\left(n + \frac{n^2 \log n}{\mu}\right)$

an exchange or a jump, where the decision is made independently and uniformly for each elementary operation. The resulting offspring replaces its parent if its fitness is not worse. The fitness function $f_{\pi_{\text{opt}}}(\pi)$ describes the sortedness of $(\pi(1), \dots, \pi(n))$. As in [14], we consider the following measures of sortedness:

- INV(π) measures the number of pairs (i, j) , $1 \leq i < j \leq n$, such that $\pi(i) < \pi(j)$ (pairs in correct order)
- HAM(π) measures the number of indices i such that $\pi(i) = i$ (elements at the correct position),
- LAS(π) equals the largest k such that $\pi(i_1) < \dots < \pi(i_k)$ for some $i_1 < \dots < i_k$ (length of the longest ascending subsequence),
- EXC(π) equals the minimal number of exchanges (of pairs $\pi(i)$ and $\pi(j)$) to sort the sequence, leading to a minimization problem.

The expected optimization time of the (1+1) EA is $\Omega(n^2)$ and $O(n^2 \log n)$ for all fitness functions. The upper bound is tight for LAS, and it is believed to be tight for INV, HAM, and EXC as well [14]. Theorem 1 yields the following.

Theorem 2. *The expected parallel optimization times of the (1+1) EA and the corresponding island model with μ islands are as in Table 1.*

For INV, all topologies guarantee a linear speedup only in case $\mu = O(\log n)$ and the bound $O(n^2 \log n)$ for the (1+1) EA is tight. The other functions allow for linear speedups up to $\mu = O(n^{1/2} \log n)$ (ring), $\mu = O(n^{2/3} \log n)$ (torus), and $\mu = O(n \log n)$ (K_μ), respectively (again assuming tightness, otherwise up to a factor of $\log n$). Note how the results improve with the density of the topology. HAM, LAS, and EXC yield much better guarantees for the island model than INV, though there is no visible performance difference for a single (1+1) EA.

5 Shortest Paths

We now consider parallel variants of the (1+1) EA for the single source shortest path problem (SSSP). Its complexity for the (1+1) EA has been first considered in [14]. An SSSP instance is given by an undirected connected graph with vertices $\{1, \dots, n\}$ and a distance matrix $D = (d_{ij})_{1 \leq i,j \leq n}$ where $d_{ij} \in \mathbb{R}_0^+ \cup \{\infty\}$ defines the length value for given edges from node i to node j . We are searching for shortest paths from a node s (w.l.o.g. $s = n$) to each other node $1 \leq i \leq n - 1$.

Table 2. Worst-case expected parallel optimization times for the (1+1) EA and the corresponding island model with μ islands for the SSSP on graphs with n vertices and m edges. The value ℓ is the maximum number of edges on any shortest path from the source to any vertex and $\ell^* := \max\{\ell, \ln n\}$. The second lines show a range of μ -values yielding a linear speedup, apart from a factor $\ln(en/\ell)$.

Algorithm	vertex-based mutation [14]	edge-based mutation [5]
(1+1) EA	$\Theta(n^2\ell^*)$ [2]	$\Theta(m\ell^*)$ [5]
island model on ring	$O\left(n^{3/2}\ell^{1/2} + \frac{n^2\ell \ln(en/\ell)}{\mu}\right)$ $\rightarrow \mu = O((n\ell)^{1/2})$	$O\left(m^{1/2}n^{1/2}\ell^{1/2} + \frac{m\ell \ln(en/\ell)}{\mu}\right)$ $\rightarrow \mu = O((m/n \cdot \ell)^{1/2})$
island model on torus	$O\left(n^{4/3}\ell^{1/3} + \frac{n^2\ell \ln(en/\ell)}{\mu}\right)$ $\rightarrow \mu = O((n\ell)^{2/3})$	$O\left(m^{1/3}n^{2/3}\ell^{1/3} + \frac{m\ell \ln(en/\ell)}{\mu}\right)$ $\rightarrow \mu = O((m/n \cdot \ell)^{2/3})$
i. m. on $K_\mu/(1+\mu)$ EA	$O\left(n + \frac{n^2\ell \ln(en/\ell)}{\mu}\right)$ $\rightarrow \mu = O(n\ell)$	$O\left(n + \frac{m\ell \ln(en/\ell)}{\mu}\right)$ $\rightarrow \mu = O(m/n \cdot \ell)$

A candidate solution is represented as a *shortest paths tree*, a tree rooted at s with directed shortest paths to all other vertices. We define a search point x as vector of length $n-1$, where position i describes the predecessor node x_i of node i in the shortest path tree. Note that infeasible solutions are possible in case the predecessors do not encode a tree. An elementary mutation chooses a vertex i uniformly at random and replaces its predecessor x_i by a vertex chosen uniformly at random from $\{1, \dots, n\} \setminus \{i, x_i\}$. We call this a vertex-based mutation. The (1+1) EA creates an offspring using S elementary mutations, where S is chosen according to a Poisson distribution with $\lambda = 1$.

The fitness function is defined as follows: Let $f(x) = (f_1(x), \dots, f_{n-1}(x))$ and $f_i(x)$ code the length of the path from s to i if it is described by x or $f_i(x) = \infty$ otherwise. The function $f(x)$ defines a partial order on the search points: $f(x) \leq f(x') \iff f_i(x) \leq f_i(x')$ for all $i \in \{1, 2, \dots, n-1\}$. That defines a multi-objective minimization problem but there is exactly one Pareto optimal fitness vector. The multi-objective (1+1) EA chooses an initial search point x uniformly at random and performs in each iteration a mutation step as described above. The new search point x' is accepted if $f(x') \leq f(x)$.

The expected parallel optimization time can be bounded as follows. Partition the vertices into *layers* $1, \dots, \ell$ such that the j -th layer contains all vertices having shortest paths of at most j edges. When shortest paths have been found for all layers $1, \dots, j$, shortest paths for vertices in layer $j+1$ can be found by assigning the correct predecessor in a lucky mutation. The probability for making an improvement is at least $i/(en^2)$, in case i vertices on layer j still need to find the right predecessors [14]. Applying Theorem 1 to all layers and considering a worst-case for the arrangement of layers yields the following upper bounds.

Theorem 3. *The expected parallel optimization times of the multi-objective (1+1) EA and the corresponding island model with μ islands are bounded according to the first column of Table 2.*

The upper bounds for the island models with constant μ match the expected time of the (1+1) EA in case $\ell = O(1)$ or $\ell = \Omega(n)$ as then $\ell \ln(en/\ell) = \Theta(\ell^*)$. In

other cases the upper bounds are off by a factor of $\ln(en/\ell)$. Table 2 also shows a range of μ -value for which the speedup is linear (if $\ell = O(1)$ or $\ell = \Omega(n)$) or almost linear, that is, when disregarding the $\ln(en/\ell)$ term. Note how the possible speedups significantly increase with the density of the topology.

Doerr and Johannsen [5] presented the following novel mutation operator. Imagine predecessors to be represented by a set of edges such that for each vertex v there is exactly one edge with end point v in the set. Each elementary mutation consists of choosing an edge (u, v) of the graph uniformly at random, adding it to the set, and removing the edge with end point v from the set. This saves the (1+1) EA from assigning predecessors that are not connected to the vertex and it decreases the expected running time of the (1+1) EA by a factor of $O(m/n^2)$. By Lemma 3 in [5] the (lower bound for the) probability of making an improvement is increased to $i/(em)$. The resulting bounds for the (1+1) EA using this mutation operator are shown in the second column of Table 2.

Note that the ranges for possible speedups are never greater than for vertex-based mutations. This is because edge-based mutations are sometimes more efficient and never worse than vertex-based mutations in the (1+1) EA.

6 Eulerian Cycles

Given an undirected, loopless Eulerian graph, the task is to find an Eulerian cycle, that is, a graph traversal where each edge is traversed exactly once. A straightforward representation leads to plateaus, i. e., regions of equal fitness that have to be overcome by an EA. The performance of EAs on Eulerian cycles has been investigated in [3, 4, 6, 11] where it has been shown that more sophisticated operators and representations lead to increasingly better performance.

Neumann [11] suggested a representation motivated by Hierholzer's algorithm. The idea of this algorithm is to subsequently concatenate cycles. This gives a *walk*, that is, a sequence of edges. When the walk includes all edges of the graph, an Eulerian cycle is created. Walks are represented by a permutation of the edges of the graph. The length of a walk (e_1, e_2, \dots, e_m) is the largest integer ℓ such that for all $1 \leq i \leq \ell - 1$ the edges e_i and e_{i+1} share a vertex. So, it is the length of a partial Euler walk. The first and last vertices of e_1 and e_ℓ are called *start* and *end* of the walk, resp. Neumann [11] as well as Doerr, Hebbinghaus, and Neumann [3] consider the length of the current walk as fitness and use jumps as mutation operators for RLS and the (1+1) EA. RLS always performs one jump, while the (1+1) EA chooses the number of jumps as in Section 4.

With the edge walk representation, fitness can be increased by appending a proper edge to the current walk. However, this operation is not always possible in case the current walk has closed a cycle. To see this, Neumann [11] defined the instance G' as the concatenation of two cycles C and C' , each consisting of $m/2$ edges, that share one common vertex v^* . This instance represents an asymptotic worst case for the time until an improvement is found.

If the current walk coincides with C , say, the current walk can only be extended by a single jump if it starts and ends with the vertex v^* . If it does not, the walk needs to be rotated until v^* becomes start and end of the current walk.

Table 3. Expected parallel optimization times for RLS and the island model running RLS on $\mu = \text{poly}(m)$ islands with topology T for computing an Eulerian cycle on G' . “Frequent migrations” is $\tau \cdot \text{diam}(T) \cdot \mu = O(m^2)$ for unrestricted jumps and $\tau \cdot \text{diam}(T) \cdot \mu = O(m)$ for symmetrically restricted ones, respectively. “Rare migrations” is $\tau \geq m^3$ and $\tau \geq m^2$, respectively.

Mutation operator	RLS	par. RLS, frequent migr.	par. RLS, rare migr.
Unrestricted	$\Theta(m^4)$ [11, 3]	$\Omega(m^4 / (\log \mu))$	$O(m^3 + 3^{-\mu} \cdot m^4)$
Restricted, symmetric	$\Theta(m^3)$	$\Omega(m^3 / (\log \mu))$	$O(m^2 + 3^{-\mu} \cdot m^3)$
Restricted, asymmetric	$\Theta(m^2)$	$O(m^2)$	$O(m^2)$

Rotations can be done by a jump with parameters $(1, m/2)$ or $(m/2, 1)$. As the fitness of all possible rotations of C is equal, the algorithm has to search on a plateau. Since the two above jumps are equally likely, rotating C with RLS corresponds to a fair random walk. With constant probability, the cycle needs to be rotated by a distance of $\Theta(m)$. This takes an expected number of $\Theta(m^2)$ steps of the random walk. As only two out of $m(m-1)$ possible jump operations are accepted, waiting for accepted jumps yields an additional factor of $\Theta(m^2)$. The expected optimization time of both RLS and (1+1) EA on G' is $\Theta(m^4)$ [11].

G' is a simple and natural instance as it represents the key features of the problem in a very clear way. It represents a worst-case for a single fitness level. It is not necessarily a global worst case as there is only one difficult fitness level, leaving a gap of m to a general upper bound of $O(m^5)$ for all Eulerian graphs [11]. For simplicity, we focus on RLS instead of the (1+1) EA—here, both have equal asymptotic performance anyway [3, 11]. Results are summarized in Table 3.

We give an example where parallelization does not reduce the parallel optimization time in any meaningful way. It can be shown that on G' a single island with constant probability arrives at a solution where the current walk equals one of the two cycles and the cycle has to be rotated by a distance of $\Theta(m)$. If the migration interval is small enough (depending on the number of islands and the diameter of the topology), there is further a constant probability that this solution was spread throughout all islands. As only strictly better immigrants are considered for inclusion, all islands perform independent random walks. As the time for completing the random walk is highly concentrated, the expected time until the first island finds an improvement is still $\Omega(m^4 / (\log \mu))$.

Theorem 4. Consider the island model with an arbitrary strongly connected topology T running RLS with jumps on each island. If $\tau \cdot \text{diam}(T) \cdot \mu = O(m^2)$ then the expected number of generations on G' is at least $\Omega(m^4 / (\log \mu))$.

Using any polynomial number of islands only reduces the expected optimization time by at most a log-factor. However, in other settings parallelization can help dramatically. One positive effect of an island model is that islands can make different decisions on how to extend the current walk. On G' this can make a difference between reaching the plateau and avoiding it completely.

In the beginning RLS typically evolves a walk on one of the two cycles C and C' . If v^* , the vertex connecting C and C' , is included in the current walk, the

walk can either be extended towards the “opposite” cycle or it can move past v^* and close the current cycle. In the former case a Eulerian cycle can be found easily by adding edges one-by-one. But in the latter case RLS has closed a cycle prematurely and it now has to rotate the walk to be able to include edges from the opposite cycle. This rotation dominates the expected running time.

Parallelization can help to make the right decision through independent evolution. If islands are run in parallel and if they evolve independently for at least $\tau \geq m^3$ generations, they tend to make independent decisions. This includes the case where no migration happens at all. The islands that have made the good decisions finish first, in expected time $\Theta(m^3)$. The remaining islands need $\Theta(m^4)$ steps in expectation. The probability of making a good decision is at least $2/3$ as a walk ending at v^* can be extended by either of 3 edges, two of which lead to the opposite cycle; all 3 edges have the same probability for being added. Hence, the probability that a rotation—and time $\Theta(m^4)$ —is needed is $3^{-\mu}$.

Theorem 5. *The island model running RLS on $\mu \leq \text{poly}(m)$ islands, $\tau \geq m^3$, and an arbitrary topology optimizes G' in expected $O(m^3 + 3^{-\mu} \cdot m^4)$ generations.*

The choice $\mu = \log_3 m$ leads to an expected parallel time of $O(m^3)$. This is a superlinear and, technically, even an exponential speedup. This is the first proof that island models can lead to a superlinear speedup on problems from combinatorial optimization.

The above result generalizes to instances where at v^* more than two cycles come together. On other graphs the probability of not closing a cycle prematurely is exponentially small [3] and no speedups are possible. Details are omitted.

The results seen so far can be improved by restricting the mutation operator. The length of the current walk can only be increased in RLS if an edge jumps to either position 1 or $\ell + 1$. Choosing the second parameter uniformly from $\{1, \ell + 1\}$ (called a symmetric restriction) decreases all time bounds by a factor of $\Theta(m)$ (see Table 3). The authors of [3] introduced an asymmetric jump operator where the second parameter is fixed to 1, i.e., all edges are prepended to the current walk. This innocent-looking modification makes rotating cycles much easier as rotations are only possible in one direction. This removes the random-walk behavior, implying that the performance difference between frequent and rare migrations breaks down. It follows from Theorem 2 in [3] that then the island model running RLS with this operator finds an optimum on G' in expected $O(m^2)$ generations, for any topology.

7 Conclusions

Considering speedups of island models has led to a surprising richness of results. For sorting linear speedups are possible, but the guarantees for parallelizability significantly depend on the measure of sortedness and the topology. The single-source shortest paths problem also allows for linear speedups, the maximum number of islands depending on the topology and the mutation operator. For Eulerian cycles results are inconclusive. Parallelization does not always help to

speed up search on plateaus. However, it can help in some cases by avoiding plateaus if decisions where to extend the current edge walk are made correctly.

Also the parameters of the island model play a key role. On the same natural instance G' speedups vary grossly from exponential up to $\mu = O(\log m)$ for rare or no migrations to at most logarithmic speedups, if migration is used too frequently and diversity is lost. Speedups also vary with the mutation operator.

References

1. Alba, E.: *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience (2005)
2. Doerr, B., Happ, E., Klein, C.: A tight analysis of the (1+1)-EA for the single source shortest path problem. In: Proc. of CEC 2007, pp. 1890–1895. IEEE (2007)
3. Doerr, B., Hebbinghaus, N., Neumann, F.: Speeding up evolutionary algorithms through asymmetric mutation operators. *Evolutionary Computation* 15, 401–410 (2007)
4. Doerr, B., Johannsen, D.: Adjacency list matchings—an ideal genotype for cycle covers. In: Proc. of GECCO 2007, pp. 1203–1210. ACM (2007)
5. Doerr, B., Johannsen, D.: Edge-based representation beats vertex-based representation in shortest path problems. In: Proc. of GECCO 2010, pp. 759–766. ACM (2010)
6. Doerr, B., Klein, C., Storch, T.: Faster evolutionary algorithms by superior graph representation. In: Proc. of FOCI 2007, pp. 245–250. IEEE (2007)
7. Lässig, J., Sudholt, D.: The benefit of migration in parallel evolutionary algorithms. In: Proc. of GECCO 2010, pp. 1105–1112 (2010)
8. Lässig, J., Sudholt, D.: Experimental Supplements to the Theoretical Analysis of Migration in the Island Model. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 224–233. Springer, Heidelberg (2010)
9. Lässig, J., Sudholt, D.: General Scheme for Analyzing Running Times of Parallel Evolutionary Algorithms. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 234–243. Springer, Heidelberg (2010)
10. Lässig, J., Sudholt, D.: Adaptive population models for offspring populations and parallel evolutionary algorithms. In: Proc. of FOGA 2011, pp. 181–192. ACM Press (2011)
11. Neumann, F.: Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. *Computers & Operations Research* 35(9), 2750–2759 (2008)
12. Neumann, F., Witt, C.: *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer, Heidelberg (2010)
13. Rudolph, G.: Takeover time in parallel populations with migration. In: BIOMA 2006, pp. 63–72 (2006)
14. Scharnow, J., Tinnefeld, K., Wegener, I.: The analysis of evolutionary algorithms on sorting and shortest paths problems. *Journal of Mathematical Modelling and Algorithms* 3, 349–366 (2004)
15. Skolicki, Z., De Jong, K.A.: The influence of migration sizes and intervals on island models. In: Proc. of GECCO 2005, pp. 1295–1302. ACM (2005)
16. Talbi, E.-G. (ed.): *Parallel Combinatorial Optimization*. Wiley (2006)
17. Tomassini, M.: *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time*. Springer, Heidelberg (2005)
18. Wegener, I.: Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In: Sarker, R., Yao, X., Mohammadian, M. (eds.) *Evolutionary Optimization*, pp. 349–369. Kluwer (2002)

Verifying Nash Equilibria in PageRank Games on Undirected Web Graphs

David Avis^{1,2}, Kazuo Iwama¹, and Daichi Paku¹

¹ School of Informatics, Kyoto University, Japan

² School of Computer Science, McGill University, Canada

Abstract. J. Hopcroft and D. Sheldon originally introduced the PageRank game to investigate the self-interested behavior of web authors who want to boost their PageRank by using game theoretical approaches. The PageRank game is a multiplayer game where players are the nodes in a directed web graph and they place their outlinks to maximize their PageRank value. They give best response strategies for each player and characterize properties of α -insensitive Nash equilibria. In this paper we consider PageRank games for undirected web graphs, where players are free to delete any of their bidirectional links if they wish. We study the problem of determining whether the given graph represents a Nash equilibrium or not. We give an $O(n^2)$ time algorithm for a tree, and a parametric $O(2^k n^4)$ time algorithm for general graphs, where k is the maximum vertex degree in any biconnected component of the graph.

Keywords: PageRank, Game theory, Nash equilibria, Fractional optimization.

1 Introduction

A PageRank value is assigned to each web page according to the stationary distribution of an α -random walk on the web graph. Here an α -random walk is a random walk modified to make a random jump with probability α at each step and a random jump is a move to a node according to a given distribution vector q . Introduced by Larry Page and Sergey Brin [12], it is an important basis of the Google search engine and possibly one of the most successful applications of a mathematical concept in the IT world.

Unlike rankings based on content such as keywords, tags, etc., PageRank focuses solely on the hyperlink structure of the given web graph. Web links themselves possess strategic worth and hence web authors often try to boost the PageRank of their web pages by carefully choosing links to other pages. Since these authors behave strategically in a self-interested way, this is a typical example of a non-cooperative game. In fact, Hopcroft and Sheldon recently introduced the PageRank game as a game theoretic model played over a directed graph [2]. Each player is a node and their strategy is to specify a set of outlinks for their node. The payoff for each player is the PageRank value for their node which is calculated on the resulting directed graph; the obvious goal of each player is to maximize their payoff.

In [2], the authors proved a nice property of this game, namely the best strategy of a player v is to place her outlinks to the nodes u having largest potential value ϕ_{uv} . The potential ϕ_{uv} measures the probability of returning to v before the first jump and does not depend on the outlinks from v if the other nodes do not change their outlinks. Thus, a simple greedy algorithm exists for deciding if a given graph is in Nash equilibrium and even a nice characterization of Nash equilibria graphs is possible. Interestingly, it turns out that such graphs representing Nash equilibria have very strong regularity properties as shown in Fig. 1(a) and (b) (see Section 2 for details).

In this paper we consider undirected rather than directed graphs, where a player cannot unilaterally create a new link but may unilaterally delete an existing one. As motivation, in the web graph model described above, what v intuitively does is to cut its links to nodes u having smaller ϕ_{uv} values, assuming that u will not delete its edge to v . In the new model, if v cuts its outlink to u we assume that u also cuts its outlink to v either automatically, or as a form of revenge. Therefore all links are necessarily bidirectional and an undirected graph model is obtained. A second motivation is that PageRank has been applied directly to settings involving undirected graphs. For example one could consider international bilateral agreements between universities, with PageRank measuring how international a university is. The ‘friend’ relationship on Facebook defines an undirected graph, and PageRank is a possible measure of the influence of a given user, etc. It turns out that the class of equilibria graphs in the undirected model is larger. For instance, the graph in Fig. 1(c) is now added to the class. Unfortunately, the nice property of the original model that ϕ_{uv} does not depend on the outlinks from v , no longer holds. Hence the greedy algorithm for the Nash decision problem does not work, either, and there seems to be no obvious way of checking the equilibrium condition.

Our Contribution. To our knowledge this is the first paper to consider PageRank games in undirected graphs even though, as described above, there are many natural applications of PageRank to undirected graphs.

Our first result is an algorithm for the case where the graph is a tree and runs in time $O(n^2)$. This is explained in Section 3. Our second result is a parametric algorithm for general graphs, where the parameter k is the maximum degree of any vertex in any biconnected component that contains it. In Section 4 we give a $O(2^k n^4)$ time algorithm for determining if G is a Nash equilibrium. It draws on many of the ideas introduced in Section 3. Biconnected components roughly correspond to local clusters of web pages, where one could expect the parameter k to be relatively small. Nodes linking biconnected clusters may have arbitrarily large degree without changing the time complexity.

Our techniques make a nontrivial application of the optimization scheme given in [2]. One of the novel ideas is the application of line arrangements to speed up Megiddo’s [3] technique for fractional optimization.

Related Work. Although it focuses less on game theoretic aspects, there is a large literature on optimal linking strategies to maximize the PageRank of given

nodes for directed graphs. On the positive side, Avrachenkov and Litvak [5] give a polynomial-time algorithm for maximizing the PageRank of a single node by selecting its outlinks. Kerchove et. al. [7] extend this result to maximizing the sum of the PageRank of a given set of nodes. Csaji et. al. [6] give a polynomial-time algorithm for maximizing the PageRank of a single node with any given set of controllable links.

On the negative side, [6] also shows that the problem becomes NP-hard if some pairs of controllable links in the set are mutually exclusive. Olsen [8] proved that maximizing the minimum PageRank in the given set of nodes is NP-hard if we are allowed to add k new links. He also proved that the problem is still NP-hard if we restrict the node set to a single node and the k links to only incoming ones to that node [9] and gives a constant factor approximation algorithm for this problem [10]. The question of whether there are α -sensitive Nash equilibria was recently affirmatively answered by Chen et. al. [11].

2 Preliminaries

2.1 PageRank Values

Initially we describe the Hopcroft-Sheldon directed graph model. Let $D = (V, E')$ be a simple directed graph on node set V and arc set E' , and let \mathbf{q} be a probability distribution on V . Throughout the paper we let $n = |V|$ denote the number of nodes. For $v \in V$, let $\Gamma(v)$ denote the set of v 's neighbours. A random jump is a move to a node according to the distribution vector \mathbf{q} instead of using one of the outlinks of the current node. An α -random walk on D is a random walk that is modified to make a random jump with fixed probability α ($0 < \alpha < 1$) at each step. The PageRank vector $\boldsymbol{\pi}$ over the n vertices in V is defined as the stationary distribution of the α -random walk. We define the potential matrix $\Phi = (\phi_{uv})$ such that for vertices $u, v \in V$, ϕ_{uv} is the probability that a random walk that starting from u visits v before the first jump ($\phi_{uv} = 1$ if $u = v$), which can be written as

$$\phi_{uv} = \frac{1 - \alpha}{|\Gamma(u)|} \sum_{i \in \Gamma(u)} \phi_{iv}. \quad (1)$$

In order to calculate $\boldsymbol{\pi}$, we have the following equation [2]:

$$\pi_v = \alpha \frac{\sum_{u \in V} q_u \phi_{uv}}{1 - \frac{(1-\alpha)}{|\Gamma(v)|} \sum_{i \in \Gamma(v)} \phi_{iv}}. \quad (2)$$

2.2 Directed PageRank games

In the PageRank games in [2] the players are the nodes V of a directed graph D and they attempt to optimize their PageRank by strategic link placement. A *strategy* for node v is a set of outlinks. An outcome is an arc set E' for D consisting of the outlinks chosen by each player. The payoff of each player is the value of PageRank which is calculated on D .

We say a player v is in *best response*, if v takes a strategy which maximizes v 's PageRank in D . A directed graph D is a *Nash equilibrium* if the set of outlinks for each node is a best response: no player can increase her PageRank value by choosing different outlinks. Several results for best response strategies and for Nash equilibria were introduced in [2]. In particular they gave a characterization of α -insensitive Nash equilibria, which are graphs being Nash equilibria for all values $0 < \alpha < 1$ of the jump parameter.

2.3 Undirected PageRank Games

In this paper we study similar games for undirected graphs. Let $G = (V, E)$ be an undirected graph on vertex set V and edge set E . Define the directed graph $D = (V, E')$ on the same vertex set V , where each edge uv in E gives rise to two arcs uv and vu in E' . In our model, the payoff of each player v for the graph G is the PageRank of v in the corresponding directed graph D .

In the undirected model, a player cannot unilaterally create a bidirected link to another node, but it can delete a bidirectional link. Therefore, we will say that a player v is in best response if v cannot increase her PageRank by any deletion of her (bidirectional) links. A Nash equilibrium is a graph for which every player is in best response. We consider the following problem:

Input: An undirected graph G , α , \mathbf{q} .

Output: Is the input a Nash equilibrium? (yes/no)

An equivalent formulation is to decide whether no player can increase her PageRank for the given input, where she is only allowed to delete edges to her neighbours. As for directed graphs, we let $\Gamma(v)$ denote the neighbours of vertex v in G . A strategy for v is to retain a subset $E_v \subseteq \Gamma(v)$ of neighbours and delete edges to her other neighbours. Let \mathbf{x} be a 0/1 vector of length $d_v = |\Gamma(v)|$, which indicates v 's strategy. Formally, if $i \in E_v$ then $x_i = 1$, otherwise $x_i = 0$, for $i = 1, \dots, d_v$. Let $\phi_{uv}(\mathbf{x})$ denote the potential function (II) for the subgraph of G formed by deleting edges vi , $i \in \Gamma(v) - E_v$. By (2) applied to the corresponding directed graph D the PageRank of v can be written as

$$\pi_v(\mathbf{x}) = \alpha \frac{\sum_{u \in V} q_u \phi_{uv}(\mathbf{x})}{1 - (1 - \alpha) \frac{\sum_{i \in \Gamma(v)} \phi_{iv}(\mathbf{x}) x_i}{\mathbf{1}^T \mathbf{x}}}, \quad (3)$$

where $\mathbf{x} \neq \mathbf{0}$. Let $\mathbf{1}_m$ denote a vector of ones of length m . Usually the length is clear by the context so for simplicity we may drop the subscript m . If the input is a Nash equilibrium then v is using a best response and no edge deletions for v will raise her PageRank. Therefore $\pi_v(\mathbf{1}_{d_v}) \geq \pi_v(\mathbf{x})$ for any 0/1 vector \mathbf{x} .

The approach we will use to solve the problem described in this section is to compute the maximum of $\pi_v(\mathbf{x})$ over all 0/1 vectors \mathbf{x} of length d_v , for each vertex v . In the next section we give a quadratic algorithm for the special case when G is a tree, and in Section 4 we give a FPT-algorithm for general graphs.

We give some examples in Fig. 1. Graphs (a), (b) are α -insensitive Nash equilibria in directed PageRank games, and are also a Nash equilibrium in our

model for any given α and \mathbf{q} . Graph (c) is an example which is not a Nash equilibrium in directed games, since v can increase its PageRank if we delete the arc from v to 2 which has less potential than 1. However (c) is a Nash equilibrium in our model for $\alpha = 0.15$ and uniform distribution \mathbf{q} , since if v cuts its edge to 2 then it decreases v 's PageRank. Graph (d) is not a Nash equilibrium in both directed and undirected games for $\alpha = 0.15$ and uniform distribution \mathbf{q} , where K_m is a m -clique, $m = 8$ for example. In this graph the potentials from 2, 3 to v are much less than 1, so v may try to cut the edge to 2 or the edge to 3, but a single edge deletion decreases v 's PageRank. Interestingly, the deletion of both edges leads to a greater PageRank for v .

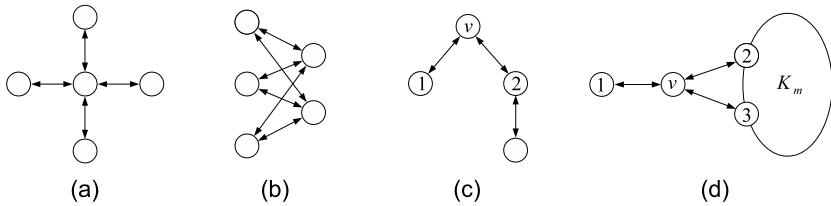


Fig. 1. Examples

3 Trees

In this section, we study the problem described in the previous section where the graph G is a tree. We prove the following theorem.

Theorem 1. *Given a tree G , jump probability α and distribution \mathbf{q} , we can determine in $O(n^2)$ time whether this is a Nash equilibrium and if not give an improving strategy for at least one player.*

The remainder of this section is devoted to the proof of this theorem.

Let v be a node in G , let $\Gamma(v)$ be the set of neighbours of v , and let $d_v = |\Gamma(v)|$. Consider any strategy for v as described in Section 2.3 and let \mathbf{x} be the 0/1 vector that represents it. For $i \in \Gamma(v)$, let N_i be the set of nodes which are descendants of i (including i itself) in the subtree of G rooted at v . For a node $u \in N_i$,

$$\phi_{uv}(\mathbf{x}) = \phi_{uv}x_i, \quad (4)$$

since potentials of all nodes in N_i depend on only link (v, i) and the other links of v do not affect these potentials. This is because if v cuts link (v, i) , all nodes in N_i are disconnected from the other nodes in G .

Therefore (3) can be rewritten:

$$\pi_v(\mathbf{x}) = \alpha \frac{\sum_{i \in \Gamma(v)} \sum_{u \in N_i} q_u \phi_{uv} x_i}{1 - (1 - \alpha) \frac{\sum_{i \in \Gamma(v)} \phi_{iv} x_i}{\mathbf{1}^T \mathbf{x}}}. \quad (5)$$

Note that the potential matrix Φ on G can be computed in $O(n^2)$ time, by using Gaussian elimination methods for each column vector $(\Phi)_v$ defined by equation (1). Since G is a tree we can apply elimination steps in post-order, where we consider v to be the root of G . There are at most n forward eliminations and backward substitutions because every node except v has only one parent. Therefore it costs $O(n^2)$ time to compute Φ .

Let $a_i = \alpha \sum_{u \in N_i} q_u \phi_{uv}$ and let $b_i = 1 - (1 - \alpha) \phi_{iv}$ for $i \in \Gamma(v)$. Consider the fractional integer programming problem,

$$P : \text{maximize } \pi_v(\mathbf{x}) = \mathbf{1}^T \mathbf{x} \frac{\mathbf{a}^T \mathbf{x}}{\mathbf{b}^T \mathbf{x}}, \quad \mathbf{x} \in \{0, 1\}^n,$$

where $a_i \geq 0$ and $b_i \geq 0$ for $i \in \Gamma(v)$ are known constants.

In order to solve problem P , we fix the Hamming weight l of \mathbf{x} and we solve the following problem for each $l = 1, \dots, d_v$:

$$Q : \text{maximize } f(\mathbf{x}) = \frac{\mathbf{a}^T \mathbf{x}}{\mathbf{b}^T \mathbf{x}} \quad \text{subject to } \mathbf{1}^T \mathbf{x} = l.$$

Problem Q can be solved directly by Megiddo's method in $O(n^2 \log^2 n)$ time [3], and it can be also solved by Newton's Method in time $O(n^2 \log n)$ [4]. However we are able to specialize Megiddo's method for our problem to obtain an $O(n^2)$ time algorithm. Our approach initially follows the technique describe in [3].

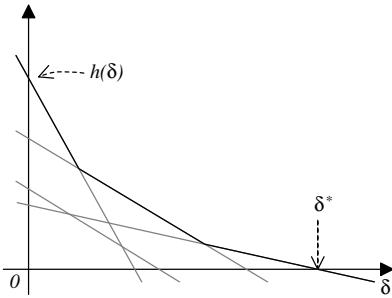
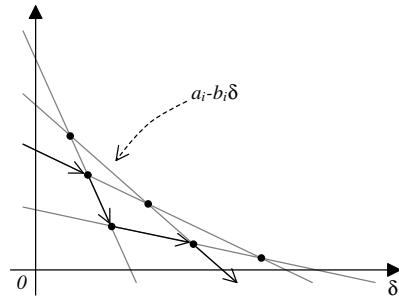
Since $\max \pi_v(\mathbf{x}) = \max_l l f(\mathbf{x})$, we can solve problem P by solving problem Q for each $l = 1, \dots, d_v$. Consider the following maximization problem for some fixed δ .

$$R : \text{maximize } g(\mathbf{x}) = (\mathbf{a} - \mathbf{b}\delta)^T \mathbf{x} \quad \text{subject to } \mathbf{1}^T \mathbf{x} = l.$$

Let $c_i = a_i - b_i \delta$, $i \in \Gamma(v)$ and let $S(\delta)$ be the decreasing sequence of indices ordered by the values of c_i . Problem R is easily solved by choosing the first l indices in $S(\delta)$. Let $h(\delta)$ be the optimal value of problem R for a given δ , that is, $h(\delta) = \max\{g(\mathbf{x}) : \mathbf{1}^T \mathbf{x} = l\}$. When $h(\delta) = 0$, then δ is equal to δ^* which is the optimal value of problem Q . On the other hand, if $h(\delta) > 0$ then $\delta < \delta^*$, and if $h(\delta) < 0$ then $\delta > \delta^*$, i. e., a root of h (see Fig. 2). The task in solving problem Q is to find the value δ^* for which $h(\delta^*) = 0$ by some tests on δ . The key point is how many values of δ have to be tested for finding δ^* . Since the optimal solution for R can change only when the order of $S(\delta)$ changes, we only have to test δ at intersection values of lines $\{c_i = a_i - b_i \delta : i \in \Gamma(v)\}$. Using some results from computational geometry, we are able to do this efficiently. First we compute the line arrangement of lines $\{c_i = a_i - b_i \delta : i \in \Gamma(v)\}$. Namely we define the planar graph H which is formed by subdivision of the plane induced by these lines. Then we look all edges in H as directed according to the positive direction of δ .

For each l we test values of δ at the change point of the l th entry in $S(\delta)$, which are the nodes in H lying on the l th layer of the arrangement. An example with $l = 3$ is shown in Fig. 3.

We summarize the algorithm for solving P . First compute constants a_i , b_i , for $i \in \Gamma(v)$, and line arrangement of lines $\{c_i = a_i - b_i \delta : i \in \Gamma(v)\}$. Compute

**Fig. 2.** Solving $h(\delta) = 0$ **Fig. 3.** Path for $l = 3$ on H

$S(0)$ by sorting the values of a_i . For $l = 1, \dots, d_v$, do following steps: let $x_i = 1$ if i is within l th entry of $S(0)$, and otherwise $x_i = 0$. Compute $A = \mathbf{a}^T \mathbf{x}$ and $B = \mathbf{b}^T \mathbf{x}$. From the starting edge, which is l th edge from the top, follow the l -th layer as follows: When we follow the edge on the line $c_i = a_i - b_i \delta$, and visit the node which is intersection of $c_i = a_i - b_i \delta$ and $c_j = a_j - b_j \delta$, let $A \leftarrow A - a_i + a_j$ and $B \leftarrow B - b_i + b_j$. If $A - B\delta < 0$ then output \mathbf{x} as the solution, otherwise let $x_i \leftarrow 0$ and $x_j \leftarrow 1$, and go to next node by following the edge on the line $c_j = a_j - b_j \delta$.

Finally, we analyze the running time of the algorithm. Computing the line arrangement takes $O(d_v^2)$ time, which is done by the incremental method or topological sort algorithms (see Edelsbrunner [1]). Since the number of nodes in H is $O(d_v^2)$ and each of them is visited twice, we can find δ^* for each l in at most $O(d_v^2)$ time. Therefore this algorithm solves problem P in $O(d_v^2)$ time. Note that if v is not in best response the solution to P gives an improving strategy for v .

As we have seen we can test whether a node v is in best response in $O(d_v^2)$ time. Because G is a tree we have $\sum_{v \in V} d_v = 2(n-1)$, and we have $\sum_{v \in V} d_v^2 < (\sum_{v \in V} d_v)^2 = 4(n-1)^2$. Therefore we can test whether all nodes are in best response in $O(n^2)$ time. This concludes the proof of Theorem 1.

4 General Graphs

In this section we give a parametric algorithm for general connected graphs based on a parameter $k = k(G)$ defined as follows. If $G = (V, E)$ is a tree we set $k = 1$ else k is the maximum vertex degree in any biconnected component of G . Note that $k(G)$ can be computed in $O(|E|)$ time by decomposing G into its biconnected components and by finding the maximum vertex degree in every such component. Note that graphs can have a large maximum vertex degree but small parameter k . This would occur whenever the large degree vertices were cut vertices. In a network setting the biconnected components could represent small groups of well connected web pages with relatively few links per page. These groups would be linked together by a few pages containing many more links. We prove the following theorem.

Theorem 2. Given a graph G with $k = k(G)$, jump probability α and distribution \mathbf{q} , in $O(2^k n^4)$ time we can determine if this is a Nash Equilibrium and if not give an improving strategy for at least one player.

The remainder of this section is devoted to the proof of this theorem.

Let v be a node in G and let $\{C_1, C_2, \dots, C_d\}$ be the set of connected components in the subgraph that is induced by deletion of v from G . It follows from the definition of $k = k(G)$ that v has at most k links to C_i for $i = 1, \dots, d$. Let $U_i = \{u : u \in C_i, u \in \Gamma(v)\}$ indicate the set of v 's neighbours in C_i , for $i = 1, \dots, d$. We have $|U_i| \leq k$ by the definition of k . Consider any strategy for v , as described in Section 2.3 and let \mathbf{x} be the 0/1 vector of length $d_v = |\Gamma(v)|$ that represents it. We write $\mathbf{x} = (x^1, x^2, \dots, x^d)$ as the concatenation of the 0/1 vectors x^i representing the strategy restricted to the component C_i , $i = 1, \dots, d$. Then, for $u \in V$, if $u \in C_i$ for $i = 1, \dots, d$, the potential of u is written as follows:

$$\phi_{uv}(\mathbf{x}) = \sum_{S \subseteq U_i} \phi_{uv}^S \prod_{s \in S} x_s^i \prod_{s \in U_v - S} (1 - x_s^i) \quad (6)$$

where ϕ_{uv}^S are the potentials from u for the subgraph of G formed by deleting edges $vu, u \in \Gamma(v) - S$. This is because potentials to v from all nodes in C_i depend only on links to U_i and never depend on other links. To compute each column vector $(\Phi)_v^S$ for $S \subseteq U_i$ for $i = 1, \dots, d$, we solve the linear systems defined on (II) by Gaussian elimination method in $O(2^k n^3)$ time.

We have the formula for PageRank of v as

$$\pi_v(\mathbf{x}) = \mathbf{1}^T \mathbf{x} \frac{\sum_i \sum_{S \subseteq U_i} a_S \prod_{s \in S} x_s^i}{\sum_i \sum_{S \subseteq U_i} b_S \prod_{s \in S} x_s^i}. \quad (7)$$

where a_S and b_S be constants such that

$$a_S = \sum_{u \in C_i : S \subseteq C_i} \alpha q_u \sum_{T \subseteq S} (-1)^{|S|-|T|} \phi_{uv}^T$$

$$b_S = \begin{cases} 1 - (1 - \alpha) \phi_{uv}^S & S = \{u\} \\ -(1 - \alpha) \sum_{u \in S} \sum_{T \subseteq S} (-1)^{|S|-|T|} \phi_{uv}^T & \text{otherwise.} \end{cases}$$

Note that $a_S \geq 0$ for all S , and the denominator is always positive.

We can determine whether v maximizes its PageRank by solving the following fractional integer programming problem:

$$P : \text{maximize } \pi_v(\mathbf{x}), \quad x_s^i \in \{0, 1\} \text{ for } s \in U_i, i = 0, \dots, d.$$

The method we use to solve P is the similar to that used in Section 3. We fix the Hamming weight of \mathbf{x} , $\mathbf{1}^T \mathbf{x} = l$, and consider the following linear fractional integer programming problem:

$$Q : \text{maximize } f(\mathbf{x}) = \frac{\sum_i \sum_{S \subseteq U_i} a_S \prod_{s \in S} x_s^i}{\sum_i \sum_{S \subseteq U_i} b_S \prod_{s \in S} x_s^i} \quad \text{subject to } \mathbf{1}^T \mathbf{x} = l.$$

Since $\max \pi_v(\mathbf{x}) = \max_l l f(\mathbf{x})$, we can solve problem P by solving problem Q for each $l = l, \dots, d_v$. Let δ be a positive real number, and let $c_S = a_S - b_S \delta$ for all $S \subseteq U_i$ for $i = 1, \dots, d$.

$$R : \text{maximize } g(\mathbf{x}) = \sum_i \sum_{S \subseteq U_i} c_S \prod_{s \in S} x_s^i \quad \text{subject to } \mathbf{1}^T \mathbf{x} = l.$$

Let $h(\delta)$ be the optimal value of problem R for some δ , i.e., $h(\delta) = \max_{\mathbf{x}} \{g(\mathbf{x}) : \mathbf{1}^T \mathbf{x} = l\}$. To solve problem Q , we can repeatedly solve R for various δ to find the root of $h(\delta)$. However, the original problem is to determine whether v maximizes its value. Namely, we do not have to maximize $f(\mathbf{x})$ subject to $\mathbf{1}^T \mathbf{x} = l$, but only determine $l\delta^* > d_v f(\mathbf{1})$. So, the task is to solve R for $\delta = \frac{d_v}{l} \delta'$, where $\delta' = f(\mathbf{1})$, and only check if $h(\delta) < 0$ or not.

For converting $g(\mathbf{x})$ in problem R to linear function, let $y_{i,t}$ denote the 0/1 variable and let $e_{i,t}$ be the constant for $1 \leq i \leq d$ and $1 \leq t \leq |U_i|$ such that,

$$y_{i,t} = \begin{cases} 1 & \sum_{s \in U_i} x_s^i = t \\ 0 & \text{otherwise} \end{cases}, \quad e_{i,t} = \max_S \left\{ \sum_{T \subseteq S} c_T : S \subseteq U_i, |S| = t \right\}.$$

$y_{i,t}$ indicates whether the number of edges used in the strategy \mathbf{x} going from v to U_i is equal to t or not. If $y_{i,t} = 1$, let $S \subseteq U_i$ be the t edges chosen. Then we consider that $g(\mathbf{x})$ earns $\sum_{T \subseteq S} c_T$, with cost t . In the optimal strategy \mathbf{x} , if $y_{i,t} = 1$ for some i, t then it must be that S maximizes $\sum_{T \subseteq S} c_T$, that is $e_{i,t}$, as any other assignment can be improved to it. We then have the equivalent integer linear program to R :

$$R' : \text{maximize } \sum_{i=1}^d \sum_{t=1}^{|U_i|} e_{i,t} y_{i,t} \quad \text{subject to } \sum_{i=1}^d \sum_{t=1}^{|U_i|} t y_{i,t} = l \quad (8)$$

$$\sum_{t=1}^{|U_i|} y_{i,t} \leq 1 \quad \text{for } i = 1, \dots, d. \quad (9)$$

Problem R' is similar to a knapsack problem where each item has positive integer weight t and value $e_{i,t}$, and the total weight must be l . The only difference is the constraint (9).

Dynamic programming can be used to solve R' in $O(l d_v)$ time. Let $w(i, t)$, for $1 \leq i \leq d$ and $1 \leq t \leq l$, denote the maximum value which has total weight t and uses only the i first items. Let $e_{i,0} = 0$, then,

$$w(i, t) = \max_{0 \leq s \leq |U_i|} \{w(i-1, t-s) + e_{i,s}\}.$$

For each $i = 1, \dots, d$, we can compute $w(i, t)$ for $1 \leq t \leq l$ in $l|U_i|$ time. Since $d_v = \sum_{1 \leq i \leq d} |U_i|$, the computation time for solving R' is $O(l d_v)$.

In order to determine whether v is in best response, we test $g(\mathbf{x})$ for $\delta = \frac{d_v}{1} \delta', \frac{d_v}{2} \delta', \dots, \frac{d_v}{d_v} \delta'$. Since $d_v \leq kd$ the running time per vertex is $O(2^k n^3 + k^2 d^3 + 2^k d)$. Moreover $\sum_{v \in V} d_v < 2kn$ from the assumption. Therefore, in order to determine whether every node is in best response, it takes $O(2^k n^4 + k^2 n^3 + 2^k n) = O(2^k n^4)$ time. This completes the proof of Theorem 2.

5 Conclusion

We have constructed a new undirected model for PageRank games and studied the problem of verifying Nash equilibria for these games. Our first result is an algorithm for trees that runs in time $O(n^2)$. Our second result is an extension of this method for general graphs. We defined a parameter k to be the maximum vertex degree in any biconnected component of the graph and gave a $O(2^k n^4)$ time algorithm for testing if it is a Nash equilibrium.

Acknowledgements. We thank Yuichi Yoshida and Junichi Teruyama for many helpful discussions.

References

1. Edelsbrunner, H.: Algorithms in Combinatorial Geometry. EATCS Monographs in Theoretical Computer Science, vol. 10. Springer, Heidelberg (1987)
2. Hopcroft, J., Sheldon, D.: Network reputation games, eCommons@Cornell (2008) (manuscript), <http://hdl.handle.net/1813/11579>
3. Megiddo, N.: Combinatorial optimization with rational objective functions. *Math. of Oper. Res.* 4, 414–424 (1979)
4. Radzik, T.: Newton’s method for fractional combinatorial optimization. In: Proceedings, 33rd Annual Symposium on Foundations of Computer Science, pp. 659–669 (1992)
5. Avrachenkov, K., Litvak, N.: The effect of new links on Google PageRank. *Stochastic Models*, 319–331 (2006)
6. Csáji, B.C., Jungers, R.M., Blondel, V.D.: PageRank Optimization in Polynomial Time by Stochastic Shortest Path Reformulation. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS, vol. 6331, pp. 89–103. Springer, Heidelberg (2010)
7. De Kerchove, C., Ninove, L., Van Dooren, P.: Maximizing pagerank via outlinks. *Linear Algebra and its Applications* 429, 1254–1276 (2008)
8. Olsen, M.: The Computational Complexity of Link Building. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 119–129. Springer, Heidelberg (2008)
9. Olsen, M.: Maximizing PageRank with New Backlinks. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 37–48. Springer, Heidelberg (2010)
10. Olsen, M., Viglas, A., Zvedeniouk, I.: A Constant-Factor Approximation Algorithm for the Link Building Problem. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 87–96. Springer, Heidelberg (2010)
11. Chen, W., Teng, S.-H., Wang, Y., Zhou, Y.: On the α -Sensitivity of Nash Equilibria in PageRank-Based Network Reputation Games. In: Deng, X., Hopcroft, J.E., Xue, J. (eds.) FAW 2009. LNCS, vol. 5598, pp. 63–73. Springer, Heidelberg (2009)
12. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: Proceedings of the 7th International World Wide Web Conference, pp. 107–117 (1998)

Improved Collaborative Filtering

Aviv Nisgav and Boaz Patt-Shamir^{*}

School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel
`{avivns, boaz}@eng.tau.ac.il`

Abstract. We consider the interactive model of collaborative filtering, where each member of a given set of users has a grade for each object in a given set of objects. The users do not know the grades at start, but a user can *probe* any object, thereby learning her grade for that object directly. We describe reconstruction algorithms which generate good estimates of all user grades (“preference vectors”) using only few probes. To this end, the outcomes of probes are posted on some public “billboard”, allowing users to adopt results of probes executed by others. We give two new algorithms for this task under very general assumptions on user preferences: both improve the best known query complexity for reconstruction, and one improving resilience in the presence of many users with esoteric taste.

1 Introduction

The *collaborative filtering* (or *interactive recommender*) problem can be described as follows. We are given n users and m objects, and each user has a *preference vector*, which consists of a grade for each object. The grades are initially unknown to the system (possibly not even to the users), but each grade can be revealed by a *probe*. For example, the objects may be books and probing is asking a user for her grade of the book (the user may need to read the book!); or the objects may be personal preference queries, and probing is presenting a query to a user by the system. The goal of collaborative filtering is to compute some function of the user grades while minimizing the *probe complexity*, defined as the maximal number of grades any user is asked to report.

The strongest possible task of collaborative filtering is reconstructing all user grades: given all grades, one can compute any function of them. Therefore one of the main questions in collaborative filtering is how expensive is it to reconstruct all grades. A basic observation is that reconstructing “esoteric” preferences (preferences that are held by only few users) may require considerably more probes than reconstructing “mainstream” preferences (preferences shared by many users). We formalize this intuitive notion using two parameters as follows. Fix a metric over preference vectors (say, Hamming distance for binary preferences). Given a *popularity parameter* $0 < \alpha \leq 1$, and a *distance parameter* $D \geq 0$, we say that a preference vector v_i is (α, D) -prevalent if there are at least αn users whose preference vectors are at distance at most D from v_i . A user whose preference vector is (α, D) -prevalent is called an (α, D) -prevalent user.¹

^{*} Supported in part by the Israel Science Foundation (grant 1372/09) and by Israel Ministry of Science and Technology.

¹ Note that α, D can be traded-off: Fix a set of preference vectors. Given any popularity parameter $\alpha \leq 1$, one can determine the smallest possible D so that a given user is (α, D) -prevalent. Similarly, given a distance parameter $D \geq 0$, the largest possible α for a user is well-defined.

Reconstruction algorithms can be distinguished according to whether their probe complexity is dependent or independent of the distance parameter D . Algorithms whose complexity depends on D may be better for small values of D : The best such algorithm known to date is Algorithm RADIUS by Alon et al. [1], whose query complexity is $O(\frac{D}{\alpha} \log n^{2.5})$ (here and below, we omit a scaling factor of $\lceil \frac{m}{n} \rceil$, applicable only when $m > n$). The algorithm reconstructs preferences of (α, D) -prevalent users with $O(D)$ errors. The best known algorithms with probe complexity independent of D for (α, D) -prevalent users are Algorithm LARGE_RADIUS (also from [1]) that guarantees $O(D/\alpha)$ errors in probe complexity $O(\log^{3.5} n / \alpha^2)$, and Algorithm CALCULATEPREFERENCES by Gilbert et al. [8], which improves the number of errors in LARGE_RADIUS to $O(D)$ at the same asymptotic complexity. However, Algorithm CALCULATEPREFERENCES suffers from an interesting weakness: it requires users tastes to be mostly homogeneous, in the sense that almost all users must be (α, D) -prevalent (the algorithm in [8] allows for $O(\alpha n)$ users to be Byzantine). More specifically, the algorithm may produce incorrect results if many esoteric (but honest) users are present. Recalling the real world, this seems to be a significant drawback: it is an accepted truth that in many contexts, as many as 40% of the users do not have “mainstream” taste (see, e.g., [9]). We note that Algorithms RADIUS and LARGE_RADIUS do not require homogeneity: users which are not (α, D) -prevalent get unpredictable results, but (α, D) -prevalent users still get correct output.

Our contribution. In this paper we present algorithms that improve both the distance-dependent and distance-independent cases. In Section 3 we describe Algorithm **S**, that reconstructs the preferences of (α, D) -prevalent users with at most $O(D)$ errors, using at most $O(\frac{D}{\alpha} \log^2 n)$ probes per user. Algorithm **S** improves on the best known probe complexity (of Algorithm RADIUS [1]), and moreover, it is much simpler. In Section 4 we describe our second result: Algorithm **A**, whose probe complexity is $O(\frac{1}{\alpha} \log^3 n)$, reconstructing the preferences of (α, D) -prevalent users with $O(D)$ errors. Algorithm **A** can work with non-homogeneous population (namely not all users must be (α, D) -prevalent), while still being able to bound the effect of Byzantine users.

Related work. Much research in collaborative filtering considers the following model. There is a large dataset that contains all past choices of users (be it purchase history, or, say, movie grades), and the goal is to predict the way a user would grade an object she did not examine yet. The problem with this approach is that it ignores the existence of feedback in the model: If the system indeed affects user choices, the dataset is biased toward objects recommended by the system, and does not necessarily reflect the “true” preference of the users.

This fundamental gap is bridged by the *interactive recommender system* model [7, 4] we use. In this model the system can observe the user’s reaction to recommendations and act on it. More specifically, the system proposes an object to the user, and the user, in response, informs the system of her grade for that object. (The system may deduce user feedbacks by some noisy heuristic, e.g., did the user click the proposed link?) It is usually assumed that the system starts out with no knowledge at all about user grades.

In the non-interactive model, it is common to assume a linear generative model for user’s grades and apply algebraic techniques such as principal component analysis [10] or singular value decomposition [15]. Papadimitriou et al. [14] and Azar et al. [5]

rigorously prove conditions under which SVD is effective. Other generative user models that were considered include simple Markov chain models [11][12], where users randomly select their “type,” and each type is a probability distribution over the objects.

Drineas et al. [7] were the first to propose the interactive model, where the algorithm tells the users which products to probe and the results of the probes are fed back to the algorithm. In [4] it was shown that in this model, a user sharing his exact preference with at least α fraction of the users ($D = 0$ in our terms), can find a product he likes in $O(\frac{\log n}{\alpha})$ probes. In [2], Awerbuch et al. show that at the same probe complexity, it is possible to reconstruct all users preference. They also prove that probe complexity $\Omega(\frac{\log n}{\alpha})$ is necessary in this case. Alon et al. [1] give the first algorithms to reconstruct preferences of (α, D) -prevalent tastes for $D > 0$, as mentioned above.

The basic interactive model was extended in a few directions. Awerbuch et al. [3] study an asynchronous model, where an adversarial (oblivious) schedule determines which user probes next. Azar et al. [6] show how to extend algorithms for binary grades to work with non-binary (discrete or continuous) grades.

Organization. In Section 2 we define the model and some notation. In Section 3 we give our algorithm with probe complexity linear in D . In Section 4 we give our second algorithm, with probe complexity independent of D , that can withstand Byzantine adversaries. Some proofs are omitted from this extended abstract.

2 Preliminaries

We first formalize the problem to be solved.

Instances. A reconstruction problem instance consists of a set P of n users, a set O of m objects, and a binary grade $A_{i,j}$ for each user $i \in P$ and object $j \in O$. The collection of grades of a given user i is called user i ’s *preference vector* or *taste*, denoted by A_i .

Given a set of objects $O' \subseteq O$, the *distance* between two users i, i' w.r.t. O' , denoted $\text{dist}_{O'}(A_i, A_{i'})$ is the number of objects in O' on which i and i' disagree. We usually omit the subscript O' when distance is taken w.r.t. all objects.

Given $0 < \alpha \leq 1$ and $D \geq 0$, a preference vector v is (α, D) -*prevalent* in a given instance if there are at least αn users whose taste is at distance at most D from v . We shall abuse notation slightly and say that a user is (α, D) -prevalent if his taste is (α, D) -prevalent. For a subset $B \subseteq P$ of the users, an instance is called (α, D, B) -*homogeneous* if all users not in B are (α, D) -prevalent.

Outputs. We are given an (α, D, B) -homogeneous instance, of which we know only the number of users n , the number of objects m , and the parameters α and D . For any user i the output is a vector \hat{A}_i , whose intended meaning is an estimate of A_i . Our algorithms are randomized, the output accuracy statement will hold with high probability, namely with probability $1 - n^{-\Omega(1)}$, when probability is taken with respect to the coin tosses of the algorithm. Note there is no requirement regarding the output of users in B .

Algorithms. We assume the following distributed computational model. Algorithms proceed in synchronous *rounds*, where in each round, the algorithm may receive, as input, at most one grade for each user. This action is called a *probe* of the user. We assume that the results of all probes are posted on a public “billboard,” i.e., they are

available to all users, and the algorithm run by user i may use the grades of all previous probes made by all users to determine (typically, in a randomized way) what object user i will probe next. The maximal number of probes any user is asked to execute in a run is the *probe complexity* of the algorithm.

Simple bounds on probe complexity. Note that it is trivial to solve the recommendation problem in $O(m)$ probe complexity, by letting each user probe all objects. On the other hand $\Omega(m/\alpha n)$ probe complexity is necessary to produce estimates with $O(D)$ errors in (α, D, B) -homogeneous instances. Informally, αn users contained in a ball of diameter D need to cover between them all m objects with probes, and hence the average number of probes per user cannot be less than $\Omega(m/\alpha n)$.

3 Algorithm S: Linear Dependence on D

In this section we present our first result: an algorithm for reconstructing preferences whose probe complexity is linear in D and $1/\alpha$, for (α, D) -prevalent users. We assume that α and D are given parameters. (Alon et al. [1] explain how to remove this restriction in an “anytime” algorithm, at the cost of increasing the probe complexity by a logarithmic factor and the number of errors by a constant factor.)

The algorithm presented in this section improves on Algorithm SMALL_RADIUS from [1] in terms of query complexity, and it is considerably simpler. It uses, as a building block, a known algorithm, as detailed below.

Tool: Exact reconstruction. We use an algorithm denoted **E** (mnemonic for “exact”), that solves the recommendation problem for $(\alpha, 0, B)$ -prevalent instances of n users and m objects, namely instances where each user in $P \setminus B$ is a member of a set of at least αn users, all with identical preferences. Algorithm **E** produces, at the cost of $T_E(n, m, \alpha)$ probe complexity, the preference vector of every user in $P \setminus B$. Several implementations of **E** are known [21]. We use the following result, adapted from [2].

Theorem 1. *There exists an algorithm **E** that for any $(\alpha, 0, B)$ -homogeneous instance with n users and m objects solves the reconstruction problem with probability at least $1 - n^{-c}$, using probe complexity $O(\lceil \frac{m}{n} \rceil \cdot \frac{\log n}{\alpha})$, for any desired constant $c > 0$.*

We note that in our algorithms, the number of invocations of **E** is polynomial in n , so by the Union Bound, we may assume that w.h.p., all invocations of **E** are successful.

Algorithm description. Algorithm **S** (see pseudo-code in Alg. 1) is very simple: The object set is broken into a few random subsets, and Algorithm **E** is applied to each of them, with popularity parameter $\alpha/4$. Repeating this procedure K times with independent random partitions of the object set yields K estimates for each object; the algorithm output at a user is, for each object, the majority of the outputs of **E** for that object. As we shall see, Algorithm **S** fails with probability $\exp(-\Omega(K))$.

Analysis. For each user $i \notin B$, define $P(i) \stackrel{\text{def}}{=} \{i' \in P : \text{dist}(A_i, A_{i'}) \leq D\}$, namely the set of users whose preference vectors differ from i by at most D objects. We shall distinguish between objects on which i has an “unusual” opinion with respect to $P(i)$, and other objects, on which i agrees with most users in $P(i)$. The first set cannot contain too many objects, and the second set can be quite reliably reconstructed using $P(i)$.

Algorithm 1. $\mathbf{S}(P, O, \alpha, D)$.	K is a confidence parameter, $c > 8$ is a constant
(1) for $k \leftarrow 1$ to K do	
(1a) Partition O randomly into $S = cD$ disjoint subsets $O = \bigcup_{s=1}^S O_s$: for each object $j \in O$, independently select $s \in \{1, \dots, S\}$ uniformly at random, and let $O_s \leftarrow O_s \cup \{j\}$.	
(1b) for $s \leftarrow 1$ to S do	
Invoke $\mathbf{E}(P, O_s, \frac{\alpha}{4})$. // all players, some objects, reduced popularity	
Let $C_{i,j}^k$ denote the output for object j by user i , for all $j \in O$ and $i \in P$.	
(2) Let $C_{i,j}$ be the majority of $\{C_{i,j}^k \mid k = 1, \dots, K\}$, for all $j \in O$.	
For each user i output $C_{i,1}, \dots, C_{i,m}$.	

Formally, we define, for each user $i \notin B$, the set of objects $O(i)$ to be the objects on which user i agrees with the majority of the users in $P(i)$, i.e.,

$$O(i) \stackrel{\text{def}}{=} \left\{ j \in O : \sum_{i' \in P(i)} |A_{i,j} - A_{i',j}| < \frac{|P(i)|}{2} \right\}.$$

We first state a “Markov’s Inequality”-type bound on $|O(i)|$ (proof is omitted).

Lemma 1. Any user $i \notin B$ agrees with at least $1 - \delta$ of the users in $P(i)$ on at least $m - D/\delta$ objects.

Next, we show that for any $j \in O(i)$, in each iteration k of Algorithm **S**, Algorithm **E** computes a correct estimate of $A_{i,j}$ in Step 1b with good probability.

Lemma 2. For all $i \in P \setminus B$, $j \in O(i)$ and $1 \leq k \leq K$: $\Pr[C_{i,j}^k = A_{i,j}] \geq 1 - \frac{4}{c}$.

Proof: Consider iteration k , and let $O_{s(j)}$ be the subset j belongs to in iteration k . Let $P_s(i)$ be the set of users that agree with i on all objects in O_s , i.e., $P_s(i) = \{i' \mid \text{dist}_{O_s}(i, i') = 0\}$. It suffices to prove that $|P_{s(j)}(i)| \geq \frac{\alpha n}{4}$, because this ensures that the preconditions to the invocation of **E** are met in iteration k , and thus the lemma follows from the correctness of **E**.

First we note that for any $\beta > 0$, as distances are non-negative integers:

$$\sum_{i' \in P(i)} \text{dist}_{O_s}(i, i') \leq \beta \cdot |P(i)| \implies |P_s(i)| \geq (1 - \beta) \cdot |P(i)| \quad (1)$$

We now turn to bound the probability that $\sum_{i' \in P(i)} \text{dist}_{O_{s(j)}}(i, i') < \frac{3}{4} \cdot |P(i)|$. By the assumption that $i \notin B$ we know that $\sum_{i' \in P(i)} \text{dist}_{O_s}(i, i') \leq D \cdot |P(i)|$. Recalling that $\sum_{i' \in P(i)} \text{dist}_O(i, i') = \sum_{s=1}^S \sum_{i' \in P(i)} \text{dist}_{O_s}(i, i')$, we may deduce that $\sum_{i' \in P(i)} \text{dist}_{O_s}(i, i') > \frac{|P(i)|}{4}$ for at most $4D$ indices s .

Consider now the random variable $\sum_{i' \in P(i)} \text{dist}_{O_{s(j)} \setminus \{j\}}(i, i')$. It is independent of the grades of j . As $j \in O(i)$, it holds $\sum_{i' \in P(i)} |A_{i,j} - A_{i',j}| \leq \frac{|P(i)|}{2}$, and hence

$$\begin{aligned} \Pr \left[\sum_{i' \in P(i)} \text{dist}_{O_{s(j)}}(i, i') \leq \frac{3|P(i)|}{4} \right] &\geq \\ \Pr \left[\sum_{i' \in P(i)} \text{dist}_{O_{s(j)} \setminus \{j\}}(i, i') \leq \frac{|P(i)|}{4} \right] &\geq \frac{S - 4D}{S} = 1 - \frac{4}{c}, \end{aligned}$$

and we are done by Equation 1 (using $\beta = \frac{3}{4}$) and the fact $|P(i)| \geq \alpha n$. ■

In each iteration the probability of a wrong prediction is less than $1/2$, and repeating the procedure diminishes it, as stated in the following lemma (proof omitted).

Lemma 3. *For any user $i \in P \setminus B$ and object $j \in O(i)$: $\Pr[C_{i,j} = A_{i,j}] = 1 - e^{-\Omega(K)}$.*

Before we summarize the performance of **S**, we note that using Chernoff bound it is easy to see that whenever $D = o(m/\log n)$, for m sufficiently large and for any s , it holds $|O_s| = \Omega(m/D)$ with high probability. We can now derive our first main result.

Theorem 2. *Suppose that $D = o(m/\log n)$ and $K = \Theta(\log(m+n))$. With probability $1 - mne^{-\Omega(K)}$, Algorithm **S** predicts for each user $i \in P \setminus B$ its preference vector with less than $2D$ errors. Moreover, the probe complexity is $KcD \cdot T_E\left(n, \left\lceil \frac{m}{(c-1)D} \right\rceil, \alpha/4\right) = O\left(\frac{1}{\alpha} \left\lceil \frac{m}{nD} \right\rceil \cdot D \log^2(m+n)\right)$.*

4 Complexity Independent of D

In this section we present our second main result: an improved algorithm estimating the preference vectors of users in an (α, D, B) -homogeneous instance, with probe complexity independent of D . An interesting problem that arises in this algorithm is the possible influence of users without (α, D) -prevalent taste. In all algorithms, the output of these “esoteric” users is unpredictable; but in the context of algorithms whose complexity is independent of D , such users may cause (α, D) -prevalent users to err too (this is the case in [8]). This problem is exacerbated in the presence of malicious users, who may fabricate their preferences on-line so as to hurt as many users as possible. Our algorithm bounds the number of errors introduced by honest but esoteric users, and the number of users affected by adaptive malicious users.

Tool: Distinguishing dissimilar users. Our algorithm uses Algorithm **Sep** essentially introduced in [13]. Algorithm **Sep** returns a users partition where each part contains users of roughly similar taste. Based on [13], the following can be proved.

Theorem 3. *Let $\mathcal{S} = \{S_1, \dots, S_k\}$ be the result of applying **Sep** (P, O, α). Then:*

- (1) *For all $i_1, i_2 \in P$ with $A_{i_1} = A_{i_2}$, there exists $S \in \mathcal{S}$ s.t. $\{i_1, i_2\} \subseteq S$.*
- (2) *Let $S \in \mathcal{S}$ be such that $|S| \geq \alpha n$. For any $j \in O$, with probability $1 - n^{-\Omega(1)}$, all users in S , except for at most $\alpha|S|$ users, have the same opinion about j .*
- (3) *The probe complexity of Algorithm **Sep** is $O(\left\lceil \frac{m}{n} \right\rceil \frac{\log n}{\alpha})$.*

As before, we note that in our algorithms **Sep** is invoked only $\text{poly}(n)$ times, and hence we shall assume w.h.p. that all invocations of **Sep** are successful.

Tool: Selecting the closest vector from a set. Another procedure we use is **SELECT**, which receives, as input, a collection V of preference vectors, and, when run by user i , outputs the vector in V which is about the closest to A_i , the preference vector of i . More precisely, in [1] the following result is proved.

Theorem 4. *Suppose user i executes **SELECT** (V). Then the return value $u \in V$, with probability $1 - n^{-\Omega(1)}$, satisfies $\text{dist}(u, A_i) = O(\min \{\text{dist}(v, A_i) \mid v \in V\})$. Procedure **SELECT** requires $O(|V|^2 \log n)$ probes by user i .*

Algorithm 2. $\mathbf{F}(P, O, \alpha)$ *c is a constant*

-
- 1: Partition O randomly into $S = 2c^2 \log n$ disjoint subsets $O = \bigcup_{s=1}^S O_s$; for each object $j \in O$, select $s \in \{1, \dots, S\}$ uniformly at random, and let $O_s \leftarrow O_s \cup \{j\}$.
 - 2: **for** $s \leftarrow 1$ **to** S **do**
 - 3: invoke $\mathbf{Sep}(P, O_s, \frac{\alpha}{6})$.
 - 4: **end for**
 - 5: **return** a $|P| \times |P|$ symmetric matrix where entry (i, i') is “close” if users i, i' ended in different subsets in less than $2c \log n$ invocations of \mathbf{Sep} , and “far” otherwise.
-

Algorithm description. The basic idea is as follows. First we take a random subset of the objects, thus reducing the expected distance parameter from D to $O(\log n)$. To this subset we apply a distance-dependent procedure. The result is used to identify (w.h.p.) similarity of users; once this relation is established, users can adopt probe results of other users that are known to have similar taste, without risking too many errors.

This idea would work if all users were (α, D) -prevalent. But the presence of many users with non-prevalent tastes may affect the results, as their distance-dependent result may be incorrect. Such users may be incorrectly identified as “similar”, and their number may overwhelm the number of (α, D) -prevalent users. To deal with this, we distinguish between two cases of non-prevalent users: honest non-prevalent users (whose taste is not determined by the execution of the algorithm) may influence only a bounded number of objects; but dishonest users, who use “bait and switch” tactics of changing their preferences on-line (as a function of the random choices taken by the algorithm), may incur much greater damage. Nevertheless, employing techniques developed for graph coalitions, we can bound the number of prevalent users influenced.

More specifically, Algorithm **F** (see Alg. 2) is used to compute a similarity relation. Note that Algorithm **F** is reminiscent of Algorithm **S**, but it uses Algorithm **Sep** instead of **E**, and its output is binary (“close” or “far”) for all pairs of users. Our top-level algorithm is Algorithm **A** (see Alg. 3). It first selects a sample Ψ^k of the objects, and sends it to Algorithm **F** (Step 5). The sample size is such that the distance parameter is reduced to $O(\log n)$, which is the distance threshold Algorithm **F** is designed for. This is repeated K times. The matrix outputs of **F** are used in Step 7 to construct a “proximity graph” that indicates which users have preferences (probably) close to theirs (we note that Gilbert et al. [8] apply a similar technique). Algorithm **A** outputs, for each user and object, the majority of opinions for that object by users in its neighborhood (Step 15). To facilitate this policy, the algorithm requires (Step 1) all users to query a sufficiently large random sample of the objects, so as to ensure that each object has sufficient coverage by “close” users. The algorithm mitigates the influence of dishonest users by repeated averaging with neighborhoods of different radii in the graph (Step 8).

Below, we analyze the case where all users are honest (this is the model used by Alon et al. [1]). The analysis of the case where some users may be dishonest (as considered in [8]) is omitted from this extended abstract.

Analysis. Here we assume all users are honest, i.e., follow the protocol and their preferences are oblivious to the unfolding execution of the algorithm. We analyze Algorithm **A** with parameter $R = 1$ ($R > 1$ is useful in the case of dishonest users). As mentioned above, adopting the majority’s opinion (Step 15) raises a problem with regard to

Algorithm 3. A (P, O, α, D, R)	R is a parameter, $c > 16$ is a constant
1: Each user probes each object j independently with probability $c \cdot \frac{\log(m+n)}{\alpha n}$.	
2: Let S_i be the set of objects probed by user i . Let $K = 2c \log n$.	
3: for $k \leftarrow 1$ to K do	
4: Select a random set $\Psi^k \subseteq O$ of $\lceil \frac{cm \log n}{D} \rceil$ objects, for some integer constant c .	
5: Evaluate $\mathbf{F}(P, \Psi^k, \alpha)$. <i>// all users, some objects</i>	
6: end for	
7: Define a graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = P$ and $\mathcal{E} = \{(i, i') \mid i, i' \text{ never marked ‘far’ by } \mathbf{F}\}$ (i.e., nodes correspond to users and edges connect ‘similar’ users).	
8: for all $r \in \{0, \dots, R-1\}$ do	
9: Let $I_i(r)$ be the set of nodes at distance at most 2^r from i in G .	
10: For all $j \in O$, let $I_{i,j}(r) \stackrel{\text{def}}{=} \{i' \in I_i(r) : j \in S_{i'}\}$, i.e. members of $I_i(r)$ that probed j .	
11: for all $j \in O$ do	
12: if $j \in S_i$ then	
13: set $A_{i,j}(r)$ to $A_{i,j}$ as probed in Step II	
14: else	
15: set $A_{i,j}(r)$ to the majority of the set $\{A_{i',j}\}_{i' \in I_{i,j}(r)}$. Break ties arbitrarily.	
16: end if	
17: end for	
18: end for	
19: return $\text{SELECT}(\{A_i(0), \dots, A_i(R-1)\})$	<i>to user i</i>

non-prevalent users, who may incorrectly be identified as close. Our goal is therefore to show that on most objects, user i agrees with the majority of $I_i(0)$ (namely the users marked as ‘close’).

Notation. We shall use the following notation. As in Section 3, we use $P(i)$ to denote the set of users whose preferences differ from those of user i on at most D objects, and $O(i)$ to denote the objects on which user i agrees with a majority of the users in $P(i)$. In addition, we denote:

- I_i^k : users not marked as ‘far’ after k iterations of \mathbf{F} . We define $I_i^0 \stackrel{\text{def}}{=} P$.
- B_i^k : users in I_i^k who disagree with user i on more than $17D$ objects.
- Q_i^k : objects on which user i disagrees with at least $|P(i)|/3$ users in B_i^k . Formally: $Q_i^k \stackrel{\text{def}}{=} \left\{ j \in O : \sum_{i' \in B_i^k} |A_{i,j} - A_{i',j}| \geq \frac{|P(i)|}{3} \right\}$.

We now turn to analyze the algorithm, focusing on a generic user i , who is (α, D) -prevalent. We start by considering members of $P(i)$ (proof is omitted).

Lemma 4. *With probability $1 - n^{-\Omega(1)}$, all members of $P(i)$ are neighbors of i in the proximity graph.*

Next, we consider the influence of the non-prevalent users. While the number of objects on which user i agrees with many ‘similar’ users can be easily bounded as in Lemma II, bounding the number of objects on which i disagrees with too many ‘dissimilar’ users (sufficient to influence the output of i) is more challenging. We bound the number of such objects by showing that each invocation of \mathbf{F} marks many ‘dissimilar’ users as ‘far.’ First we state a technical lemma, analogous to Lemmas I and II.

Lemma 5. Fix an invocation of Algorithm **F**. There are at most $4c \log n$ subsets O_s for which $O_s \not\subseteq O(i)$. Moreover, there are at most $4c \log n$ subsets O_s for which $O_s \subseteq O(i)$ and user i doesn't agree with at least $|P(i)|/2$ users on all object in O_s .

We can finally bound the objects dominated by users not in $P(i)$.

Lemma 6. With high probability, $|Q_i^K| < 17D$.

Proof: We show that if $|Q_i^{k-1}| \geq 17D$ then $|B_i^k| \leq (1 - \frac{1}{c})|B_i^{k-1}|$. The idea is that by Theorem 3, **Sep** does not assign most users in B_i^{k-1} to the same set as i , so whenever B_i^{k-1} is big enough, many of its users are marked as "far" in iteration k . We start by identifying, for each invocation of **F**, the object subsets on which the premise of Theorem 3(2) is fulfilled, and then use a counting argument.

Consider the k th iteration of Line 5. Let S' be the set of indices such that for any $s \in S'$ it holds (1) O_s contains at least one object from Q_i^{k-1} , and (2) i agrees with at least half the users in $P(i)$ on all objects in O_s . In other words, for $P_s(i) \stackrel{\text{def}}{=} \{i' \mid \text{dist}_{O_s}(i, i') = 0\}$, let $S' \stackrel{\text{def}}{=} \{s : O_s \cap Q_i^{k-1} \neq \emptyset \text{ and } |P_s(i)| \geq P(i)/2\}$.

Assume $|Q_i^{k-1}| \geq 17D$. Then $\mathbb{E}[|\Psi^k \cap Q_i^{k-1}|] = |Q_i^{k-1}| \frac{c \log n}{D} \geq 17c \log n$, and therefore, by the Chernoff Bound, w.h.p., there are at least $16c \log n$ object-subsets in **F** containing an object from Q_i^{k-1} . By Lemma 5 there are at most $8c \log n$ subsets in which user i doesn't agree with at least $P(i)/2$ other users, so we get $|S'| \geq 8c \log n$.

For every user b in B_i^{k-1} , let ϕ_b^k be the total number of **Sep** invocations at which user b was assigned to different subset than i at the k 'th iteration of **F**. Consider $s \in S'$: On one hand, as $O_s \cap Q_i^{k-1} \neq \emptyset$, then by Q_i^k definition, there is an object $j \in O_s$ on which user i disagrees with at least $P(i)/3$ of the users in B_i^{k-1} . On the other hand, as $|P_s(i)| \geq \frac{P(i)}{2}$, by Theorem 3, at most $\alpha n/6$ of those users are assigned by **Sep** to the same partition as user i . By definition of Q_i^k , $|Q_i^{k-1}| < 17D$ whenever $|B_i^{k-1}| < \frac{|P(i)|}{3}$, so we can bound the number of times "dissimilar" users are assigned to different partition as i by using the assumption $|B_i^{k-1}| \geq \frac{|P(i)|}{3} \geq \frac{\alpha n}{3}$:

$$\begin{aligned} \sum_{b \in B_i^k} \phi_b^k &= \sum_{s \in S'} |\{b \in B_i^k : b \text{ is separated from } i \text{ by } \mathbf{Sep} \text{ in } O_s\}| \\ &\geq \sum_{s \in S''} \left(|B_i^{k-1}| - \frac{\alpha n}{6} \right) \geq |S''| \frac{|B_i^{k-1}|}{2} \geq 4c \log n |B_i^{k-1}|. \end{aligned}$$

As Algorithm **F** marks every user with $\phi_b^k \geq 2c \log n$ as "far" and there are $2c^2 \log n$ object-subsets, then out of the users in B_i^{k-1} at least $\frac{4c \log n |B_i^{k-1}| - 2c \log n |B_i^{k-1}|}{2c^2 \log n} = \frac{1}{c} |B_i^{k-1}|$ users are marked as such, and hence $|B_i^k| \leq (1 - \frac{1}{c}) |B_i^{k-1}|$. Now, since $|Q_i^K| \geq 17D$ implies $|Q_i^k| \geq 17D$ for $k = 1, \dots, K$, we may conclude that if $|Q_i^K| \geq 17D$ then $B_i^K \leq 1$, contradiction. ■

We can now summarize the properties of Algorithm **A** for the case of honest users.

Theorem 5. Algorithm **A**($P, O, \alpha, D, 1$) predicts for each (α, D) -prevalent user its preference vector with $O(D)$ errors, with probability $1 - n^{\Omega(c)}$. Moreover, the probe complexity is $O\left(\frac{1}{\alpha} \left\lceil \frac{m}{nD} \right\rceil \cdot \log^3(m+n)\right)$.

We note that Theorem 5 improves on Algorithm CALCULATEPREFERENCES by Gilbert et al. [8] in the case of honest users, both in terms of probe complexity, and in terms of resilience to non-prevalent users.

References

1. Alon, N., Awerbuch, B., Azar, Y., Patt-Shamir, B.: Tell me who I am: an interactive recommendation system. In: Proc. 18th Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 1–10 (2006)
2. Awerbuch, B., Azar, Y., Lotker, Z., Patt-Shamir, B., Tuttle, M.: Collaborate with strangers to find own preferences. In: Proc. 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 263–269 (2005)
3. Awerbuch, B., Nisgav, A., Patt-Shamir, B.: Asynchronous Active Recommendation Systems. In: Tovar, E., Tsigas, P., Fouchal, H. (eds.) OPODIS 2007. LNCS, vol. 4878, pp. 48–61. Springer, Heidelberg (2007)
4. Awerbuch, B., Patt-Shamir, B., Peleg, D., Tuttle, M.: Improved recommendation systems. In: Proc. 16th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 1174–1183 (2005)
5. Azar, Y., Fiat, A., Karlin, A., McSherry, F., Saia, J.: Spectral analysis of data. In: Proc. 33rd ACM Symp. on Theory of Computing (STOC), pp. 619–626 (2001)
6. Azar, Y., Nisgav, A., Patt-Shamir, B.: Recommender systems with non-binary grades. In: Proc. 23rd Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), San Jose, CA (June 2011)
7. Drineas, P., Kerenidis, I., Raghavan, P.: Competitive recommendation systems. In: Proc. 34th ACM Symp. on Theory of Computing (STOC), pp. 82–90 (2002)
8. Gilbert, S., Guerraoui, R., Rad, F.M., Zadimoghaddam, M.: Collaborative scoring with dishonest participants. In: Proc. 22nd Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA), pp. 41–49 (2010)
9. Goel, S., Broder, A., Gabrilovich, E., Pang, B.: Anatomy of the long tail: ordinary people with extraordinary tastes. In: Proc. 3rd ACM Int. Conf. on Web Search and Data Mining (WSDM), pp. 201–210. ACM, New York (2010)
10. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval Journal 4(2), 133–151 (2001)
11. Kleinberg, J., Sandler, M.: Convergent algorithms for collaborative filtering. In: Proc. 4th ACM Conf. on Electronic Commerce (EC), pp. 1–10 (2003)
12. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Recommendation systems: A probabilistic analysis. In: Proc. 39th IEEE Symp. on Foundations of Computer Science (FOCS), pp. 664–673 (1998)
13. Nisgav, A., Patt-Shamir, B.: Finding similar users in social networks: extended abstract. In: Proc. 21st Ann. ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 169–177 (2009)
14. Papadimitriou, C.H., Raghavan, P., Tamaki, H., Vempala, S.: Latent semantic indexing: A probabilistic analysis. In: Proc. 17th ACM Symp. on Principles of Database Systems (PODS), pp. 159–168. ACM Press (1998)
15. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Analysis of recommendation algorithms for e-commerce. In: Proc. 2nd ACM Conf. on Electronic Commerce (EC), pp. 158–167. ACM Press (2000)

Asymptotic Modularity of Some Graph Classes *

Fabien de Montgolfier¹, Mauricio Soto¹, and Laurent Viennot²

¹ LIAFA, UMR 7089 CNRS - Université Paris Diderot

² INRIA and Université Paris Diderot

{fm,mausoto}@liafa.jussieu.fr Laurent.Viennot@inria.fr

Abstract. Modularity has been introduced as a quality measure for graph partitioning. It has received considerable attention in several disciplines, especially complex systems. In order to better understand this measure from a graph theoretical point of view, we study the modularity of a variety of graph classes. We first consider simple graph classes such as tori and hypercubes. We show that these regular graph families have asymptotic modularity 1 (that is the maximum possible). We extend this result to the general class of unit ball graphs of bounded growth metrics. Our most striking result concerns trees with bounded degree which also appear to have asymptotic modularity 1. This last result can be extended to graphs with constant average degree and to some power-law graphs.

1 Introduction

Graph partitioning (also known as graph clustering, see [1]) is a fundamental problem that recently became popular in the context of partitioning a network into communities. In that line of work, modularity [2,3] has been introduced as a quality measure of a network partitioning. It has rapidly become popular for various applications from biological networks to social networks measurement (see, e.g. [4,5,6]) or modelling [7,8]. It is now a standard measure for comparing the partitions obtained by various algorithms on practical networks.

From a graph-theoretic point of view, modularity has mainly been considered as a computational problem. Several heuristics have been proposed for computing a high modularity partition of a given network [9,10,11]. However, computing a maximum modularity partition is NP-complete [12].

This paper follows the approach of [2] with the goal to give some insight about what high modularity means from a graph-theoretic perspective. We are not interested in computing the modularity of a given graph extracted from “real” data. Instead, we analyze the modularity of *graph classes*. Given some well-known families of graphs, we wonder how good (or how bad) their members can be clustered. Our results are expressed as asymptotic modularity (limit when the graph size goes to infinity).

Informally, the modularity of a partition is, up to a normalization constant, the number of edges falling within clusters of the partition minus the expected

* Supported by the european project “EULER” (Grant No.258307) and by the INRIA project-team “GANG”.

number in an equivalent network with edges placed at random. It is normalized so that it amounts to a number between -1 and 1. The modularity of a graph is the maximum modularity of a partition of the graph over all partitions. As a single cluster partition trivially leads to a modularity of 0, the modularity of a graph is always between 0 and 1.

Our contribution resides in the analysis of the asymptotic modularity of various classes of graphs. We show that grids, tori, and hypercubes have asymptotic modularity 1, and can thus be well clustered despite their very regular structure. We extend this result to unit ball graphs of bounded growth metrics, showing a lower bound of the modularity of such graphs depending on the growth constant of the metric. On the other hand, stars (who are trees of unbounded maximum degree) have modularity as low as 0. However, trees of bounded degree have asymptotic modularity 1. This result can be extended to any class of constant average degree d connected graphs, showing a lower bound of $\frac{2}{d}$ for their asymptotic modularity.

As a consequence, high modularity is not necessarily the sign of an intrinsic community structure. Grids for example can be clustered in various manners to obtain high modularity. Second the modularity obtained for a connected graph with average degree d should be compared to $\frac{2}{d}$. A value not significantly larger cannot be interpreted as the sign of intrinsically clustered data.

The paper is organized as follows. Section 2 gives the formal definition and some preliminary remarks. Section 3 introduces a *decomposable graphs* framework and is devoted to tori, and hypercubes. Section 4 introduces the class of unit ball graphs of a bounded growth metric (which generalizes grids for instance) and show a lower bound on its asymptotic modularity. Finally, Section 5 is devoted to trees with small maximum degree, and graphs with constant average degree, and power-law graphs.

2 Revisiting Modularity

Given a graph $G = (V(G), E(G))$, we denote by n and m its number of vertices and edges respectively. Given a subset of vertices $S \subset V(G)$, the *size* $|S|$ of S is its number of vertices and its *volume* $\text{vol}(S) = \sum_{v \in S} \deg(v)$ is its degree sum. Let $E(S)$ denote the edge-set of the graph induced by S . $|E(S)|$ is thus the number of inner-edges in S . Similarly, $E(G)$ denotes the set of edges of G and $\text{vol}(G)$ denotes the volume of $V(G)$ (note that $\text{vol}(G) = 2m$).

A *clustering* is a partition of $V(G)$ into disjoint sets called *clusters*. Many quality measures can be defined for judging how good a clustering is. A popular one was introduced by Newman and Girvan [23] and is called *modularity*.

Definition 1 ([23]). *Let G be a graph and $\mathcal{C} = \{C_1, \dots, C_k\}$ a partition of $V(G)$ into k clusters. The modularity of the clustering \mathcal{C} is defined as:*

$$\mathcal{Q}(\mathcal{C}) = \sum_{i=1}^k \left[\frac{|E(C_i)|}{|E(G)|} - \frac{\text{vol}(C_i)^2}{\text{vol}(G)^2} \right]$$

The modularity of graph G , denoted $\mathcal{Q}(G)$ is the maximum modularity among all possible clusterings of G .

In this definition both the *left term* (densities) and the *right term* (volumes) take values between 0 and 1 so $\mathcal{Q}(\mathcal{C})$ belongs to $[-1, 1]$ (and in fact to $[-1/2, 1[$, see [12]). The left term is the ratio of internal edges. The less clusters you have, the greater it will be. To counterbalance this tendency to have few clusters, the right term is the quality of the same clustering of a random graph with same degree sequence. Detailed motivation for the definition is given by Newman [5]. Computing the modularity of a given clustering clearly takes $O(n + m)$ time, but giving the modularity of a graph is an NP-hard problem [12].

According to Brandes *et al.* [12], for any graph G , we have $0 \leq \mathcal{Q}(G) < 1$. We can refine the result further:

Lemma 1. *If the maximum degree of Graph G is Δ then $\mathcal{Q}(G) \leq 1 - \frac{\Delta^2}{4m^2}$.*

Proof. Let x be a degree Δ vertex. If C denotes the cluster containing x then $\text{vol}(C) \geq \Delta$. So the right term is at least $\Delta^2/4m^2$, and the left term at most 1.

Lemma 1 motivates to work on *asymptotic* modularity: no graph has modularity exactly 1, but the modularity of a *sequence* of graphs may have limit 1. A graph class has asymptotic modularity ℓ if any sequence of graph in the class taken with increasing number of vertices has a limit modularity of ℓ . We focus on $\ell = 1$, the ideal case. We now give some simple facts about modularity.

Lemma 2. *Given a cluster C , if $G[C]$ is not connected, then breaking C in $C_1 \sqcup C_2$ by assigning each component of $G[C]$ to C_1 or C_2 improves modularity.*

Proof. We have $\text{vol}(C)^2 \geq \text{vol}(C_1)^2 + \text{vol}(C_2)^2$ (since $x \mapsto x^2$ is superlinear) so right term of Definition 1 decreases. And the left term is the ratio of internal edge, so it remains unchanged as each edge of C goes within C_1 or C_2 . (In fact $\text{vol}(C)^2 > \text{vol}(C_1)^2 + \text{vol}(C_2)^2$ as soon as both C_1 and C_2 contain an edge.)

Corollary 1 (Brandes *et al.* [12]). *There exists a maximum modularity clustering where each cluster is connected.*

Lemma 3 (DasGupta and Devendra [13]). *If Clustering \mathcal{C} contains k clusters then $\mathcal{Q}(\mathcal{C}) \leq 1 - 1/k$.*

As a consequence, note that a modularity close to 1 can only be obtained through a large number of clusters.

Proposition 1. *A star of m rays ($K_{1,m}$ graph) has modularity 0.*

Proof. Every clustering consists in a first cluster C_0 containing the center together with $a \leq m$ other vertices, plus other clusters that are stable sets. According to Corollary 1, in a clustering with maximal modularity, each one contains one vertex. Their number of edges is 0 and their volume is 1. Then $\mathcal{Q}(\mathcal{C}) =$

$$\sum_{i=1}^k \left[\frac{|E(C_i)|}{|E(G)|} - \frac{\text{vol}(C_i)^2}{\text{vol}(G)^2} \right] = \frac{a}{m} - \frac{(m+a)^2 + (m-a)}{4m^2} = \frac{(m-a)(a-m-1)}{4m^2}$$

Maximum is for $a = m$, i.e. an unique cluster C_0 . The modularity is then 0.

Consequently, trees may have modularity as low as 0. We shall now present, on the other hand, classes having modularity 1. We shall see further that bounded degree trees also have modularity 1 (Section 5.1).

3 Modularity of Decomposable Graphs

3.1 Decomposable graphs

To reach asymptotic modularity 1, it is necessary that the limit (when n goes to infinity) of the left term be 1 and that of the right term be 0. Furthermore, Lemma 3 implies that the number k of clusters must also tends to infinity. That leads us to the following definition:

Definition 2. A graph G is (k, c, e) -decomposable if it can be split into k clusters such that each cluster has volume at most $c \frac{\text{vol}(G)}{k}$, and the number of inter-cluster edges is at most $|E(G)| \times e$.

Intuitively, a graph is decomposable if it can be split into k clusters of roughly balanced volume with few edges in-between clusters. The following lemma bounds the modularity of a decomposable graph.

Lemma 4. If G is (k, c, e) -decomposable, then $\mathcal{Q}(G) \geq 1 - e - \frac{c^2}{k}$.

Proof. Consider a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$ such that G is (k, c, e) -decomposable. Then $\sum_{i=1}^k |E(C_i)| \geq m - em$ and $\sum_{i=1}^k \text{vol}(C_i)^2 \leq \sum_{i=1}^k (c \text{vol}(G)/k)^2 \leq \frac{c^2}{k} \text{vol}(G)^2$. By Definition 1 we get $\mathcal{Q}(\mathcal{C}) \geq \frac{m-em}{m} - \frac{\frac{c^2}{k} \text{vol}(G)^2}{\text{vol}(G)^2} \geq 1 - e - \frac{c^2}{k}$.

Let us apply this tool to computation of asymptotic modularity of some classes.

3.2 Multidimensional Torus

Given a vector \mathbf{p} of d numbers p_1, \dots, p_d let the d -dimensional torus $G_{\mathbf{p}} = C_{p_1} \times \dots \times C_{p_d}$ (where C_a is a cycle of a vertices and \times is the Cartesian product of graphs). We have $n = \prod_{i=1}^d p_i$ and $m = dn$.

Lemma 5. d -dimensional tori are $(b, 2, \frac{2b}{dn^{1/d}})$ -decomposable.

Proof. Consider the largest dimension $p = \max_{i=1}^d p_i$ of the torus. We have $p \geq n^{1/d}$. Let \mathcal{C}_b be the clustering where the torus is split into b slices according to the largest dimension. The cluster C_i , with $0 \leq i < b$, corresponds to nodes whose coordinate in the largest dimension falls in the interval $[i \lceil p/b \rceil, (i+1) \lceil p/b \rceil]$.

The nodes with same coordinate in the largest dimension form an $d-1$ -hyperplane of size n/p . The number of edges outgoing a cluster is thus at most $2n/p$. So $e \leq \frac{2bn/p}{dn} \leq \frac{2b}{dn^{1/d}}$. Cluster C_i contains at most $n/p \lceil p/b \rceil \leq \frac{n}{b} + \frac{n}{p} \leq \frac{2n}{b}$ nodes as $b \leq p$. The degree of each node is $2d$ so $\text{vol}(C_i) \leq \frac{4dn}{b}$. We have $\frac{\text{vol}(C_i)}{\text{vol}(G)} \leq \frac{4dn}{b} \frac{1}{2dn} \leq \frac{2}{b}$.

Theorem 1. A d -dimensional torus with n nodes has modularity $1 - O(n^{-1/2d})$.

Proof. According to Lemma 4 $\mathcal{Q}(G) \geq 1 - \frac{2b}{dn^{1/d}} - \frac{2^2}{b}$. This value is maximal for $\frac{2n^{-1/d}}{d} = \frac{4}{b^2}$, i.e. $b = \sqrt{2dn^{1/2d}}$. For this value of b , we obtain $\mathcal{Q}(\mathcal{C}_b) \geq 1 - \frac{4\sqrt{2}}{\sqrt{d}} n^{-1/2d} \geq 1 - O(n^{-1/2d})$

A very similar proof (not presented here) can show the same bound applies to grids (with $c = 4$ instead of 2). Note that tori and grids have many natural cuts and there are several possible partitions with asymptotic modularity 1.

3.3 Hypercube

The d -dimensional hypercube has $n = 2^d$ vertices and $\text{vol}(G) = d2^d$. Its vertices are identified with the d -digits binary numbers. So we may say, for instance, that there is an edge uv iff u and v differ on one bit.

Lemma 6. d -dimensional hypercubes are $(2^b, 1, \frac{b}{d})$ -decomposable.

Proof. Given an integer $b < d$, let the b -prefix clustering \mathcal{C}_b of d -dimensional hypercube be the clustering with 2^b clusters such that cluster C_a contains vertices beginning with prefix a where a is a b bits binary string.

Among the d edges incident to a given vertex, b are external (go to another cluster, if vertices differ on a bit of number $i \leq b$) and $d - b$ are internal. So clearly $e = \frac{b}{d}$. Cluster C_i contains 2^{d-b} vertices and has thus volume $d2^{d-b}$. Then $\frac{\text{vol}(C_i)}{\text{vol}(G)} = 2^{-b} = 1/k$, i.e. $c = 1$.

Theorem 2. A hypercube of dimension d has modularity $1 - O(\frac{\log \log n}{\log n})$.

Proof. According to Lemma 4, $\mathcal{Q}(G) \geq 1 - \frac{b}{d} - \frac{1}{2^b}$. It reaches its maximum when $b = \log_2(d \ln 2)$ and: $\mathcal{Q}(G) \geq 1 - \frac{\log_2(d \ln 2)}{d} - \frac{1}{d \ln 2} = 1 - O(\frac{\log \log n}{\log n})$ since $d = \log_2 n$.

4 Unit Ball Graph of a Bounded Growth Metric

We now turn to the general class of unit ball graphs. Note that a grid can be defined as the unit ball graph of a regular mesh of points in a d -dimensional space. Varying the unit radius allows to play with the density of the grid. Varying the point positions yields non uniform grids. We restrict to the case where the metric induced by the points has bounded growth.

We are now interested in a cloud of points V from a metric space \mathcal{E} (in the most usual cases, \mathcal{E} is \mathbb{R}^d or a d -dimensional \mathbb{R} -space, but we do not require that). We suppose there is a metric called dist (a function $\mathcal{E} \times \mathcal{E} \rightarrow \mathbb{R}^+$). Given

a length $R \in \mathbb{R}^+$, we define $E_R = \{uv | dist(u, v) \leq R\}$. We call *R-ball graph* or (since all balls have same radius) *unit ball graph* the graph $G_R = (V, E_R)$, and the *modularity of a clustering \mathcal{C} of V* is then the modularity computed in G_R . In this section, we lower bound the asymptotic modularity of G_R provided that the radius R is taken in an appropriate range and that the metric has bounded growth.

4.1 Bounded Growth Metrics

The *bounded growth* property is a generalization of the Euclidean dimension. We let $B(u, r) = \{v \in V | dist(u, v) \leq r\}$ denote the *ball* with center u and radius r .

Definition 3 (Grid Dimension [14] also known as bounded growth property). Space V has growth $\gamma > 0$ if doubling the radius of any ball does not increase its volume by more than γ : $\forall x \in V, \forall r > 0, |B(x, 2r)| \leq \gamma \cdot |B(x, r)|$

For instance, the d -dimensional torus has growth 2^d . This definition also applies to continuous spaces. For instance a d -dimensional Euclidean space with the L_∞ norm has growth $\gamma = 2^d$.

4.2 *R*-Nets

Now let us recall an algorithmic tool: the *R*-net. It is a covering of the space by points mutually at least R apart. Among its many uses, it allows to define a clustering where the radius of each cluster is bounded by R . The current section presents such clusterings, the next one proves that they have good modularity.

Definition 4 (*R*-net). A subset U of V is a *R*-net if $\forall u, u' \in U, dist(u, u') > R$ (points are R apart), and $\forall v \in V \exists u \in U dist(u, v) \leq R$ (U covers V)

A greedy process can easily construct an *R*-net: take any vertex $u \in V$, put it in U , and recurse on $V - B(u, R)$. Notice we are not interested in minimizing $|U|$ nor maximizing $B(u, R)$.

Then an *R*-net U allows to define a clustering (denoted \mathcal{C}_U) of V . It consists in $|U|$ clusters. For each $u_i \in U$, we define cluster C_i as the points of V whose nearest neighbors in U is u_i (ties are arbitrarily broken). The *cover* part of the definition implies that for any $v \in C_i$, we have $dist(u_i, v) \leq R$. Cluster radius is thus at most R . Additionally, as a consequence of the nearest neighbor choice, a point $v \in B(u_i, R/2)$ cannot be in C_j with $j \neq i$ as $dist(u_i, u_j) > R$. Cluster C_i thus contains $B(u_i, R/2)$.

4.3 Modularity of an *R*-Net Clustering

Theorem 3. Let V be a finite space of n points, together with Metric $dist$, and having growth at most γ , and $R \geq 0$ such that for all $v \in V$ we have $|B(v, R/2)| > 1$ and $|B(v, R/2)| = o(\sqrt{n})$. We have:

$$\mathcal{Q}(G_R) \geq \frac{1}{2\gamma^3} - o(1)$$

Proof. Let $U = \{u_1, \dots, u_k\}$ be an R -net of V and let $\mathcal{C}_U = \{C_i\}_{i \in \{1, \dots, k\}}$ be the associated clustering as defined previously. Then, by construction, $B(u_i, R/2) \subseteq C_i \subseteq B(u_i, R)$. Let $b_i = |B(u_i, R/2)|$. The bounded growth hypothesis gives: $b_i \leq |C_i| \leq \gamma b_i$. Points from $B(u_i, R/2)$ are all mutually linked in G_R (from its definition) and form a clique of size b_i , included within C_i . Then we have:

$$\mathcal{Q}(\mathcal{C}_U) = \sum_{i=1}^k \left[\frac{|E(C_i)|}{m} - \frac{(\sum_{v \in C_i} d_v)^2}{4m^2} \right] \geq \sum_{i=1}^k \left[\frac{b_i(b_i-1)}{2m} - \frac{(\sum_{v \in C_i} d_v)^2}{4m^2} \right]$$

for every $v \in C_i$, its degree is $d_v = |B(v, R)| \leq |B(u_i, 2R)| \leq \gamma^2 b_i$, and thus:

$$\mathcal{Q}(\mathcal{C}_U) \geq \sum_{i=1}^k \left[\frac{b_i(b_i-1)}{2m} - \frac{(|C_i|\gamma^2 b_i)^2}{4m^2} \right] \geq \sum_{i=1}^k \left[\frac{b_i(b_i-1)}{2m} - \frac{\gamma^6 b_i^4}{4m^2} \right]$$

$$\text{As we have: } \sum_{i=1}^k b_i(b_i-1) \leq 2m = \sum_{u \in V(G)} d_u \leq \sum_{i=1}^k |C_i|\gamma^2 b_i \leq \gamma^3 \sum_{i=1}^k b_i^2,$$

$$\text{we get: } \mathcal{Q}(\mathcal{C}_U) \geq \frac{\sum_{i=1}^k b_i(b_i-1)}{\gamma^3 \sum_{i=1}^k b_i^2} - \frac{\gamma^6 \bar{b}^2 \sum_{i=1}^k b_i^2}{2m \sum_{i=1}^k b_i(b_i-1)}$$

where $\bar{b} = \max_i \{b_i\}$. The ball size hypothesis of the theorem implies $b_i \geq 2$ and $b_i = o(\sqrt{n})$. We have thence $\frac{\sum_{i=1}^k b_i(b_i-1)}{\sum_{i=1}^k b_i^2} = 1 - \frac{\sum_{i=1}^k b_i}{\sum_{i=1}^k b_i^2} \geq 1 - \frac{\sum_{i=1}^k b_i}{\sum_{i=1}^k 2b_i} = \frac{1}{2}$.

Back to modularity: $\mathcal{Q}(\mathcal{C}_U) \geq \frac{1}{2\gamma^3} - \frac{\gamma^6 \bar{b}^2}{m}$. Finally, as $\bar{b} = o(\sqrt{n})$ and $m \geq \frac{n}{2}$, we get $\mathcal{Q}(\mathcal{C}_U) \geq \frac{1}{2\gamma^3} - o(1)$: the asymptotic modularity of the class is $\frac{1}{2\gamma^3}$.

5 Constant Average Degree Graphs

We now investigate some sparse classes of graphs, in the sense that the average degree is a constant d and does no go to infinity with n . Trees are such a class. We prove that they have asymptotic modularity 1 when maximum degree is low enough. We extend then the result to some constant average degree graph classes using tree spanners.

5.1 Trees of Small Maximum Degree

Let T be a tree. The removal of any edge of T splits T into two parts. We are interested in the size of the smaller part. A *centroid edge* of T is an edge chosen to maximize the size of the smaller part (a centroid edge is thus incident with the unique or the two *centroid vertices* of the tree).

Let us now introduce a clustering tool, Algorithm $\text{greedy-decompose}_{\leq h}$. Given a forest F and a fixed integer $h \geq 1$, the algorithm works as follows. As long as F contains a tree T_0 with strictly more than h vertices, then it finds a centroid edge e of T_0 and removes it ($F := F - e$). The clustering \mathcal{C}_h of T is the forest computed by $\text{greedy-decompose}_{\leq h}$ using T as initial forest.

Lemma 7. *If a tree has maximum degree at most Δ then $\text{greedy-decompose}_{\leq h}$ computes clusters of size $\frac{h}{\Delta} \leq |C_i| \leq h$.*

Proof. Clearly each cluster has size at most h . Let e be the centroid edge removed from a tree T_0 with n vertices. Let s be the size of the smallest part of $T_0 - e$ (we have $s \leq n/2$). Let x be the vertex incident with e and belonging to the largest part of $T_0 - e$. For every edge e' incident with x , the part of $T_0 - e'$ not containing x has at most s vertices (otherwise e is not a centroid edge). As x has degree at most Δ then $n \leq \Delta s + 1$ (each of the $\leq \Delta$ parts size is bounded by s , and $+1$ counts x itself). So $s \geq \frac{n-1}{\Delta}$. In the algorithm we split only trees with size $n > h$ so $s \geq \frac{h}{\Delta}$.

So the cluster sizes are roughly balanced (up to a ratio Δ) and parametrized with h . Indeed, as the sum of cluster sizes is n we have that $\text{greedy-decompose}_{\leq h}$ splits T into $\frac{n}{h} \leq k \leq \frac{\Delta n}{h}$ clusters.

Lemma 8. *For any tree T of degree bounded by Δ there exists k such that T is $(k, \Delta^2, \frac{k-1}{n-1})$ -decomposable.*

Proof. Splitting a tree of n vertices into k connected clusters using $\text{greedy-decompose}_{\leq h}$ yields a fraction $\frac{k-1}{n-1}$ of external edges. As each vertex has degree at most Δ , for each cluster $\text{vol}(C_i) \leq \Delta h$. As $h \leq \frac{\Delta n}{k}$ we get $\text{vol}(C_i) \leq \Delta^2 \frac{\text{vol}(T)}{k}$ (as $\text{vol}(T) = 2(n-1) \geq n$ for $n \geq 2$).

Theorem 4. *Trees with max. degree $\Delta = o(\sqrt[5]{n})$ have asymptotic modularity 1.*

Proof. Let us find a good k . We have control over h and it gives k with $\frac{n}{h} \leq k \leq \frac{\Delta n}{h}$. Lemma 4 gives

$$\mathcal{Q}(T) \geq 1 - \frac{k-1}{n-1} - \frac{\Delta^4}{k} \geq 1 - \frac{(\Delta n/h) - 1}{n-1} - \frac{\Delta^4}{(n/h)} \geq 1 + \frac{1}{n-1} - \frac{\Delta n}{h(n-1)} - \frac{\Delta^4}{n} h$$

For maximizing \mathcal{Q} we take derivative: $\frac{d\mathcal{Q}}{dh} = \frac{\Delta n}{h^2(n-1)} - \frac{\Delta^4}{n}$. Solving $\frac{d\mathcal{Q}}{dh} = 0$ gives $h = \frac{n}{\Delta \sqrt{\Delta(n-1)}}$. Using the clustering from $\text{greedy-decompose}_{\leq \frac{n}{\Delta \sqrt{\Delta(n-1)}}}$ we get

$$\begin{aligned} \mathcal{Q}(T) &\geq 1 + \frac{1}{n-1} - \frac{\Delta n}{(n-1)} \cdot \frac{\Delta \sqrt{\Delta(n-1)}}{n} - \frac{\Delta^4}{n} \cdot \frac{n}{\Delta \sqrt{\Delta(n-1)}} \\ &\geq 1 + \frac{1}{n-1} - \frac{2\Delta^{2.5}}{\sqrt{n-1}}. \text{ If } \Delta = o(\sqrt[5]{n}) \text{ then } \frac{2\Delta^{2.5}}{\sqrt{n-1}} = o(1). \end{aligned}$$

On the other hand, given a tree class, if there is a constant $c \geq 0$ such that there is a sequence T_i of trees of the class, each one containing a vertex x of degree cn then according to Lemma 1 $\mathcal{Q}(T_i) \leq 1 - \frac{cn^2}{4(n-1)^2} < 1 - \frac{c}{4}$. The asymptotic modularity of a tree class with maximal degree $\Omega(n)$ is strictly less than 1.

5.2 Graphs of Average Degree d

Let $d = 2m/n$ be the average degree of a graph. In this section we prove that a class of graphs with constant average degree d and bounded maximum degree have good asymptotic modularity (maximum degree is bounded by a function of n but may go to infinity). It is an extension to graphs of results from the previous section. Indeed, given a connected graph G of maximum degree Δ , we take a spanning tree T of G (thus having maximum degree at most Δ) and apply *greedy-decompose* $_{\leq h}$ on T . Lemma 7 remains true and we still have $\frac{n}{h} \leq k \leq \frac{\Delta n}{h}$ clusters. It is a clustering of G as T spans G . Each cluster is connected and has volume at most $\Delta|C_i| \leq \Delta h$.

Lemma 9. *For any connected graph G of maximum degree bounded by Δ and of average degree d there exists k such that G is $(k, \frac{\Delta^2}{d}, 1 - \frac{n-k}{m})$ -decomposable.*

Proof. Among the m edges of G , $n - k$ belong to the clusters of T and are thus internal (we can not say anything about edges not in T). So we have $e \leq 1 - \frac{n-k}{m}$.

On the other hand $k \frac{\text{vol}(C_i)}{\text{vol}(G)} \leq k \frac{\Delta h}{2m} \leq \frac{\Delta^2}{d}$ since $k \leq \frac{\Delta n}{h}$ and $d = \frac{2m}{n}$.

Theorem 5. *Connected graphs of average degree d and of maximum degree $\Delta = o(\sqrt[5]{d^3 n})$ have asymptotic modularity at least $2/d$.*

Proof. Using Lemma 4 we have: $\mathcal{Q}(G) \geq 1 - \left(1 - \frac{n-k}{m}\right) - \left(\frac{\Delta^2}{d}\right)^2 \cdot \frac{1}{k} \geq \frac{n}{m} - \frac{\Delta n}{hm} - \frac{\Delta^4 h}{nd^2}$ since $\frac{1}{k} \leq \frac{h}{n}$ and $k \leq \frac{\Delta n}{h}$. Finally: $\mathcal{Q}(G) \geq \frac{2}{d} - \frac{2\Delta}{dh} - \frac{\Delta^4}{nd^2}h$. For maximizing \mathcal{Q} we take the derivative in h : $\mathcal{Q}' = \frac{2\Delta}{d} \frac{1}{h^2} - \frac{\Delta^4}{nd^2}$. It is zero for $h = \frac{\sqrt{2dn}}{\Delta^{1.5}}$. So taking clustering *greedy-decompose* $_{\frac{\sqrt{2dn}}{\Delta^{1.5}}}$ we have $\mathcal{Q}(G) \geq \frac{2}{d} - \frac{2\Delta}{d} \frac{\Delta^{1.5}}{\sqrt{2dn}} - \frac{\Delta^4}{nd^2} \frac{\sqrt{2dn}}{\Delta^{1.5}} \geq \frac{2}{d} - \frac{2^{1.5} \Delta^{2.5}}{d^{1.5} \sqrt{n}} \geq \frac{2}{d} - o(1)$ since $\Delta = o(\sqrt[5]{d^3 n})$.

Notice that for trees $d = \frac{2(n-1)}{n}$ has limit 2. This result thus generalizes the previous one.

5.3 Power-Law Graphs

Let us say a graph class has the *power-law* property of parameter α if for any graph the proportion of vertices of degree at least k is $O(k^{-\alpha})$. Note that our definition is much broader than the usual definition of power law graph.

Theorem 6. *A power-law connected graph class of parameter $\alpha > 5$ with constant average degree d has asymptotic modularity at least $\frac{2}{d}$.*

Proof. We first prove that the maximum degree of each graph is $\Delta = o(\sqrt[5]{n})$. The number of vertices of degree at least k is $nO(k^{-\alpha})$. Consider $k = n^\gamma$ for some $\gamma \in (\frac{1}{\alpha}, \frac{1}{5})$. Then we have $nO(k^{-\alpha}) = O(n^{1-\alpha\gamma}) = o(1)$. For n large enough, there are thus no vertex of degree at least n^γ . We thus have $\Delta = O(n^\gamma) = o(\sqrt[5]{n})$. We can finally apply the previous theorem since d is a constant.

6 Conclusion

Usually, people consider that a graph has a good modularity if there is some intrinsic clustering that induces the existence of edges, but is somehow blurred by adding or removing a few edges. Then a clustering algorithm has to “retrieve” this hidden structure. For instance, according to Newman and Girval [3], “*Values approaching $Q = 1$, which is the maximum, indicate strong community structure. In practice, values for such networks typically fall in the range from about 0.3 to 0.7. Higher values are rare.*” We show however that some very regular graphs, like tori or hypercubes, where no “hidden naturally clustered structure” seems to exist, may also have a high quality clustering. That relativizes the use of modularity as a measurement of the “*clustering*” property of data: we think it should be seen only as the objective function of an algorithm.

References

1. Schaeffer, S.: Graph clustering. Computer Science Review 1(1), 27–64 (2007)
2. Girvan, M., Newman, M.: Community structure in social and biological networks. P.N.A.S 99(12), 7821–7826 (2002)
3. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E 69(066133) (2004)
4. Guimera, R., Amaral, L.A.N.: Functional cartography of complex metabolic networks. Nature 433(7028), 895–900 (2005)
5. Newman, M.E.: Modularity and community structure in networks. PNAS 103(23), 8577–8582 (2006)
6. Olesen, J., Bascompte, J., Dupont, Y., Jordano, P.: The modularity of pollination networks. Proceedings of the National Academy of Sciences 104(50), 19891–19896 (2007)
7. Guimera, R., Sales-Pardo, M., Amaral, L.A.N.: Modularity from fluctuations in random graphs and complex networks. Phys. Rev. E 70, 025101 (2004)
8. Kashtan, N., Alon, U.: Spontaneous evolution of modularity and network motifs. P.N.A.S 102(39), 13773–13778 (2005)
9. Duch, J., Arenas, A.: Community detection in complex networks using extremal optimization. Phys. Rev. E 72(2), 27104 (2005)
10. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E 70, 066111 (2004)
11. Blondel, V., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment (2008)
12. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Transactions on Knowledge and Data Engineering 20, 172–188 (2008)
13. DasGupta, B., Desai, D.: On the complexity of newman’s community finding approach for biological and social networks (2011),
<http://arxiv.org/abs/1102.0969>
14. Karger, D., Ruhl, M.: Finding nearest neighbors in growth-restricted metrics. In: Proceedings of STOC, pp. 741–750. ACM (2002)

Program Size and Temperature in Self-assembly*

Ho-Lin Chen¹, David Doty¹, and Shinnosuke Seki²

¹ California Institute of Technology,

Department of Computing and Mathematical Sciences, Pasadena, CA, USA

holinc@gmail.com, ddoty@caltech.edu

² Department of System Sciences for Drug Discovery, Kyoto University, 46-29,

Yoshida-Shimo-Adachi-cho, Sakyo-ku, Kyoto, 606-8501, Japan

sseki@pharm.kyoto-u.ac.jp

Abstract. Winfree’s abstract Tile Assembly Model (aTAM) is a model of molecular self-assembly of DNA complexes known as tiles, which float freely in solution and attach one at a time to a growing “seed” assembly based on specific binding sites on their four sides. We show that there is a polynomial-time algorithm that, given an $n \times n$ square, finds the minimal tile system (i.e., the system with the smallest number of distinct tile types) that uniquely self-assembles the square, answering an open question of Adleman, Cheng, Goel, Huang, Kempe, Moisset de Espanés, and Rothemund (*Combinatorial Optimization Problems in Self-Assembly*, STOC 2002). Our investigation leading to this algorithm reveals other positive and negative results about the relationship between the size of a tile system and its “temperature” (the binding strength threshold required for a tile to attach).

1 Introduction

Tile self-assembly is an algorithmically rich model of “programmable crystal growth”. It is possible to design monomers (square-like “tiles”) with specific binding sites so that, even subject to the chaotic nature of molecules floating randomly in a well-mixed chemical soup, they are guaranteed to bind so as to deterministically form a single target shape. This is despite the number of different types of tiles possibly being much smaller than the size of the shape and therefore having only “local information” to guide their attachment. The ability to control nanoscale structures and machines to atomic-level precision will rely crucially on sophisticated self-assembling systems that automatically control their own behavior where no top-down externally controlled device could fit.

* The first author was supported by the Molecular Programming Project under NSF grant 0832824, the second author was supported by a Computing Innovation Fellowship under NSF grant 1019343, and the third author was supported by NSERC Discovery Grant R2824A01 and the Canada Research Chair in Biocomputing to Lila Kari and the Funding Program for Next Generation World-Leading Researchers (NEXT program) to Professor Yasushi Okuno.

A practical implementation of self-assembling molecular tiles was proved experimentally feasible in 1982 by Seeman [8] using DNA complexes formed from artificially synthesized strands. Experimental advances have delivered increasingly reliable assembly of algorithmic DNA tiles with error rates of 10% per tile in 2004 [7], 1.4% in 2007 [5], 0.13% in 2009 [3], and 0.05% in 2010 [4]. Erik Winfree [11] introduced the abstract Tile Assembly Model (aTAM) – based on a constructive version of Wang tiling [9,10] – as a simplified mathematical model of self-assembling DNA tiles. Winfree demonstrated the computational universality of the aTAM by showing how to simulate an arbitrary cellular automaton with a tile assembly system. Building on these connections to computability, Rothemund and Winfree [6] investigated a self-assembly resource bound known as *tile complexity*, the minimum number of tile types needed to assemble a shape. They showed that for most n , the problem of assembling an $n \times n$ square has tile complexity $\Omega(\frac{\log n}{\log \log n})$, and Adleman, Cheng, Goel, and Huang [1] exhibited a construction showing that this lower bound is asymptotically tight.

The results of this paper are motivated by the following problem posed in 2002. Adleman, Cheng, Goel, Huang, Kempe, Moisset de Espanés, and Rothemund [2] showed that there is a polynomial time algorithm for finding a minimum size tile system (i.e., system with the smallest number of distinct tile types) to uniquely self-assemble a given $n \times n$ square¹ subject to the constraint that the tile system’s “temperature” (binding strength threshold required for a tile to attach) is 2. They asked whether their algorithm could be modified to remove the temperature 2 constraint. Our main theorem answers this question affirmatively. Their algorithm works by brute-force search over the set of all temperature-2 tile systems with at most $O(\frac{\log n}{\log \log n})$ tile types, using the fact proven by Adleman, Cheng, Goel, and Huang [1] that such an upper bound on tile complexity suffices to assemble any $n \times n$ square. A simple counting argument shows that for any constant τ , the number of tile systems with glue strengths and temperature at most τ and $O(\frac{\log n}{\log \log n})$ tile types is bounded by a polynomial in n . One conceivable approach to extending the algorithm to arbitrary temperature is to prove that for any tile system with K tile types, the strengths and temperature can be re-assigned so that they are upper-bounded by a constant or slow-growing function of K , without affecting the behavior of the tile system.² However, we show that this approach cannot work, by demonstrating that for each K , there is a tile system with K tile types whose behavior cannot be preserved using any temperature less than $2^{K/4}$. The proof crucially uses 3-cooperative binding, meaning attachment events that require three different glues of a tile to match the assembly. On the other hand, we show that any 2-cooperative tile

¹ The square is encoded by a list of its points, so the algorithm’s running time is polynomial in n .

² We define “behavior” more formally in Section 3. Briefly, we consider a tile system’s behavior unaltered by a reassignment of strengths and temperature if, for each tile type t , the reassignment has not altered the collection of subsets of sides of t that have sufficient strength to bind.

system (which captures a wide and useful class of systems) with K tile types is behaviorally equivalent to a system with temperature at most $2K + 2$.

Of course, the choice of integer glue strengths is an artifact of the model. Nonetheless, our investigation does reflect fundamental questions about how finely divided molecular binding energies must be in a real molecular self-assembly system. The requirement of integer strengths is simply one way of “quantizing” the minimum distinction we are willing to make between energies and then rescaling so that this quantity is normalized to 1.³ Our 3-cooperative lower bound therefore shows that in general, certain self-assembling systems that have very large gaps between some of their binding energies nonetheless require other binding energies to be extremely close (exponentially small in terms of the larger gaps) and yet still unequal. This can be interpreted as an infeasibility result if one defines “exponentially fine control” of binding energies as “infeasible” to execute in any real laboratory, since no implementation of the specified tile behavior can use courser energies.

As a converse to the exponential temperature lower bound stated above, we show that there is a polynomial-time algorithm that, given any tile system \mathcal{T} with K tile types specified by its desired binding behavior, finds a temperature and glue strengths at most $2^{O(K)}$ that implement this behavior or reports that no such strengths exist. This algorithm is used to show our main result, that there is a polynomial-time algorithm that, given an $n \times n$ square, determines the smallest tile assembly system (at any temperature) that uniquely self-assembles the square, answering the open question of [2].

2 Abstract Tile Assembly Model

This section gives a brief informal sketch of the abstract Tile Assembly Model (aTAM).

A *tile type* is a unit square with four sides, each having a *glue label* (often represented as a finite string). We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile*. An *assembly* (a.k.a., *supertile*) is a positioning of tiles on (part of) the integer lattice \mathbb{Z}^2 ; i.e., a partial function $\mathbb{Z}^2 \dashrightarrow T$. For a set of tile types T , let $\Lambda(T)$ denote the set of all glue labels of tile types in T . We may think of a tile type as a function $t : \{\text{N, S, E, W}\} \rightarrow \Lambda(T)$ indicating, for each direction $d \in \{\text{N, S, E, W}\}$ (“north, south, east, west”), the glue label $t(d)$ appearing on side d . A *strength function* is a function $g : \Lambda(T) \rightarrow \mathbb{N}$ indicating, for each glue label ℓ , the strength $g(\ell)$ with which it binds. Two adjacent tiles in an assembly *interact* if the glue labels on their abutting sides are equal and have positive strength according to g . Each assembly induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The assembly is τ -*stable* if every cut of its binding graph has strength at least τ , where the weight of an edge is

³ Indeed, our proof does not require that strengths be integer, merely that the distance between the smallest energy strong enough to bind and the largest energy too weak to bind be at least 1.

the strength of the glue it represents. That is, the assembly is stable if at least energy τ is required to separate the assembly into two parts.

A *tile assembly system* (TAS) is a quadruple $\mathcal{T} = (T, \sigma, g, \tau)$, where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is a finite, τ -stable *seed assembly*, $g : \Lambda(T) \rightarrow \mathbb{N}$ is a *strength function*, and τ is a *temperature*. In this paper, we assume that all seed assemblies σ consist of a single tile type (i.e., $|\text{dom } \sigma| = 1$). Given a TAS $\mathcal{T} = (T, \sigma, g, \tau)$, an assembly α is *producible* if either $\alpha = \sigma$ or if β is a producible assembly and α can be obtained from β by placing a single tile type t on empty space (a position $p \in \mathbb{Z}^2$ such that $\beta(p)$ is undefined), such that the resulting assembly α is τ -stable. In this case write $\beta \rightarrow_1 \alpha$ (α is producible from β by the stable attachment of one tile), and write $\beta \rightarrow \alpha$ if $\beta \rightarrow_1^* \alpha$ (α is producible from β by the stable attachment of zero or more tiles). An assembly is *terminal* if no tile can be stably attached to it. Let $\mathcal{A}[\mathcal{T}]$ be the set of producible assemblies of \mathcal{T} , and let $\mathcal{A}_\square[\mathcal{T}] \subseteq \mathcal{A}[\mathcal{T}]$ be the set of producible, terminal assemblies of \mathcal{T} . Given a connected shape $S \subseteq \mathbb{Z}^2$, a TAS \mathcal{T} *uniquely self-assembles* S if $\mathcal{A}_\square[\mathcal{T}] = \{\hat{\alpha}\}$ and $\text{dom } \hat{\alpha} = S$.

3 Finding Strengths to Implement a Tile System

In this section we show that there is a polynomial-time algorithm that, given a desired behavior of a tile set with unspecified glue strengths or temperature, can find strengths and temperature to implement that behavior that are at most exponential in the number of tile types, or report that no such strengths and temperature exist. This algorithm is the primary technical tool used in the proof of our main result, Theorem 4.2.

First, we formalize what we mean by the “behavior” of a tile system. Let T be a set of tile types, and let $t \in T$. Given a strength function $g : \Lambda(T) \rightarrow \mathbb{N}$ and a temperature $\tau \in \mathbb{Z}^+$, define the *cooperation set of t with respect to g and τ* to be the collection $\mathcal{D}_{g,\tau}(t) = \{D \subseteq \{N, S, E, W\} \mid \sum_{d \in D} g(t(d)) \geq \tau\}$, i.e., the collection of subsets of sides of t whose glues have sufficient strength to bind cooperatively. Let $\sigma : \mathbb{Z}^2 \dashrightarrow T$ be a seed assembly, let $\tau_1, \tau_2 \in \mathbb{Z}^+$ be temperatures, and let $g_1, g_2 : \Lambda(T) \rightarrow \mathbb{N}$ be strength functions. We say that the TAS’s $\mathcal{T}_1 = (T, \sigma, g_1, \tau_1)$ and $\mathcal{T}_2 = (T, \sigma, g_2, \tau_2)$, differing only on their strength function and temperature, are *locally equivalent* if, for each tile type $t \in T$, $\mathcal{D}_{g_1, \tau_1}(t) = \mathcal{D}_{g_2, \tau_2}(t)$ ⁴. The behavior of an individual tile type during assembly is completely determined by its cooperation set, in the sense that if \mathcal{T}_1 and \mathcal{T}_2 are locally equivalent, then $\mathcal{A}[\mathcal{T}_1] = \mathcal{A}[\mathcal{T}_2]$ and $\mathcal{A}_\square[\mathcal{T}_1] = \mathcal{A}_\square[\mathcal{T}_2]$ ⁵.

Even without any strength function or temperature, by specifying a cooperation set for each tile type, one can describe a “behavior” of a TAS in the

⁴ Note that the definition of equivalence is independent of the seed assembly; we include it only to be able to talk about the equivalence of TAS’s rather than the more cumbersome “equivalence of triples of the form (T, g, τ) .”

⁵ The converse does not hold, however. For instance, some tile types may have a subset of sides whose glues never appear together at a binding site during assembly, so it would be irrelevant to the definition of $\mathcal{A}[\mathcal{T}]$ and $\mathcal{A}_\square[\mathcal{T}]$ whether or not that combination of glues have enough strength to bind.

sense that its dynamic evolution can be simulated knowing only the cooperation set of each tile type. We call a system thus specified a strength-free TAS. More formally, a *strength-free TAS* is a triple (T, σ, \mathcal{D}) , where T is a finite set of tile types, $\sigma : \mathbb{Z}^2 \dashrightarrow T$ is the size-1 seed assembly, and $\mathcal{D} : T \rightarrow \mathcal{P}(\mathcal{P}(\{\text{N, S, E, W}\}))$ is a function from a tile type $t \in T$ to a cooperation set $\mathcal{D}(t)$. For a standard TAS $\mathcal{T} = (T, \sigma, g, \tau)$ and a strength-free TAS $\mathcal{T}_{sf} = (T, \sigma, \mathcal{D})$ sharing the same tile set and seed assembly, we say that \mathcal{T} and \mathcal{T}_{sf} are *locally equivalent* if $\mathcal{D}_{g,\tau}(t) = \mathcal{D}(t)$ for each tile type $t \in T$.

Note that every TAS has a unique cooperation set for each tile type, and hence, has a locally equivalent strength-free TAS. Say that a strength-free TAS is *implementable* if there exists a TAS locally equivalent to it. Not every strength-free TAS is implementable. This is because cooperation sets could be contradictory; for instance, two tile types t_1 and t_2 could satisfy $t_1(\text{N}) = t_2(\text{N})$ and $\{\text{N}\} \in \mathcal{D}(t_1)$ but $\{\text{N}\} \notin \mathcal{D}(t_2)$.

Theorem 3.1. *There is a polynomial time algorithm that, given a strength-free TAS $\mathcal{T}_{sf} = (T, \sigma, \mathcal{D})$, determines whether there exists a locally equivalent TAS $\mathcal{T} = (T, \sigma, g, \tau)$ and outputs such a \mathcal{T} if it exists. Furthermore, it is guaranteed that $\tau \leq 2^{O(|T|)}$.*

Intuitively, we count the number of different total behaviors a tile can have, based on equating its behavior with its cooperation set. We then cast the problem of finding strengths to implement a given strength-free tile system as finding integer solutions to a certain system of linear inequalities. We relax the integer constraint and solve the system by Gaussian elimination. We then argue that the vertices of the polytope defined by this system have rational numbers with numerator and denominator that are at most exponential in the number of inequalities, which is itself linear in the number of tile types. This implies that by multiplying the rational numbers at one of these vertices by their least common multiple, which preserves the inequalities, we obtain an integer solution with values at most exponential in the number of tile types.

4 Finding the Minimum Tile Assembly System Assembling a Square at any Temperature

This section shows the main result of this paper, that there is a polynomial-time algorithm that, given an $n \times n$ square S_n , computes the smallest TAS that uniquely self-assembles S_n . Adleman, Cheng, Goel, Huang, Kempe, Moisset de Espanés, and Rothemund [2] showed that this problem is polynomial-time solvable when the temperature is restricted to be 2, and asked whether there is an algorithm that works when the temperature is unrestricted, which we answer affirmatively.

The next proposition is useful in the proof of Theorem 4.2.

Proposition 4.1. *For each $k \in \mathbb{Z}^+$, there are at most $168^k k^{4k+2}$ implementable strength-free TAS's with at most k tile types.*

The following theorem is the main result of this paper.

Theorem 4.2. *There is a polynomial-time algorithm that, given an $n \times n$ square S_n , outputs a minimal TAS T that uniquely self-assembles S_n .*

Intuitively, we conduct a search over all strength-free tile systems with at most $O(\frac{\log n}{\log \log n})$ tile types, using Proposition 4.1 to show that there are at most a polynomial in n number of such tile systems. We apply the algorithm of Theorem 3.1 to this strength-free tile system to obtain a tile system if it exists. We then use the algorithm of [2] to determine which of these tile systems is directed and strictly self-assembles the square, choosing the smallest such tile system to output.

We note that while we have stated the theorem for the family of square shapes, our method, as well as that of [2], works for any family of shapes S_1, S_2, \dots where $|S_n| = \text{poly}(n)$ and the tile complexity of S_n is at most $O(\frac{\log n}{\log \log n})$. This includes, for instance, the family $\{T_1, T_2, \dots\}$, where T_n is a width- n right triangle, and for each $q \in \mathbb{Q}^+$ the family $\{R_{q,1}, R_{q,2}, \dots\}$, where $R_{q,n}$ is the $n \times \lfloor qn \rfloor$ rectangle.

5 Bounds on Temperature Relative to Number of Tile Types

This section reports two bounds relating the number of tile types in a TAS to its temperature. The first bound, Theorem 5.1, shows that there are TAS's that require temperature *exponential* in the number of tile types (in the sense of local equivalence as defined in Section 3), if any combination of sides may be used for binding. This result can be interpreted to mean that the algorithm of [1] to find the minimum temperature-2 TAS for assembling an $n \times n$ square, which searches over all possible assignments of strengths to the glues, cannot be extended in a straightforward manner to handle larger temperatures, which is why it is necessary for the algorithm of Theorem 4.2 to “shortcut” through the behaviors of tile types rather than enumerating strengths. The second bound, Theorem 5.2, on the other hand, shows that if we restrict attention to those (quite prevalent) classes of tile systems that use only one or two sides of tiles to bind, then *linear* temperature always suffices.

5.1 Tile Assembly Systems Requiring Temperature Exponential in Number of Tile Types

In this section, we argue that a temperature that is exponential in the number of tile types given by Theorem 3.1 is optimal, although there is a gap between the exponents ($2^{|T|}/4$ for Theorem 5.1 below versus $O(2^{6|T|})$ for Theorem 3.1).

Theorem 5.1. *For every $n \in \mathbb{Z}^+$, there is a TAS $T = (T, \sigma, g, \tau)$ such that $|T| = 4n$ and for every TAS $T' = (T, \sigma, g', \tau')$ that is locally equivalent to T , $\tau' \geq 2^n$.*

The tile set T is shown in Figure 1. The proof uses induction on n to show that $A''_n \geq A_n + 2^n$.

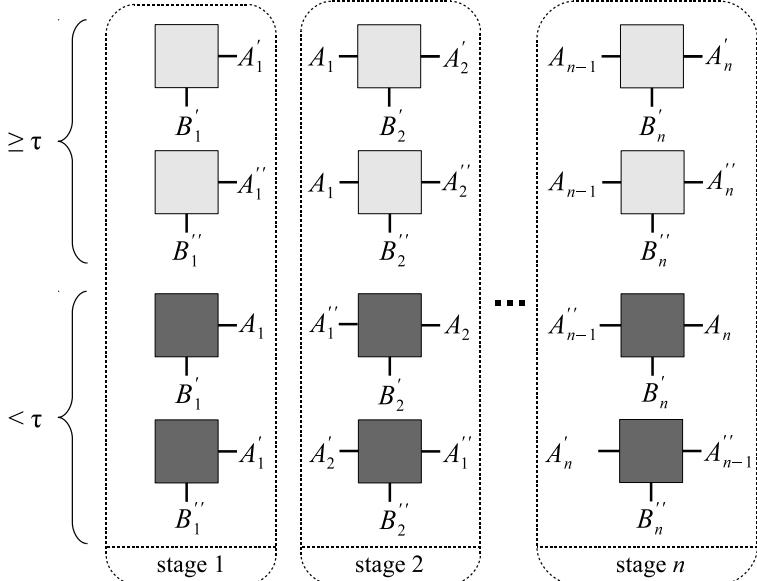


Fig. 1. A set of tile types requiring temperature that is exponentially larger than the number of tile types. There are stages $1, 2, \dots, n$, with stage i containing 4 tiles, and stage i ensuring that the gap between the largest and smallest strength in the stage is at least 2^i . In each stage, each of the top two light tiles represents a triple (a pair in stage 1) of glues whose sum is at least τ . Each of the bottom two dark tiles represents a triple of glues whose sum is less than τ . The inequalities are satisfiable, for instance, by setting $A_n = 3^{n-1}$, $A'_n = 2A_n$, $A''_n = 3A_n$, $B_n = \tau - A_n$, $B'_n = \tau - A'_n$, $B''_n = \tau - A''_n$.

5.2 Temperature Linear in the Number of Tile Types Suffices for 2-Cooperative Equivalence

Theorem 5.1 shows that temperature exponentially larger than the number of tile types is sometimes necessary for a TAS's behavior to be realized by integer strengths. However, the definition of local equivalence assumes that all possible combinations of sides of a tile type may be present in an assembly. Many TAS's are more constrained than this. There is a wide class of TAS's that we term *2-cooperative*, meaning that all binding events during all assembly sequences use only 1 or 2 sides that bind with positive strength. Nearly all theoretical TAS's found in the literature are 2-cooperative (indeed, temperature 2 systems by definition cannot *require* three sides to be present, although the model allows tile attachments with excess strength). In this section we show that the 3-cooperativity of Figure 1 is necessary, by showing that 2-cooperative systems can always be realized by strengths linear in the number of tile types.

Theorem 5.2. Let $\mathcal{T} = (T, \sigma, g, \tau)$ be a TAS, and let $\mathcal{D}_{g,\tau}^{(2)}(t) \subseteq \mathcal{D}_{g,t}(t)$ be the cooperation set of t with respect to g and τ restricted to containing only subsets of $\{\text{N}, \text{S}, \text{E}, \text{W}\}$ of cardinality 1 or 2. Then there is a TAS $\mathcal{T}' = (T, \sigma, g', \tau')$ with $\tau' \leq 2|T| + 2$ such that, for each $t \in T$, $\mathcal{D}_{g,\tau}^{(2)}(t) = \mathcal{D}_{g',\tau'}^{(2)}(t)$.

That is, \mathcal{T}' is equivalent in behavior to \mathcal{T} , so long as all attachments involve only 1 or 2 sides. Intuitively, we separate glue strengths into those at most $\tau/2$ (“low”) and those at least $\tau/2$ (“high”). The 2-cooperative property ensures that each low glue must cooperate with a high glue to bind. We then partition the high glues into equivalence classes based on those low glues with which they can cooperate, such that two high glues are equivalent iff they share the same minimal low glue with enough strength to cooperate. These k equivalence classes are then used to re-assign the glue strengths to the integers $\{1, \dots, k\}$. Since the number of equivalence classes is at most the number of low glues, which is itself bounded by the number of tile types, this gives the desired bound.

6 Open Questions

Our polynomial-time algorithm for finding the minimal tile system to self-assemble a square made crucial use of our polynomial-time algorithm that, given a strength-free tile system \mathcal{T}_{sf} , finds strengths and a temperature to implement a locally equivalent TAS \mathcal{T} , or reports that none exists. An open question is whether there is a polynomial-time algorithm that, given a strength-free TAS \mathcal{T}_{sf} , outputs a TAS of minimal temperature that is locally equivalent to \mathcal{T}_{sf} , or reports that none exists. In the proof of Theorem 3.1, we expressed the problem as an integer linear program, but needed only to find an integer feasible solution that is not necessarily optimal. If we instead wanted to find an *optimal* solution to the program, it is not clear whether the problem restricted to such instances is NP-hard.

The next question is less formal. Our results relating to 3-cooperative and 2-cooperative systems (Theorems 5.1 and 5.2, respectively), show that there is a difference in self-assembly “power” between these two classes of systems when we consider two tile systems to “behave the same” if and only if they are locally equivalent, which is a quite strict notion of behavioral equivalence. For example, two tile systems could uniquely self-assemble the same shape even though they have different tile types (hence could not possibly be locally equivalent). It would be interesting to know to what extent 3-cooperative systems – or high temperature tile systems in general – are strictly more powerful than temperature 2 systems under more relaxed notions of equivalence. For example, it is not difficult to design a shape that can be uniquely self-assembled with slightly fewer tile types at temperature 3 than at temperature 2. How far apart can this gap be pushed? Are there shapes with temperature-2 tile complexity that is “significantly” greater than their absolute (i.e., unrestricted temperature) tile complexity?

Acknowledgement. The authors thank Ehsan Chiniforooshan especially, for many fruitful and illuminating discussions that led to the results on temperature, and also Adam Marblestone and the members of Erik Winfree's group, particularly David Soloveichik, Joe Schaeffer, Damien Woods, and Erik Winfree, for insightful discussion and comments. The second author is grateful to Aaron Meyerowitz (via the website <http://mathoverflow.net>) for pointing out the Dedekind numbers as a way to count the number of collections of subsets of a given set that are closed under the superset operation.

References

1. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.-D.: Running time and program size for self-assembled squares. In: STOC 2001: Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing, Hersonissos, Greece, pp. 740–748. ACM (2001)
2. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.-D.A., Kempe, D., de Espanés, P.M., Rothemund, P.W.K.: Combinatorial optimization problems in self-assembly. In: STOC 2002: Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, pp. 23–32 (2002)
3. Barish, R.D., Schulman, R., Rothemund, P.W., Winfree, E.: An information-bearing seed for nucleating algorithmic self-assembly. Proceedings of the National Academy of Sciences 106(15), 6054–6059 (2009)
4. Evans, C.: Personal communication
5. Fujibayashi, K., Hariadi, R., Park, S.H., Winfree, E., Murata, S.: Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern. Nano Letters 8(7), 1791–1797 (2007)
6. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: STOC 2000: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 459–468 (2000)
7. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology 2(12), 2041–2053 (2004)
8. Seeman, N.C.: Nucleic-acid junctions and lattices. Journal of Theoretical Biology 99, 237–247 (1982)
9. Wang, H.: Proving theorems by pattern recognition – II. The Bell System Technical Journal XL(1), 1–41 (1961)
10. Wang, H.: Dominoes and the AEA case of the decision problem. In: Proceedings of the Symposium on Mathematical Theory of Automata, New York, pp. 23–55 (1962); Polytechnic Press of Polytechnic Inst. of Brooklyn, Brooklyn, N.Y. (1963)
11. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Institute of Technology (June 1998)

Optimization, Randomized Approximability, and Boolean Constraint Satisfaction Problems

Tomoyuki Yamakami

Department of Information Science, University of Fukui,
3-9-1 Bunkyo, Fukui 910-8507, Japan

Abstract. We give a unified treatment to optimization problems that can be expressed in the form of nonnegative-real-weighted Boolean constraint satisfaction problems. Creignou, Khanna, Sudan, Trevisan, and Williamson studied the complexity of approximating their optimal solutions whose optimality is measured by the sums of outcomes of constraints. To explore a wider range of optimization constraint satisfaction problems, following an early work of Marchetti-Spaccamela and Romano, we study the case where the optimality is measured by products of constraints' outcomes. We completely classify those problems into three categories: PO problems, NPO-hard problems, and intermediate problems that lie between the former two categories. To prove this trichotomy theorem, we analyze characteristics of nonnegative-real-weighted constraints using a variant of the notion of T-constructibility developed earlier for complex-weighted counting constraint satisfaction problems.

Keywords: optimization problem, approximation algorithm, constraint satisfaction problem, PO, APX, approximation-preserving reducibility.

1 Maximization by Multiplicative Measure

In the 1980s started extensive studies that have greatly improved our understandings of the exotic behaviors of various optimization problems within a scope of computational complexity theory. These studies have brought us deep insights into the approximability and inapproximability of optimization problems; however, many studies have targeted individual problems by cultivating different and independent methods for them. To push our insights deeper, we are focused on a collection of “unified” optimization problems, whose foundations are all formed in terms of *Boolean constraint satisfaction problems* (or *CSPs*, in short). Creignou is the first to have given a formal treatment to maximization problems derived from CSPs [3]. The *maximization constraint satisfaction problems* (or MAX-CSPs for succinctness) are, in general, optimization problems in which we seek a truth assignment σ of Boolean variables that maximizes an *objective value*¹ (or a *measure*) of σ , which equals the number of constraints being satisfied at once. Creignou presented three criteria (which are 0-validity, 1-validity, and

¹ A function that associates an objective value (or a measure) to each solution is called an *objective function* or (a *measure function*).

2-monotonicity) under which we can solve the MAX-CSPs in polynomial time; that is, the problems belong to PO.

Creignou's result was later reinforced by Khanna, Sudan, Trevisan, and Williamson [7], who gave a unified treatment to several types of CSP-based optimization problems, including MAX-CSP, MIN-CSP, and MAX-ONE-CSP. With constraints limited to “nonnegative” integer values, Khanna et al. defined MAX-CSP(\mathcal{F}) as the maximization problem in which constraints are all taken from constraint set \mathcal{F} and the maximization is measured by the “sum” of the objective values of constraints. For a later comparison, we call such a measure an *additive measure*. More formally, MAX-CSP(\mathcal{F}) is defined as:

MAX-CSP(\mathcal{F}):

- INSTANCE: a *finite* set H of elements of the form $\langle h, (x_{i_1}, \dots, x_{i_k}) \rangle$ on Boolean variables x_1, \dots, x_n , where $h \in \mathcal{F}$, $\{i_1, \dots, i_k\} \subseteq [n]$, and k is the arity of h .
- SOLUTION: a truth assignment σ to the variables x_1, \dots, x_n .
- MEASURE: the sum $\sum_{\langle h, x' \rangle \in H} h(\sigma(x_{i_1}), \dots, \sigma(x_{i_k}))$, where $x' = (x_{i_1}, \dots, x_{i_k})$.

For instance, MAX-CSP(*XOR*) coincides with the optimization problem MAX-CUT, which is known to be MAX-SNP-complete [9]. Khanna et al. reproved Creignou's classification theorem that, for every set \mathcal{F} of constraints, if \mathcal{F} is one of 0-valid, 1-valid, and 2-monotone, then MAX-CSP(\mathcal{F}) is in PO, and otherwise, MAX-CSP(\mathcal{F}) is APX-complete under their approximation-preserving reducibility. This classification theorem was proven by an extensive use of a notion of *strict/perfect implementation*.

From a different and meaningful perspective, Marchetti-Spaccamela and Romano [8] made a general discussion on max NPSP problems whose maximization is measured by the “products” of the objective values of chosen (feasible) solutions. From their general theorem follows an early result of [1] that the maximization problem, MAX-PROD-KNAPSACK, has a *fully polynomial(-time) approximation scheme* (or FPTAS). This result can be compared with another result of Ibarra and Kim [5] that MAX-KNAPSACK (with additive measures) admits an FPTAS. The general theorem of [8] requires development of a new proof technique, called *variable partitioning*. A similar approach was taken in a study of linear multiplicative programming to minimize the “product” of two positive linear cost functions, subject to linear constraints [6]. In contrast with the previous additive measures, we call this different type of measures *multiplicative measures*, and we wish to study the behaviors of MAX-CSPs whose maximization is taken over multiplicative measures.

Our approach clearly fills a part of what Creignou [3] and Khanna et al. [7] left unexplored. Let us formalize our objectives for clarity. To differentiate our multiplicative measures from additive measures, we develop a new notation MAX-PROD-CSP(\cdot), in which “PROD” refers to a use of “product” of objective values.

MAX-PROD-CSP(\mathcal{F}):

- INSTANCE: a *finite* set H of elements of the form $\langle h, (x_{i_1}, \dots, x_{i_k}) \rangle$ on Boolean variables x_1, \dots, x_n , where $h \in \mathcal{F}$ and $\{i_1, \dots, i_k\} \subseteq [n]$.
- SOLUTION: a truth assignment σ to x_1, \dots, x_n .

- MEASURE: the product $\prod_{\langle h, x' \rangle \in H} h(\sigma(x_{i_1}), \dots, \sigma(x_{i_k}))$, where $x' = (x_{i_1}, \dots, x_{i_k})$.

There is a natural, straightforward way to relate MAX-CSP(\mathcal{F})’s to MAX-PROD-CSP(\mathcal{F})’s. For example, consider the problem MAX-CUT and its multiplicatively-measured counterpart, MAX-PROD-CUT. Given any cut for MAX-CUT, we set the weight of each vertex to be 2 if it belongs to the cut, and set the weight to be 1 otherwise. When the cardinality of a cut is maximized, the same cut incurs the maximum product value as well. In other words, an algorithm that “exactly” finds an optimal solution for MAX-CUT also computes an optimal solution for MAX-PROD-CUT. However, when an algorithm only tries to “approximate” such an optimal solution, its performance rates for MAX-CUT and for MAX-PROD-CUT significantly differ. In a case of geometric programming, a certain type of product objective function is known to be “reduced” to additive ones if function values are all positive (see an survey [2]). In our setting, a different complication comes in when some values of h ’s accidentally fall into zero and thus the entire product value vanishes. Thus, an approximation algorithm using an additive measure does not seem to lead to an approximation algorithm with a multiplicative measure. This circumstance indicates that multiplicative measures appear to endow their associated optimization problems with much higher approximation complexity than additive measures do. We then need to develop a quite different methodology for a study on the approximation complexity of multiplicatively-measured optimization problems.

Within the framework of MAX-PROD-CSPs, we can precisely express many maximization problems, such as MAX-PROD-CUT, MAX-PROD-SAT, MAX-PROD-IS (independent set), and MAX-PROD-BIS (bipartite independent set), which are naturally induced from their corresponding additively-measured maximization problems. Some of their formal definitions will be given in Section 2.2. In a context of approximability, since the multiplicative measures can provide a richer structure than the additive measures, classification theorems for MAX-CSPs and for MAX-PROD-CSPs are essentially different. The classification theorem for MAX-PROD-CSPs—our main result—is formally stated in Theorem I, in which we use an abbreviation, MAX-PROD-CSP*(\mathcal{F}), to mean a MAX-PROD-CSP whose unary constraints are provided for free² and other nonnegative-real-weighted constraints are drawn from \mathcal{F} .

Theorem 1. *Let \mathcal{F} be any set of constraints. If either $\mathcal{F} \subseteq \mathcal{AF}$ or $\mathcal{F} \subseteq \mathcal{ED}$, then MAX-PROD-CSP*(\mathcal{F}) is in PO. Otherwise, if $\mathcal{F} \subseteq \mathcal{IM}_{opt}$, then MAX-PROD-CSP*(\mathcal{F}) is APT-reduced from MAX-PROD-BIS and is APT-reduced to MAX-PROD-FLOW. Otherwise, MAX-PROD-CSP*(\mathcal{F}) is APT-reduced from MAX-PROD-IS.*

Here, “APT” stands for “approximation preserving Turing” in the sense of [4]. Moreover, \mathcal{AF} is the set of “affine”-like constraints, \mathcal{ED} is related to the binary equality and disequality constraints, and \mathcal{IM}_{opt} is characterized by

² Allowing a free use of arbitrary unary constraints is a commonly used assumption for decision CSPs and counting CSPs.

“implication”-like constraints. The problem MAX-PROD-FLOW is a maximization problem of finding a design that maximizes the amount of water flow in a given direct graph. See Sections 2.1–2.2 for their formal definitions.

The purpose of the rest of this paper is to prove Theorem II. For this purpose, we will introduce a new notion of T_{\max} -constructibility, which is a variant of the notion of T-constructibility invented in [10]. This T_{\max} -constructibility is proven to be a powerful tool in dealing with MAX-PROD-CSPs.

2 Formal Definitions and Basic Properties

Let \mathbb{N} denote the set of all *natural numbers* (i.e., nonnegative integers) and let \mathbb{R} denote the set of all real numbers. For convenience, \mathbb{N}^+ expresses $\mathbb{N} - \{0\}$ and, for each $n \in \mathbb{N}^+$, $[n]$ denotes the integer set $\{1, 2, \dots, n\}$. Moreover, the notation $\mathbb{R}^{\geq 0}$ stands for the set $\{r \in \mathbb{R} \mid r \geq 0\}$.

2.1 Constraints and Relations

Here, we use terminology given in [10]. A (*nonnegative-real-weighted*) constraint is a function mapping from $\{0, 1\}^k$ to $\mathbb{R}^{\geq 0}$, where k is the *arity* of f . Assuming the standard lexicographic order on $\{0, 1\}^k$, we express f as a series of its output values. For instance, if $k = 2$, then f is $(f(00), f(01), f(10), f(11))$. We set $EQ = (1, 0, 0, 1)$, $\Delta_0 = (1, 0)$, and $\Delta_1 = (0, 1)$.

A *relation* of arity k is a subset of $\{0, 1\}^k$. Such a relation can be also viewed as a function mapping Boolean variables to $\{0, 1\}$ (i.e., $x \in R$ iff $R(x) = 1$, for every $x \in \{0, 1\}^k$) and it can be treated as a Boolean constraint. For instance, logical relations *OR*, *NAND*, *XOR*, and *Implies* are all expressed as appropriate constraints in the following manner: $OR = (0, 1, 1, 1)$, $NAND = (1, 1, 1, 0)$, $XOR = (0, 1, 1, 0)$, and $Implies = (1, 1, 0, 1)$. A relation R is *affine* if it is expressed as a set of solutions to a certain system of linear equations over $GF(2)$. An *underlying relation* R_f of f is defined as $R_f = \{x \mid f(x) \neq 0\}$.

We introduce the following six special sets of constraints.

1. Let \mathcal{U} denote the set of all unary constraints.
2. The notation \mathcal{NZ} expresses the set of all non-zero constraints.
3. Denote by \mathcal{DG} the set of all constraints that are expressed by products of unary constraints, each of which is applied to a different variable. Such a constraint is called *degenerate*.
4. Let \mathcal{ED} denote the set of all constraints that are expressed as products of some of unary constraints, the equality EQ , and the disequality XOR .
5. The set \mathcal{AF} is defined as the collection of all constraints of the form $g(x_1, \dots, x_k) \prod_{j:j \neq i} R_j(x_i, x_j)$ for a certain fixed index $i \in [k]$, where g is in \mathcal{DG} and each R_j is an affine relation.
6. Define \mathcal{IM}_{opt} to be the collection of all constraints that are products of some of the following constraints: unary constraints and constraints of the form $(1, 1, \lambda, 1)$ with $0 \leq \lambda < 1$. This is different from \mathcal{IM} defined in [10].

Lemma 1. *For any constraint $f = (x, y, z, w)$ with $x, y, z, w \in \mathbb{R}^{\geq 0}$, if $xw > yz$, then f belongs to \mathcal{IM}_{opt} .*

2.2 Optimization Problems with Multiplicative Measures

A (*combinatorial*) optimization problem $P = (I, \text{sol}, m)$ takes input instances of “admissible data” to the target problem. We often write I to denote the set of all such instances and $\text{sol}(x)$ denotes a set of (*feasible*) *solutions* associated with instance x . A *measure function* (or *objective function*) m associates a nonnegative real number to each solution y in $\text{sol}(x)$; that is, $m(x, y)$ is an *objective value* (or a *measure*) of the solution y on the instance x . We conveniently assume $m(x, y) = 0$ for any element $y \notin \text{sol}(x)$. The goal of the problem P is to find a solution y in $\text{sol}(x)$ that has an optimum value (such y is called an *optimal solution*), where the optimality is measured by either the maximization or minimization of an objective value $m(x, y)$, taken over all solutions $y \in \text{sol}(x)$. When y is an optimal solution, we set $m^*(x)$ to be $m(x, y)$.

Let NPO denote the class of all optimization problems P such that (1) input instances and solutions can be recognized in polynomial time; (2) solutions are polynomially-bounded in input size; and (3) a measure function can be computed in polynomial time. Define PO as the class of all problems P in NPO such that there exists a deterministic algorithm that, for every instance $x \in I$, returns an optimal solution y in $\text{sol}(x)$ in time polynomial in the size $|x|$ of the instance x .

We say that, for a fixed real-valued function α with $\alpha(n) \geq 1$ for any input size $n \in \mathbb{N}$, an algorithm \mathcal{A} for an optimization problem $P = (I, \text{sol}, m)$ is an α -*approximation algorithm* if, for every instance $x \in I$, \mathcal{A} produces a solution $y \in \text{sol}(x)$ satisfying that $1/\alpha(|x|) \leq |m(x, y)/m^*(x)| \leq \alpha(|x|)$, except that, whenever $m^*(x) = 0$, we always demand that $m(x, y) = 0$. Such a y is called an $\alpha(n)$ -*approximate solution* for input instance x of size n . The class APX (resp., exp-APX) consists of all problems P in NPO such that there are a constant $r \geq 1$ (resp., an exponentially-bounded³ function α) and a polynomial-time r -approximation (resp., α -approximation) algorithm for P . Approximation algorithms are often randomized. A *randomized approximation scheme* for P is a randomized algorithm that takes a standard input instance $x \in I$ together with an error tolerance parameter $\varepsilon \in (0, 1)$, and outputs a 2^ε -approximate solution $y \in \text{sol}(x)$ with probability at least $3/4$.

Of numerous existing notions of approximation-preserving reducibilities, we choose a notion introduced recently by Dyer, Goldberg, Greenhill, and Jerrum [4], which can be viewed as a randomized variant of Turing reducibility, based on a mechanism of *oracle Turing machine*. Since the purpose of Dyer et al. is to solve counting problems approximately, we need to modify their notion so that we can deal with optimization problems. Given two optimization problems $P = (I, \text{sol}, m)$ and $Q = (I', \text{sol}', m')$, a *polynomial-time (randomized) approximation-preserving Turing reduction* (or *APT-reduction*, in short) from P to Q is a randomized algorithm N that takes a pair $(x, \varepsilon) \in I \times (0, 1)$ as input, uses an arbitrary randomized approximation scheme (not necessarily polynomial time-bounded) M for Q as *oracle*, and satisfies the following conditions: (i) N is a randomized approximation scheme for P for any choice of oracle M for Q ;

³ This function α must satisfy that there exists a positive polynomial p for which $1 \leq \alpha(n) \leq 2^{p(n)}$ for any number $n \in \mathbb{N}$.

(ii) every *oracle call* made by N is of the form $(w, \delta) \in I' \times (0, 1)$ satisfying $1/\delta \leq p(|x|, 1/\varepsilon)$, where p is a certain absolute polynomial, and an oracle answer is an outcome of M on the input (w, δ) ; and (iii) the running time of N is bounded from above by a certain polynomial in $(|x|, 1/\varepsilon)$, not depending on the choice of the oracle M . In this case, we write $P \leq_{\text{APT}} Q$ and we also say that P is *APT-reducible* (or *APT-reduced*) to Q . Note that APT-reducibility composes. If $P \leq_{\text{APT}} Q$ and $Q \leq_{\text{APT}} P$, then P and Q are said to be *APT-equivalent* and we use the notation $P \equiv_{\text{APT}} Q$.

In the definition of MAX-PROD-CSP(\mathcal{F}) given in Section 11, we also write $h(x_{i_1}, \dots, x_{i_k})$ to mean $\langle h, (x_{i_1}, \dots, x_{i_k}) \rangle$ in H . For notational simplicity, we intend to write, for example, MAX-PROD-CSP($f, \mathcal{F}, \mathcal{G}$) instead of MAX-PROD-CSP($\{f\} \cup \mathcal{F} \cup \mathcal{G}$). In addition, we abbreviate as MAX-PROD-CSP*(\mathcal{F}) the maximization problem MAX-PROD-CSP($\mathcal{F} \cup \mathcal{U}$).

For any optimization problem P and any class \mathcal{C} of optimization problems, we write $P \leq_{\text{APT}} \mathcal{C}$ if there exists a problem $Q \in \mathcal{C}$ such that $P \leq_{\text{APT}} Q$. Our choice of APT-reducibility makes it possible to prove Lemma 2; however, it is not clear whether the lemma implies that MAX-PROD-CSP(\mathcal{F}) $\in \text{exp-APX}$.

Lemma 2. MAX-PROD-CSP(\mathcal{F}) $\leq_{\text{APT}} \text{exp-APX}$ for any constraint set \mathcal{F} .

Hereafter, we introduce the maximization problems stated in Theorem 11.

MAX-PROD-IS: (maximum product independent set)

- INSTANCE: an undirected graph $G = (V, E)$ and a series $\{w_x\}_{x \in V}$ of vertex weights with $w_x \in \mathbb{R}^{\geq 0}$;
- SOLUTION: an independent set A on G ;
- MEASURE: the product $\prod_{x \in A} w_x$.

This maximization problem MAX-PROD-IS literally coincides with MAX-PROD-CSP($NAND, \mathcal{G}_0$), where $\mathcal{G}_0 = \{[1, \lambda] \mid \lambda \geq 0\}$, and it can be easily shown to be NPO-complete. When all input graphs are limited to bipartite graphs, the corresponding problem is called MAX-PROD-BIS.

MAX-PROD-BIS: (maximum product bipartite independent set)

- In MAX-PROD-IS, all input graphs are limited to bipartite graphs.

Since the above two problems can be expressed in the form of MAX-PROD-CSP*(\cdot), we can draw the following important conclusion, which becomes part of the proof of the main theorem.

Lemma 3. 1. MAX-PROD-IS $\leq_{\text{APT}} \text{MAX-PROD-CSP}^*(OR)$.
 2. MAX-PROD-BIS $\leq_{\text{APT}} \text{MAX-PROD-CSP}^*(Implies)$.

Next, we introduce a special maximization problem, called MAX-PROD-FLOW, whose intuitive setting is explained as follows. Suppose that water flows from point u to point v through a one-way pipe at flow rate $\rho_{(u,v)}$. A value $\sigma(x)$ expresses an elevation (indicating either the *bottom level* or the *top level*) of point x so that water runs from point u to point v whenever $\sigma(u) \geq \sigma(v)$. More water is added at influx rate w_v at point v for which $\sigma(v) = 1$.

MAX-PROD-FLOW: (maximum product flow)

- INSTANCE: a directed graph $G = (V, E)$, a series $\{\rho_e\}_{e \in E}$ of flow rates with $\rho_e \geq 1$, and a series $\{w_x\}_{x \in V}$ of influx rates with $w_x \geq 0$;
- SOLUTION: a Boolean assignment σ to V ;
- MEASURE: the product $\left(\prod_{(x,y) \in E, \sigma(x) \geq \sigma(y)} \rho_{(x,y)}\right) \left(\prod_{z \in V, \sigma(z)=1} w_z\right)$.

From the above definition, it is not difficult to prove the following statement.

Lemma 4. *For any constraint set $\mathcal{F} \subseteq \mathcal{IM}_{opt}$, MAX-PROD-CSP $^*(\mathcal{F})$ is APT-reducible to MAX-PROD-FLOW.*

2.3 T_{\max} -Constructibility

To pursue notational succinctness, we use the following notations. Let f be any arity- k constraint. For any two distinct indices $i, j \in [k]$ and any bit $c \in \{0, 1\}$, let $f^{x_i=c}$ denote the function g satisfying that $g(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k) = f(x_1, \dots, x_{i-1}, c, x_{i+1}, \dots, x_k)$ and let $f^{x_i=x_j}$ be the function g defined as $g(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_k) = f(x_1, \dots, x_{i-1}, x_j, x_{i+1}, \dots, x_k)$. Moreover, we denote by $\max_{y_1, \dots, y_d}(f)$ the function g defined as $g(x_1, \dots, x_k) = \max_{(y_1, \dots, y_d) \in \{0, 1\}^d} \{f(x_1, \dots, x_k, y_1, \dots, y_d)\}$, where y_1, \dots, y_d are all distinct and different from x_1, \dots, x_k , and let $\lambda \cdot f$ denote the function satisfying $(\lambda \cdot f)(x_1, \dots, x_k) = \lambda f(x_1, \dots, x_k)$.

A helpful tool invented in [10] for counting CSPs is a notion of *T-constructibility*. For our purpose of proving the main theorem, we wish to modify this notion and introduce a notion of T_{\max} -constructibility. We say that an arity- k constraint f is *T_{\max} -constructible* (or *T_{\max} -constructed*) from a constraint set \mathcal{G} if f can be obtained, initially from constraints in \mathcal{G} , by applying recursively a finite number (possibly zero) of seven functional operations described below.

1. PERMUTATION: for two indices $i, j \in [k]$ with $i < j$, by exchanging two columns x_i and x_j in $(x_1, \dots, x_i, \dots, x_j, \dots, x_k)$, transform g into g' , where g' is defined as $g'(x_1, \dots, x_i, \dots, x_j, \dots, x_k) = g(x_1, \dots, x_j, \dots, x_i, \dots, x_k)$.
2. PINNING: for an index $i \in [k]$ and a bit $c \in \{0, 1\}$, build $g^{x_i=c}$ from g .
3. LINKING: for two distinct indices $i, j \in [k]$, build $g^{x_i=x_j}$ from g .
4. EXPANSION: for an index $i \in [k]$, introduce a new “free” variable, say, y and transform g into g' that is defined by $g'(x_1, \dots, x_i, y, x_{i+1}, \dots, x_k) = g(x_1, \dots, x_i, x_{i+1}, \dots, x_k)$.
5. MULTIPLICATION: from two constraints g_1 and g_2 of arity k that share the same input variable series (x_1, \dots, x_k) , build the new constraint $g_1 \cdot g_2$, where $(g_1 \cdot g_2)(x_1, \dots, x_k) = g_1(x_1, \dots, x_k)g_2(x_1, \dots, x_k)$.
6. MAXIMIZATION: build $\max_{y_1, \dots, y_d}(g)$ from g , where y_1, \dots, y_d are not shared with any other constraint other than this particular constraint g .
7. NORMALIZATION: for a positive constant λ , build $\lambda \cdot g$ from g .

When f is T_{\max} -constructible from \mathcal{G} , we use the notation $f \leq_{con}^{\max} \mathcal{G}$. In particular, when \mathcal{G} is a singleton $\{g\}$, we also write $f \leq_{con}^{\max} g$ instead of $f \leq_{con}^{\max} \{g\}$.

It holds that T_{\max} -constructibility between constraints guarantees APT-reducibility between their corresponding MAX-PROD-CSP $^*(\cdot)$'s.

Lemma 5. *If $f \leq_{con}^{\max} \mathcal{G}$, then MAX-PROD-CSP*(f, \mathcal{F}) is APT-reducible to MAX-PROD-CSP*(\mathcal{G}, \mathcal{F}) for any constraint set \mathcal{F} .*

3 Proof of the Main Theorem

Our main theorem—Theorem 1—states that all maximization problems of the form of MAX-PROD-CSP*(\cdot) can be classified into three categories. This trichotomy theorem sheds a clear contrast with the dichotomy theorem of Khanna et al. [7] for MAX-CSPs. Hereafter, we will present the proof of Theorem 1.

3.1 First Step Toward the Proof

We begin with MAX-PROD-CSP*(\cdot)'s that can be solved in polynomial time.

Proposition 1. *If either $\mathcal{F} \subseteq \mathcal{AF}$ or $\mathcal{F} \subseteq \mathcal{ED}$, then MAX-PROD-CSP*(\mathcal{F}) belongs to PO.*

Proof Sketch. For every target problem, as in the proof of [10, Lemma 6.1], we can greatly simplify the structure of each input instance so that it depends only on polynomially many solutions. By examining all such solutions deterministically, we surely find its optimal solution. Hence, the target problem belongs to PO. \square

It thus remains to deal with only the case where $\mathcal{F} \not\subseteq \mathcal{AF}$ and $\mathcal{F} \not\subseteq \mathcal{ED}$. In this case, we first make the following key claim that leads to the main theorem.

Proposition 2. *Let f be any constraint and assume that $f \notin \mathcal{AF} \cup \mathcal{ED}$. Let \mathcal{F} be any set of constraints.*

1. *If $f \in \mathcal{IM}_{opt}$, then MAX-PROD-CSP*(*Implies*, \mathcal{F}) is APT-reducible to MAX-PROD-CSP*(f, \mathcal{F}).*
2. *If $f \notin \mathcal{IM}_{opt}$, then there exists a constraint $g \in \{\text{OR}, \text{NAND}\}$ such that MAX-PROD-CSP*(g, \mathcal{F}) is APT-reducible to MAX-PROD-CSP*(f, \mathcal{F}).*

We postpone the proof of the above proposition and, meanwhile, we want to prove Theorem 1 using the proposition.

Proof of Theorem 1 If $\mathcal{F} \subseteq \mathcal{AF}$ or $\mathcal{F} \subseteq \mathcal{ED}$, then Proposition 1 implies that MAX-PROD-CSP*(\mathcal{F}) belongs to PO. Henceforth, we assume that $\mathcal{F} \not\subseteq \mathcal{AF}$ and $\mathcal{F} \not\subseteq \mathcal{ED}$. If $\mathcal{F} \subseteq \mathcal{IM}_{opt}$, then Lemma 4 helps APT-reduce MAX-PROD-CSP*(\mathcal{F}) to MAX-PROD-FLOW. Next, we choose a constraint $f \in \mathcal{F}$ for which $f \notin \mathcal{AF} \cup \mathcal{ED}$. Proposition 2(1) then yields an APT-reduction from MAX-PROD-CSP*(*Implies*) to MAX-PROD-CSP*(f). By Lemma 3(2), we obtain $\text{MAX-PROD-BIS} \leq_{\text{APT}} \text{MAX-PROD-CSP*(Implies)}$. Since $\text{MAX-PROD-CSP}(f) \leq_{\text{APT}} \text{MAX-PROD-CSP}(\mathcal{F})$, it follows that MAX-PROD-BIS is APT-reducible to MAX-PROD-CSP*(\mathcal{F}).

Finally, we assume that $\mathcal{F} \not\subseteq \mathcal{IM}_{opt}$. Take a constraint $f \in \mathcal{F}$ satisfying that $f \notin \mathcal{AF} \cup \mathcal{ED} \cup \mathcal{IM}_{opt}$. In this case, Proposition 2(2) yields

an APT-reduction from MAX-PROD-CSP*(*OR*) to MAX-PROD-CSP*(*f*), since MAX-PROD-CSP*(*OR*) \equiv_{APT} MAX-PROD-CSP*(*NAND*). From MAX-PROD-CSP*(*f*) \leq_{APT} MAX-PROD-CSP*(\mathcal{F}), it immediately follows that MAX-PROD-CSP*(*OR*) is APT-reducible to MAX-PROD-CSP*(\mathcal{F}). By Lemma 3(1), MAX-PROD-IS \leq_{APT} MAX-PROD-CSP*(*OR*). Therefore, we conclude that MAX-PROD-IS is APT-reducible to MAX-PROD-CSP*(\mathcal{F}). \square

3.2 Second Step Toward the Proof

To finish the proof of Theorem 1, we still need to prove Proposition 2. Proving this proposition requires three properties. To describe them, we first review two existing notions from [10]. We say that a constraint *f* has *affine support* if R_f is an affine relation and that *f* has *imp support* if R_f is logically equivalent to a conjunction of a certain “positive” number of relations of the form $\Delta_0(x)$, $\Delta_1(x)$, and *Implies*(*x*, *y*). The notation *AFFINE* denotes the set of all affine relations.

In the following three statements, \mathcal{F} denotes an arbitrary set of constraints.

Lemma 6. *If *f* is a non-degenerate constraint in \mathcal{IM}_{opt} and has no imp support, then $\text{MAX-PROD-CSP}^*(\text{Implies}, \mathcal{F}) \leq_{APT} \text{MAX-PROD-CSP}^*(*f*, \mathcal{F})$.*

Proposition 3. *Let *f* be any constraint having imp support. If either *f* has no affine support or *f* $\notin \mathcal{ED}$, then $\text{MAX-PROD-CSP}^*(\text{Implies}, \mathcal{F})$ is APT-reducible to $\text{MAX-PROD-CSP}^*(*f*, \mathcal{F})$.*

Proposition 4. *Let *f* $\notin \mathcal{NZ}$ be any constraint. If *f* has neither affine support nor imp support, then there exists a constraint *g* $\in \{\text{OR}, \text{NAND}\}$ such that $\text{MAX-PROD-CSP}^*(*g*, \mathcal{F}) \leq_{APT} \text{MAX-PROD-CSP}^*(*f*, \mathcal{F})$.*

With a help of the above statements, we can prove Proposition 2 as follows.

Proof Sketch of Proposition 2. Let *f* $\notin \mathcal{AF} \cup \mathcal{ED}$ be any constraint. We proceed our proof by induction on the arity *k* of *f*. For the proposition’s claims, (1) and (2), the basis case *k* = 1 is trivial since \mathcal{ED} contains all unary constraints. Next, we prove the induction step *k* ≥ 3 . In the remainder of this proof, as our induction hypothesis, we assume that the proposition holds for any arity less than *k*. The claims (1) and (2) will be shown separately.

(1) Assume that *f* is in \mathcal{IM}_{opt} . If *f* has imp support, since *f* $\notin \mathcal{ED}$, we can apply Proposition 3 and immediately obtain the desired APT-reduction $\text{MAX-PROD-CSP}^*(\text{Implies}, \mathcal{F}) \leq_{APT} \text{MAX-PROD-CSP}^*(*f*, \mathcal{F})$. Otherwise, by Lemma 6, we have the desired APT-reduction.

(2) Since *f* has no imp support, if R_f is not affine, then Proposition 4 implies that, for a certain $g_0 \in \{\text{OR}, \text{NAND}\}$, $\text{MAX-PROD-CSP}^*(g_0, \mathcal{F}) \leq_{APT} \text{MAX-PROD-CSP}^*(*f*, \mathcal{F})$; therefore, the desired result follows. To finish the proof, we hereafter assume the affine property of R_f .

[Case: $f \in \mathcal{NZ}$] Recall that *f* $\notin \mathcal{ED}$ and $R_f \in \text{AFFINE}$. Since *f* $\in \mathcal{NZ}$, we have $|R_f| = 2^k$, and thus *f* should be in *clean form* (i.e., *f* contains no factor of the

form: $\Delta_0(x)$, $\Delta_1(x)$, and $EQ(x, y)$). As shown in [10, Lemma 7.5], there exists a constraint $p = (1, x, y, z) \notin \mathcal{ED}$ with $xyz \neq 0$, $z \neq xy$, and $p \leq_{con}^{\max} f$. When $z < xy$, we can prove that, for a certain $g_0 \in \{OR, NAND\}$, $\text{MAX-PROD-CSP}^*(g_0, \mathcal{F}) \leq_{\text{APT}} \text{MAX-PROD-CSP}^*(p, \mathcal{F})$. In the case where $z > xy$, Lemma 10 implies $p \in \mathcal{IM}_{opt}$. Since p is obtained from f by pinning operations only, we conclude that $f \in \mathcal{IM}_{opt}$, a contradiction. This finishes the induction step.

[Case: $f \notin \mathcal{NZ}$] We first claim that $k \geq 3$. Assume otherwise that $k = 2$. Since $R_f \in \text{AFFINE}$, it is possible to write f in the form $f(x_1, x_2) = \xi_A(x_1, x_2)g(x_1)$ after appropriately permuting variable indices. This places f within \mathcal{AF} , a contradiction against the choice of f . Hence, $k \geq 3$ holds. Moreover, we can prove the existence of a constraint $g \notin \mathcal{AF}$ of arity m for which $2 \leq m < k$, $g \leq_{con}^{\max} f$, and either $g \in \mathcal{NZ}$ or $R_g \notin \text{AFFINE}$. If we can show that (*) there exists a constraint $g_0 \in \{OR, NAND\}$ satisfying $\text{MAX-PROD-CSP}^*(g_0, \mathcal{F}) \leq_{\text{APT}} \text{MAX-PROD-CSP}^*(g, \mathcal{F})$, then the proposition immediately follows from $g \leq_{con}^{\max} f$. The claim (*) is split into two cases: (i) $R_g \in \text{AFFINE}$ and (ii) $R_g \notin \text{AFFINE}$. For (i), we apply the induction hypothesis. For (ii), we apply Propositions 3–4. Thus, we have completed the induction step. \square

In this end, we have completed the proof of the main theorem. The detailed proofs omitted in this extended abstract will be published shortly. We hope that our systematic treatment of MAX-PROD-CSP*'s would lead to a study on a far wider class of optimization problems.

References

1. Ausiello, G., Marchetti-Spaccamela, A., Protasi, M.: Full Approximability of a Class of Problems Over Power Sets. In: Astesiano, E., Böhm, C. (eds.) CAAP 1981. LNCS, vol. 112, pp. 76–87. Springer, Heidelberg (1981)
2. Boyd, S., Kim, S.J., Vandenberghe, L., Hassibi, A.: A tutorial on geometric programming. *Optm. Eng.* 8, 67–127 (2007)
3. Creignou, N.: A dichotomy theorem for maximum generalized satisfiability problems. *J. Comput. System Sci.* 51, 511–522 (1995)
4. Dyer, M., Goldberg, L.A., Greenhill, C., Jerrum, M.: The relative complexity of approximating counting problems. *Algorithmica* 38, 471–500 (2003)
5. Ibarra, O.H., Kim, C.E.: Fast approximation for the knapsack and sum of subset problems. *J. ACM* 22, 463–468 (1975)
6. Konno, H., Kuno, T.: Linear multiplicative programming. *Math. Program.* 56, 51–64 (1992)
7. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. *SIAM J. Comput.* 30, 1863–1920 (2001)
8. Marchetti-Spaccamela, A., Romano, S.: On different approximation criteria for subset product problems. *Inform. Process. Lett.* 21, 213–218 (1985)
9. Papadimitriou, C., Yannakakis, M.: Optimization, approximation and complexity classes. *J. Comput. System Sci.* 43, 425–440 (1991)
10. Yamakami, T.: Approximate Counting for Complex-Weighted Boolean Constraint Satisfaction Problems. In: Jansen, K., Solis-Oba, R. (eds.) WAOA 2010. LNCS, vol. 6534, pp. 261–272. Springer, Heidelberg (2011)

Lower Bounds for Myopic DPLL Algorithms with a Cut Heuristic*

Dmitry Itsykson and Dmitry Sokolov

Steklov Institute of Mathematics at St. Petersburg,
27 Fontanka, 191023, St. Petersburg, Russia

dmitrits@pdmi.ras.ru,
sokolov.dmt@gmail.com

Abstract. The paper is devoted to lower bounds on the time complexity of DPLL algorithms that solve the satisfiability problem using a splitting strategy. Exponential lower bounds on the running time of DPLL algorithms on unsatisfiable formulas follow from the lower bounds for resolution proofs. Lower bounds on satisfiable instances are also known for some classes of DPLL algorithms; these lower bounds are usually based on reductions to unsatisfiable instances. In this paper we consider DPLL algorithms with a cut heuristic that may decide that some branch of the splitting tree will not be investigated. DPLL algorithms with a cut heuristic always return correct answer on unsatisfiable formulas while they may err on satisfiable instances. We prove the theorem about effectiveness vs. correctness trade-off for deterministic myopic DPLL algorithms with cut heuristic. Myopic algorithms can see formulas with erased signs of negations; they may also request a small number of clauses to read them precisely.

We construct a family of unsatisfiable formulas $\Phi^{(n)}$ and a polynomial time samplable ensemble of distributions Q_n concentrated on satisfiable formulas such that every deterministic myopic algorithm that gives a correct answer with probability $1 - o(1)$ on a random formula from the ensemble Q_n runs exponential time on the formulas $\Phi^{(n)}$.

1 Introduction

DPLL (are named by the authors: Davis, Putnam, Logemann and Loveland) algorithms are one of the most popular approach to the problem of satisfiability of Boolean formulas (SAT). DPLL algorithm is a recursive algorithm that takes the input formula ϕ , uses a procedure **A** to choose a variable x , uses a procedure **B** that chooses the value $a \in \{0, 1\}$ for the variable x that would be investigated first, and makes two recursive calls on inputs $\phi[x := a]$ and $\phi[x := 1 - a]$. Note that the second call is not necessary if the first one returns the result, that the formula is satisfiable.

* The work is partially supported by RAS Program for Fundamental Research, the president grants NSh-5282.2010.1 and MK-4089.2010.1, by RFBR grants 11-01-00760-a and 11-01-12135-ofi-m-2011, by Rokhlin's scholarship and by PDMI Computer Science Club scholarship.

There is a number of works concerning lower bounds for DPLL algorithms: for unsatisfiable formulas exponential lower bounds follow from lower bounds on the complexity of resolution proofs [9], [8]. In case of satisfiable formulas we have no hope to prove superpolynomial lower bound since if $P = NP$, then procedure **B** may always choose the correct value of the variable according to some satisfying assignment. The paper [1] gives exponential lower bounds for two wide enough classes of DPLL algorithms: myopic and drunken algorithms. In the myopic case procedures **A** and **B** can see formula with erased signs of negation, they can request the number of positive and negative occurrences for every variable and also may read $K = n^{1-\epsilon}$ clauses precisely. In the drunken algorithms the procedure **A** may be arbitrary while the procedure **B** chooses the value uniformly at random. The paper [2] shows that myopic algorithms invert Goldreich's function ([3]) based on a random graph in at least exponential time, [5] and [7] extend this result for drunken algorithms. The paper [6] presents the explicit Goldreich's function based on any expander that is hard to invert for drunken and myopic algorithms.

All discussed lower bounds for satisfiable instances are based on the fact that during several first steps algorithm falls into a hard unsatisfiable formula, and algorithm should investigate the whole it' splitting tree. In this work we extend the class of DPLL algorithms by adding the procedure **C** that may decide that some branch of the splitting tree will not be investigated since it is not too "perspective". More precisely, before each recursive call an algorithm calls the procedure **C** that decides whether to make this recursive call or not. DPLL algorithms with cut heuristic are always give a correct answer on unsatisfiable formulas; however they may err on satisfiable formulas. On the other hand if the presence of a cut heuristic gives the substantial improvement on the time complexity while the bad instances (i.e. instances on which the algorithm errs) are not easy to find, then such algorithms become reasonable.

In this work we show that it is possible to construct the family of unsatisfiable formulas $\Phi^{(n)}$ in polynomial time such that for every myopic deterministic heuristics **A** and **C** there exists a polynomial time samplable ensemble of distributions R_n such that the DPLL algorithm based on procedures **A**, **B** and **C** for some **B** either errs on 99% of random inputs according R_n or runs exponential time on formulas $\Phi^{(n)}$. In case **A** and **C** are not restricted we show that a statement similar to above is equivalent to $P \neq NP$. The case of randomized myopic procedures **A** and **C** is left open.

In the proof of the main result we use the technique of closures for expander graphs invented by Alekhnovich. We suggest a constructive variant of the closure notion and apply it to the construction of the ensemble of distributions R_n .

The constructed distribution R_n depends on procedures **A** and **C**. We also construct a universal ensemble of distributions Q_n that dominates all possible ensembles R_n . In particularly we prove that if myopic DPLL with a cut heuristic errs with $o(1)$ probability on a random formula from the ensemble Q_n , then it runs exponential time on formulas $\Phi^{(n)}$.

Heuristic acceptors. The study of DPLL algorithms with cut heuristic was also motivated by the study of heuristic acceptors [4]. The distributional proving problem is a pair (L, D) of a language L and a polynomial time samplable distribution D concentrated on the complement of L . An algorithm A is called a heuristic acceptor if it has additional input d that represents the parameter of the error and for every $x \in L$ and $d \in \mathbb{N}$, $A(x, d)$ returns 1 and $\Pr_{x \leftarrow D_n}[A(x) = 1] < 1/d$ for every integer n . We call an acceptor polynomially bounded if for every $x \in L$ running time of $A(x, d)$ is bounded by polynomial in $|x| \cdot d$. The paper [4] shows that the existence of distributed proving problems that have no polynomially bounded acceptors is equivalent to the existence of infinitely often one-way functions.

Let D be some distribution concentrated on satisfiable formulas. We consider DPLL algorithm with a cut heuristic supplied with an additional parameter d that is available for procedures **A**, **B**, **C**. We call such an algorithm a heuristic DPLL acceptor if it satisfies the definition of a heuristic acceptor. Our result implies that there are no deterministic polynomially bounded myopic DPLL acceptors for the proving problem (UNSAT, Q) .

2 Preliminaries

By partial substitution we mean the set of assignments $x_i := a_i$, where x_i is a propositional variable and $a_i \in \{0, 1\}$ such that every propositional variable has at most one occurrence in it. For partial substitution ρ we denote $\text{vars}(\rho)$ the set of variable that appears in ρ . For propositional formula ϕ we denote by $\phi|_\rho$ the formula that is obtained from ϕ after applying all assignments from ρ and elementary simplifications.

An ensemble of distributions is a family D_n , where D_n is a distribution on the finite set of strings. The ensemble of distributions D_n is polynomial time samplable if there is a polynomial time randomized algorithm S (a sampler) such that for every n outputs of $S(1^n)$ are distributed according to D_n .

3 DPLL Algorithms with a Cut Heuristic

We define a family of algorithms that solve the satisfiability problem of CNF formulas. DPLL algorithms with a cut heuristic are parameterized by three procedures: **A**, **B** and **C**. The procedure **A** takes a formula as the input and returns a variable for splitting; the procedure **B** takes a formula and a variable as the input and returns a value that would be investigated first; the procedure **C** takes a formula, variable and its value and decides whether an algorithm should investigate this branch.

Formally algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ is defined by the following way.

Algorithm 1 *Algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$.*

- *Input: formula φ*
- *If φ contains no clauses, return 1.*
- *If φ contains empty clause, return 0.*
- $x := \mathbf{A}(\varphi);$
- $b := \mathbf{B}(\varphi, x);$
- *If $\mathbf{C}(\varphi, x, b) = 1$, then if $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\varphi[x := b]) = 1$, return 1.*
- *If $\mathbf{C}(\varphi, x, \neg b) = 1$, then if $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\varphi[x := \neg b]) = 1$, return 1.*
- *Return 0.*

Let **1** denote the constant 1 function. Algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{1}}$ obviously correctly solves the satisfiability problem. For every procedures **A**, **B**, **C** the algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ returns the correct answer on all unsatisfiable formulas, however its answer on satisfiable formulas may be incorrect.

For every formula φ execution of an algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ corresponds to a splitting tree. Vertices of a splitting tree correspond to recursive calls, edges correspond to substitutions of a value to a variable. Every leaf of a splitting tree corresponds to one of the three possible situations: 1) some clause of initial formula is refuted; 2) the substitution is a satisfying assignment; 3) the procedure **C** reports 0 two times. It is not hard to see that if procedures **A** and **B** are deterministic, then the splitting tree of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ is a subtree of the splitting tree of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{1}}$; if the input formula φ is unsatisfiable, then the above statement holds even if only **A** is deterministic.

By running time of the algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ we mean the number of recursive calls that equals to the number of vertices in the splitting tree.

Our main goal is to prove lower bounds for almost correct DPLL algorithms with a cut heuristic. Namely we construct the polynomial time samplable ensemble of distributions R_n concentrated on satisfiable formulas and the sequence of unsatisfiable formulas $\Phi^{(n)}$ such that if $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ with high probability correctly solves the satisfiability problem for instances distributed according to R_n , then the algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ will run exponential time on formulas $\Phi^{(n)}$. Without restrictions on procedures **A**, **B**, **C** it is unlikely that we prove the above statement since if $P = NP$, the polynomial time procedure **C** may cut all unsatisfiable branches. Therefore we need to restrict the power of heuristic **C** and we require it to be myopic.

A myopic procedure has access to the formula with erased signs of negations. For every variable it also has access to the number of its positive and negative occurrences. A myopic procedure is also able to read $K = n^{1-\varepsilon}$ clauses of the formula precisely (with all negations).

However it is not enough to restrict only the procedure **C** since the procedure **A** may transmit information to the procedure **C**; for example **A** may return lexicographically first variable for satisfiable formulas and lexicographically last value for unsatisfiable formulas and the procedure **C** will just cut unsatisfiable branches. Therefore we restrict **A** to be myopic. We also require for **A** and **C** to be deterministic. The case of randomized myopic **A** and **C** is left open.

4 Distribution

Here and after we assume that procedures **A** and **C** are deterministic and polynomial time myopic algorithms.

Let Φ be an unsatisfiable formula and T_Φ be the splitting tree of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\Phi)$. We say that the set of vertices $\{v_1, v_2, \dots, v_S\}$ of the tree T_Φ is *the system of myopic copies of Φ* , if the following is satisfied:

- for every $i, j \in \{1, \dots, S\}$ a vertex v_i is not a descendant of v_j ;
- For every vertex v_i there exists a satisfiable formula Φ_i with the only satisfying assignment such that the substitution made on the path from the root of T_Φ to v_i is consistent with the satisfying assignment of Φ_i .
- Myopic algorithms **A** and **C** can't distinguish formulas Φ_i and Φ on the path from the root of T_Φ to v_i . Formally it means that for every substitution ρ that corresponds to this path formulas $\Phi_i|_\rho$ and $\Phi|_\rho$ with erased negation signs have no differences; every variable has the same number of positive and negative occurrences and sets of clauses that **A** and **C** read with negations are the same for formulas $\Phi_i|_\rho$ and $\Phi|_\rho$.

We call formulas $\{\Phi_1, \Phi_2, \dots, \Phi_S\}$ *myopic representatives* of Φ .

Lemma 1. *Let Φ be an unsatisfiable formula and $\{v_1, v_2, \dots, v_S\}$ is the system of myopic copies of Φ with myopic representatives $\{\Phi_1, \Phi_2, \dots, \Phi_S\}$. Let U_Φ be a uniform distribution on the set of formulas $\{\Phi_1, \Phi_2, \dots, \Phi_S\}$. Suppose that $\Pr_{\phi \leftarrow U_\Phi}[\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\phi) = 1] \geq 1 - \epsilon$, then the running time of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\Phi)$ is at least $(1 - \epsilon)S$. (The probability also includes random bits of the algorithm **B**).*

Proof. Note that the usage of random bits by **B** does not affect the running time of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ on unsatisfiable formulas. We fix the sequence of random bits in such a way that if **B** uses this sequence then $\Pr_{\phi \leftarrow U_\Phi}[\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\phi) = 1] \geq 1 - \epsilon$. Since now we assume that **B** uses only this sequence of random bits.

Lets consider one of the formulas Φ_j such that $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\Phi_j) = 1$. We note that the splitting tree of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ on the input Φ_j contains the path from the root to v_j from the tree T_Φ . Since the only satisfying assignment of Φ_j is consistent with the substitution made by this path, if $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ diverges from this path it will necessary come back since Φ_j has the unique satisfying assignment. During this path the procedure **A** will select the same variables for splitting since it can't differ Φ from Φ_j . The procedure **C** also can't differ Φ from Φ_j during this path, therefore the vertex v_j is necessary in the splitting tree of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ on the input Φ . Thus at least $(1 - \epsilon)S$ vertices from the set $\{v_1, \dots, v_S\}$ must be visited by $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ on the input Φ . \square

So our goal is the construction of the formula Φ that has the system of myopic copies of exponential size. And moreover the uniform distribution on the myopic representatives of Φ should be polynomial time samplable.

We consider formulas that code a system of linear equations $Ax = b$ over the field \mathbb{F}_2 , where A is a $n \times n$ matrix.

We require that in every row of the matrix A the number of ones is bounded by the constant d . In this case the linear system consists of equations of type $x_{i_1} + x_{i_2} + \dots + x_{i_s} = b_i$, where $s \leq d$. Such equation may be written in d -CNF form with at most 2^d clauses. Note that the d -CNF form of equalities with right hand sides 0 and 1 differs only in the literal signs and also the number of positive and negative occurrences for every variable is the same. Therefore myopic procedure can't see a bit of the right hand side until it requests at least one of corresponding clauses. In what follows we say that a myopic procedure opens a bit of the vector b if it requests a clause that corresponds to this bit.

We will choose a nonsingular matrix A , in this case the system $Ax = b$ has a unique solution. It means that the variable x_1 can't take some value $\alpha \in \{0, 1\}$. The formula Φ denotes the formula that encodes the system $Ax = b$ after the substitution $x_1 := \alpha$. Φ is unsatisfiable by the choice of α .

Now we may describe some intuition how to construct the system of myopic representatives of the formula Φ . For vertices v_i from the splitting tree of $\mathcal{D}_{A,B,1}$ we have to find $b' \in \{0, 1\}^n$ such that the solution of the system $Ax = b'$ would be consistent with substitutions made during the path from the root to v_i and with $x_1 := \alpha$. However it is not clear why it is possible. We will do it by the appropriate choice of matrix A .

5 Boundary Expanders and the Closure

We consider a bipartite graph G (multiple edges are allowed), we denote its parts by X and Y . The boundary of the set $I \subset Y$ is a set $\delta(I) = \{x \in X \mid \text{there is exactly one edge between } I \text{ and } x\}$. The graph G is called (r, d, c) -boundary expander if the degrees of all vertices from Y do not exceed d and for every set $I \subseteq Y$ if $|I| \leq r$, then $|\delta(I)| \geq c|I|$.

Let G be an (r, d, c) -boundary expander. Let's $J \subseteq X$; a closure of the set J is the maximum subset $I \subseteq Y$ that satisfies the following properties: 1) $|I| \leq r$; 2) $\delta(I) \subseteq J$. The closure may be not uniquely defined; we denote lexicographically first closure of the set J by $Cl(J)$.

Proposition 1. *Let G be an (r, d, c) -boundary expander with $c \geq 1$, then the following properties hold. 1) If I is a closure of the set J , then $|I| \leq \frac{|J|}{c}$. 2) If $|J| < r/2$, there is the unique closure of the set J . 3) Let's $J \subseteq J'$ and $|J'| < r/2$, then $Cl(J) \subseteq Cl(J')$.*

For every bipartite graph G with parts X and Y we define a $|Y| \times |X|$ matrix A . Elements of X correspond to columns, elements of Y correspond to Y . $A_{y,x}$ is the number of edges between x and y modulo 2. Let G be an (r, d, c) -boundary expander with $c \geq 1$. We call A an adjacency matrix of G . We consider a linear system $Ax = b$, where x is a vector of unknowns of size $|X|$. Let ρ be a partial substitution for variables of x . A partial substitution ρ is called *locally consistent* if $|\rho| < r/2$ and the system of equalities $Ax|_{Cl(\text{vars}(\rho))} = b|_{Cl(\text{vars}(\rho))}$ has a solution that is consistent with ρ . Here and after for vector z by $z|_E$ we denote the projection of z to the coordinates from a set E .

Lemma 2 (□). *If a partial substitution ρ is locally consistent, then for every $I \subseteq Y, |I| \leq r/2$ the system $Ax|_I = b|_I$ has a solution that is consistent with ρ .*

6 Refined Splitting Tree

Lemma 3. *There exists an algorithm that given $n \in \mathbb{N}$ as an input in polynomial in n time returns an (r, d, c) -boundary expander G with n vertices in each part with nonsingular adjacency matrix, where $r = \Omega(n)$, $c > 2$, d is a constant and degrees of all vertices from X are bounded by a constant D .*

Let G be a graph from Lemma 3 and A be its adjacency matrix. Let $x_1 := \alpha$ be such a substitution to the variable x_1 that makes the system $Ax = b$ unsatisfiable. Let formula Φ encode the system of equalities $Ax = b$ after the substitution $x_1 := \alpha$.

We consider the splitting tree T_Φ made by algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, 1}$ on the input Φ . We may assume that every vertex of T_Φ has a partial substitution that consists of all substitutions made on the path from the root to the current vertex. We also include $x_1 := \alpha$ in all such substitutions.

We go through all vertices of the tree T_Φ in the order of increasing of their depth. For some vertices we will delete the subtree rooted by them from T_Φ . If a vertex is deleted we will not consider it latter. Suppose we consider a vertex v ; if the partial substitution ρ_v that corresponds to v is not locally consistent, then we remove the subtree rooted with v from T_Φ . We denote the resulting tree by T'_Φ .

Lemma 4. 1) *The length of every path in T'_Φ from the root to a leaf is at least $r/2 - 2$. 2) At least one half of first $r/2 - 3$ vertices on every path from the root to a leaf in T'_Φ are splitting points (that is they have two direct descendants).*

Proof. 1) Since $c > 2$ we get $Cl(\{x_1\}) = \emptyset$ and therefore the substitution $\{x_1 := \alpha\}$ is locally consistent. Let ρ be a locally consistent substitution that corresponds to a vertex v in the tree T_Φ . Let x_v be a splitting variable in v and $|\rho| < r/2 - 1$; item 1 of Proposition □ implies $Cl(vars(\rho) \cup \{x_v\}) < r/2$. Since ρ is locally consistent, there exists such α_v that $\rho \cup \{x_v := \alpha_v\}$ is locally consistent. Finally we note that a leaf of the tree T_Φ can't correspond to a locally consistent substitution since Φ is unsatisfiable.

2) Let v be some vertex of T'_Φ that belongs to the path from the root of length at most $r/2 - 3$ and u be the vertex that belongs to the path from the root to a leaf that contains v , and the length of the path from the root to u equals $\lceil r/2 \rceil - 1$. Let's $|\rho_v| \leq r/2 - 2$ and ρ_v is locally consistent. Let's $\rho'_v = \rho_v \cup \{x_v := \alpha_v\}$ is not locally consistent. The latter means that the value of x_v follows from the substitution ρ_v and equalities that correspond to $Cl(vars(\rho'_v))$. The item 3 of Proposition □ implies that $Cl(vars(\rho'_v)) \subseteq Cl(vars(\rho_u))$. We split the set $vars(\rho_u)$ into $P \cup Q$, where P corresponds to variables in splitting vertices (i.e. the corresponding vertex has two direct descendants) and Q corresponds to variables with unique descendant. The values of variables from Q are followed from

the value of variables P and equalities corresponding to the set $Cl(vars(\rho_u))$. Note that equalities corresponding to the partial substitution ρ_u are linearly independent. Therefore $|Q| \leq |Cl(vars(\rho_u))| \leq \frac{|vars(\rho_u)|}{c} < \frac{|vars(\rho_u)|}{2}$. \square

Corollary 1. *The size of T'_Φ (and therefore T_Φ) is at least $2^{r/4-2}$.*

Let K be the upper bound on the number of bits of right hand side of linear system that procedures **A** and **C** can open per step. Now we construct the system of myopic copies for the formula Φ . Let's consider the tree T'_Φ ; on every path from the root to a leaf we select a vertex such that among its ancestor there are exactly N splitting points, where $N \leq \frac{r/4-2}{K}$. Lemma 4 implies that the distance from every selected vertex to the root is at most $2N$ and the number of selected vertices equals 2^N . Let's denote the set of selected vertices by $\{v_1, v_2, \dots, v_{2^N}\}$. We consider the execution of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ along the path from the root to v_i . Along this path procedures **A** and **C** open at most $2NK$ bits of b . Since $2NK < r/2$ and the substitution ρ_{v_i} is locally consistent there exists a vector b' such that the system $Ax = b'$ has a solution that is consistent with ρ_{v_i} and b and b' agree on all bits open along the path. Let Φ_i be a formula that encodes the linear system $Ax = b'$ after the substitution $x_1 := \alpha$.

We still have to prove that the uniform distribution on formulas Φ_i is polynomial time samplable. Assume that there exists a polynomial time algorithm that given a vertex from T_Φ decides whether it has one or two direct descendants in the tree T'_Φ and if it has only one descendant it says which one. Under this assumption we may generate uniform distribution on the set $\{v_1, v_2, \dots, v_{2^N}\}$. Namely we simulate the running of algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{1}}$ in the following way: the procedure **A** chooses a variable for splitting. If the current vertex is a splitting point, then we choose a value for the variable uniformly at random, otherwise we choose the value that leads to the descendant in T'_Φ . We stop after N splitting points; let v_i be the vertex where we stopped. We also should save the bits of b opened by procedures **A** and **C**. Based on this information it is not hard to construct the formula Φ_i by Gaussing elimination.

Now we have to describe a way to determine whether a given vertex is a splitting point or not and if it is not a splitting point to find a correct descendant. Unfortunately it is not so obvious since it is not clear whether we may calculate a closure in polynomial time. However Lemma 2 implies that the substitution is locally consistent if it is consistent with every set of rows of size at most $r/2$. If the substitution is not locally correct then it contradicts the set of rows corresponding to the closure and therefore to any superset of the closure. So it is sufficient to construct in polynomial time a superset of the closure of size at most $r/2$. To do this we show that the closure of the set J is contained in a relatively small neighborhood of the set J . And this neighborhood is an easy computable superset of $Cl(J)$.

Let $\Gamma^k(J)$ denote the set of all vertices connected with J by a path of a length at most k .

Lemma 5. *If $|J| < r/2$, then $Cl(J) \subseteq \Gamma^k(J)$, where $k = O(\log |J|)$.*

Proof. Let's $I = Cl(J)$. We split I into disjoint parts $I_1 = I \cap \Gamma^1(J)$, $I_{j+1} = (I \setminus \bigcup_{i=1}^j I_i) \cap \Gamma^{2j+1}(J)$; let I_ℓ be the last nonempty set. We denote $S_i = \bigcup_{j=i}^\ell I_j$. Proposition 11 implies that $|S_i| \leq |I| < r/2$. Since $\delta(I) \subseteq J$, then for $i \geq 2$ the following is satisfied $\delta(S_i) \subseteq \Gamma(I_{i-1})$. Hence $c|S_i| \leq d|I_{i-1}|$. The latter inequality implies $|S_i|\frac{c}{d} \leq |I_{i-1}|$, hence $|S_i|(1 + \frac{c}{d}) \leq |S_{i-1}|$. And therefore $1 \leq |I_\ell| = |S_\ell| \leq \frac{|J|}{(1 + \frac{c}{d})^{\ell-1}}$, hence $\ell \leq \log_{1+c/d} |J| + 1$. \square

By Lemma 6 degrees of all vertices from G are bounded by a constant, therefore we may conclude the following corollary.

Corollary 2. *There exists $\delta > 0$ such that if $|J| < n^\delta$, then the inequality $|\Gamma^k(J)| < r/2$ is satisfied.*

Alltogether we prove the following statement.

Lemma 6. *There exists a polynomial time algorithm that given $n \in \mathbb{N}$ returns an unsatisfiable formula Φ ; there exist a constant δ such that for all myopic polynomial time procedures \mathbf{A} and \mathbf{C} there exists a system of myopic copies for Φ of size 2^N with myopic representatives $\Phi_1, \Phi_2, \dots, \Phi_{2^N}$ in the splitting tree of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{1}}$ for any heuristic \mathbf{B} , where $N = \min\{n^\delta, \frac{r/4-2}{K}\}$ and K is a total number of clauses that procedures \mathbf{A}, \mathbf{C} may request per step. Moreover the uniform distribution on the set $\{\Phi_1, \Phi_2, \dots, \Phi_{2^N}\}$ is samplable in polynomial in n time.*

Lemma 11 and Lemma 6 imply

Theorem 1. *There exists a polynomial time algorithm that outputs unsatisfiable formula $\Phi^{(n)}$ in polynomial in n time. There exists $\delta > 0$ such that for every myopic polynomial time procedures \mathbf{A} and \mathbf{C} there exists polynomial time samplable ensemble of distributions R_n concentrated on satisfiable formulas such that if for some procedure \mathbf{B} and some $\epsilon > 0$ the inequality $\Pr_{\phi \leftarrow R_n}[\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\phi) = 1] \geq 1 - \epsilon$ is satisfied, then running time of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}(\Phi)$ is at least $(1 - \epsilon)2^N$, where $N = \min\{n^\delta, r/K\}$ and $r = \Omega(n)$.*

Based on the fact that formula $\Phi^{(n)}$ doesn't depend on procedures \mathbf{A} and \mathbf{C} and that sampling time of R_n is bounded by a polynomial in runing time of \mathbf{A} and \mathbf{C} , we construct the universal distribution that would dominate all others distributions. Let $(\mathcal{R}^{(i)}, c_i)$ be an enumeration of all samplers $\mathcal{R}^{(i)}$ with time bounded by n^{c_i} for all distributions given by Theorem 11 for all DPLL algorithms based on myopic polynomial time procedures \mathbf{A} and \mathbf{C} . We construct a sampler \mathcal{Q} , that defines the ensemble of distributions Q_n . \mathcal{Q} on the input 1^n with probability $\frac{1}{2^i}$ outputs the answer of the sampler $\mathcal{R}^{(i)}(1^{\lfloor n^{1/c_i} \rfloor})$ executed on at most n^{c_i} steps for all $1 \leq i \leq n$, and with probability $\frac{1}{2^n}$ outputs some fixed satisfiable formula ϕ_0 .

Theorem 2. *For every polynomial time myopic procedures \mathbf{A} and \mathbf{C} there exists such positive ϵ that for every procedure \mathbf{B} algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ for all n errs on*

a random formula distributed according to Q_n with probability less than ϵ , then for all large enough n the running time of $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ on the input $\Phi^{(n)}$ is at least 2^{N-1} , where $N = \min\{n^\delta, r/K\}$ and $r = \Omega(n)$, δ is a positive constant.

Proof. Let \mathcal{R}_i be a sampler that corresponds to procedures **A** and **C** and its running time is bounded by n^{c_i} . Then for all $n \geq i$ the sampler $\mathcal{R}^{(i)}$ is present in the enumeration in the definition of Q . Let $\epsilon = \frac{1}{2^{i+1}}$; for $i \geq n$ the algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ works correctly on a random input according to $R_{\lfloor n^{1/c_i} \rfloor}^{(i)}$ with probability at least $\frac{1}{2}$, where $R_n^{(i)}$ is an ensemble of distributions generated by $\mathcal{R}^{(i)}$. Let $m = \lfloor n^{1/c_i} \rfloor$; then the algorithm $\mathcal{D}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ runs on the input Φ_m at least 2^{N-1} steps, where $N = \min\{m^\delta, r/K(m)\}$ and $r = \Omega(m)$, δ is a positive constant from Theorem II. We are done since m goes through all large enough natural numbers. \square

Acknowledgments. The authors thank Edward A. Hirsch for the statement of the problem and also thank Ilya Posov and anonymous reviewers for useful comments.

References

1. Alekhnovich, M., Hirsch, E.A., Itsykson, D.: Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. *J. Autom. Reason.* 35(1-3), 51–72 (2005)
2. Cook, J., Etesami, O., Miller, R., Trevisan, L.: Goldreich’s One-Way Function Candidate and Myopic Backtracking Algorithms. In: Reingold, O. (ed.) *TCC 2009. LNCS*, vol. 5444, pp. 521–538. Springer, Heidelberg (2009)
3. Goldreich, O.: Candidate one-way functions based on expander graphs. Technical Report 00-090, Electronic Colloquium on Computational Complexity (2000)
4. Hirsch, E.A., Itsykson, D., Monakhov, I., Smal, A.: On optimal heuristic randomized semidecision procedures, with applications to proof complexity and cryptography. *Theory of Computing Systems* (2011); Extended abstract appeared in the proceedings of STACS 2010
5. Itsykson, D.: Lower Bound on Average-Case Complexity of Inversion of Goldreich’s Function by Drunken Backtracking Algorithms. In: Ablayev, F., Mayr, E.W. (eds.) *CSR 2010. LNCS*, vol. 6072, pp. 204–215. Springer, Heidelberg (2010)
6. Itsykson, D., Sokolov, D.: The Complexity of Inversion of Explicit Goldreich’s Function by DPLL Algorithms. In: Kulikov, A., Vereshchagin, N. (eds.) *CSR 2011. LNCS*, vol. 6651, pp. 134–147. Springer, Heidelberg (2011)
7. Miller, R.: Goldreich’s one-way function candidate and drunken backtracking algorithms. Master’s thesis, University of Virginia, Distinguished Majors Thesis (2009)
8. Tseitin, G.S.: On the complexity of derivation in the propositional calculus. *Zapiski nauchnykh seminarov LOMI* 8, 234–259 (1968); English translation of this volume: Consultants Bureau, N.Y., pp. 115–125 (1970)
9. Urquhart, A.: Hard examples for resolution. *JACM* 34(1), 209–219 (1987)

Algorithm for Single Allocation Problem on Hub-and-Spoke Networks in 2-Dimensional Plane

Ryuta Ando and Tomomi Matsui

Department of Information and System Engineering,
Faculty of Science and Engineering, Chuo University, Tokyo 112-8551, Japan

Abstract. This paper deals with a single allocation problem in hub-and-spoke networks. We handle the case that all the nodes are embedded in a 2-dimensional plane and a transportation cost (per unit flow) is proportional to Euclidean distance. We propose a randomized $(1 + 2/\pi)$ -approximation algorithm based on a linear relaxation problem and a dependent rounding procedure.

1 Introduction

This paper deals with a single allocation problem in hub-and-spoke networks. Given a set of hub nodes and a set of non-hub nodes, the problem allocates each non-hub node to exactly one of the hub nodes so that the total transportation cost is minimized where the required amount of flow and a transportation cost per unit flow are given. We show that the problem is polynomial time solvable if a cost matrix between hub nodes has Monge property. When all the nodes are embedded in a 2-dimensional plane and transportation costs are proportional to Euclidean distance, we propose $(1 + 2/\pi)$ -approximation algorithm based on a dependent rounding procedure.

The hub-and-spoke structure is based on the situation where some nodes, called non-hub nodes, can interact only via a set of completely interconnected nodes, called hub nodes. The structure arises in the airline industry, telecommunications and postal delivery systems. In 1987, O’Kelly [12] considered a hub location problem, which chooses hub nodes from given nodes and allocates remaining nodes to exactly one of the hub nodes so that a total transportation cost is minimized. After his work, a wide variety of studies have been done on this topic (e.g., [3,5]). Many heuristics are surveyed by Bryan and O’Kelly [3]. Exact algorithms are found, for example, in [8,10,11].

The single allocation problem is a subproblem of the hub location problem obtained by fixing hub locations. In many practical situations, the hub locations are fixed for some time interval because of costs of moving equipment on hubs. In this case, the decision of optimally allocating non-hub nodes to one of given hub nodes is important for efficient operation of the network. Sohn and Park [15] showed the polynomial time solvability of the problem when the number of hub nodes is equal to two. They also showed NP-hardness of the problem when

the number of hub nodes is greater than two [16]. Computational experiments with CAB data [12] are performed in [14]. Ge, Ye and Zhang discussed some approximation algorithms [7]. Iwasa, Saito and Matsui [9] proposed a randomized 2-approximation algorithm for the problem.

2 Problem Formulations and Algorithm

Let H and N be sets of hub nodes with $|H| = h$ and non-hub nodes with $|N| = n$, respectively. We define $\widetilde{N}^2 \stackrel{\text{def.}}{=} \{(p, q) \in N^2 \mid p \neq q\}$. For any pair of nodes $(p, q) \in \widetilde{N}^2 \cup (N \times H) \cup (H \times N)$, a given non-negative amount of flow from p to q is denoted by $w_{pq} (\geq 0)$. For any pair of nodes $(i, j) \in (H \times H) \cup (H \times N) \cup (N \times H)$, a given non-negative transportation cost per unit flow is denoted by $c_{ij} (\geq 0)$.

First, we introduce two naive assumptions of the cost vector.

Assumption 1. *A given cost c_{ij} satisfies*

- (i) $c_{ii} = 0$ for any $i \in H$,
- (ii) triangle inequalities among hubs, i.e., $c_{ij} \leq c_{ik} + c_{kj}$ for any $(i, j, k) \in H^3$,
- (iii) symmetry, i.e., $c_{ij} = c_{ji}$ ($\forall (i, j) \in (H \times H) \cup (H \times N) \cup (N \times H)$).

Assumption 2. *A given cost c_{ij} satisfies $c_{ij} \leq (c_{pi} + c_{pj})$ ($\forall (p, i, j) \in N \times H^2$).*

This paper discusses situations under following assumptions.

Assumption 3. *Every node $p \in N \cup H$ is embedded in a 2-dimensional plane and a position of node p is denoted by a vector $\mathbf{v}(p) \in \mathbb{R}^2$. For any pair of nodes $(i, j) \in (H \times N) \cup (N \times H)$, the transportation cost c_{ij} is defined by the Euclidean distance $\|\mathbf{v}(i) - \mathbf{v}(j)\|$. Given a value $0 \leq \alpha \leq 1$, called “discount rate,” we define $c_{ij} = \alpha \|\mathbf{v}(i) - \mathbf{v}(j)\|$ for any pair of hub nodes $(i, j) \in H^2$.*

The discount rate represents an economy of scale with respect to the transportation among hubs. Obviously, Assumption 3 implies Assumptions 1 and 2.

Assumption 4. *A matrix of distances between hub nodes, $(c_{ij} : (i, j) \in H^2)$, satisfies the Monge property, i.e., $c_{ij} + c_{i'j'} \leq c_{ij'} + c_{i'j}$ for all $1 \leq i < i' \leq h$, $1 \leq j < j' \leq h$ where $H = \{1, 2, \dots, h\}$.*

It is well-known that if a transportation problem has a cost matrix with Monge property, then the north-west corner rule gives an optimal solution (e.g., see [4]). When all the hub nodes lie on a line in the order $(1, 2, \dots, h)$ and the transportation costs between hub nodes is proportional to Euclidean distance, Assumption 4 holds.

We introduce variables $x_{pi} \in \{0, 1\}$ ($\forall (p, i) \in N \times H$) where $x_{pi} = 1$ when non-hub node p is connected to hub node i and $x_{pi} = 0$ otherwise. Then the single allocation problem is formulated as follows:

$$\begin{aligned}
\text{QIP :} \quad \min. \quad & \sum_{(p,q) \in \widetilde{N}^2} w_{pq} \left(\sum_{i \in H} c_{pi} x_{pi} + \sum_{i \in H} \sum_{j \in H} c_{ij} x_{pi} x_{qj} + \sum_{j \in H} c_{jq} x_{qj} \right) \\
& + \sum_{(p,j) \in N \times H} w_{pj} \sum_{i \in H} (c_{pi} + c_{ij}) x_{pi} + \sum_{(i,q) \in H \times N} w_{iq} \sum_{j \in H} (c_{ij} + c_{jq}) x_{qj} \\
\text{s. t.} \quad & \sum_{i \in H} x_{pi} = 1 \quad (\forall p \in N), \\
& x_{pi} \in \{0, 1\} \quad (\forall (p, i) \in N \times H).
\end{aligned}$$

Adams and Sheralli [1] proposed a tight linearization for general zero-one quadratic programming problems. By simply applying their method, we can transform QIP to the following mixed integer programming (MIP) problem:

$$\begin{aligned}
\text{MIP :} \quad \min. \quad & \sum_{(p,q) \in \widetilde{N}^2} w_{pq} \left(\sum_{i \in H} c_{pi} x_{pi} + \sum_{i \in H} \sum_{j \in H} c_{ij} y_{piqj} + \sum_{j \in H} c_{jq} x_{qj} \right) \\
& + \sum_{(p,j) \in N \times H} w_{pj} \sum_{i \in H} (c_{pi} + c_{ij}) x_{pi} + \sum_{(i,q) \in H \times N} w_{iq} \sum_{j \in H} (c_{ij} + c_{jq}) x_{qj} \\
\text{s. t.} \quad & \sum_{i \in H} x_{pi} = 1 \quad (\forall p \in N), \\
& \sum_{j \in H} y_{piqj} = x_{pi} \quad (\forall (p, q) \in \widetilde{N}^2, \forall i \in H), \\
& \sum_{i \in H} y_{piqj} = x_{qj} \quad (\forall (p, q) \in \widetilde{N}^2, \forall j \in H), \\
& x_{pi} \in \{0, 1\} \quad (\forall (p, i) \in N \times H), \\
& y_{piqj} \geq 0 \quad (\forall (p, q) \in \widetilde{N}^2, \forall (i, j) \in H^2).
\end{aligned}$$

It is easy to show that every optimal solution of MIP is an 0-1 vector. The formulation MIP can be obtained by replacing $x_{pi}x_{qj}$ with a new variable y_{piqj} , multiplying $\sum_{i \in H} x_{pi} = 1$ by x_{qj} to derive $\sum_{i \in H} y_{piqj} = x_{qj}$ ($\sum_{j \in H} y_{piqj} = x_{pi}$ is derived in the same manner). Throughout this paper, the objective function of MIP is denoted by $\widehat{\mathbf{w}}^\top \mathbf{x} + \widetilde{\mathbf{w}}^\top \mathbf{y}$ for simplicity. These QIP and MIP formulations are also studied under the name of the quadratic semi-assignment problem (for details, see [13] and references therein).

We consider the linear programming relaxation of MIP, called LPR, obtained by substituting non-negativity constraint $x_{pi} \geq 0$ for 0-1 constraint $x_{pi} \in \{0, 1\}$. Next, we propose a rounding procedure for an optimal solution of LPR.

Let Π be the set of all the total orders of hubs, i.e., for any $\sigma \in \Pi$, $H = \{\sigma_1, \sigma_2, \dots, \sigma_h\}$. A dependent rounding procedure is defined as follows.

Dependent Rounding ($\mathbf{x}, \mathbf{y}; \sigma$)

Input: A feasible solution (\mathbf{x}, \mathbf{y}) of LPR and a total order of hubs $\sigma \in \Pi$.

Step 1: Choose a random variable U uniformly at random from $[0, 1)$.

Step 2: For each non-hub node $p \in N$, we connect p to a hub σ_i where $i \in \{1, 2, \dots, h\}$ is the minimum index satisfying $U < x_{p\sigma_1} + \dots + x_{p\sigma_i}$.

Randomized rounding method with such a dependency among 0-1 variables was named “dependent rounding” by Bertsimas, Teo, and Vohra in [2].

Lemma 1. Given a feasible solution (\mathbf{x}, \mathbf{y}) of LPR, the vector of random variables \mathbf{X}^σ obtained by executing **Dependent Rounding** $(\mathbf{x}, \mathbf{y}; \sigma)$ satisfies that $\forall (p, i) \in N \times H$, $E[X_{pi}^\sigma] = x_{pi}$.

Proof. Let $j \in H$ be a unique node satisfying $i = \sigma_j$. The definition of the procedure **Dependent Rounding** implies that

$$\begin{aligned} E[X_{pi}^\sigma] &= E[X_{p\sigma_j}^\sigma] = \Pr[X_{p\sigma_j}^\sigma = 1] \\ &= \Pr[x_{p\sigma_1} + \cdots + x_{p\sigma_{j-1}} \leq U < x_{p\sigma_1} + \cdots + x_{p\sigma_j}] = x_{p\sigma_j} = x_{pi}. \end{aligned} \quad \square$$

Now we describe our algorithm. Given an angle $\theta \in [0, 2\pi)$, and a node $p \in N \cup H$, we rotate $\mathbf{v}(p)$ by an angle θ and obtain a vector

$$\begin{pmatrix} f_1(p, \theta) \\ f_2(p, \theta) \end{pmatrix} \stackrel{\text{def.}}{=} \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \mathbf{v}(p).$$

Algorithm A

Step 1: Solve the problem LPR and obtain an optimal solution $(\mathbf{x}^*, \mathbf{y}^*)$.

Step 2: Choose an angle θ [radian] uniformly at random from an interval $[0, 2\pi)$.

Rotate every node $p \in N \cup H$ by setting the position of p to $(f_1(p, \theta), f_2(p, \theta))^\top$.

Let $\sigma(\theta) \in \Pi$ be a total order of H satisfying that a sequence of first coordinates $(f_1(\sigma(\theta)_1, \theta), f_1(\sigma(\theta)_2, \theta), \dots, f_1(\sigma(\theta)_h, \theta))$ is non-increasing.

Step 3: Execute **Dependent Rounding** $(\mathbf{x}^*, \mathbf{y}^*; \sigma(\theta))$ and output a solution.

3 North-West Corner Rule and Dependent Rounding

In this section, we describe a relation between a classical north-west corner rule solution and a solution obtained by procedure **Dependent Rounding**. Throughout this section, we denote the set of hub nodes by $H = \{1, 2, \dots, h\}$. Given a feasible solution (\mathbf{x}, \mathbf{y}) of LPR, and a pair $(p, q) \in \widetilde{N^2}$, we introduce a set of variables $\{\tilde{y}_{piqj} \mid (i, j) \in H^2\}$ and define a transportation problem:

$$\begin{aligned} \text{HTP}_{pq}(\mathbf{x}, \mathbf{y}) : \min. \quad & \sum_{i \in H} \sum_{j \in H} c_{ij} \tilde{y}_{piqj} \\ \text{s. t.} \quad & \sum_{j \in H} \tilde{y}_{piqj} = x_{pi} \quad (\forall i \in H), \\ & \sum_{i \in H} \tilde{y}_{piqj} = x_{qj} \quad (\forall j \in H), \\ & \tilde{y}_{piqj} \geq 0 \quad (\forall (i, j) \in H^2). \end{aligned}$$

Clearly, the subvector $\mathbf{y}|_{pq}$ of \mathbf{y} induced by restriction to index subset $\{piqj \mid (i, j) \in H^2\}$ is feasible to $\text{HTP}_{pq}(\mathbf{x}, \mathbf{y})$.

We discuss the well-known north-west corner rule for finding a feasible solution of the transportation problem. Given a feasible solution (\mathbf{x}, \mathbf{y}) of LPR and a total order $\sigma \in \Pi$, it is not hard to see that there exists a unique vector \mathbf{y}^σ satisfying a system of equalities:

$$\sum_{i=1}^{i'} \sum_{j=1}^{j'} y_{p\sigma_i q\sigma_j}^\sigma = \min \left\{ \sum_{i=1}^{i'} x_{p\sigma_i}, \sum_{j=1}^{j'} x_{q\sigma_j} \right\} \left(\begin{array}{l} \forall (p, q) \in \widetilde{N^2}, \\ \forall (i', j') \in H^2 = \{1, 2, \dots, h\}^2 \end{array} \right).$$

We say that $(\mathbf{x}, \mathbf{y}^\sigma)$ is a *north-west corner rule solution with respect to $(\mathbf{x}, \mathbf{y}; \sigma)$* . Clearly, $(\mathbf{x}, \mathbf{y}^\sigma)$ is also feasible to LPR. Let σ^I be the identity permutation ($\sigma_i^I = i$ for all $i \in H$). It is easy to show that subvector $\mathbf{y}^{\sigma^I}|_{pq}$ of \mathbf{y}^{σ^I} is equivalent to a solution obtained by classical north-west corner rule to $\text{HTP}_{pq}(\mathbf{x}, \mathbf{y})$. Under Assumption 4 (Monge property), the north-west corner rule solution $(\mathbf{x}, \mathbf{y}^{\sigma^I})$ implies the optimality of subvector $\mathbf{y}^{\sigma^I}|_{pq}$ to $\text{HTP}_{pq}(\mathbf{x}, \mathbf{y})$ (e.g., see [4]). Thus, we have the following.

Lemma 2. *Let (\mathbf{x}, \mathbf{y}) be a feasible solution of LPR and σ^I be the identity permutation on H . Under Assumption 4, the north-west corner rule solution $(\mathbf{x}, \mathbf{y}^{\sigma^I})$ satisfies that $\sum_{i \in H} \sum_{j \in H} c_{ij} y_{piqj}^{\sigma^I} \leq \sum_{i \in H} \sum_{j \in H} c_{ij} y_{piqj}$.*

We show a lemma useful to analyze procedure **Dependent Rounding**.

Lemma 3. *Let (\mathbf{x}, \mathbf{y}) be a feasible solution of LPR and $\sigma \in \Pi$ a total order of H . A vector of random variables \mathbf{X}^σ obtained by **Dependent Rounding** $(\mathbf{x}, \mathbf{y}; \sigma)$ satisfies that $\Pr[X_{pi}^\sigma X_{qj}^\sigma = 1] = y_{piqj}^\sigma$ ($\forall (p, q) \in \widetilde{N}^2, \forall (i, j) \in H^2$) where $(\mathbf{x}, \mathbf{y}^\sigma)$ is the north-west corner rule solution with respect to $(\mathbf{x}, \mathbf{y}; \sigma)$.*

Proof. We denote $\Pr[X_{pi}^\sigma X_{qj}^\sigma = 1]$ by y'_{piqj} for simplicity. Then the vector \mathbf{y}' satisfies that for any pairs $(p, q) \in \widetilde{N}^2$ and $(i', j') \in H^2$,

$$\begin{aligned} \sum_{i=1}^{i'} \sum_{j=1}^{j'} y'_{p\sigma_i q\sigma_j} &= \Pr \left[\left[\sum_{i=1}^{i'} X_{p\sigma_i} = 1 \right] \wedge \left[\sum_{j=1}^{j'} X_{q\sigma_j} = 1 \right] \right] \\ &= \Pr \left[\left[U < \sum_{i=1}^{i'} x_{p\sigma_i} \right] \wedge \left[U < \sum_{j=1}^{j'} x_{q\sigma_j} \right] \right] \\ &= \Pr \left[U < \min \left\{ \sum_{i=1}^{i'} x_{p\sigma_i}, \sum_{j=1}^{j'} x_{q\sigma_j} \right\} \right] = \min \left\{ \sum_{i=1}^{i'} x_{p\sigma_i}, \sum_{j=1}^{j'} x_{q\sigma_j} \right\}. \end{aligned}$$

From the above, $(\mathbf{x}, \mathbf{y}')$ is a unique north-west corner rule solution with respect to $(\mathbf{x}, \mathbf{y}; \sigma)$ and thus $\mathbf{y}' = \mathbf{y}^\sigma$. \square

We will close this section by showing that under Assumption 4, the original problem (MIP) is polynomial time solvable.

Theorem 1. *Let $(\mathbf{x}^*, \mathbf{y}^*)$ be an optimal solution of LPR. Under Assumption 4, dependent rounding $(\mathbf{x}^*, \mathbf{y}^*; \sigma^I)$ finds an optimal solution of MIP where σ^I is the identity permutation ($\sigma_i^I = i$ for all $i \in H$).*

Proof. Let \mathbf{X}^{σ^I} be a vector of random variables obtained by executing **Dependent Rounding** $(\mathbf{x}^*, \mathbf{y}^*; \sigma^I)$ and Z be the corresponding objective value. Clearly, the optimal value Z^* of MIP gives a lower bound of the random variable Z and also satisfies the following:

$$Z^* \leq \mathbb{E}[Z] \stackrel{\text{def.}}{=} \mathbb{E} \left[\hat{\mathbf{w}}^\top \mathbf{X}^{\sigma^I} + \sum_{(p,q) \in \widetilde{N}^2} w_{pq} \sum_{i \in H} \sum_{j \in H} c_{ij} X_{pi}^{\sigma^I} X_{qj}^{\sigma^I} \right]$$

$$\begin{aligned}
&= \sum_{(p,i) \in N \times H} \widehat{w}_{pi} \mathbb{E}[X^{\sigma^I}] + \sum_{(p,q) \in \widetilde{N}^2} w_{pq} \sum_{i \in H} \sum_{j \in H} c_{ij} \mathbb{E}[X_{pi}^{\sigma^I} X_{qj}^{\sigma^I}] \\
&= \sum_{(p,i) \in N \times H} \widehat{w}_{pi} x_{pi}^* + \sum_{(p,q) \in \widetilde{N}^2} w_{pq} \sum_{i \in H} \sum_{j \in H} c_{ij} y_{piqj}^{\sigma^I} \\
&\leq \widehat{\mathbf{w}}^\top \mathbf{x}^* + \sum_{(p,q) \in \widetilde{N}^2} w_{pq} \sum_{i \in H} \sum_{j \in H} c_{ij} y_{piqj}^* = \widehat{\mathbf{w}}^\top \mathbf{x}^* + \widetilde{\mathbf{w}}^\top \mathbf{y}^* \leq Z^*.
\end{aligned}$$

From the above, random variable Z is always equal to Z^* . Thus, **dependent rounding** $(\mathbf{x}, \mathbf{y}; \sigma^I)$ always finds a solution optimal to MIP (even if it is a randomized procedure). \square

The above theorem directly implies that under Assumption 4, LPR has a 0-1 valued optimal solution, which is also optimal to MIP. Chekuri et al. [6] discussed a dependent rounding procedure in the context of the metric labeling problem. They dealt with a line metric case and pointed out a relation to Monge property.

4 Analysis of Approximation Ratio

In this section, we discuss the approximation ratio of our algorithm. First, we briefly review a key lemma proved in [9].

Lemma 4. [9] Let (\mathbf{x}, \mathbf{y}) and $(\mathbf{x}', \mathbf{y}')$ be feasible solutions of LPR with $\mathbf{x} = \mathbf{x}'$. Under Assumptions 1 and 2, the inequality

$$\sum_{i \in H} \sum_{j \in H} c_{ij} y'_{piqj} \leq \sum_{i \in H} c_{pi} x_{pi} + \sum_{i \in H} \sum_{j \in H} c_{ij} y_{piqj} + \sum_{j \in H} c_{jq} x_{qj}$$

holds for any $(p, q) \in \widetilde{N}^2$.

For any pair of nodes $(i, j) \in (H \times N) \cup (N \times H) \cup (H \times H)$, we define $d_{ij} \stackrel{\text{def.}}{=} \|v(i) - v(j)\|$, $d_1(\theta)_{ij} \stackrel{\text{def.}}{=} |f_1(i, \theta) - f_1(j, \theta)|$, and $d_2(\theta)_{ij} \stackrel{\text{def.}}{=} |f_2(i, \theta) - f_2(j, \theta)|$. The following lemma shows some properties of procedure **Dependent Rounding**.

Lemma 5. Let (\mathbf{x}, \mathbf{y}) be a feasible solution of LPR. For any (fixed) angle $\theta \in [0, 2\pi]$, the vector of random variables \mathbf{X}^θ obtained by executing the procedure **Dependent Rounding** $(\mathbf{x}, \mathbf{y}; \sigma(\theta))$ satisfies that

$$\begin{aligned}
(1) \quad &\mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_{ij} X_{pi}^\theta X_{qj}^\theta \right] \\
&\leq \sum_{i \in H} d_{pi} x_{pi} + \sum_{i \in H} \sum_{j \in H} d_{ij} y_{piqj} + \sum_{j \in H} d_{jq} x_{qj} \quad (\forall (p, q) \in \widetilde{N}^2),
\end{aligned}$$

$$\begin{aligned}
(2) \quad & \mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_2(\theta)_{ij} X_{pi}^\theta X_{qj}^\theta \right] \\
& \leq \sum_{i \in H} d_2(\theta)_{pi} x_{pi} + \sum_{i \in H} \sum_{j \in H} d_2(\theta)_{ij} y_{piqj} + \sum_{j \in q} d_2(\theta)_{jq} x_{qj} \quad (\forall (p, q) \in \widetilde{N^2}), \\
(3) \quad & \mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} X_{pi}^\theta X_{qj}^\theta \right] \leq \sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} y_{piqj} \quad (\forall (p, q) \in \widetilde{N^2}).
\end{aligned}$$

Proof.

Case (1). Since both (\mathbf{x}, \mathbf{y}) and $(\mathbf{x}, \mathbf{y}^{\sigma(\theta)})$ are feasible to LPR and distance d_{ij} satisfies Assumptions 1 and 2, Lemmas 3 and 4 directly imply that

$$\begin{aligned}
\mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_{ij} X_{pi}^\theta X_{qj}^\theta \right] &= \sum_{i \in H} \sum_{j \in H} d_{ij} \Pr[X_{pi}^\theta X_{qj}^\theta = 1] = \sum_{i \in H} \sum_{j \in H} d_{ij} y_{piqj}^{\sigma(\theta)} \\
&\leq \sum_{i \in H} d_{pi} x_{pi} + \sum_{i \in H} \sum_{j \in H} d_{ij} y_{piqj} + \sum_{j \in H} d_{jq} x_{qj}.
\end{aligned}$$

Case (2). We can show the second inequality in a similar way with Case (1). Since distance $d_2(\theta)_{ij}$ also satisfies Assumptions 1 and 2, Lemmas 3 and 4 imply the desired inequality.

Case (3). The total order $\sigma(\theta)$ satisfies that a matrix of (first coordinate) distances $(d_1(\theta)_{ij} : (i, j) \in H^2)$ whose rows and columns are ordered by $\sigma(\theta)$ has Monge property. Hence, the north-west corner rule solution $(\mathbf{x}, \mathbf{y}^{\sigma(\theta)})$ satisfies that subvector $\mathbf{y}^{\sigma(\theta)}|_{pq}$ of $\mathbf{y}^{\sigma(\theta)}$ is optimal to a transportation problem $\text{HTP}_{pq}(\mathbf{x}, \mathbf{y})$ by substituting $d_1(\theta)_{ij}$ for c_{ij} . The optimality of the north-west corner rule solution implies that

$$\begin{aligned}
\mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} X_{pi}^\theta X_{qj}^\theta \right] &= \sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} \Pr[X_{pi}^\theta X_{qj}^\theta = 1] \\
&= \sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} y_{piqj}^{\sigma(\theta)} \leq \sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} y_{piqj}.
\end{aligned}$$

□

Now, we discuss the expectation of the objective value.

Theorem 2. Let $(\mathbf{x}^*, \mathbf{y}^*)$ be an optimal solution of LPR. Under Assumption 3, Algorithm A finds a solution whose objective value Z satisfies

$$\begin{aligned}
(1) \quad & \mathbb{E}[Z] \leq (1 + \alpha) \hat{\mathbf{w}}^\top \mathbf{x}^* + \tilde{\mathbf{w}}^\top \mathbf{y}^*, \text{ and} \\
(2) \quad & \mathbb{E}[Z] \leq (1 + \frac{2\alpha}{\pi}) \hat{\mathbf{w}}^\top \mathbf{x}^* + \frac{4}{\pi} \tilde{\mathbf{w}}^\top \mathbf{y}^*.
\end{aligned}$$

Proof. Let \mathbf{X}^θ be a vector of random variables obtained by executing the procedure **Dependent Rounding** ($\mathbf{x}, \mathbf{y}; \sigma(\theta)$). First, we fix the angle θ and discuss the expectation, denoted by $Z(\theta)$, of the objective value corresponding to \mathbf{X}^θ .

Case (1). Lemma 2 and Lemma 5 (1) imply the following:

$$\begin{aligned}
Z(\theta) &\stackrel{\text{def.}}{=} \mathbb{E} \left[\widehat{\mathbf{w}}^\top \mathbf{X}^\theta + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \sum_{i \in H} \sum_{j \in H} c_{ij} X_{pi}^\theta X_{qj}^\theta \right] \\
&= \sum_{(p,i) \in N \times H} \widehat{w}_{pi} \mathbb{E}[X_{pi}^\theta] + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_{ij} X_{pi}^\theta X_{qj}^\theta \right] \\
&\leq \sum_{(p,i) \in N \times H} \widehat{w}_{pi} x_{pi}^* + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \left(\sum_{i \in H} d_{pi} x_{pi}^* + \sum_{i \in H} \sum_{j \in H} d_{ij} y_{piqj}^* + \sum_{j \in H} d_{jq} x_{qj}^* \right) \\
&= \widehat{\mathbf{w}}^\top \mathbf{x}^* + \alpha \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \left(\sum_{i \in H} d_{pi} x_{pi}^* + \sum_{j \in H} d_{jq} x_{qj}^* \right) \\
&\quad + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \left(\sum_{i \in H} \sum_{j \in H} \alpha d_{ij} y_{piqj}^* \right) \leq (1 + \alpha) \widehat{\mathbf{w}}^\top \mathbf{x}^* + \tilde{\mathbf{w}}^\top \mathbf{y}^*.
\end{aligned}$$

Even if we choose an angle $\theta \in [0, 2\pi)$ uniformly at random, it is obvious that the expectation $\mathbb{E}[Z] = \mathbb{E}[Z(\theta)] \leq (1 + \alpha) \widehat{\mathbf{w}}^\top \mathbf{x}^* + \tilde{\mathbf{w}}^\top \mathbf{y}^*$.

Case (2). From Assumption 3, all the pairs of hub nodes $(i, j) \in H^2$ satisfy $c_{ij} = \alpha d_{ij} \leq \alpha(d_1(\theta)_{ij} + d_2(\theta)_{ij})$. Thus, Lemma 5 (2) and (3) imply the following.

$$\begin{aligned}
Z(\theta) &= \mathbb{E} \left[\widehat{\mathbf{w}}^\top \mathbf{X}^\theta + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \sum_{i \in H} \sum_{j \in H} c_{ij} X_{pi}^\theta X_{qj}^\theta \right] \\
&\leq \widehat{\mathbf{w}}^\top \mathbf{x}^* + \mathbb{E} \left[\sum_{(p,q) \in \widetilde{N^2}} w_{pq} \sum_{i \in H} \sum_{j \in H} \alpha(d_1(\theta)_{ij} + d_2(\theta)_{ij}) X_{pi}^\theta X_{qj}^\theta \right] \\
&= \widehat{\mathbf{w}}^\top \mathbf{x}^* + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \left(\mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} X_{pi}^\theta X_{qj}^\theta \right] + \mathbb{E} \left[\sum_{i \in H} \sum_{j \in H} d_2(\theta)_{ij} X_{pi}^\theta X_{qj}^\theta \right] \right) \\
&\leq \widehat{\mathbf{w}}^\top \mathbf{x}^* + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \left(\sum_{i \in H} \sum_{j \in H} d_1(\theta)_{ij} y_{piqj}^* \right) \\
&\quad + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \left(\sum_{i \in H} d_2(\theta)_{pi} x_{pi}^* + \sum_{i \in H} \sum_{j \in H} d_2(\theta)_{ij} y_{piqj}^* + \sum_{j \in H} d_2(\theta)_{jq} x_{qj}^* \right).
\end{aligned}$$

When we choose $\theta \in [0, 2\pi)$ uniformly at random, it is clear that $E[d_1(\theta)_{ij}] = E[d_2(\theta)_{ij}] = \int_0^{2\pi} \frac{d_{ij} |\sin \theta|}{2\pi} d\theta = \frac{2d_{ij}}{\pi}$. The expectation $E[Z(\theta)]$ satisfies that

$$\begin{aligned} E[Z] &= E[Z(\theta)] \leq \hat{\mathbf{w}}^\top \mathbf{x}^* + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \left(\sum_{i \in H} \sum_{j \in H} E[d_1(\theta)_{ij}] y_{piqj}^* \right) \\ &\quad + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \alpha \left(\sum_{i \in H} E[d_2(\theta)_{pi}] x_{pi}^* + \sum_{i \in H} \sum_{j \in H} E[d_2(\theta)_{ij}] y_{piqj}^* + \sum_{j \in H} E[d_2(\theta)_{jq}] x_{qj}^* \right) \\ &= \hat{\mathbf{w}}^\top \mathbf{x}^* + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \left(\sum_{i \in H} \sum_{j \in H} \frac{2\alpha d_{ij}}{\pi} y_{piqj}^* \right) \\ &\quad + \alpha \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \left(\sum_{i \in H} \frac{2d_{pi}}{\pi} x_{pi}^* + \sum_{j \in H} \frac{2d_{jq}}{\pi} x_{qj}^* \right) + \sum_{(p,q) \in \widetilde{N^2}} w_{pq} \left(\sum_{i \in H} \sum_{j \in H} \frac{2\alpha d_{ij}}{\pi} y_{piqj}^* \right) \\ &\leq \hat{\mathbf{w}}^\top \mathbf{x}^* + \frac{2}{\pi} \tilde{\mathbf{w}}^\top \mathbf{y}^* + \frac{2\alpha}{\pi} \hat{\mathbf{w}}^\top \mathbf{x}^* + \frac{2}{\pi} \tilde{\mathbf{w}}^\top \mathbf{y}^* = \left(1 + \frac{2\alpha}{\pi} \right) \hat{\mathbf{w}}^\top \mathbf{x}^* + \frac{4}{\pi} \tilde{\mathbf{w}}^\top \mathbf{y}^* \end{aligned}$$

□

Lastly, we show an approximation ratio of our algorithm.

Theorem 3. *Under Assumption 3, an approximation ratio, denoted by ρ , of Algorithm A satisfies that*

$$\rho \leq \begin{cases} 1 + \frac{\alpha(4 - \pi)}{(4 - \pi) + \alpha(\pi - 2)} & \left(0 \leq \alpha \leq \frac{4 - \pi}{2} \right), \\ 1 + \frac{2\alpha}{\pi} & \left(\frac{4 - \pi}{2} < \alpha \leq 1 \right). \end{cases}$$

Proof. The optimal value $\hat{\mathbf{w}}^\top \mathbf{x}^* + \tilde{\mathbf{w}}^\top \mathbf{y}^*$ of LPR is less than or equal to the optimal value Z^* of the original problem MIP. If $\frac{4-\pi}{2} < \alpha \leq 1$, Theorem 2 (2) directly implies the desired inequality.

When α satisfies $0 \leq \alpha \leq \frac{4-\pi}{2}$, inequalities $0 \leq \frac{\alpha\pi}{(4-\pi)+\alpha(\pi-2)} \leq 1$ hold and thus it is easy to show that

$$\begin{aligned} E[Z] &\leq \frac{\alpha\pi}{(4 - \pi) + \alpha(\pi - 2)} \left(\left(1 + \frac{2\alpha}{\pi} \right) \hat{\mathbf{w}}^\top \mathbf{x}^* + \frac{4}{\pi} \tilde{\mathbf{w}}^\top \mathbf{y}^* \right) \\ &\quad + \left(1 - \frac{\alpha\pi}{(4 - \pi) + \alpha(\pi - 2)} \right) \left((1 + \alpha) \hat{\mathbf{w}}^\top \mathbf{x}^* + \tilde{\mathbf{w}}^\top \mathbf{y}^* \right) \\ &= \left(1 + \frac{\alpha(4 - \pi)}{(4 - \pi) + \alpha(\pi - 2)} \right) (\hat{\mathbf{w}}^\top \mathbf{x}^* + \tilde{\mathbf{w}}^\top \mathbf{y}^*) \\ &\leq \left(1 + \frac{\alpha(4 - \pi)}{(4 - \pi) + \alpha(\pi - 2)} \right) Z^*. \end{aligned}$$

□

5 Conclusion

We proposed an approximation algorithm for the single allocation problem in hub-and-spoke networks, when nodes are embedded on a 2-dimensional plane and transportation cost per unit flow is proportional to Euclidean distance. When discount rate α is equal to 1, the approximation ratio is bounded by $(1 + 2/\pi)$. Our algorithms can be derandomized by simply checking all the possible cases in polynomial time.

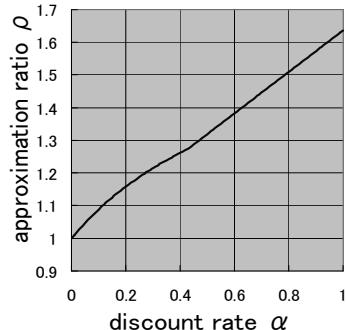


Fig. 1. Approximation ratio

References

1. Adams, W.P., Sherali, H.D.: A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science* 32, 1274–1290 (1986)
2. Bertsimas, D., Teo, C., Vohra, R.: On dependent randomized rounding algorithms. *Operations Research Letters* 24, 105–114 (1999)
3. Bryan, D.L., O’Kelly, M.E.: Hub-and-spoke networks in air transportation: an analytical review. *Journal of Regional Science* 39, 275–295 (1999)
4. Burkard, R.E., Klinz, B., Rudolf, R.: Perspectives of Monge properties in optimization. *Discrete Applied Mathematics* 70, 95–161 (1996)
5. Campbell, J.F.: Integer programming formulations of discrete hub location problems. *European Journal of Operational Research* 72, 387–405 (1994)
6. Chekuri, C., Khanna, S., Naor, J., Zosin, L.: A linear programming formulation and approximation algorithms for the metric labeling problem. *SIAM Journal on Discrete Mathematics* 18, 608–625 (2005)
7. Ge, D., Ye, Y., Zhang, J.: The Fixed-Hub Single Allocation Problem: A Geometric Rounding Approach (October 2007) (manuscript)
8. Hamacher, H.W., Labb  , M., Nickel, S., Sonneborn, T.: Adapting polyhedral properties from facility to hub location problems. *Discrete Applied Mathematics* 145, 104–116 (2004)
9. Iwasa, M., Saito, H., Matsui, T.: Approximation Algorithms for the Single Allocation Problem in Hub-and-Spoke Networks and Related Metric Labeling Problems. *Discrete Applied Mathematics* 157, 2078–2088 (2009)
10. Labb  , M., Yaman, H.: Projecting the flow variables for hub location problems. *Networks* 44, 84–93 (2004)
11. Labb  , M., Yaman, H., Gourdin, E.: A branch and cut algorithm for hub location problems with single assignment. *Mathematical Programming* 102, 371–405 (2005)
12. O’Kelly, M.E.: A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research* 32, 393–404 (1987)
13. Saito, H., Fujie, T., Matsui, T., Matuura, S.: The Quadratic semi-assignment polytope. *Discrete Optimization* 6, 37–50 (2009)
14. Saito, H., Matuura, S., Matsui, T.: A linear relaxation for hub network design problems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E85-A, 1000–1005 (2002)
15. Sohn, J., Park, S.: A linear program for the two-hub location problem. *European Journal of Operational Research* 100, 617–622 (1997)
16. Sohn, J., Park, S.: The single allocation problem in the interacting three-hub network. *Networks* 35, 17–25 (2000)

Packing-Based Approximation Algorithm for the k -Set Cover Problem

Martin Fürer* and Huiwen Yu

Department of Computer Science and Engineering,
The Pennsylvania State University, University Park, PA 16802, USA

Abstract. We present a packing-based approximation algorithm for the k -Set Cover problem. We introduce a new local search-based k -set packing heuristic, and call it Restricted k -Set Packing. We analyze its tight approximation ratio via a complicated combinatorial argument. Equipped with the Restricted k -Set Packing algorithm, our k -Set Cover algorithm is composed of the k -Set Packing heuristic [8] for $k \geq 7$, Restricted k -Set Packing for $k = 6, 5, 4$ and the semi-local $(2, 1)$ -improvement [2] for 3-Set Cover. We show that our algorithm obtains a tight approximation ratio of $H_k - 0.6402 + \Theta(\frac{1}{k})$, where H_k is the k -th harmonic number. For small k , our results are 1.8667 for $k = 6$, 1.7333 for $k = 5$ and 1.5208 for $k = 4$. Our algorithm improves the currently best approximation ratio for the k -Set Cover problem of any $k \geq 4$.

1 Introduction

Given a set of elements U and a collection of subsets \mathcal{S} of U with each subset of \mathcal{S} having size at most k and the union of \mathcal{S} being U , the k -Set Cover problem is to find a minimal size sub-collection of \mathcal{S} whose union remains U . Without loss of generality, we assume that \mathcal{S} is closed under subsets. Then the objective of the k -Set Cover problem can be viewed as finding a disjoint union of sets of \mathcal{S} which covers U .

The k -Set Cover problem is NP-hard for any $k \geq 3$. For $k = 2$, the 2-Set Cover problem is polynomial-time solvable by a maximum matching algorithm. The greedy approach for approximating the k -Set Cover problem chooses a maximal collection of i -sets (sets with size i) for each i from k down to 1. It achieves a tight approximation ratio H_k (the k -th harmonic number) [10]. The hardness result by Feige [3] shows that for $n = |U|$, the Set Cover problem is not approximable within $(1 - \epsilon) \ln n$ for any $\epsilon > 0$ unless $\text{NP} \subseteq \text{DTIME}(n^{\log \log n})$. For the k -Set Cover problem, Trevisan [12] shows that no polynomial-time algorithm has an approximation ratio better than $\ln k - \Omega(\ln \ln k)$ unless subexponential-time deterministic algorithms for NP-hard problems exist. Therefore, it is unlikely that a tremendous improvement of the approximation ratio is possible.

There is no evidence that the $\ln k - \Omega(\ln \ln k)$ lower bound can be achieved. Research on approximating the k -Set Cover problem has been focused on improving the positive constant c in the approximation ratio $H_k - c$. Small improvements on the constant might lead us closer to the optimal ratio. One of the main ideas based on greedy algorithms is

* Research supported in part by NSF Grant CCF-0728921 and CCF-0964655.

to handle small sets separately. Goldschmidt et al. [5] give a heuristic using a matching computation to deal with sets of size 2 and obtain an $H_k - \frac{1}{6}$ approximation ratio. Halldórsson [6] improves c to $\frac{1}{3}$ via his “t-change” and “augmenting path” techniques. Duh and Fürer [2] give a semi-local search algorithm for the 3-Set Cover problem and further improve c to $\frac{1}{2}$. They also present a tight example for their semi-local search algorithm.

A different idea is to replace the greedy approach by a set-packing approach. Levin [11] uses a set-packing algorithm for packing 4-sets and improves c to 0.5026 for $k \geq 4$. Athanassopoulos et al. [1] substitute the greedy phases for $k \geq 6$ with packing phases and reach an approximation ratio $H_k - 0.5902$ for $k \geq 6$.

The goal of this paper is not to provide incremental improvement in the approximation ratio for k -Set Cover. We rather want to obtain the best such result achievable by current methods. It might be the best possible result, as we conjecture the lower bound presented in [12] not to be optimal.

In this paper, we give a complete packing-based approximation algorithm (in short, PRPSLI) for the k -Set Cover problem. For $k \geq 7$, we use the k -set packing heuristic introduced by Hurkens and Shrijver [8], which achieves the best known to date approximation ratio $\frac{2}{k} - \epsilon$ for the k -Set Packing problem for any $\epsilon > 0$. On the other hand, the best hardness result by Hazan et al. [7] shows that it is NP-hard to approximate the k -Set Packing problem within $\Omega(\frac{\ln k}{k})$.

For $k = 6, 5, 4$, we use the same packing heuristic with the restriction that any local improvement should not increase the number of 1-sets which are needed to finish the disjoint set cover. We call this new heuristic Restricted k -Set Packing. We prove that for any $k \geq 5$, the Restricted k -Set Packing algorithm achieves the same approximation ratio as the corresponding unrestricted set packing heuristic. For $k = 4$, this is not the case. The approximation ratio of the Restricted 4-Set Packing algorithm is $\frac{7}{16}$, which is worse than the $\frac{1}{2} - \epsilon$ ratio of the 4-set packing heuristic but it is also tight. For $k = 3$, we use the semi-local optimization technique [2]. We thereby obtain the currently best approximation ratio for the k -Set Cover problem. We also show that our result is indeed tight. Thus, k -Set Cover algorithms which are based on packing heuristic can hardly be improved. Our novel Restricted k -Set Packing algorithm is quite simple and natural, but its analysis is complicated. It is essentially based on combinatorial arguments. We use the factor-revealing linear programming analysis for the k -Set Cover problem. The factor-revealing linear program is introduced by Jain et al. [9] for analyzing the facility location problem. Athanassopoulos et al. [1] are the first to apply it to the k -Set Cover problem.

The paper is organized as follows. In Section 2, we give the description of our algorithm and present the main results. In Section 3, we prove the approximation ratio of the Restricted k -Set Packing algorithm. In Section 4, we analyze our k -Set Cover algorithm via the factor-revealing linear program. For detailed proofs in each section, we refer the readers to full version [4].

2 Algorithm Description and the Main Theorem

In this section, we describe our packing-based k -Set Cover approximation algorithm. We first give an overview of some existing results.

Duh and Fürer [2] introduce a semi-local (s, t) -improvement for the 3-Set Cover problem. First, it greedily selects a maximal disjoint union of 3-sets. Then each local improvement replaces t 3-sets with s 3-sets, if and only if after computing a maximum matching of the remaining elements, either the total number of sets in the cover decreases, or it remains the same, while the number of 1-sets decreases. They also show that the $(2, 1)$ -improvement algorithm gives the best performance ratio for the 3-Set Cover problem among all semi-local (s, t) -improvement algorithms. The ratio is proved to be tight.

Theorem 1 ([2]). *The semi-local $(2, 1)$ -optimization algorithm for 3-Set Cover produces a solution with performance ratio $\frac{4}{3}$. It uses a minimal number of 1-sets.*

We use the semi-local $(2, 1)$ -improvement for 3-Set Cover as the basis of our k -Set Cover algorithm. Other phases of the algorithm are based on the set packing heuristic [8]. For fixed s , the heuristic starts with an arbitrary maximal packing, it replaces $p \leq s$ sets in the packing with $p + 1$ sets if the resulting collection is still a packing. Hurkens and Shrijver [8] show the following result.

Theorem 2 ([8]). *For all $\epsilon > 0$, the local search k -Set Packing algorithm for parameter $s = O(\log_k \frac{1}{\epsilon})$ has an approximation ratio $\frac{2}{k} - \epsilon$.*

The worst-case ratio is also known to be tight. We apply this packing heuristic for $k \geq 7$. For $k = 6, 5, 4$, we follow the intuition of the semi-local improvement and modify the local search of the packing heuristic, requiring that any improvement does not increase the number of 1-sets. We use the semi-local $(2, 1)$ -improvement for 3-Set Cover to compute the number of 1-sets required to finish the cover. Lemma 2.2 in [2] guarantees that the number of 1-sets returned by the semi-local $(2, 1)$ -improvement is no more than this number in any optimal solution. We compute this number first at the beginning of the restricted phase. Each time we want to make a replacement via the packing heuristic, we compute the number of 1-sets needed to finish the cover after making the replacement. If this number increases, the replacement is prohibited. To summarize, we call our algorithm the Restricted Packing-based k -Set Cover algorithm (PRPSLI) and give the pseudo-code in Algorithm 1. For input parameter $\epsilon > 0$, s_i is the parameter of the local improvement in Phase i . For any $i \neq 5, 6$, we set s_i in the same way as in Theorem 2. For $i = 5, 6$, we set $s_i = \lceil \frac{2}{i\epsilon} \rceil$.

The algorithm clearly runs in polynomial time. The approximation ratio of PRPSLI is presented in the following main theorem. For completeness, we also state the approximation ratio for the 3-Set Cover problem, which is obtained by Duh and Fürer [2] and remains the best result. Let ρ_k be the approximation ratio of the k -Set Cover problem.

Theorem 3 (Main). *For all $\epsilon > 0$, the Packing-based k -Set Cover algorithm has an approximation ratio $\rho_k = 2H_k - H_{\frac{k}{2}} + \frac{2}{k} - \frac{1}{k-1} - \frac{4}{3} + \epsilon$ for even k and $k \geq 6$; $\rho_k = 2H_k - H_{\frac{k-1}{2}} - \frac{4}{3} + \epsilon$ for odd k and $k \geq 7$; $\rho_5 = 1.7333$; $\rho_4 = 1.5208$; $\rho_3 = \frac{4}{3}$.*

Remark 1. For odd $k \geq 7$, the approximation ratio ρ_k is derived from the expression $\rho_k = \frac{2}{k} + \dots + \frac{2}{5} + \frac{1}{3} + 1 + \epsilon$. We can further obtain the asymptotic representation of ρ_k , i.e., $\rho_k = 2H_k - H_{\frac{k-1}{2}} - \frac{4}{3} + \epsilon = H_k + \ln 2 - \frac{4}{3} + \Theta(\frac{1}{k}) + \epsilon = H_k - 0.6402 + \Theta(\frac{1}{k}) + \epsilon$.

Algorithm 1. Packing-based k -Set Cover Algorithm (PRPSLI)

```

// The  $k$ -Set Packing Phase
for  $i \leftarrow k$  down to 7 do
    Select a maximal collection of disjoint  $i$ -sets.
    repeat
        Select  $p \leq s_i$   $i$ -sets and replace them with  $p + 1$   $i$ -sets.
    until there exist no more such improvements.
end for
// The Restricted  $k$ -Set Packing Phase
Run the semi-local  $(2, 1)$ -improvement algorithm for 3-Set Cover on the remaining uncovered elements to obtain the number of 1-sets.
for  $i \leftarrow 6$  to 4 do
    repeat
        Try to replace  $p \leq s_i$   $i$ -sets with  $p + 1$   $i$ -sets. Commit to the replacement only if the number of 1-sets computed by the semi-local  $(2, 1)$ -improvement for 3-Set Cover on the remaining uncovered elements does not increase.
    until there exist no more such improvements.
end for
// The Semi-Local Optimization Phase
Run the semi-local  $(2, 1)$ -improvement algorithm on the remaining uncovered elements.

```

Similarly, for even $k \geq 6$, $\rho_k = \frac{2}{k} + \frac{1}{k-1} + \frac{2}{k-3} + \dots + \frac{2}{5} + \frac{1}{3} + 1 + \epsilon = 2H_k - H_{\frac{k}{2}} + \frac{2}{k} - \frac{1}{k-1} - \frac{4}{3} + \epsilon = H_k + \ln 2 - \frac{4}{3} + \Theta(\frac{1}{k}) + \epsilon = H_k - 0.6402 + \Theta(\frac{1}{k}) + \epsilon$. The previous best asymptotic result is $H_k - 0.5902$ [1]. Finally, $\rho_5 = \frac{2}{5} + \frac{1}{3} + 1$ and $\rho_4 = \frac{7}{16} + \frac{1}{12} + 1$.

Remark 2. Restriction on Phase 6 is only required for obtaining the approximation ratio ρ_k for even k and $k \leq 12$. In other cases, only restriction on Phase 5 and Phase 4 are necessary.

We prove the main theorem in Section 4. Before that, we analyze the approximation ratio of the Restricted k -Set Packing algorithm for $k \geq 4$ in Section 3. We state the result of the approximation ratio of the Restricted k -Set Packing algorithm as follows.

Theorem 4 (Restricted k -Set Packing). *There exists a Restricted 4-Set Packing algorithm which has an approximation ratio $\frac{7}{16}$. For all $\epsilon > 0$ and for any $k \geq 5$, there exists a Restricted k -Set Packing algorithm which has an approximation ratio $\frac{2}{k} - \epsilon$.*

Remark 3. Without loss of generality, we assume that optimal solution of the Restricted k -Set Packing problem also has the property that it does not increase the number of 1-sets needed to finish the cover of the remaining uncovered elements. This assumption is justified in Section 4.

3 The Restricted k -Set Packing Algorithm

We fix one optimal solution \mathcal{O} of the Restricted k -Set Packing algorithm. We refer to the sets in \mathcal{O} as optimal sets. For fixed s , a local improvement replaces $p \leq s$ k -sets with

$p+1$ k -sets. We pick a packing of k -sets \mathcal{A} that cannot be improved by the Restricted k -Set Packing algorithm. We say an optimal set is an i -level set if exactly i of its elements are covered by sets in \mathcal{A} . For the sake of analysis, we call a local improvement an i - j -improvement if it replaces i sets in \mathcal{A} with j sets in \mathcal{O} . As a convention in the rest of the paper, small letters represent elements, capital letters represent subsets of U , and calligraphic letters represent collections of sets. We first introduce the notion of blocking.

3.1 Blocking

The main difference between unrestricted k -set packing and restricted k -set packing is the restriction on the number of 1-sets which are needed to finish the covering via the semi-local (2,1)-improvement. This restriction can prohibit a local improvement. If any i - j -improvement is prohibited because of an increase of 1-sets, we say there exists a *blocking*. In Example 1 we construct an instance of 4-set packing to help explain how blocking works.

Example 1 (Blocking). Consider an instance (U, \mathcal{S}) of the 4-Set Cover problem. Suppose there is an optimal solution \mathcal{O} of the Restricted 4-Set Packing algorithm which consists of only disjoint 4-sets that cover all elements. Let $\mathcal{O} = \{O_i\}_{i=1}^{16} \cup \{B_i\}_{i=1}^m$, $m > 12$. Let $\{A_i\}_{i=1}^7$ be a collection of 4-sets chosen by the algorithm. $\{O_i\}_{i=1}^{16}$ is a collection of 1-level or 2-level sets. Denote the j -th element of O_i by o_i^j , for $j = 1, 2, 3, 4$. If $A_i = (o_{i1}^{j_1}, o_{i2}^{j_2}, o_{i3}^{j_3}, o_{i4}^{j_4})$, we say that A_i covers the elements $o_{i1}^{j_1}, o_{i2}^{j_2}, o_{i3}^{j_3}$ and $o_{i4}^{j_4}$. Denote the following unit by \mathcal{U} ,

$$\mathcal{U} = \begin{cases} A_1 = (o_1^1, o_5^1, o_9^1, o_{13}^1) \\ A_2 = (o_2^1, o_6^1, o_{10}^1, o_{14}^1) \\ A_3 = (o_3^1, o_7^1, o_{11}^1, o_{15}^1) \\ A_4 = (o_4^1, o_8^1, o_{12}^1, o_{16}^1) \\ A_5 = (o_1^2, o_2^2, o_3^2, o_4^2) \\ A_6 = (o_5^2, o_6^2, o_7^2, o_8^2) \\ A_7 = (o_9^2, o_{10}^2, o_{11}^2, o_{12}^2) \end{cases}$$

We visualize this construction in Fig. 1. Let each cube represent a 4-set in $\{A_i\}_{i=1}^7$ and we place $\{O_i\}_{i=1}^{16}$ vertically within a 4×4 square (not shown in the figure), such that each O_i intersects with one or two sets in $\{A_i\}_{i=1}^7$. $\{A_i\}_{i=1}^7$ are placed horizontally. Notice that $O_{13}, O_{14}, O_{15}, O_{16}$ which intersects with A_1, A_2, A_3, A_4 respectively are 1-level sets. The other 12 sets in $\{O_i\}_{i=1}^{16}$ are 2-level sets.

$\{B_i\}_{i=1}^m$ is a collection of 3-level sets. Notice that for our fixed optimal solution, all elements can be covered by 4-sets, so there is no 1-set needed to finish the cover. For given \mathcal{S} , when we compute an extension of the packing to a full cover via the semi-local (2,1)-improvement, assume the unpacked element of B_i ($1 \leq i \leq 12$) can only be covered by a 2-set intersecting with both B_i and O_i , or it introduces a 1-set in the cover. The remaining unpacked elements of $\{O_i\}_{i=1}^{16}$ and $\{B_i\}_{i=1}^m$ can be covered arbitrarily by 2-sets and 3-sets. In unrestricted packing, one of the local improvements consists of replacing A_1, A_2, A_3, A_4, A_5 by $O_1, O_2, O_3, O_4, O_{13}, O_{14}, O_{15}, O_{16}$. However, in

restricted packing, for $1 \leq i \leq 12$, adding any O_i to the packing would create a 1-set covering the unpacked element of B_i during the semi-local $(2,1)$ -improvement. Hence this local improvement is prohibited as a result of restricting on the number of 1-sets.

We remark that blocking can be much more complicated than in this simple example. As we shall see later in Section 3.2, for the Restricted 4-Set Packing problem, a 3-level set can initiate a blocking of many optimal sets.

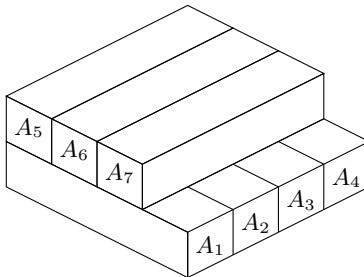


Fig. 1. Placement of A_1 to A_7

We now define blocking formally. We are given a fixed optimal k -set packing \mathcal{O} of U and a k -set packing \mathcal{A} chosen by the Restricted k -Set Packing algorithm. We consider all possible extensions of \mathcal{A} to a disjoint cover of U by 1-sets, 2-sets and 3-sets. We order these extensions lexicographically, first by the number of 1-sets, second by the total number of 2-sets and 3-sets which are not within a k -set of \mathcal{O} , and third by the number of 3-sets which are not within a k -set of \mathcal{O} . We are interested in the lexicographically first extension. Notice that we pick this specific extension for analysis only. We cannot obtain this ordering without access to \mathcal{O} .

Suppose we finish the cover from the packing \mathcal{A} with the lexicographically first extension. Let \mathcal{F} be an undirected graph such that each vertex in \mathcal{F} represents an optimal set. Two vertices are adjacent if and only if there is a 2-set in the extension intersecting with the corresponding optimal sets. Since the number of 2-sets and 3-sets not within an optimal set is minimized, there are no multiple edges in the graph. For brevity, when we talk about a node V in \mathcal{F} , we also refer to V as the corresponding optimal set. Moreover, when we say the *degree* of a node V , we refer to the number of neighbors of V .

Proposition 1. \mathcal{F} is a forest.

Proposition 2. For any $i < k - 1$, there is no 1-set inside an i -level set. i.e. 1-set can only appear in $(k - 1)$ -level sets.

Proposition 3. For any tree \mathcal{T} in \mathcal{F} , there is at most one node which represents an i -level set, such that the degree of the node is smaller than $k - i$.

For any tree, if there exists a node with property in Proposition 3, we define it to be the root. Otherwise, we know that all degree 1 nodes represent $(k - 1)$ -level sets. We define

an arbitrary node not representing a $(k - 1)$ -level set to be the root. If there are only $(k - 1)$ -level sets in the tree, i.e. the tree degenerates to one edge or a single point, we define an arbitrary $(k - 1)$ -level set to be the root. All leaves represent $(k - 1)$ -level sets. (The root is not considered to be a leaf.) We call such a tree a *blocking tree*. For any subtree, we say that the leaves *block* the nodes in this subtree. We also call the set represented by a leaf a *blocking set*.

We consider one further property of the root.

Proposition 4. *Let $k \geq 4$. In any blocking tree, there exists at most one node of either 0-level or 1-level that is of degree 2. If such a node exists, it is the root.*

Based on these simple structures of the blocking tree, we are now ready to prove the approximation ratio of the Restricted k -Set Packing algorithm.

3.2 Analysis of the Restricted 4-Set Packing Algorithm

We prove in this section that the Restricted 4-Set Packing algorithm has an approximation ratio $\frac{7}{16}$. We first explain how this $\frac{7}{16}$ ratio is derived. We use the unit \mathcal{U} defined in Example 1. Assume when the algorithm stops, we have $n \gg 1$ copies of \mathcal{U} and a relatively small number of 3-level sets. We denote the i -th copy of \mathcal{U} by \mathcal{U}_i . For each i and $1 \leq j \leq 12$, the set O_j in \mathcal{U}_i and \mathcal{U}_{i+1} are adjacent. This chain of O_j 's starts from and ends at a 3-level set respectively. Then the performance ratio of this instance is slightly larger than $\frac{7}{16}$. We first prove that the approximation ratio of the Restricted 4-Set Packing algorithm is at least $\frac{7}{16}$.

Given \mathcal{T} , a collection of blocking trees. We assign 4 tokens to every element covered by sets chosen by the restricted packing algorithm. We say a set has a free token if after distributing the token, this set retains at least 7 tokens. We show that we can always distribute the tokens among all the optimal sets \mathcal{O} , so that there are at least 7 tokens in each optimal set.

Proof. We present the first round of redistribution.

Round 1 - Redistribution in each blocking tree \mathcal{T} . Every leaf in \mathcal{T} has 4 free tokens to distribute. Every internal node V of degree d requests $4(d - 2)$ tokens from a leaf. We consider each node with nonzero request in the reverse order given by breadth first search (BFS).

- If $d = 3$, V requests 4 tokens from any leaf in the subtree rooted at V which has 12 tokens.
- If $d = 4$, V has three children V_1, V_2, V_3 . V sends requests of 4 tokens to any leaf in the subtree rooted at V_1, V_2, V_3 , one for each subtree. V takes any two donations of 4 tokens.
- The root of degree r receives the rest of the tokens contributed by the leaves.

We can show that after Round 1, every internal node in \mathcal{T} has at least 8 tokens, and the root of degree r has $4r$ tokens.

According to Proposition 4, we know that after the first round of redistribution, every node has at least 8 tokens except any 0-level roots which are of degree 1 and any singletons in \mathcal{F} which are 1-level sets.

We first consider the collection of 1-level sets \mathcal{S}_1 which are singletons in \mathcal{F} . Let S_1 be such a 1-level set that intersects with a 4-set A chosen by the algorithm. Assume A also intersects with j other optimal sets $\{O_i\}_{i=1}^j$.

We point out that no O_i belongs to \mathcal{S}_1 . Otherwise suppose $O_i \in \mathcal{S}_1$. Then there is a 1-2-improvement (replace A with S_1 and O_i).

We give the second round of redistribution, such that after this round, every set in \mathcal{S}_1 has at least 7 tokens. For optimal sets O, W , consider each token request sent to O from W . We say it is an *internal request*, if $W \in \mathcal{S}_1$, and W and O intersect with a set A chosen by the algorithm. Otherwise, we say it is an *external request*.

Round 2 - Redistribution for $S_1 \in \mathcal{S}_1$. S_1 sends τ requests of one token to O_i if $|O_i \cap A| = \tau$. For each node V in \mathcal{T} , internal requests are considered prior to external requests.

For each request sent to V from W ,

- If V has at least 8 tokens, give one to W .
- If V has only 7 tokens.
 - (1) If V is a leaf, it requests from the node which has received 4 tokens from it during the first round of redistribution.
 - (2) If V is not a leaf,
 - (2.1) If $W \in \mathcal{S}_1$, V requests a token from a leaf which has at least 8 tokens in the subtree rooted at V .
 - (2.2) If W is a node in \mathcal{T} . Suppose V has children V_1, \dots, V_d and W belongs to the subtree rooted at V_1 . V requests from a leaf which has at least 8 tokens in the subtree rooted at V_2, \dots, V_d .

We can show that after the second round of distribution, every set in \mathcal{S}_1 has at least 7 tokens. Moreover, every root of level 0 and degree 1 retains 4 tokens.

We consider a root R which is a 0-level set of degree 1. R receives 4 tokens from leaf B . Assume B is covered by $\{A_i\}_{i=1}^j \in \mathcal{A}$ and $\{A_i\}_{i=1}^j$ intersect with $\{O_i\}_{i=1}^l \in \mathcal{O}$. We prove that, $\forall O \in \{O_i\}_{i=1}^l$, O does not receive any token request during Round 2 of redistribution. Hence, we can think of a token request from R to B as an internal request to O . We describe the third round of redistribution.

Round 3 - Redistribution for root of level 0 and degree 1. Request 1 token from each of O_1, \dots, O_l following Round 2.

By showing $l \geq 3$, we prove that a root of level 0 and degree 1 has 7 tokens after the third round of redistribution.

Therefore, after the three rounds of token redistribution, each optimal set has at least 7 tokens, then the approximation ratio of the Restricted 4-Set Packing algorithm is at least $\frac{7}{16}$. \square

We also give a randomized construction of tight example. We thus conclude that the approximation ratio of the Restricted 4-Set Packing algorithm is $\frac{7}{16}$.

3.3 Analysis of the Restricted k -Set Packing Algorithm, $k \geq 5$

For $k \geq 5$, we prove that the approximation ratio of the Restricted k -Set Packing algorithm is the same as the set packing heuristic [8].

The proof strategy is similar to that of the Restricted 4-Set Packing algorithm. We first create a forest of blocking trees \mathcal{T} . We give every element covered by sets chosen by the algorithm one unit of tokens. We then redistribute the tokens among all the optimal sets.

Round 1 - Redistribution in each blocking tree \mathcal{T} . Every leaf in \mathcal{T} has $k - 3$ units of free tokens to distribute. Every internal node V of degree d receives $d - 2$ units of tokens from a leaf. The root receives the remaining tokens.

We can show that after Round 1, every node has at least 2 units of tokens, except singletons which are 1-level sets. Round 2 is redistribution for those 1-level singletons. The detail is rather complicated which we omit here. After the second round of redistribution, every optimal set gets at least $2 - k\epsilon$ units of tokens.

Moreover, the tight example of the k -set packing heuristic [8] can also serve as a tight example of the Restricted k -Set Packing algorithm for $k \geq 5$. Hence, the approximation ratio of the Restricted k -Set Packing algorithm is $\frac{2}{k} - \epsilon$.

4 Analysis of the Algorithm PRPSLI

We use the factor-revealing linear program introduced by Jain et al. [9] to analyze the approximation ratio of the algorithm PRPSLI. Athanassopoulos et al. [11] first apply this method to the k -Set Cover problem. Notice that the cover produced by the restricted set packing algorithms is a cover which minimizes the number of 1-sets. Before using the factor-revealing linear program, we show that for any $k \geq 4$, there exists a k -set cover which simultaneously minimizes the size of the cover and the number of 1-sets in the cover. Then the proof of Theorem 3 is similar as the proof of Theorem 6 in [11]. Namely, we set up the factor-revealing linear program (LP) for the k -Set Cover problem, we then find a feasible solution to the dual program of (LP), which makes the objective function of the dual program equal to the value of ρ_k defined in Theorem 3, thus ρ_k is an upper bound of the approximation ratio of PRPSLI. On the other side, we give an instance for each k , such that PRPSLI does not achieve a better ratio than ρ_k on this instance. We thus prove Theorem 3.

Acknowledgements. We thank the anonymous reviewers for their helpful comments.

References

1. Athanassopoulos, S., Caragiannis, I., Kaklamanis, C.: Analysis of approximation algorithms for k -set cover using factor-revealing linear programs. *Theory of Computing Systems* 45(3), 555–576 (2009)
2. Duh, R., Fürer, M.: Approximation of k -set cover by semi-local optimization. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 256–264 (1997)

3. Feige, U.: A threshold of $\ln n$ for approximating set cover. *Journal of ACM* 45(4), 634–652 (1998)
4. Fürer, M., Yu, H.: Packing-based approximating algorithm for the k -set cover problem, <http://arxiv.org/abs/1109.3418>
5. Goldschmidt, O., Hochbaum, D.S., Yu, G.: A modified greedy heuristic for the set covering problem with improved worst case bound. *Information Processing Letters* 48, 305–310 (1993)
6. Halldórsson, M.M.: Approximating k -Set Cover and Complementary Graph Coloring. In: Cunningham, W.H., Queyranne, M., McCormick, S.T. (eds.) IPCO 1996. LNCS, vol. 1084, pp. 118–131. Springer, Heidelberg (1996)
7. Hazan, E., Safra, S., Schwartz, O.: On the complexity of approximating k -set packing. *Computational Complexity* 15, 20–39 (2006)
8. Hurkens, C.A., Shrijver, J.: On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal of Discrete Math.* 2(1), 68–72 (1989)
9. Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of ACM* 50(6), 795–824 (2003)
10. Johnson, D.S.: Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences* 9, 256–278 (1974)
11. Levin, A.: Approximating the unweighted k -set cover problem: greedy meets local search. *SIAM J. Discrete Math.* 23(1), 251–264 (2008)
12. Trevisan, L.: Non-approximability results for optimization problems on bounded degree instances. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, pp. 453–461 (2001)

Capacitated Domination: Constant Factor Approximations for Planar Graphs*

Mong-Jen Kao** and D.T. Lee***

Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
d97021@csie.ntu.edu.tw, dtlee@nchu.edu.tw

Abstract. We consider the capacitated domination problem, which models a service-requirement assigning scenario and which is also a generalization of the dominating set problem. In this problem, we are given a graph with three parameters defined on the vertex set, which are cost, capacity, and demand. The objective of this problem is to compute a demand assignment of least cost, such that the demand of each vertex is fully-assigned to some of its closed neighbours without exceeding the amount of capacity they provide. In this paper, we provide the first constant factor approximation for this problem on planar graphs, based on a new perspective on the hierarchical structure of outer-planar graphs. We believe that this new perspective and technique can be applied to other capacitated covering problems to help tackle vertices of large degrees.

1 Introduction

For decades, *Dominating Set* problem has been one of the most fundamental and well-known problems in both graph theory and combinatorial optimization. Given a graph $G = (V, E)$ and an integer k , *Dominating Set* asks for a subset $D \subseteq V$ whose cardinality does not exceed k such that every vertex in the graph either belongs to this set or has a neighbour which does. As this problem is known to be NP-hard, approximation algorithms have been proposed in the literature [11, 10, 11].

A series of study on capacitated covering problem was initiated by Guha et al., [9], which addressed the capacitated vertex cover problem from a scenario of Glycomolecule ID (GMID) placement. Several follow-up papers have appeared since then, studying both this topic and related variations [4, 7, 8]. These problems are also closely related to work on the capacitated facility location problem, which has drawn a lot of attention since 1990s. See [3, 16].

* Supported in part by the National Science Council, Taiwan, under Grants NSC99-2911-I-002-055-2, NSC98-2221-E-001-007-MY3, and NSC98-2221-E-001-008-MY3.

** This work was done when the author was with Karlsruhe Institute of Technology (KIT), Germany, as a visiting scholar under the NSC-DAAD-sponsored sandwich program (grant number NSC99-2911-I-002-055-2).

*** The author's present address is Department of Computer Science and Engineering, National Chung-Hsing University, Taichung, Taiwan.

Motivated by a general service-requirement assignment scenario, Kao et al., [12][14] considered a generalization of the dominating set problem called *Capacitated Domination*, which is defined as follows. Let $G = (V, E)$ be a graph with three non-negative parameters defined on each vertex $u \in V$, referred to as the cost, the capacity, and the demand, further denoted by $w(u)$, $c(u)$, and $d(u)$, respectively. The demand of a vertex stands for the amount of service it requires from its adjacent vertices, including the vertex itself, while the capacity of a vertex represents the amount of service each multiplicity (copy) of that vertex can provide.

By a demand assignment function f we mean a function which maps pairs of vertices to non-negative real numbers. Intuitively, $f(u, v)$ denotes the amount of demand of u that is assigned to v . We use $N_G(v)$ to denote the set of neighbours of a vertex $v \in V$.

Definition 1 (feasible demand assignment function). A demand assignment function f is said to be feasible if $\sum_{u \in N_G[v]} f(v, u) \geq d(v)$, for each $v \in V$, where $N_G[v] = N_G(v) \cup \{v\}$ denotes the neighbours of v unions v itself.

Given a demand assignment function f , the corresponding capacitated dominating multi-set $\mathcal{D}(f)$ is defined as follows. For each vertex $v \in V$, the multiplicity of v in $\mathcal{D}(f)$ is defined to be $x_f(v) = \left\lceil \frac{\sum_{u \in N_G[v]} f(u, v)}{c(v)} \right\rceil$. The cost of the assignment function f , denoted $w(f)$, is defined to be $w(f) = \sum_{u \in V} w(u) \cdot x_f(u)$.

Definition 2 (Capacitated Domination Problem). Given a graph $G = (V, E)$ with cost, capacity, and demand defined on each vertex, the capacitated domination problem asks for a feasible demand assignment function f such that $w(f)$ is minimized.

For this problem, Kao et al., [14], presented a $(\Delta + 1)$ -approximation for general graphs, where Δ is the maximum vertex degree of the graph, and a polynomial time approximation scheme for trees, which they proved to be NP-hard. In a following work [12], they provided more approximation algorithms and complexity results for this problem. On the other hand, Dom et al., [6] considered a variation of this problem where the number of multiplicities available at each vertex is limited and proved the $W[1]$ -hardness when parameterized by treewidth and solution size. Cygan et al., [5], made an attempt toward the exact solution and presented an $O(1.89^n)$ algorithm when each vertex has unit demand. This result was further improved by Liedloff et al., [15].

Our Contributions. We provide the first constant factor approximation algorithms for the capacitated domination problem on planar graphs. This result can be considered a break-through with respect to the pseudo-polynomial time approximations given in [12], which is based on a dynamic programming on graphs of bounded treewidth. The approach used in [12] stems from the fact that vertices of large degrees will fail most of the techniques that transform a pseudo-polynomial time dynamic programming algorithm into approximations, i.e., the error accumulated at vertices of large degrees could not be bounded.

In this work, we tackle this problem using a new approach. Specifically, we give a new perspective toward the hierarchical structure of outer-planar graphs, which enables us to further tackle vertices of large degrees. Then we analyse both the primal and the dual linear programs of this problem to obtain the claimed result. We believe that the approach we provided in this paper can be applied to other capacitated covering problems to help tackle vertices of large degrees as well. Due to the space limit, the proofs as well as certain technical details are omitted. Please refer to our full article [13] for further reference.

2 Preliminary

We assume that all the graphs considered in this paper are simple and undirected. Let $G = (V, E)$ be a graph. We denote the number of vertices, $|V|$, by n . The set of neighbors of a vertex $v \in V$ is denoted by $N_G(v) = \{u : (u, v) \in E\}$. The closed neighborhood of $v \in V$ is denoted by $N_G[v] = N_G(v) \cup \{v\}$. We use $\deg_G(v)$ and $\deg_G[v]$ to denote the cardinality of $N_G(v)$ and $N_G[v]$, respectively. The subscript G in $N_G[v]$ and $\deg_G[v]$ will be omitted when there is no confusion.

A planar embedding of a graph G is a drawing of G in the plane such that the edges intersect only at their endpoints. A graph is said to be planar if it has a planar embedding. An outer-planar graph is a graph which adopts a planar embedding such that all the vertices lie on a fixed circle, and all the edges are straight lines drawn inside the circle. For $k \geq 1$, k -outerplanar graphs are defined as follows. A graph is 1-outerplanar if and only if it is outer-planar. For $k > 1$, a graph is called k -outerplanar if it has a planar embedding such that the removal of the vertices on the unbounded face results in a $(k - 1)$ -outerplanar graph.

An integer linear program (ILP) for capacitated domination is given in (II). The first inequality ensures the feasibility of the demand assignment function f required in Definition II. In the second inequality, we model the multiplicity function x as defined. The third constraint, $d(v)x(u) - f(v, u) \geq 0$, which seems unnecessary in the problem formulation, is required to bound the integrality gap between the optimal solution of this ILP and that of its relaxation. To see that this additional constraint does not alter the optimality of any optimal solution, we have the following lemma.

Lemma 1. *Let f be an arbitrary optimal demand assignment function. We have $d(v) \cdot x_f(u) - f(v, u) \geq 0$ for all $u \in V$ and $v \in N[u]$.*

However, without this constraint, the integrality gap can be arbitrarily large. This is illustrated by the following example. Let $\alpha > 1$ be an arbitrary constant, and $\mathcal{T}(\alpha)$ be an n -vertex star, where each vertex has unit demand and unit cost.

$$\begin{aligned} & \text{Minimize} \quad \sum_{u \in V} w(u)x(u) \quad (1) \\ & \text{subject to} \\ & \sum_{v \in N[u]} f(u, v) - d(u) \geq 0, \quad u \in V \\ & c(u)x(u) - \sum_{v \in N[u]} f(v, u) \geq 0, \quad u \in V \\ & d(v)x(u) - f(v, u) \geq 0, \quad v \in N[u], u \in V \\ & f(u, v) \geq 0, \quad x(u) \in \mathbb{Z}^+ \cup \{0\}, \quad u, v \in V \end{aligned}$$

The capacity of the central vertex is set to be n , which is sufficient to cover the demand of the entire graph, while the capacity of each of remaining $n - 1$ petal vertices is set to be αn .

Lemma 2. *Without the additional constraint $d(v)x(u) - f(v,u) \geq 0$, the integrality gap of the ILP (II) on $\mathcal{T}(\alpha)$ is α , where $\alpha > 1$ is an arbitrary constant.*

Indeed, with the additional constraint applied, we can refrain from unreasonably assigning a small amount of demand to any vertex in any fractional solution. Take a petal vertex, say v , from $\mathcal{T}(\alpha)$ as example, given that $d(v) = 1$ and $f(v,v) = 1$, this constraint would force $x(v)$ to be at least 1, which prevents the aforementioned situation from being optimal.

For the rest of this paper, for any graph G , we denote the optimal values to the integer linear program (II) and to its relaxation by $OPT(G)$ and $OPT_f(G)$, respectively. Note that $OPT_f(G) \leq OPT(G)$.

3 Constant Approximation for Outer-Planar Graphs

Without loss of generality, we assume that the graphs are connected. Otherwise we simply apply the algorithm to each of the connected component separately. In the following, we first classify the outer-planar graphs into a class of graphs called general-ladders and show how the corresponding general-ladder representation can be extracted in $O(n \log^3 n)$ time in §3.1. Then we consider in §3.2 and §3.3 both the primal and the dual programs of the relaxation of (II) to further reduce a given general-ladder and obtain a constant factor approximation. We analyse the algorithm in §3.4 and extend our result to planar graphs in §3.5.

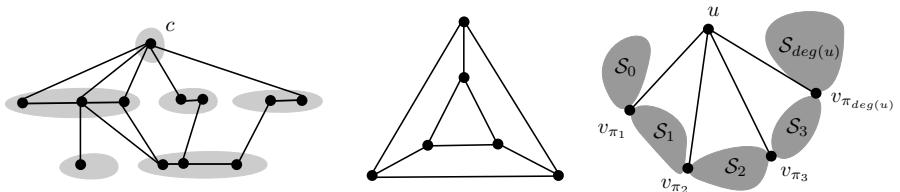


Fig. 1. (a) A general-ladder with anchor c . (b) A 2-outerplanar graph which fails to be a general-ladder. (c) The subdivision formed by a vertex u in an outer-planar embedding.

3.1 The Structure

First we define the notation which we will use later on. By a total order of a set we mean that each pair of elements in the set can be compared, and therefore an ascending order of the elements is well-defined. Let $P = (v_1, v_2, \dots, v_k)$ be a path. We say that P is an *ordered path* if a total order $v_1 \prec v_2 \prec \dots \prec v_k$ or $v_k \prec v_{k-1} \prec \dots \prec v_1$ is defined on the set of vertices.

Definition 3 (General-Ladder). *A graph $G = (V, E)$ is said to be a general-ladder if a total order on the set of vertices is defined, and G is composed of*

a set of layers $\{\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_k\}$, where each layer is a collection of subpaths of an ordered path such that the following holds. The top layer, \mathcal{L}_1 , consists of a single vertex, which is referred to as the anchor, and for each $1 < j < k$ and $u, v \in \mathcal{L}_j$, we have (1) $N[u] \subseteq \mathcal{L}_{j-1} \cup \mathcal{L}_j \cup \mathcal{L}_{j+1}$, and (2) $u \prec v$ implies $\max_{p \in N[u] \cap \mathcal{L}_{j+1}} p \preccurlyeq \min_{q \in N[v] \cap \mathcal{L}_{j+1}} q$.

Note that each layer in a general-ladder consists of a set of ordered paths which are possibly connected only to vertices in the neighbouring layers. See Fig. 2(a). Although the definition of general-ladders captures the essence and simplicity of an ordered hierarchical structure, there are planar graphs which fall outside this framework. See also Fig. 2(b).

In the following, we state and argue that every outerplanar graph meets the requirements of a general-ladder. We assume that an outer-planar embedding for any outer-planar graph is given as well. Otherwise we apply the $O(n \log^3 n)$ algorithm provided by Bose [2] to compute such an embedding.

Let $G = (V, E)$ be an outer-planar graph, $u \in V$ be an arbitrary vertex, and \mathcal{E} be an outer-planar embedding of G . We fix u to be the smallest element and define a total order on the vertices of G according to their orders of appearances on the outer face of \mathcal{E} in a counter-clockwise order. For convenience, we label the vertices such that $u = v_1$ and $v_1 \prec v_2 \prec v_3 \prec \dots \prec v_n$.

Let $N(u) = \{v_{\pi_1}, v_{\pi_2}, \dots, v_{\pi_{deg(u)}}\}$ denote the neighbours of u such that $v_{\pi_1} \prec v_{\pi_2} \prec \dots \prec v_{\pi_{deg(u)}}$. $N(u)$ divides the set of vertices except u into $deg(u) + 1$ subsets, namely, $\mathcal{S}_0 = \{v_2, v_3, \dots, v_{\pi_1}\}$, $\mathcal{S}_i = \{v_{\pi_i}, v_{\pi_i+1}, \dots, v_{\pi_{i+1}}\}$ for $1 \leq i < deg(u)$, and $\mathcal{S}_{deg(u)} = \{v_{\pi_{deg(u)}}, v_{\pi_{deg(u)}+1}, \dots, v_n\}$. See Fig. 2(c) for an illustration. For $1 \leq i < deg(u)$, we partition \mathcal{S}_i into two sets L_i and R_i as follows. Let $d_{\mathcal{S}_i}$ denote the distance function defined on the induced subgraph of \mathcal{S}_i . Let $L_i = \{v : v \in \mathcal{S}_i, d_{\mathcal{S}_i}(v_{\pi_i}, v) \leq d_{\mathcal{S}_i}(v, v_{\pi_{i+1}})\}$ and $R_i = \mathcal{S}_i \setminus L_i$.

Denote $\ell(v) \equiv d_G(u, v)$. Now consider the set of the edges connecting L_i and R_i . Note that, this is exactly the set of edges connecting vertices on the shortest path between v_{π_i} and $\max_{a \in L_i} a$ and vertices on the shortest path between $v_{\pi_{i+1}}$ and $\min_{b \in R_i} b$. See also Fig. 2. Below we present our structural lemma, which states that, when the vertices are classified by their distances to u , these edges can only connect vertices between neighbouring sets and do not form any crossing.

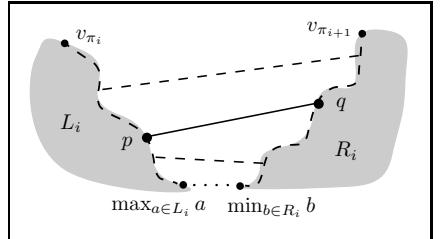


Fig. 2. Partition of \mathcal{S}_i into L_i and R_i

Lemma 3. Any outer-planar graph $G = (V, E)$ together with an arbitrary vertex $u \in V$ is a general-ladder anchored at u , where the set of vertices in each layer are classified by their distances to the anchor u .

Extracting the General-ladder. Let $\mathcal{G} = (V, E)$ be the input outer-planar graph and $u \in V$ be an arbitrary vertex. We can extract the corresponding general-ladder as stated below.

Theorem 1. *Given an outer-planar graph \mathcal{G} and its outer-planar embedding, we can compute in linear time a general-ladder representation for \mathcal{G} .*

For the rest of this paper we will denote the layers of this particular general-ladder representation by $\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_M$. The following additional structural property comes from the outer-planarity of \mathcal{G} and our construction scheme.

Lemma 4. *For any $0 < i \leq M$ and $v \in \mathcal{L}_i$, we have $|N(v) \cap \mathcal{L}_{i-1}| \leq 2$. Moreover, if v has two neighbours in \mathcal{L}_i , say, v_1 and v_2 with $v_1 \prec v \prec v_2$, then there is an edge joining v_1 (and v_2 , respectively) and each neighbouring vertex of v in \mathcal{L}_{i-1} that is smaller (larger) than v .*

The Decomposition. The idea behind this decomposition is to help reduce the dependency between vertices of large degrees and their neighbours such that further techniques can be applied. To this end, we tackle the demands of vertices from every three layers separately.

For each $0 \leq i < 3$, let $\mathcal{R}_i = \bigcup_{j \geq 0} \mathcal{L}_{3j+i}$. Let $\mathcal{G}_i = (V_i, E_i)$ consist of the induced subgraph of \mathcal{R}_i and the set of edges connecting vertices in \mathcal{R}_i to their neighbours. Formally, $V_i = \bigcup_{v \in \mathcal{R}_i} N[v]$ and $E_i = \bigcup_{v \in \mathcal{R}_i} \bigcup_{u \in N[v]} e(u, v)$. In addition, we set $d(v) = 0$ for all $v \in \mathcal{G}_i \setminus \mathcal{R}_i$. Other parameters remain unchanged.

Lemma 5. *Let f_i , $0 \leq i < 3$, be an optimal demand assignment function for \mathcal{G}_i . The assignment function $f = \sum_{0 \leq i < 3} f_i$ is a 3-approximation of \mathcal{G} .*

3.2 Removing More Edges

We describe an approach to further simplifying the graphs \mathcal{G}_i , for $0 \leq i < 3$. Given any feasible demand assignment for \mathcal{G}_i , we can properly reassign the demand of a vertex to a constant number of neighbours while the increase in terms of fractional cost remains bounded.

For each $v \in \mathcal{R}_i$, we sort the closed neighbours of v according to their cost in ascending order such that $w(\pi_v(1)) \leq w(\pi_v(2)) \leq \dots \leq w(\pi_v(\deg[v]))$, where $\pi_v : \{1, 2, \dots, \deg[v]\} \rightarrow N[v]$ is an injective function. For convenience, we set $\pi_v(\deg[v] + 1) = \phi$. Suppose that $v \in \mathcal{L}_\ell$. We identify the following four vertices.

- Let j_v , $1 \leq j_v \leq \deg[v]$, be the smallest integer such that $c(\pi_v(j_v)) > d(v)$. If $c(\pi_v(j_v)) \leq d(v)$ for all $1 \leq j \leq \deg[v]$, then we let $j_v = \deg[v] + 1$.
- Let k_v , $1 \leq k_v < j_v$, be the integer such that $w(\pi_v(k_v)) / c(\pi_v(k_v))$ is minimized. k_v is defined only when $j_v > 1$.
- Let $p_v = \max_{u \in N[v] \cap \mathcal{L}_{\ell-1}} u$ and
 $q_v = \max_{u \in N[v] \cap \mathcal{L}_{\ell+1}} u$.

Intuitively, $\pi_v(j_v)$ is the first vertex in the sorted list whose capacity is greater than $d(v)$, and $\pi_v(k_v)$ is the vertex with best cost-capacity ratio among the first $j_v - 1$ vertices. p_v and q_v are the rightmost neighbour of v in layer $\mathcal{L}_{\ell-1}$ and $\mathcal{L}_{\ell+1}$, respectively.

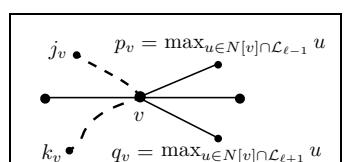


Fig. 3. Incident edges of a vertex $v \in \mathcal{L}_\ell$ to be kept

We will omit the function π_v and use j_v, k_v to denote $\pi_v(j_v), \pi_v(k_v)$ without confusion. The reduced graph \mathcal{H}_i is defined as follows. Denote the set of neighbours to be disconnected from v by $R(v) = N[v] \setminus (\mathcal{L}_\ell \cup \{j_v \cup k_v \cup p_v \cup q_v\})$, and let $\mathcal{H}_i = \mathcal{G}_i \setminus \bigcup_{v \in \mathcal{R}_i} \bigcup_{u \in R(v)} \{e(u, v)\}$. Roughly speaking, in graph \mathcal{H}_i we remove the edges which connect vertices in \mathcal{R}_i , say v , to vertices not in \mathcal{R}_i , except possibly for j_v, k_v, p_v , and q_v . See Fig. B. Note that, although our reassigning argument applies to arbitrary graphs, only when two vertices are unimportant to each other can we remove the edge between them.

Lemma 6. *In the subgraph \mathcal{H}_i , we have*

- For each $v \notin \mathcal{R}_i$, at most one incident edge of v which was previously in \mathcal{G}_i will be removed.
- For each $v \in \mathcal{R}_i$, the degree of v in \mathcal{H}_i is upper-bounded by 6.
- $OPT_f(\mathcal{H}_i) \leq 2 \cdot OPT_f(\mathcal{G}_i)$.

We also remark that, although $OPT_f(\mathcal{H}_i)$ is bounded in terms of $OPT_f(\mathcal{G}_i)$, an α -approximation for \mathcal{H}_i is not necessarily a 2α -approximation for \mathcal{G}_i . That is, having an approximation \mathcal{A} with $OPT(\mathcal{A}) \leq \alpha \cdot OPT(\mathcal{H}_i)$ does not imply that $OPT(\mathcal{A}) \leq 2\alpha \cdot OPT(\mathcal{G}_i)$, for $OPT(\mathcal{H}_i)$ could be strictly larger than $OPT_f(\mathcal{H}_i)$. Instead, to obtain our claimed result, an approximation with a stronger bound, in terms of $OPT_f(\mathcal{H}_i)$, is desired.

3.3 Greedy Charging Scheme

We show how we can further approximate the optimal solution for the reduced graph \mathcal{H}_i by a primal-dual charging argument. We apply a technique from [4] to obtain a feasible solution for the dual program of the relaxation of (I), which is given in (2). By Lemma 4, we can further tighten the approximation ratio.

We first describe an approach to obtaining a feasible solution to (2) and how a corresponding feasible demand assignment can be found. Note that any feasible solution to (2) will serve as a lower bound to any feasible solution of (I) by the linear program duality.

During the process, we will maintain a vertex subset, V^ϕ , which contains the set of vertices with non-zero unassigned demand. For each $u \in V$, let $d^\phi(u) = \sum_{v \in N[u] \cap V^\phi} d(v)$ denote the amount of unassigned demand from the closed neighbours of u . We distinguish between two cases. If $c(u) < d^\phi(u)$, then we say that u is heavily-loaded. Otherwise, u is lightly-loaded. During the process, some heavily-loaded vertices might turn into lightly-loaded due to the demand assignments of its closed neighbours. For each of these vertices, say v , we will maintain a vertex subset $D^*(v)$, which contains the set of unassigned vertices in $N[v] \cap V^\phi$ when v is about to fall into lightly-loaded. For other vertices, $D^*(v)$ is defined to be an empty set.

$\begin{aligned} & \text{Maximize} && \sum_{u \in V} d(u)y_u \\ & \text{subject to} && \\ & c(u)z_u + \sum_{v \in N[u]} d(v)g_{u,v} \leq w(u), && u \in V \\ & y_u \leq z_v + g_{v,u}, && v \in N[u], \quad u \in V \\ & y_u \geq 0, \quad z_u \geq 0, \quad g_{v,u} \geq 0, && v \in N[u], \quad u \in V \end{aligned} \tag{2}$
--

Initially, $V^\phi \equiv \{u : u \in \mathcal{L}_i, d(u) \neq 0\}$ and all the dual variables are set to be zero. We increase the dual variable y_u simultaneously, for each $u \in V^\phi$. To maintain the dual feasibility, as we increase y_u , we have to raise either z_v or $g_{v,u}$, for each $v \in N[u]$. If v is heavily-loaded, then we raise z_v . Otherwise, we raise $g_{v,u}$. Note that, during this process, for each vertex u that has a closed neighbour in V^ϕ , the left-hand side of the inequality $c(u)z_u + \sum_{v \in N[u]} d(v)g_{u,v} \leq w(u)$ is constantly raising. As soon as one of the inequalities $c(u)z_u + \sum_{v \in N[u]} d(v)g_{u,v} \leq w(u)$ is met with equality (saturated) for some vertex $u \in V$, we perform the following operations.

If u is lightly-loaded, we assign all the unassigned demand from $N[u] \cap V^\phi$ to u . In this case, there are still $c(u) - d^\phi(u)$ units of capacity free at u . We assign the unassigned demand from $D^*(u)$, if there is any, to u until either all the demand from $D^*(u)$ is assigned or all the free capacity in u is used. On the other hand, if u is heavily-loaded, we mark it as heavy and delay the demand assignment from its closed neighbours.

Then we set $\mathcal{Q}_u \equiv N[u] \cap V^\phi$ and remove $N[u]$ from V^ϕ . Note that, due to the definition of d^ϕ , even when u is heavily-loaded, we still update $d^\phi(p)$ for each $p \in V$ with $N[p] \cap N[u] \neq \emptyset$, if needed, as if the demand was assigned. During the above operation, some heavily-loaded vertices might turn into lightly-loaded due to the demand assignments (or simply due to the update of d^ϕ). For each of these vertices, say v , we set $D^*(v) \equiv N[v] \cap (V^\phi \cup \mathcal{Q}_u)$. Intuitively, $D^*(v)$ contains the set of unassigned vertices from $N[v] \cap V^\phi$ when v is about to fall into lightly-loaded.

This process is continued until $V^\phi = \emptyset$. For those vertices which are marked as heavy, we iterate over them according to their chronological order of being saturated and assign at this moment all the remaining unassigned demand from their closed neighbours to them.

Let $f^* : V \times V \rightarrow R^+ \cup \{0\}$ denote the resulting demand assignment function, and $x^* : V \rightarrow Z^+ \cup \{0\}$ denotes the corresponding multiplicity function. The following lemma bounds the cost of the solution produced by our algorithm.

Lemma 7. *For any \mathcal{H}_i obtained from a general-ladder \mathcal{G}_i , we have $w(f^*) \leq 7 \cdot OPT_f(\mathcal{H}_i)$.*

Thanks to the structural property provided in Lemma 4, given the fact that the input graph is outer-planar, we can modify the algorithm slightly and further improve the bound given in the previous lemma. To this end, we consider the situations when a unit demand from a vertex u with $\deg[u] = 7$ and argue that, either it is not fully-charged by all its closed neighbours, or we can modify the demand assignment, without raising the cost, to make it so.

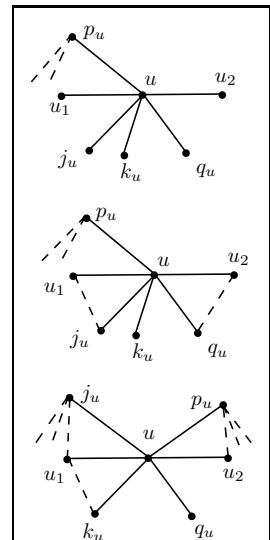


Fig. 4. Situations when a unit demand of u is fully-charged

Lemma 8. *Given the fact that \mathcal{H}_i comes from an outerplanar graph, we can modify the algorithm to obtain a demand assignment function f^* such that $w(f^*) \leq 6 \cdot OPT_f(\mathcal{H}_i)$.*

3.4 Overall Analysis

We summarize the whole algorithm and our main theorem. Given an outer-planar graph $G = (V, E)$, we use the algorithm described in §3.1 to compute a general-ladder representation of G , followed by applying the decomposition to obtain three subproblems, \mathcal{G}_0 , \mathcal{G}_1 , and \mathcal{G}_2 . For each \mathcal{G}_i , we use the approach described in §3.2 to further remove more edges and obtain the reduced subgraph \mathcal{H}_i , for which we apply the algorithm described in §3.3 to obtain an approximation, which is a demand assignment function f_i for \mathcal{H}_i . The overall approximation, e.g., the demand assignment function f , for G is defined as $f = \sum_{0 \leq i < 3} f_i$.

Theorem 2. *Given an outerplanar graph G as an instance of capacitated domination, we can compute a constant factor approximation for G in $O(n^2)$ time.*

3.5 Extension to Planar Graphs

In this section we extend our outer-planar result to a constant factor approximation for planar graphs under a general framework due to [1]. As our algorithm is designed mainly for outerplanar graphs, to meet the minimum requirement of this framework, which is the ability to deal with planar graphs of at least three levels, we have to modify our algorithm to undertake this difference.

In the following, we assume that the input graph, G , is 3-outerplanar and sketch only the key changes we made on our algorithm. Let L_0 , L_1 , and L_2 be the sets of vertices from the three levels of G . In addition, we also have $d(v) = 0$ for each $v \notin L_1$. See Fig. 5 (a).

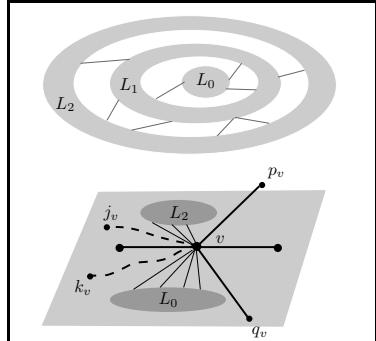


Fig. 5. (a) 3-outerplanar graph. (b) Local connections w.r.t. a vertex v . Bold edges represent links in the ladder extracted from L_1 . Thin edges represent links between L_1 and L_0 , or L_1 and L_2 .

Obtaining the General Ladders. For each level L_i , $0 \leq i < 3$, we define a total order according to the counter-clockwise order of appearances of the vertices. The general-ladder is extracted from L_1 as we did before. Furthermore, for each vertex in the ladder, its incident edges to vertices in L_0 and L_2 are also included.

Removing Redundant Edges. In addition to the four vertices we identified for each vertex v with non-zero demand, we identify two more vertices, which literally corresponds to the rightmost neighbours of v in levels L_0 and L_2 , respectively. See also Fig. 5 (b).

Theorem 3. *Given a planar graph G as an instance of capacitated domination, we can compute a constant factor approximation for the G in polynomial time.*

4 Conclusion

Due to the flexibility of the ways the demand can be assigned, the results we provided here seem to have room for further improvements. However, when the demand cannot be split, it is not difficult to prove a constant approximation threshold. Therefore, it would be very interesting to investigate the problem complexity on planar graphs.

Second, as we have shown in §3.1, the concept of general-ladders does not extend directly to k -outerplanar graphs for $k \geq 2$. It would be interesting to formalize and extend this concept to k -outerplanar graphs, for it seems helpful not only to our problem, but also to most capacitated covering problems as well.

Acknowledgements. The author would like to thank the anonymous referees for their very helpful comments on the layout of this work.

References

1. Baker, B.S.: Approximation algorithms for np-complete problems on planar graphs. *J. ACM* 41, 153–180 (1994)
2. Bose, P.: On embedding an outer-planar graph in a point set. *CGTA: Computational Geometry: Theory and Applications* 23, 2002 (1997)
3. Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. *Math. Program.* 102, 207–222 (2005)
4. Chuzhoy, J.: Covering problems with hard capacities. *SIAM J. Comput.* 36, 498–515 (2006)
5. Cygan, M., Pilipczuk, M., Wojtaszczyk, J.O.: Capacitated Domination Faster Than $O(2^n)$. In: Kaplan, H. (ed.) *SWAT 2010*. LNCS, vol. 6139, pp. 74–80. Springer, Heidelberg (2010), doi:10.1007/978-3-642-13731-0_8
6. Dom, M., Lokshtanov, D., Saurabh, S., Villanger, Y.: Capacitated Domination and Covering: A Parameterized Perspective. In: Grohe, M., Niedermeier, R. (eds.) *IWPEC 2008*. LNCS, vol. 5018, pp. 78–90. Springer, Heidelberg (2008)
7. Gandhi, R., Halperin, E., Khuller, S., Kortsarz, G., Srinivasan, A.: An improved approximation algorithm for vertex cover with hard capacities. *J. Comput. Syst. Sci.* 72, 16–33 (2006)
8. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding in bipartite graphs. In: *FOCS 2002*, pp. 323–332 (2002)
9. Guha, S., Hassin, R., Khuller, S., Or, E.: Capacitated vertex covering. *J. Algorithms* 48(1), 257–270 (2003)
10. Hochbaum, D.: Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing* 11, 555–556 (1982)
11. Johnson, D.S.: Approximation algorithms for combinatorial problems. In: *The 5th Annual ACM Symposium on Theory of Computing*, pp. 38–49 (1973)
12. Kao, M.-J., Chen, H.-L.: Approximation Algorithms for the Capacitated Domination Problem. In: Lee, D.-T., Chen, D.Z., Ying, S. (eds.) *FAW 2010*. LNCS, vol. 6213, pp. 185–196. Springer, Heidelberg (2010)
13. Kao, M.-J., Lee, D.: Capacitated domination: Constant factor approximation for planar graphs (manuscript, 2011), <http://arxiv.org/abs/1108.4606>
14. Kao, M.-J., Liao, C.-S., Lee, D.T.: Capacitated domination problem. *Algorithmica* 60, 274–300 (2011), doi:10.1007/s00453-009-9336-x
15. Liedloff, M., Todinca, I., Villanger, Y.: Solving Capacitated Dominating Set by Using Covering by Subsets and Maximum Matching. In: Thilikos, D.M. (ed.) *WG 2010*. LNCS, vol. 6410, pp. 88–99. Springer, Heidelberg (2010)
16. Shmoys, D.B., Tardos, E., Aardal, K.: Approximation algorithms for facility location problems (extended abstract). In: *STOC 1997*, pp. 265–274 (1997)

On Power-Law Distributed Balls in Bins and Its Applications to View Size Estimation

Ioannis Atsonios, Olivier Beaumont, Nicolas Hanusse, and Yusik Kim

CNRS and INRIA Bordeaux – Sud-Ouest, University of Bordeaux, LaBRI, France

Abstract. The view size estimation plays an important role in query optimization. It has been observed that many data follow a power law distribution. In this paper, we consider the balls in bins problem where we place balls into N bins when the bin selection probabilities follow a power law distribution. As a generalization to the coupon collector’s problem, we address the problem of determining the expected number of balls that need to be thrown in order to have at least one ball in each of the N bins. We prove that $\Theta(\frac{N^\alpha \ln N}{c_N^\alpha})$ balls are needed to achieve this where α is the parameter of the power law distribution and $c_N^\alpha = \frac{\alpha-1}{\alpha-N^{\alpha-1}}$ for $\alpha \neq 1$ and $c_N^\alpha = \frac{1}{\ln N}$ for $\alpha = 1$. Next, when fixing the number of balls that are thrown to T , we provide closed form upper and lower bounds on the expected number of bins that have at least one occupant. For n large and $\alpha > 1$, we prove that our bounds are tight up to a constant factor of $\left(\frac{\alpha}{\alpha-1}\right)^{1-\frac{1}{\alpha}} \leq e^{1/e} \simeq 1.4$.

1 Introduction

1.1 Context

Datacubes and Query Optimizations. Query optimization can be decomposed into several steps. One of the most important deals with the estimation of the memory and time requirements of some possible sequences of operations in order to choose the cheapest. In the particular case of OLAP queries, the use of a data structure called a *datacube* [GBLP96] allows to speed up the queries. It consists of the storage of some *views* corresponding to intermediate results. Usually, each materialized view is a cuboid, that is the set of aggregative values populating a fact table for a given combination of attributes. View selection algorithms rely on the fast estimation of the cuboids.

The simplest way to estimate a view size is to scan the whole dataset. In practice, scanning once a dataset of a few millions of entries can take 1 minute. For a d -dimensional data set of large size, the computation of the exact size of the 2^d views can take up to a few days. Whenever data are dynamic, this is not realistic. A quick but naive way to estimate the size of a view v is to use Cardenas formula [Car75],

$$\mathbf{E}[\text{viewsize}] = m(v) \left(1 - \left(1 - \frac{1}{m(v)} \right)^T \right). \quad (1)$$

where $m(v)$ denotes the number of all possible value combinations of view v , and $T = |\mathcal{T}|$, the number of entries in fact table \mathcal{T} . This formula holds true if the entries of \mathcal{T} are chosen uniformly at random from the set of all possible attribute combinations. However, it turns out that this estimate leads to overestimate the real size as soon as the uniformity assumption does not hold. From this observation, several more sophisticated approaches have been proposed (see [AL07] for an experimental comparison). For instance, it has been proposed in [CPKH05, DF03, GT01, FM85] (for a data stream setting [KNW10]) to scan \mathcal{T} but with a light consumption of memory, without any assumption on the distribution of data. These algorithms are based on independent and uniform hashing and provide a theoretical accuracy of $1 + \Theta(1/\sqrt{M})$. However, in practice, getting independent and uniform hashing is not realistic. In [AL07], it is shown that using only few kbytes is enough to get a good accuracy. When \mathcal{T} can be partitioned, distributed computation of such estimates has been proposed in [BGH⁺09], but the time complexity remains proportional to T . Nevertheless, whenever the number of view size requests becomes too large, this approach cannot be considered.

A second approach is based on sampling. In this context, the fact table is sampled and the skew of the data distribution is approximated by a statistical model. It turns out that without any assumption on the distribution of data, it is impossible to get an estimate with a good accuracy using a small sample size [CCMN00]. For instance, Faloutsos *et al.* [FMS96] choose a multifractal model based only on the knowledge of the number of occurrences of the most frequent tuples in the sample \mathcal{T}' and $|\mathcal{T}'|$. Haas *et al.* [HNSS95] propose an estimator based on the distribution of data in the sample. Some work [NT03, AL07] report the relevance of the multifractal model with respect to the accuracy but there is no theoretical guarantee on the accuracy related to the size of the sample. We point out that in [NT03], an estimator dedicated to Pareto distribution (similar to power law for $\alpha > 1$) is proposed. In this approach, time and memory space are proportional to $|\mathcal{T}'|$. Motwani and Vassilvitskii [MV06] propose a sampling based method under the power law assumption that provides near accurate results with positive probability. More precisely, denoting by $\mathbf{F}_\alpha(T, n)$ the expected number of filled bins after T trials over n bins, and assuming that the exponent of the power law distribution α is known, they propose an algorithm providing an estimator of $\mathbf{F}_\alpha(T, n)$ with an accuracy of $(1 + \epsilon)$ with probability $2 \exp(-\epsilon^2 \mathbf{F}_\alpha(T, n))$, using a sample of size $\Theta((1 + \epsilon)^{1+\alpha} \mathbf{F}_\alpha(T, n) c_n^\alpha)$ with c_n^α being a normalizing factor.

Boneh and Hofri [AM97] provide the distribution of the number of filled bins for general bin selection distributions and derive the expected value (See Equation 3), which, in this form, is intractable to compute due to $|\mathcal{T}|$ being excessively large. To summarize, to our knowledge, no existing estimate provides at low computational cost an accurate estimate of $\mathbf{F}_\alpha(T, n)$ for power law distributed data, even if power law parameter α is given.

1.2 Model

We abstract the problem of determining the view size as a *balls in bins* problem. The setting of the balls in bins problem is that there are some empty bins where balls are thrown into. Balls decide, independently of each other, which bin to fall into according to some given probability distribution over the bins. Among the many variants of problems arising from this setting, we are particularly interested in the following two questions. Given a fixed number of bins and bin selection probabilities, what is the expected number of balls that need to be thrown in order to have at least 1 ball in all bins? (equivalent to the coupon collector's problem) *and* Given a fixed number of bins, a fixed number of balls, and bin selection probabilities, what is the expected number of bins that have at least 1 ball in it?

The relation between the view size of a fact table and the balls in bins problem is based on the following analogy. A **bin** corresponds to a possible combination of attribute values, i.e., a single cell of the cuboid and a **ball** corresponds to a single row of the fact table that contributes to the count of exactly one cell of the cuboid. Thus, the number of “occupied” bins corresponds to the view size. From a theoretical point of view, once the bin selection probabilities $P = \{p_1, p_2, \dots, p_n\}$ are given, estimating a view size from a fact table of T entries can be modeled by a generalized version of balls and bins problem. At each time step t going from 1 to T , a ball is thrown in one of the bins, and bin i is chosen with probability p_i which follows a power law, *i.e.* $p_i = \frac{c_n^\alpha}{i^\alpha}$ where $\alpha \geq 0$ and c_n^α is the normalizing factor of the form $c_n^\alpha = [\sum_{i=1}^n \frac{1}{i^\alpha}]^{-1}$.

We focus on $\mathbf{F}_P(T, n)$, the expected number of filled bins after T trials, corresponding to the number of different values of a combination of attributes. For convenience, whenever P follows a power law distribution of parameter α , we use the notation $\mathbf{F}_\alpha(T, n)$. $\mathbf{F}_0(T, n)$ is a very well known uniform and $\mathbf{F}_P(T, n)$ is given by the Cardenas formula [Car75]. From the coupon collector problem [MR95], we know that whenever $T < n \ln n$, $\mathbf{F}_0(T, n) < n$ and if $T = n \ln n$, $\mathbf{F}_0(T, n) = n$ with constant probability. For $\alpha = 1$, T needs to be $n \ln_2 n$ in order to get $\mathbf{F}_1(T, n)$ with constant probability [BP96]. To the best of our knowledge, for $\alpha \notin \{0, 1\}$, no other closed formula or bounds on $F_\alpha(T, n)$ are known.

1.3 Our Contribution

Expected time to fill all the bins. The coupon collector is a well-known problem related to estimating the number of balls required to fill at least once every bin. In our setting and assuming that the frequencies of tuples follow a power law of parameter α , we raise the question on computing the expected number of entries $\mathbf{E}[T]$ required in order to make a view v saturated, that is of size $n = m(v)$. As an easily obtainable upper bound of $\mathbf{E}[T]$, we have $O(n^{\alpha+1} \log n)$. This can be derived by making bin i reject a ball with probability $(p_i - p_n)/p_i$. In this paper, we prove that (Theorem II)

$$\frac{(n - \sqrt{n})^\alpha}{c_n^\alpha} \ln(\sqrt{n} + 1) \leq \mathbf{E}[T] \leq \frac{n^\alpha}{c_n^\alpha} (1 + \ln n)$$

$$\text{and } \lim_{n \rightarrow \infty} \frac{\mathbf{E}_n[T]}{n^\alpha(1 + \ln n)/c_n^\alpha} = 1,$$

where $\mathbf{E}_n[T]$ denotes the expected time to fill all n bins (Theorem 2).

Expected number of bins filled when throwing T balls We provide bounds on $\mathbf{F}_\alpha(T, n)$. For upper bounds, we prove that

- When $\alpha \geq 0, \alpha \neq 1$, then $\mathbf{F}_\alpha(T, n) \leq \frac{\alpha}{\alpha-1}(c_{n,\alpha}T)^{1/\alpha} - \frac{c_{n,\alpha}T}{\alpha-1}n^{1-\alpha}$ (Th. 6)
- When $\alpha = 1$, then $\mathbf{F}_\alpha(T, n) \leq \frac{T}{\ln n + \gamma} - 1 + T \left(1 - \frac{\ln T - \ln(\ln n + \gamma) + \gamma}{\ln n + \gamma}\right)$ (Th. 6), where $\gamma \simeq 0.5772$ is the Euler-Mascheroni constant.

For lower bounds, we have results for the general case $n \geq 1, \alpha \geq 0$ and tighter bounds when n is sufficiently large

- When $n \geq 1$ and $\alpha \geq 0$, then $\mathbf{F}_\alpha(T, n) \geq \left(1 - \left(1 - \frac{1}{T}\right)^T\right)(c_n^\alpha T)^{1/\alpha}$ (Th. 5)
- When n is large enough and $\alpha > 1$, then $\mathbf{F}_\alpha(T, n) \geq (T+1)^{1/\alpha} - 1$ (Th. 3)
- For $n \geq 1$ and $0 \leq \alpha < 1$, $\mathbf{F}_\alpha(T, n) \geq (n+1) \left(1 - e^{-\frac{(1-\alpha)T}{n+1} - \ln \frac{n+1}{n}}\right)$ (Th. 4)

At last, when n and $\alpha > 1$, we prove that the ratio between the upper and lower bounds is bounded by $e^{1/e}$, i.e., $UB/LB \leq e^{1/e} \simeq 1.4447$ (Corollary 2).

2 Bounds for the Expected Number of Balls Needed to Fill All Bins

Consider having n bins where we throw balls into. Assume that bin i has a probability p_i of being chosen to put a ball into in each trial. We are interested in finding the expected number of balls (or equivalently, trials), which we denote by T , necessary to fill each of the n bins at least once when the probabilities p_i follow a power law distribution.

The classical coupon collector's problem is the special case of this problem when the probabilities p_i follow a uniform distribution instead of the power law. For this special case, the expected number of balls necessary to fill all bins is given by nH_n , where H_n is the harmonic number defined as $H_n = \sum_{k=1}^n \frac{1}{k}$. The simple form for the solution of the classical problem benefits from the simple state space only requiring the number of currently occupied bins, whereas in the general case (without the uniformity assumption), it is necessary to keep track of the specific combination of bins that are currently occupied when doing the calculation. For large n , this calculation is intractable, and therefore, we focus on finding good bounds for T .

2.1 Stochastic Majorization Scheme for Finding Bounds

The main idea of our method to find upper and lower bounds for T is to consider models which are different from our original model where it is possible to

“order” the expected number of necessary balls through stochastic majorization arguments. We provide basic definitions and properties of stochastic orders. See [SS94] for details.

Definition 1. A random variable (r.v.) X is greater than a r.v. Y in the stochastic order, written as $X \geq_{st} Y$ if $P(X > x) \geq P(Y > x)$ for all x .

An alternative characterization of stochastic order is that $X \geq_{st} Y$ if and only if there exist r.v.’s X' and Y' having the same distribution as X and Y , respectively and satisfies $P(X' \geq Y') = 1$. Therefore, we use a coupling argument as a method of proof. From the definition, it is straightforward to verify that stochastic order implies ordering of the mean. To establish an upper bound, we need to have a model that is more pessimistic than the original model in occupying an empty bin in each trial. Let us denote the original model by O and let us consider a new model, denoted model U , that consists in $n + 1$ bins where $p_i = p_n$ for $i = 1, 2, \dots, n$ and $p_0 = 1 - np_n$ representing a “trash” bin (See Figure 1 for an illustration). Comparing this with our original model, intuitively, balls that would normally go into a particular empty bin under model O will now have a non zero probability of going into the trash bin under the model U , failing to increment the number of occupied bins. Thus, the overall number of balls required to fill bins 1 through n under the model U will be in some sense larger than under the model O as formally stated in the following lemma.

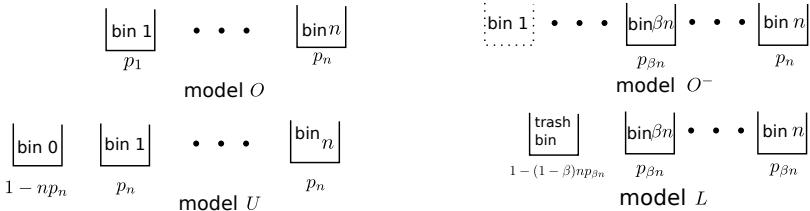


Fig. 1. Illustration of various models used. The probability of selecting a certain bin written below the bins.

Lemma 1. Consider the random experiment of throwing balls independently into n bins with selection probabilities $p_1 \geq \dots \geq p_n$ for the n bins. Call this experiment O and denote the random variable counting the number of balls thrown until there are no longer any empty bins by X . Now consider a random experiment of throwing balls independently into $n + 1$ bins with selection probabilities p_n, \dots, p_n for the first n bins and $1 - np_n$ for the last bin. Call this experiment U and denote the random variable counting the number of balls thrown until there are no longer any empty bins among the first n bins by X' . Then $X' \geq_{st} X$ and consequently, $\mathbf{E}[X'] \geq \mathbf{E}[X]$.

Similarly, to find a lower bound, we need a model that is more optimistic. Consider a subset of the original model, which we will call model O^- that only

includes the last $(1 - \beta)n$ bins, i.e., $i = \beta n, \dots, n$, where $0 \leq \beta \leq 1$. Suppose n is large enough so that the effect of rounding to an integer is negligible. Now consider an alternative model, which we will call model L , with $(1 - \beta)n$ bins each having probability $p_{\beta n}$ of being chosen plus a trash bin having probability $1 - (1 - \beta)np_{\beta n}$ of being chosen. Since $p_{\beta n}$ is the largest choice probability among the bins in model O^- , loosely speaking, it will take less time to fill the $(1 - \beta)n$ bins of model L than the last $(1 - \beta)n$ bins of model O^- . Since filling the last $(1 - \beta)n$ bins is a necessary condition for filling all n bins of the original problem, the number of balls required to fill all bins in model L provides a lower bound to the original problem (see Figure 1 for an illustration).

Lemma 2. *Consider the random experiment of throwing balls independently into $n - m + 1$ bins with selection probabilities $p_m \geq \dots \geq p_n$ and $\sum_{i=m}^n p_i \leq 1$. Note that it is possible that no bin is selected. Call this experiment O^- and denote the random variable counting the number of balls thrown until there are no longer any empty bins by X . Now consider a random experiment of throwing balls independently into $n - m + 1$ bins with selection probabilities p_m, \dots, p_m , provided $(n - m + 1)p_m \leq 1$. Call this experiment L and denote the random variable counting the number of balls thrown until there are no longer any empty bins among the first n bins by X' . Then $X' \leq_{st} X$ and consequently, $\mathbf{E}[X'] \leq \mathbf{E}[X]$.*

To find $\mathbf{E}[X']$, we rely on the following lemma.

Lemma 3. *We are given n bins with equal probability of being filled with $p_i = p$, $i = 1, \dots, n$ and $np \leq 1$. Then the expected time to fill all the bins is H_n/p .*

We are now in a position to propose an upper and lower bound for $\mathbf{E}[T]$. For the upper and lower bounds, we apply Lemma 3 to models U and L respectively.

Corollary 1. *Under the power law assumption for the p_i ,*

$$\frac{(\beta n)^\alpha H_{(1-\beta)n}}{c_n^\alpha} \leq \mathbf{E}[T] \leq \frac{n^\alpha H_n}{c_n^\alpha} \quad (2)$$

for any β that satisfies the conditions $(1 - \beta)np_{\beta n} \leq 1$ and $0 \leq \beta \leq 1$.

2.2 General Bounds When $n \geq 1$

In the view estimation context, n represents the number of different values an attribute can take. Thus, it is important to prove results that still hold true for smaller values of n . In this section, we provide bounds for $\mathbf{E}[T]$ when $n \geq 1$. This is achieved by finding bounds for H_n and applying Corollary 1.

Theorem 1. *For $\alpha \geq 0$ and $n \geq 1$ bins, the expected number of balls needed to fill all bins satisfies $\frac{1}{c_n^\alpha} \cdot (n - \sqrt{n})^\alpha \ln(\sqrt{n} + 1) \leq \mathbf{E}[T] \leq \frac{n^\alpha}{c_n^\alpha}(1 + \ln n)$.*

2.3 Asymptotically Accurate Estimator for the Expected Number of Balls

The following theorem proves that the expected number of balls needed to fill all bins when n is large can be approximated by $n^\alpha(1 + \ln n)/c_n^\alpha$.

Theorem 2. Let $\mathbf{E}_n[T]$ denote the expected number of balls needed to fill all n bins. Then, for $\alpha \geq 0$, $\lim_{n \rightarrow \infty} \frac{\mathbf{E}_n[T]}{n^\alpha(1 + \ln n)/c_n^\alpha} = 1$.

3 Estimating the Expected Number of Occupied Bins Given a Fixed Number of Balls

In this section, we consider the problem of estimating the expected number of filled bins given that a fixed number of balls are thrown. Here, *filled bin* means that there is at least one ball in the bin. Let us assume that there are n bins having probabilities p_i for $i = 1, 2, \dots, n$ and we throw T balls. To find an expression for the expected number of filled bins, let $Z_i = \mathbf{1}\{\text{bin } i \text{ is filled after } T \text{ throws}\}$. Then the number of filled bins at the end of T throws is $\sum_i Z_i$ and therefore the expected value of the number of filled bins after T throws is

$$\mathbf{E} \left[\sum_{i=1}^n Z_i \right] = \sum_{i=1}^n \Pr\{\text{bin } i \text{ is filled after } T \text{ throws}\} = n - \sum_{i=1}^n (1 - p_i)^T \quad (3)$$

When bin probabilities follow a uniform distribution, the expression reduces to a closed form without the summation. This yields to Cardenas' formula

$$n - n(1 - 1/n)^T. \quad (4)$$

However, in the general case and even under our power law distribution assumption on the bin probabilities, it is difficult to express it without the summation. Moreover, in the context of computing the view size for database queries, n can be as large as 10^{20} in some cases, which makes the summation intractable. So we seek upper and lower bounds to our quantity of interest.

3.1 Lower Bound

To obtain a lower bound to the number of filled bins, we must find a scheme that is pessimistic in the sense that the probability that a new bin will be filled at each trial is minimized. So we consider the following process. After each throw, if the ball goes into a previously empty bin, the ball is relocated to the remaining empty bin with the highest probability. This is to ensure that the probability that a new bin will be occupied at the next trial is minimized. After T throws, the random variable that counts the number of filled bins for this scheme is stochastically smaller than that of the original problem and therefore the expected values are ordered accordingly. Under this modified process, let X_T

denote the number of filled bins after T throws. Then $\{X_T\}$ is a Markov Chain with the following transition rule

$$X_{T+1} = \begin{cases} X_T & \text{with prob. } \sum_{i=1}^{X_T} \frac{c_i^\alpha}{i^\alpha} \\ X_T + 1 & \text{with prob. } \sum_{i=X_T+1}^n \frac{c_i^\alpha}{i^\alpha}. \end{cases}$$

Note that it is not computationally hard to evaluate c once α and n are given. However, we use the approximation

$$c_n^\alpha \simeq \left(1 + \int_1^n x^{-\alpha} dx \right)^{-1} = \begin{cases} \frac{\alpha-1}{\alpha-n^{1-\alpha}} & \text{when } \alpha \neq 1 \\ \frac{1}{\ln n + \gamma} & \text{when } \alpha = 1 \end{cases}$$

where γ is the Euler-Mascheroni constant. In particular, when n is large and $\alpha > 1$, $c_n^\alpha \simeq (\alpha - 1)/\alpha$.

For large n and $\alpha > 1$. When the number of bins n is sufficiently large and $\alpha > 1$, we provide a lower bound for the expected number of filled bins when T balls are thrown.

Theorem 3. *Let Z be the number of filled bins when T balls are thrown into n bins. When n is large and $\alpha > 1$, $\mathbf{E}[Z] \geq (T + 1)^{1/\alpha} - 1$.*

For any $n \geq 1$ and $\alpha < 1$. For the case $\alpha < 1$, we do not need n to be large to obtain a lower bound for $\mathbf{F}_\alpha(T, n)$.

Theorem 4. *For $0 \leq \alpha < 1$, a lower bound on the expected number of filled bins when T balls are thrown is given by $(n + 1) \left(1 - e^{-\frac{(1-\alpha)T}{n+1} - \ln \frac{n+1}{n}} \right) - 1$.*

For any $n \geq 1$ and any $\alpha \geq 0$. Here we provide lower bounds for the most general case. Notably, it covers the case when $\alpha = 0$.

Theorem 5. *For $\alpha > 0$ and $n \geq 1$, a lower bound on the expected number of filled bins when T balls are thrown is given by $\left(1 - \left(1 - \frac{1}{T} \right)^T \right) (c_n^\alpha T)^{1/\alpha}$.*

Note that when T is large, the lower bound can be approximated by $(1 - e^{-1}) (c_n^\alpha T)^{1/\alpha}$.

3.2 Upper Bound

Let us provide an upper bound of $\mathbf{F}_\alpha(T, n)$ that holds for $n \geq 1$ and $\alpha \geq 0$.

Theorem 6. *For $n \geq 1$, an upper bound on the expected number of filled bins when T balls are thrown is given by*

$$\begin{cases} \min \left\{ n - n \left(1 - \frac{1}{n} \right)^T, \frac{T}{\ln n + \gamma} - 1 + T \left(1 - \frac{\ln T - \ln(\ln n + \gamma) + \gamma}{\ln n + \gamma} \right) \right\} & \alpha = 1 \\ \min \left\{ n - n \left(1 - \frac{1}{n} \right)^T, \frac{\alpha}{\alpha-1} (cT)^{1/\alpha} - \frac{cT}{\alpha-1} n^{1-\alpha} \right\} & \alpha \neq 1 \end{cases}$$

3.3 Bound Performance

Corollary 2. *When n is sufficiently large and $\alpha > 1$, the ratio between the lower bound of Theorem 3 and the upper bound of Theorem 6 is less than or equal to $\left(\frac{\alpha}{\alpha-1}\right)^{1-\frac{1}{\alpha}} \leq e^{1/e}$.*

4 Conclusion

The power law is a distribution frequently observed in real data sets. For the balls into bins problem, we studied the special but important case where the bin selection probabilities follow a power law. Asymptotically accurate estimators for the expected number of balls needed to be thrown in order to have all bins occupied as well as closed form expressions for the lower and upper bounds for the expected number of bins occupied when throwing a fixed number of balls are provided.

References

- AL07. Aouiche, K., Lemire, D.: A comparison of five probabilistic view-size estimation techniques in olap. In: Song, I.-Y., Pedersen, T.B. (eds.) DOLAP, pp. 17–24. ACM (2007)
- AM97. Boneh, A., Hofri, M.: The coupon-collector problem revisited -A survey of engineering problems and computational methods. Stochastic Models 13(1), 39–66 (1997)
- BGH⁺09. Beyer, K., Gemulla, R., Haas, P.J., Reinwald, B., Sismanis, Y.: Distinct-value synopses for multiset operations. Commun. ACM 52, 87–95 (2009)
- BP96. Boneh, S., Papanicolaou, V.G.: General asymptotic estimates for the coupon collector problem. J. Comput. Appl. Math. 67, 277–289 (1996)
- Car75. Cardenas, A.F.: Analysis and performance of inverted data base structures. Commun. ACM 18(5), 253–263 (1975)
- CCMN00. Charikar, M., Chaudhuri, S., Motwani, R., Narasayya, V.R.: Towards estimation error guarantees for distinct values. In: PODS, pp. 268–279. ACM (2000)
- CPKH05. Cai, M., Pan, J., Kwok, Y.-K., Hwang, K.: Fast and accurate traffic matrix measurement using adaptive cardinality counting. In: Sen, S., Ji, C., Saha, D., McCloskey, J. (eds.) MineNet, pp. 205–206. ACM (2005)
- DF03. Durand, M., Flajolet, P.: Loglog counting of large cardinalities (extended abstract). In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 605–617. Springer, Heidelberg (2003)
- FM85. Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. 31(2), 182–209 (1985)
- FMS96. Faloutsos, C., Matias, Y., Silberschatz, A.: Modeling skewed distribution using multifractals and the ‘80-20’ law. In: Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L. (eds.) Proceedings of 22th International Conference on Very Large Data Bases, VLDB 1996, Mumbai, India, September 3-6, pp. 307–317. Morgan Kaufmann (1996)

- GBLP96. Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: ICDE, pp. 152–159 (1996)
- GT01. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: SPAA, pp. 281–291 (2001)
- HNSS95. Haas, P.J., Naughton, J.F., Seshadri, S., Stokes, L.: Sampling-based estimation of the number of distinct values of an attribute. In: VLDB, pp. 311–322 (1995)
- KNW10. Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: Paredaens, J., Van Gucht, D. (eds.) PODS, pp. 41–52. ACM (2010)
- MR95. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press (1995)
- MV06. Motwani, R., Vassilvitskii, S.: Distinct values estimators for power law distributions. In: 2006 Proceedings of the Third Workshop on Analytic Algorithmics and Combinatorics (2006)
- NT03. Nadeau, T.P., Teorey, T.J.: A pareto model for olap view size estimation. Information Systems Frontiers 5(2), 137–147 (2003)
- SS94. Shaked, M., Shanthikumar, J.G.: Stochastic orders and their applications. Academic Press (1994)

A Randomized Algorithm for Finding Frequent Elements in Streams Using $O(\log \log N)$ Space

Masatora Ogata, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita

Department of Informatics,
Graduate School of Information Science and Electrical Engineering,
Kyushu University, Fukuoka 819-0395, Japan
`{masatora.ogata,yamauchi,kijima,mak}@inf.kyushu-u.ac.jp`

Abstract. Finding frequent items in a data stream is a fundamental problem; Given a threshold $\theta \in (0, 1)$, find items appearing more than $\theta \cdot N$ times in an input *stream* with length N . Karp, Shenker, Papadimiriou (2003) gave a simple deterministic online algorithm, which allows false positive outputs using memory of $O(\theta^{-1} \log N)$ bits, while they also gave a lower bound. Motivated by the theoretical bound of the space complexity, this paper proposes a simple randomized online algorithm using memory of $O(\theta^{-2} \log^2 \theta^{-1} + \log \log N)$ bits where parameters for approximation are hidden in the constant. Our algorithm is *robust* for memory overflow, compared with other naïve randomized algorithms, or deterministic algorithms using memory of $O(\log N)$ bits. We also give some randomized algorithms for approximate counting.

Keywords: Streaming algorithm, space complexity.

1 Introduction

Today, many computers, sensors, and applications generate streams of data, and processing data streams attracts more and more attention. For example, POS data from supermarkets generates data streams of trade names, packet monitoring in the Internet treats data streams such as IP addresses and port numbers, and so on. For such data streams, finding *frequent items* (a.k.a. *iceberg query* [9]) is a fundamental processing; For a prescribed frequency threshold $\theta \in (0, 1)$, find all items appearing more than or equal to $\theta \cdot N$ times in an input data stream $\mathbf{x} = (x_1, \dots, x_N) \in \Sigma^N$ where Σ denotes the set of items.

In processing stream data, algorithms are often required to work without knowing N in advance, or without knowing even an approximate value of $\log N$. Hence, it is an important task to consider the size of memory of an algorithm up to *bits*, otherwise memory overflow may cause serious troubles. The goal of this paper is to provide an online algorithm for finding frequent items which works without knowing N in advance using memory of $O(\log \log N)$ bits; thus the size of memory required by our algorithm is insensitive to N compared to existing algorithms using memory of $O(\log N)$ bits.

There are many known deterministic algorithms for finding frequent items in a stream. Boyer and Moore [34] gave an algorithm, known as *majority* algorithm, for $\theta = 1/2$ using $O(\log N)$ bits, while Fischer and Salzburg [10] independently gave an essentially same algorithm. Karp, Shenker, and Papadimitriou [14] gave a simple and elegant deterministic algorithm, which allows false positive outputs using memory of $O(\theta^{-1} \log N)$ bits. They also showed a lower bound that any online algorithm requires $\Omega(|\Sigma| \log(N/|\Sigma|))$ bits to find frequent items exactly. Demaine, López-Ortiz, and Munro [8] gave an algorithm essentially same as [14]. Manku and Motwani [16] proposed another deterministic algorithm, called *lossy counting*, which is based on approximate counting using $O(\varepsilon^{-1} \log(\varepsilon^{-1} N))$ bits, that outputs items of frequency more than $(\theta - \varepsilon)$ for $\varepsilon \in (0, 1)$.

There are many known randomized algorithms for finding frequent items, too. Toivonen [18] proposed an algorithm using memory of $O(\varepsilon^{-2} \log \delta^{-1} \log N)$ bits, which is based on uniform sampling of data in a stream, and outputs all frequent items with probability $1 - \delta$. Manku and Motwani [16] gave another algorithm, called *sticky sampling*, which uses memory of $O(\varepsilon^{-1} \log \delta^{-1} \log N)$ bits and outputs all frequent items with probability $1 - \delta$. Their algorithm samples items by adaptively changing the sampling rate according to the number of items that have arrived. All previous randomized algorithms are equipped with a counter of items, and hence, they require memory of $O(\log N)$ bits.

Our results. This paper proposes a simple randomized algorithm for finding frequent items in a stream. Our algorithm uses memory of $O(\theta^{-2} \log^2(\theta^{-1}) + \log \log N)$ bits, meaning that algorithm requires incremental memory of only $O(\log \log N)$ bits for the unknown length N of the data stream. Our algorithm is robust for memory overflow compared with other randomized algorithm such as [18][16] which proves only the expected size of memory required. Our algorithm is essentially based on the same strategy as Morris [17] and/or Flajolet [11] for probabilistic counting using $\lg \lg N$ bits, while we need to retain likely frequent items for our purpose using memory of *constant* number of bits for the “*significand*.” In fact, our algorithm works in constant time per input in *average*; however, we omit the detail, in order to simplify the description of the algorithm which achieves our desired space complexity in terms of bits.

Related Works. There are many known techniques to extract summary from a data stream, for example, hot-list queries [2], frequency moments [1], quantiles [2], sketches [5], etc. There are many papers on algorithms based on random sampling for these problems. Gibbons and Matias proposed a footprinting of data stream based on a random sampling with fixed space [12]. Based on the frequency count of the samples, the algorithm returns the top k symbols (hot-list). The moment of frequencies shows the skew of items. Alon, Matias and Szegedy [1] proposed a randomized sampling algorithm for computing frequency moments. However, their algorithm is *multi-pass* that requires to check the entire data stream many times. Good surveys are available in [6][15].

2 Preliminary

2.1 Problem Description

Let Σ denote a set of items. For simplicity of discussion, we in this paper assume that Σ is finite and that every item is identified with $\sigma := \log |\Sigma|$ bits. For a sequence $\mathbf{x} = (x_1, \dots, x_N)$ of items in Σ , let $f(s) \in \{0, \dots, N\}$ for $s \in \Sigma$ denote the frequency of s in \mathbf{x} . Given a (constant) $\theta \in (0, 1)$, we say an item $s \in \Sigma$ is *frequent* (in \mathbf{x}) if $f(s) \geq \theta N$.

2.2 A Basic Idea

If we know the length N of the stream data \mathbf{x} in advance, there is a simple Monte Carlo algorithm for finding frequent items (see e.g., [18]); for each item x_i sequentially arriving, record it with sampling rate c/N where c is a parameter for approximation. At the end of the input stream, output every item in the sample if the item appears no less than $\theta \cdot c$ times in the sample. If we set c sufficient large, we obtain the exact candidates of frequent items with high probability.

However, it is quite unnatural assumption, especially in practice, that an algorithm knows the length of a stream in advance. To get rid of the issue, the *bit-shifting* may be a good strategy (see e.g., [16]); count the size of a stream, and modify the sampling rate according to the size of a stream adaptively. In fact, we can design an algorithm based on this idea, while the analysis of it is usually complicated in contrast to the simple idea, see e.g., [16] or Section 4 of this paper.

At a glance, the bit-shifting strategy seems a successful idea, in spite of the complicated analysis. Nevertheless, it may not be entirely satisfactory because it uses $O(\log N)$ bits to count the size of the stream as it is, meaning that we have to prepare $O(\log N)$ bits without knowing approximate $\log N$ in advance. It is possible to code a program that increases the working space according to the size of the input stream, however it is clearly better off with an algorithm using memory of $O(\log \log N)$ bits or less.

In fact, Morris [7] gave an online algorithm for *probabilistic counting* which approximately counts the size of data stream using $O(\log \log N)$ bits, and Flajolet [11] analyzed it in detail. Hence, one may think that a randomized algorithm for finding frequent items is easily established using the probabilistic counting with $O(\log \log N)$ bits. Unfortunately, this is not the fact because we cannot predict which items are frequent, and we have to memorize all possibly frequent items. Remark that a similar issue also appears in deterministic algorithms; The result by Karp, Shenker, Papadimitriou [14] indicates that a simple counting scheme requires $\Omega(|\Sigma| \log(N/|\Sigma|))$ bits in the worst scenario, under a natural assumption of $N \gg |\Sigma| \gg 1/\theta$.

In the next section, we give a simple algorithm using memory of $O(\theta^{-2} \log^2(\theta^{-1}) + \log \log N)$ bits, independent of the size of $|\Sigma|$ on the assumption of $N \gg |\Sigma| \gg 1/\theta$, as did Karp, Shenker, Papadimitriou [14].

3 Algorithm

Parameters $\theta \in (0, 1)$, $\gamma \in (0, 1)$, and $\delta \in (0, 1)$ are supposed to be prescribed by a user, for which our algorithm satisfies the following properties: with probability at least $1 - \delta$, the output contains all frequent items $\{s \in \Sigma \mid f(s) \geq \theta N\}$ and does not contain any rare item of $\{s \in \Sigma \mid f(s) < (1 - \gamma)\theta N\}$. The following shows our algorithm for an input stream $\mathbf{x} = (x_1, \dots, x_N)$, in which $b \in \mathbb{N}$ is a parameter for approximation.

Algorithm 1

Let K denote a data structure which stores 2^b pairs of an item and a number $(s, K[s])$, where s uses σ bits, and $K[s]$ uses b bits.

0. **Initialize K .**

Set “exponent” $h := 0$.

1. **Read** x_i . Suppose $x_i = s^*$ for convenience.

If no more input, then **goto** 5.

2. With probability 2^{-h} , “**record**” s^* in K :

add s^* in K unless s^* in K , and **increment** $K[s^*]$ by one.

3. If $\sum_{s \in K} K[s] = 2^b$, then “**flush**”:

 3-(i) **Increment** h by one.

 3-(ii) **For** each s in K ,

choose k with probability $\binom{K[s]}{k} / 2^{K[s]}$, and
 set $K[s] := k$.

 3-(iii) **Delete** symbol $s \in K$ if $K[s] = 0$.

4. **Goto** 1.

5. **Output** every s satisfying that $K(s) \geq (1 - \gamma/2)\theta \sum_{s' \in K} K[s']$.

It is not difficult to see that Algorithm 1 uses $(b + \sigma)2^b + \lceil \lg h \rceil$ bits, and some extra working space of $O(\lg h)$ bits only. In precise, we can realize the probability 2^{-h} at Step 2 by flipping an unbiased coin h times using a counter of $\lceil \lg h \rceil$ bits; record x_i if all h trials are heads. Note that $h + b$ is approximately equal to $\lg N$ for $N \geq 2^b$, which we will discuss later, and hence $2^h \cdot K[s]$ approximates $f(s)$, intuitively. We also remark that the number of items output by Algorithm 1 is at most $1/(\theta(1 - \gamma/2))$.

Theorem 3.1. *Arbitrarily given $\theta \in (0, 1)$, $\gamma \in (0, 1)$ and $\delta \in (0, 1)$, set $b \geq \lceil \lg((\theta\gamma/2)^{-2} \ln(3((1 - \gamma/2)\theta\delta)^{-1})) \rceil + 3$. Then the output of Algorithm 1 contains all frequent items $\{s \in \Sigma \mid f(s) \geq \theta N\}$ and does not contain any rare item of $\{s \in \Sigma \mid f(s) < (1 - \gamma)\theta N\}$ with probability at least $1 - \delta$ for any input stream $\mathbf{x} = (x_1, \dots, x_N)$.*

Theorem 3.1 indicates that our algorithm uses memory of

$$\begin{aligned} 2^b(b + \sigma) + \lg \lg N &\simeq \frac{32 \ln \left(\frac{3}{(1 - \gamma/2)\theta\delta} \right)}{(\gamma\theta)^2} \lg \left(\frac{\ln \left(\frac{3}{(1 - \gamma/2)\theta\delta} \right)}{(\gamma\theta)^2} \right) + \lg \lg N \\ &= O \left(\frac{\log^2(\theta^{-1})}{\theta^2} + \log \log N \right) \end{aligned}$$

bits. Note that the bound on b is not optimized, for simplicity of the following arguments.

Now we show Theorem 3.1 in the following. To begin with, we observe the following fact, that is a key idea of our analysis.

Observation 3.2. *At the end of Algorithm 1, the sampling probability of every item x_i is the same, 2^{-h} , where h denotes the exponent at the end of Algorithm 1.*

Proof. Suppose the exponent was h' when x_i was read. Then x_i was recorded with probability $2^{-h'}$. To realize Step 3-(ii), we consider the following process: Keep x_i with probability $1/2$, otherwise delete it. Clearly the resulting frequency of element $s \in \Sigma$ recorded in K follows binomial distribution, i.e., $\binom{K[s]}{k}/2^{K[s]}$. Thus x_i survives at the end of Algorithm 1 with probability 2^{-h} exactly. \square

Lemma 3.3. *Set $b \geq \lceil \lg((\theta\gamma/2)^{-2} \ln(3((1-\gamma/2)\theta\delta)^{-1})) \rceil + 3$. If $h \geq 1$ then*

$$\Pr \left[\sum_{s' \in K} K[s'] \geq 2^{b-2} \right] \geq 1 - \frac{\delta}{3}$$

holds at the end of Algorithm 1.

Proof. Remark that $\sum_{s' \in K} K[s']$ is monotone nondecreasing if $\sum_{s' \in K} K[s'] < 2^b$, while $\sum_{s' \in K} K[s']$ decreases when $\sum_{s' \in K} K[s']$ becomes 2^b by flushing in Step 3-(ii). It is not difficult to see that the probability that $\sum_{s' \in K} K[s']$ becomes k at Step 3-(ii) follows a binomial distribution $\binom{2^b}{k}/2^{2^b}$. Thus

$$\Pr \left[\sum_{s' \in K} K[s'] < 2^{b-2} \right] \leq \sum_{j=0}^{2^{b-2}} \binom{n}{j} \frac{1}{2^{2^b}}$$

holds. Now we use the following inequality for the binomial distribution (c.f. 7) that

$$\sum_{j=0}^{(p-t)n} \binom{n}{j} p^j (1-p)^{n-j} \leq e^{-2t^2 n} \quad (1)$$

for $t \in (0, p)$. Let $n = 2^b$, $p = 1/2$, and $t = 1/4$ in (1), we have

$$\sum_{j=0}^{2^{b-2}} \binom{n}{j} \frac{1}{2^{2^b}} \leq e^{-2(1/4)^2 2^b} = e^{-2^{b-3}} \leq e^{-\ln(\delta/3)} = \frac{\delta}{3}. \quad \square$$

Next we remark the following.

Observation 3.4. *On condition that $n = \sum_{s' \in K} K[s']$, the frequency of $s \in \Sigma$ recorded in K follows the hypergeometric distribution; i.e.,*

$$\Pr[K[s] = j] = \frac{\binom{f(s)}{j} \cdot \binom{N-f(s)}{n-j}}{\binom{N}{n}}.$$

Using Observation 3.4, we obtain the following two lemmas.

Lemma 3.5. Set $b \geq \lceil \lg((\theta\gamma/2)^{-2} \ln(3((1-\gamma/2)\theta\delta)^{-1})) \rceil + 3$. On the condition that $\sum_{s' \in K} K[s'] \geq 2^{b-2}$ holds at the end of Algorithm 1, the probability that the output of Algorithm 1 contains all frequent elements $\{s \in \Sigma \mid f(s) \geq \theta N\}$ is at least $1 - \delta/3$.

Proof. Suppose $s \in \Sigma$ is a frequent element satisfying $f(s) \geq \theta N$. Now we use the following inequality for a hypergeometric distribution that

$$\sum_{j=0}^k \frac{\binom{M}{j} \cdot \binom{N-M}{n-j}}{\binom{N}{n}} \leq e^{-2(\frac{k}{n} - \frac{M}{N})^2 n} \quad (2)$$

due to Chvatal [7]. Let $N = |\mathcal{X}|$, $M = f(s)$, $n = \sum_{s' \in K} K[s'] \geq 2^{b-2}$, and $k = (1 - \gamma/2)\theta n$ in inequality (2), then $k/n = (1 - \gamma/2)\theta$ and $M/N \geq \theta$ hold, and we have

$$\begin{aligned} \Pr \left[K[s] < (1 - \gamma/2)\theta \sum_{s' \in K} K[s'] \right] &\leq \sum_{j=0}^{(1-\gamma/2)\theta n} \frac{\binom{M}{j} \cdot \binom{N-M}{n-j}}{\binom{N}{n}} \\ &\leq e^{-2((1-\gamma/2)\theta - \theta)^2 2^{b-2}} = e^{-2(\theta\gamma/2)^2 2^{b-2}} \leq \frac{(1 - \gamma/2)\theta\delta}{3}. \end{aligned}$$

Since there are at most $1/\theta$ frequent elements, the probability of false negative is at most $\lfloor 1/\theta \rfloor (1 - \gamma/2)\theta\delta/3 \leq \delta/3$. \square

Lemma 3.6. Set $b \geq \lceil \lg((\theta\gamma/2)^{-2} \ln(3((1-\gamma/2)\theta\delta)^{-1})) \rceil + 3$. On the condition that $\sum_{s' \in K} K[s'] \geq 2^{b-2}$ holds at the end of Algorithm 1, the probability that the output of Algorithm 1 contains a rare element s satisfying that $f(s) < (1 - \gamma)\theta N$ is at most $\delta/3$.

Proof. Suppose $s \in \Sigma$ is a rare element satisfying $f(s) < (1 - \gamma)\theta N$. Now we use the inequality (2) again. Let $N = |\mathcal{X}|$, $M = N - f(s)$, $n = \sum_{s' \in K} K[s'] \geq 2^{b-2}$, and $k = (1 - (1 - \gamma/2)\theta)n$ in the inequality (2), then $k/n = 1 - (1 - \gamma/2)\theta$ and $M/N \geq 1 - (1 - \gamma)\theta$ hold, and we have

$$\begin{aligned} \Pr [K[s] \geq (1 - \gamma/2)\theta n] &= \Pr [n - K[s] < (1 - (1 - \gamma/2)\theta)n] \\ &\leq \sum_{j=0}^{(1-(1-\gamma/2)\theta)n} \frac{\binom{M}{j} \cdot \binom{N-M}{n-j}}{\binom{N}{n}} \leq e^{-2((1-(1-\gamma/2)\theta) - (1-(1-\gamma)\theta))^2 2^{b-2}} \\ &= e^{-2(\theta\gamma/2)^2 2^{b-2}} \leq \frac{(1 - \gamma/2)\theta\delta}{3}. \end{aligned}$$

Remark that Algorithm 1 outputs at most $1/(\theta(1 - \gamma/2))$ elements, thus the probability of a false positive is at most

$$\frac{1}{\theta(1 - \gamma/2)} \cdot \frac{(1 - \gamma/2)\theta\delta}{3} \leq \frac{\delta}{3}. \quad \square$$

From Lemmas 3.3, 3.5, 3.6, we obtain Theorem 3.1.

Finally we discuss that $h + b$ approximates $\lg N$ well.

Lemma 3.7. *Set $b \geq \lceil \lg((\theta\gamma/2)^{-2} \ln(3((1-\gamma/2)\theta\delta)^{-1})) \rceil + 3$, then*

$$\Pr[\lfloor \lg n \rfloor - 1 \leq h + b \leq \lceil \lg n \rceil + 1] \geq 1 - \delta$$

holds when Algorithm 2 reads x_n .

Proof. Notice that

$$\begin{aligned} & \Pr[\lfloor \lg n \rfloor - 1 \leq h + b \leq \lceil \lg n \rceil + 1] \\ &= 1 - (\Pr[h + b < \lg n - 1] + \Pr[h + b > \lg n + 1]) \end{aligned}$$

holds. Let $X(n; \alpha)$ denote a random variable according to the binomial distribution which n time Bernoulli trials with head probability $1/2^\alpha$ follows.

Remembering Observation 3.2, $[h + b < \lfloor \lg(n) \rfloor - 1]$ if and only if $[X(n; \lfloor \lg(n) \rfloor - b - 1) < 2^b]$. Using the Chernoff bound, we have

$$\begin{aligned} \Pr[h + b < \lfloor \lg(n) \rfloor - 1] &\leq \Pr[X(n; \lg(n) - b - 1) < 2^b] \\ &= \Pr\left[X(n; \lg(n) - b - 1) < \frac{2^b}{\mathbb{E}[X(n; \lg(n) - b - 1)]}\mathbb{E}[X(n; \lg(n) - b - 1)]\right] \\ &= \Pr\left[X(n; \lg(n) - b - 1) < \frac{2^b}{n \cdot 2^{-\lg(n)+b+1}}\mathbb{E}[X(n; \lg(n) - b - 1)]\right] \\ &= \Pr\left[X(n; \lg(n) - b - 1) < \frac{2^b}{2^{b+1}}\mathbb{E}[X(n; \lg(n) - b - 1)]\right] \\ &= \Pr\left[X(n; \lg(n) - b - 1) < \left(1 - \frac{1}{2}\right)\mathbb{E}[X(n; \lg(n) - b - 1)]\right] \\ &\leq e^{-\frac{1}{2} \cdot \frac{1}{2^2} \mathbb{E}[X(n; \lg(n) - b - 1)]} \leq e^{-\frac{1}{2^3} n \cdot 2^{-\lg(n)+b+1}} \leq e^{-2^{b-2}} \leq \frac{\delta}{2}. \end{aligned}$$

In a similar way, $[h + b > \lceil \lg(n) \rceil + 1]$ if and only if $[X(n; \lceil \lg(n) \rceil - b + 1) \geq 2^b]$. Using the Chernoff bound, we have

$$\begin{aligned} \Pr[h + b \geq \lceil \lg(n) \rceil + 1] &\leq \Pr[X(n; \lg(n) - b + 1) \geq 2^b] \\ &= \Pr\left[X(n; \lg(n) - b + 1) \geq \frac{2^b}{\mathbb{E}[X(n; \lg(n) - b + 1)]}\mathbb{E}[X(n; \lg(n) - b + 1)]\right] \\ &= \Pr\left[X(n; \lg(n) - b + 1) \geq \frac{2^b}{n \cdot 2^{-\lg(n)+b-1}}\mathbb{E}[X(n; \lg(n) - b + 1)]\right] \\ &= \Pr\left[X(n; \lg(n) - b + 1) \geq \frac{2^b}{2^{b-1}}\mathbb{E}[X(n; \lg(n) - b + 1)]\right] \\ &= \Pr[X(n; \lg(n) - b + 1) \geq 2\mathbb{E}[X(n; \lg(n) - b + 1)]] \\ &\leq e^{-\frac{1}{3}\mathbb{E}[X(n; \lg(n) - b + 1)]} \leq e^{-\frac{1}{3}n \cdot 2^{-\lg(n)+b-1}} \leq e^{-2^{b-3}} \leq \frac{\delta}{2} \end{aligned}$$

and we obtain the claim. \square

4 Other Naïve Algorithms

In this section, we show two naïve randomized algorithms using $O(\frac{1}{\theta} \lg^2 \frac{1}{\theta} + \lg \lg N)$ bits. For any prescribed parameters θ, δ and ε , Algorithm 2 outputs all frequent items with probability at least $1 - \delta$ in a similar way as Algorithms I, II, as well as gives approximate counts of their frequency.

Algorithm 2

0. **Initialize** K consisting of $\frac{8t}{\theta}$ pairs of an item and a number $(s, K[s])$, where s uses σ bits, and $K[s]$ uses $3 + \lg \frac{t}{\theta}$ bits.
Set *exponent* $h = 0$.
Set $p = 0$, $cnt = 0$, $h' = 0$.
1. **Read** input x_i . Suppose $x_i = s^*$ for convenience.
If no more input, then **goto** 5.
2. **If** $cnt > 2t$, then with probability $\frac{t}{\theta \cdot 2^{h'+1}}$, **record** s^* in K ;
 Add s^* in K unless s^* in K , and $K[s^*]++$.
 else with probability $\frac{t}{\theta \cdot 2^h}$, **record** s^* in K ;
 add s^* in K unless s^* in K , and $K[s^*]++$.
3. **Update** “exponent” h
If $cnt > 2t$ and $h' < h$ then
 3-(i) For each $s \in K$
 Set $K[s] := \lfloor K[s]/2 \rfloor$, and
 $K[s]++$ with probability $1/2$.
 3-(ii) **Set** $h' := h$.
4. **Goto** 1.
5. For each entry of K **do**
 If $K[s] \geq (1 - \varepsilon) \frac{t}{4}$ **then** output $(s, f(s; \mathbf{x}), h)$.

We use the exponent introduced in Algorithm I. Lemma B.7 showed that the exponent h and the size of the additional memory of b bits approximates $\lg N$ well. The value of the exponent used in Algorithm 2 is the sum of the exponent and b in Algorithm I. We note that we can obtain the accuracy shown in Lemma B.7 with $\lg \delta^{-1}$ bits.

Algorithm 2 uses $(\frac{8t}{\theta} + \sigma) \lg (\frac{8t}{\theta})$ bits and some extra working space for $O(\lg h + b)$ bits only. Step 2 and 3 are realized by $\lceil \lg h \rceil$ bits as discussed in Section 3.

Theorem 4.1. *For any arbitrary $\theta \in (0, 1)$, $\delta \in (0, 1)$, $\varepsilon \in (0, 1)$, set $t \geq 48\varepsilon^{-2} \ln(5\theta^{-1}\delta^{-1})$. Then, the output of Algorithm 2 contains all frequent items and the counter value of each frequent item approximates its frequency within ε with probability at least $1 - \delta$ for any input stream $\mathbf{x} = (x_1, \dots, x_N)$.*

Theorem 4.1 indicates that Algorithm 2 uses memory of

$$\begin{aligned} \left(\frac{8t}{\theta} + \sigma \right) \lg \left(\frac{8t}{\theta} \right) + \lg \lg N &\simeq \frac{\frac{1}{\varepsilon^2} \ln \frac{1}{\theta}}{\theta} \lg \left(\frac{\frac{1}{\varepsilon^2} \ln \frac{1}{\theta}}{\theta} \right) + \lg \lg N \\ &= O \left(\frac{\lg^2(\theta^{-1})}{\theta} + \log \log N \right) \end{aligned}$$

bits.

Theorem 4.1 is obtained by replacing 3-(i) of Algorithm 2 with the following *bit-shifting* procedure.

Algorithm 3

- 3-(i) **For** each $s \in K$,
choose k with probability $\binom{K[s]}{k} / 2^{K[s]}$, and
Set $K[s] := k$.

Again, this procedure is realized with two $\lceil \lg h \rceil$ bits counters as discussed in Section 3. The variance of $K[s]$ after the execution of line 3-(i) of Algorithm 2 is smaller than that of Algorithm 3. Hence, we prove Theorem 4.1 with Algorithm 3 and the results also hold for Algorithm 2.

5 Concluding Remark

We have proposed a simple randomized algorithm for finding frequent items. Our algorithm uses $O(\theta^{-2} \log^2(\theta^{-1}) + \lg \lg N)$ bits, and is robust for memory overflow. For comparison, we also mentioned two naïve randomized algorithms briefly, for which we can also give proofs of approximation. While algorithms proposed in this paper uses $O(\log \log N)$ bits, in terms of N , Algorithm 2 in Section 3 is simpler to analyse and is more robust than ones in Section 4.

Acknowledgment. The authors thank anonymous reviewers for their valuable comments.

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. *J. of Computer and System Sciences* 58, 137–147 (1999)
2. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proc. of 23rd ACM Symposium on Principles of Database Systems, pp. 286–296 (2004)
3. Boyer, R.S., Moore, J.S.: A fast majority vote algorithm, Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas (1981)
4. Boyer, R.S., Moore, J.S.: MJRTY—a fast majority vote algorithm. In: Automated Reasoning: Essays in Honor of Woody Bledsoe. Automated Reasoning Series, pp. 105–117. Kluwer, Dordrecht (1991)
5. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. *Theoretical Computer Science* 312, 3–15 (2004)
6. Cormode, G., Hadjieleftheriou, M.: Methods for finding frequent items in data streams. *The VLDB Journal* 19, 3–20 (2010)
7. Chvátal, V.: The tail of the hypergeometric distribution. *Discrete Mathematics* 25, 285–287 (1979)
8. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of Internet packet streams with limited space. In: Proc. of 10th Annual European Symposium on Algorithms, pp. 348–360 (2002)

9. Fang, M., Shivakumar, N., Gracia-Molina, H., Motwani, R., Ullman, J.D.: Computing iceberg queries efficiently. In: Proc. of 24th Intl. Conf. on Very Large Data Bases, pp. 299–310 (1998)
10. Fischer, M., Salzburg, S.: Finding a majority among n votes: solution to problem 81-5. *J. Algorithms* 3, 376–379 (1982)
11. Flajolet, P.: Approximate counting: a detailed analysis. *BIT* 25, 113–134 (1985)
12. Gibbons, P.B., Matias, Y.: New sampling-based summary statistics for improving approximate query answer. In: Proc. of ACM SIGMOD Intl. Conf. on Management of Data, pp. 331–342 (1998)
13. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association* 58, 13–30 (1963)
14. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.* 28, 51–55 (2003)
15. Liu, H., Yuan, Y., Han, J.: Methods for mining frequent items in data streams. *Knowl. and Inf. Syst.* 26, 1–30 (2011)
16. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proc. of 28th Intl. Conf. on Very Large Data Bases, pp. 346–357 (2002)
17. Morris, R.: Counting large numbers of events in small registers. *Communications of the ACM* 21, 840–842 (1978)
18. Toivonen, H.: Sampling large database for association rules. In: Proc. of 22nd Intl. Conf. on Very Large Data Bases, pp. 134–145 (1996)

A Nearly-Quadratic Gap between Adaptive and Non-adaptive Property Testers

(Extended Abstract)

Jeremy Hurwitz*

Department of Computing and Mathematical Sciences,
California Institute of Technology, Pasadena California 91125
`jhurwitz@cs.caltech.edu`

Abstract. We show that for all integers $t \geq 8$ and arbitrarily small $\epsilon > 0$, there exists a graph property Π (which depends on ϵ) such that ϵ -testing Π has non-adaptive query complexity $Q = \tilde{\Theta}(q^{2-2/t})$, where $q = \tilde{\Theta}(\epsilon^{-1})$ is the adaptive query complexity. This resolves the question of how beneficial adaptivity is, in the context of proximity-dependent properties. This also gives evidence that the canonical transformation of Goldreich and Trevisan is essentially optimal when converting an adaptive property tester to a non-adaptive property tester.

To do so, we provide optimal adaptive and non-adaptive testers for the combined property of having maximum degree $O(\epsilon N)$ and being a *blow-up collection* of an arbitrary base graph H .

Keywords: Sublinear-Time Algorithms, Property Testing, Dense-Graph Model, Adaptive vs Non-adaptive Queries, Hierarchy Theorem.

1 Introduction

In this paper, we consider the power of adaptive versus non-adaptive queries for property testers in the dense-graph model. In this model, the algorithm is given access to a graph $G = ([N], E)$ via an oracle $g : [N] \times [N] \rightarrow \{0, 1\}$ such that $g(u, v) = 1$ if and only if $(u, v) \in E$. A graph is said to be ϵ -*far* from a particular property if at least ϵN^2 edges must be added or deleted to yield a graph with the property. If less than ϵN^2 edits are required, the graph is ϵ -*close* to the property.

Given a graph property Π , we say that a (randomized) algorithm \mathcal{A} is an ϵ -*tester* for Π if $\Pr[\mathcal{A}(G) = \text{ACCEPT}] > 2/3$ for all $G \in \Pi$ and $\Pr[\mathcal{A}(G) = \text{REJECT}] > 2/3$ for all G which are ϵ -far from Π .

1.1 Adaptive, Non-adaptive, and Canonical Testers

Property testers can be broadly classified into two types according to how the queries are determined. In an *adaptive algorithm*, the results of previous queries can be used when determining the next query. A *non-adaptive algorithm*, on the other hand, determines all of its queries in advance.

* Supported by NSF grants CCF-0830787, CCF-0829909, and CCF-1116111.

Most of the initial work in the dense-graph model focused on non-adaptive algorithms. In fact, these early algorithms used an even more restricted framework, termed *canonical algorithms* by [8]. In a canonical algorithm, the tester chooses a random subset of the vertices and queries the entire induced subgraph.

Given a property Π and an error parameter $\epsilon > 0$, let q be the adaptive query-complexity, Q be the non-adaptive query complexity, and \tilde{Q} be the canonical query complexity. By definition, $q \leq Q \leq \tilde{Q}$. A natural question, then, is to determine the exact relationship between these parameters.

In [8] and [2], the authors note that any algorithm which makes q queries considers at most $2q$ vertices. It therefore suffices to query the subgraph induced by a random set of $2q$ vertices and then locally simulate the adaptive algorithm. This *canonical transformation* shows that, for any property, $\tilde{Q} < 2q^2$.

It is easy to show that there exist properties with $\tilde{Q} = \Omega(Q^2)$, thereby showing that the canonical transformation is optimal when converting (adaptive or non-adaptive) algorithms into canonical algorithms. Indeed, [5] and [7] show that a quadratic performance gap exists between non-adaptive and canonical testers for many natural properties.

Similarly, we can consider the relative power of adaptive versus non-adaptive (but not necessarily canonical) property testers. It is widely believed that the canonical transformation remains optimal, in the worst case, even for this more modest goal. In other words, it is believed that there exist properties such that $Q = \Omega(q^2)$. However, proving such a separation between the adaptive and non-adaptive complexities has proven elusive – no unconditional gap was known until Goldreich and Ron demonstrated a property where $Q = \tilde{\Omega}(q^{3/2})$ in [7].

In light of this difficulty, researchers have considered two modifications of the problem. The first approach, used by [9], considers *proximity-dependent* properties which depend on the tolerance parameter ϵ . In this context, they achieved a gap of $Q = \tilde{\Theta}(q^{4/3})$.

The second approach, used by [2], is to consider *promise problems*. In this context, the authors showed that there exist properties such that $Q = \Omega(q^{2-\delta})$ for all $\delta > 0$, exhibiting the first nearly-quadratic gap between adaptive and non-adaptive queries in the dense-graph model. More specifically, the authors demonstrated a hierarchy of gaps of the form $Q = \Theta(q^{2-2/t})$ for each integer $t \geq 2$. Unfortunately, the promise they use, while natural, is quite strong, and it is currently unclear how to remove the promise.

In this paper, we achieve a nearly-quadratic gap without using a promise, at the expense of making the properties proximity-dependent. This proves that for all $\delta > 0$ and arbitrarily small ϵ , there exists a property Π (which depends on ϵ), such that ϵ -testing Π requires $Q > q^{2-\delta}$ non-adaptive queries, where q is the adaptive query complexity. As in [7], we also establish a hierarchy of relationships between the adaptive and non-adaptive query complexities.

Theorem 1 (Main Theorem). *For all $t \geq 8$ and arbitrarily small ϵ , there exists a graph property Π (which depends on ϵ) such that ϵ -testing Π has non-adaptive query complexity $Q = \tilde{\Theta}(q^{2-2/t})$, where $q = \tilde{\Theta}(\epsilon^{-1})$ is the adaptive query complexity.*

Theorem 11 and [7] both provide strong evidence that the canonical transformation is optimal in the general case. Although each result technically leaves open the possibility of a better transformation between adaptive and non-adaptive testers, each does so in a different, and very restricted, way. As a result, any such transformation would have to be very unnatural and would have to depend sensitively on the internal structure of the adaptive tester.

1.2 Graph Blow-Ups and Blow-Up Collections

A *graph blow-up* consists of replacing each vertex of a graph with a cluster of vertices (a rigorous definition is given in section 2). This operation is frequently used in studying the dense-graph model (see, for example, [1], [3], and [6]).

The complexity of testing whether a graph is a blow-up was essentially resolved in [4] (see also [5]), where it is shown that, for any H , the adaptive query complexity is $O(\epsilon^{-1})$ and the non-adaptive query complexity is $\tilde{O}(\epsilon^{-1})$.

A graph is a *blow-up collection* if it can be partitioned into disjoint subgraphs, each of which is a blow-up of H . This notion was implicitly introduced in [7], which showed the following lower bound.

Lemma 2 ([7], Lemma 5.6). *Let H be a simple t -cycle, with $t \geq 4$. Testing whether G is a blow-up collection of H requires $\Omega(\epsilon^{-(2-2/t)})$ non-adaptive queries, even given the promise that G has maximum degree at most $2t\epsilon N$.*

In this paper, we prove tight upper-bounds for both the adaptive and non-adaptive cases. Specifically, we show that a tester can determine whether G is a blow-up collection of any given H , given that same promise on the degrees, using only $O(\epsilon^{-1})$ adaptive queries or $O(\epsilon^{-2+1/(\Delta+2)} + \epsilon^{-2+2/W})$ non-adaptive queries, where Δ and W are parameters depending only on H . When H is a simple t -cycle, $\Delta = 2$ and $W = t$, and the non-adaptive upper bound reduces to $O(\epsilon^{-2+2/t})$, matching the lower bound in Lemma 2.

Indeed, it suffices for G to be $O(\epsilon)$ -close to satisfying the promise. Since [9] gives an efficient non-adaptive tester for the property of having maximum degree $O(\epsilon N)$, we obtain the following two theorems.

Theorem 3 (Adaptive Tester). *For all graphs H and constants $c > 1$, there exists an adaptive property tester (with two-sided error) for the (proximity-dependent) combined property of having maximum degree at most ceN and being a blow-up collection of H . The tester has query complexity $O(\epsilon^{-1} \lg^3 \epsilon^{-1})$.*

Since any tester must make $\Omega(\epsilon^{-1})$ queries, Theorem 3 is essentially optimal.

Theorem 3, combined with Lemma 2, suffices to show a gap of size $Q = \Omega(q^{2-\delta})$ for all $\delta > 0$. In the following theorem, we strengthen this result by proving a tight upper-bound on the non-adaptive query complexity. This shows that there is an infinite hierarchy of achievable relationships between the adaptive and non-adaptive query complexities of proximity-dependent properties.

Theorem 4 (Non-Adaptive Tester). *For all graphs H and constants $c > 1$, there exists a non-adaptive property tester (with two-sided error) for the*

(proximity-dependent) combined property of having maximum degree at most $c\epsilon N$ and being a blow-up collection of H . The tester has query complexity $O(\epsilon^{-2+1/(\Delta+2)} + \epsilon^{-2+2/W})$, where $\Delta = \deg(H)$ is the maximum degree of H and $W < |H|^2$ is a bound on the size of a witness against H (see Definition 9).

As mentioned previously, when H is a simple t -cycle, $\Delta = 2$ and $W = t$. Therefore, combining Theorems 3 and 4 with Lemma 2 yields Theorem 1.

We note that the algorithms in both theorems have running time polynomial in the query complexity. In the adaptive case, the query complexity can also be made polynomial in c and $|H|$. However, we do not do so here.

1.3 Organization of the Paper

Section 2 contains basic definitions and notation. In section 3, we show how to adaptively test whether a graph is a blow-up collection, given the promise that the graph is close to having maximum degree $O(\epsilon N)$. In section 4, we present the non-adaptive algorithm, under the same promise. Finally, section 5 removes the promise by explicitly testing that the input is close to having low degree.

Due to space constraints, most proofs are omitted here. The complete arguments can be found in the full version of the paper.

2 Notation and Basics

All graphs are assumed to be undirected. Following standard graph-theoretic notation, we let $\Gamma(v) = \{u : (u, v) \in E\}$ denote the neighbors of v . Given $S \subset V$, we let $G|_S$ denote the subgraph induced by S and $\Gamma_S(v) = \Gamma(v) \cap S$ denote the neighbors of v in S . Given $S, T \subset V$, we let $E(S, T) = \{(u, v) \in E : u \in S, v \in T\}$ denote the set of edges between S and T and $S \Delta T$ denote the symmetric difference of S and T .

Definition 5 (Graph Blow-Up). A graph $G = ([N], E)$ is a *blow-up* of the graph $H = ([h], F)$ if there exists a partition of $[N]$ into $V_1 \cup \dots \cup V_h$ such that for every $i, j \in [h]$ and $(u, v) \in V_i \times V_j$, $(u, v) \in E$ if and only if $(i, j) \in F$. We denote the set of blow-ups of H by $\mathcal{BU}(H)$.

Note that no requirement is made as to the relative sizes of the V_j . In particular, we allow the case where $|V_j| = 0$.

Definition 6 (Blow-Up Collection). A graph $G = ([N], E)$ is a *blow-up collection* of the graph H if there exists a partition of $[N]$ into $V^1 \cup \dots \cup V^k$, for some k , such that $G|_{V^i} \in \mathcal{BU}(H)$ for all i and $E(V^i, V^j) = \emptyset$ for all $i \neq j$. We denote the set of blow-up collections of H by $\mathcal{BUC}(H)$.

Throughout this paper, H will be an arbitrary fixed graph and $c > 1$ will be an arbitrary fixed constant. We let $h = |H|$, $\Delta = \deg(H)$, and $\mathcal{LD}_{c\epsilon} = \{G : \deg(G) \leq c\epsilon N\}$.

To prove Theorems 3 and 4, we will repeatedly use the concept of a set being (k, α) -partitionable. Informally, a vertex is (k, α) -partitionable if almost all of its

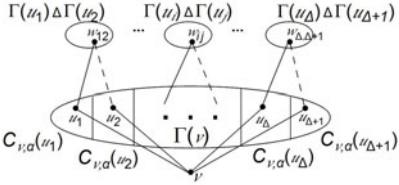
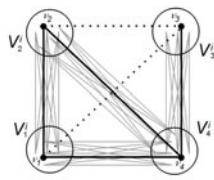
(a) A witness against v being $(\Delta, 0)$ -partitionable(b) A witness against G having the correct high-level structure, for $H = K_4$

Fig. 1. The two types of witnesses used by Algorithm 10 to determine whether G is a member of $\mathcal{BUC}(H)$. In each diagram, a line indicates that the algorithm queried that edge, with a solid line indicating the presence of an edge and a dashed line indicating the absence of an edge.

neighbors can be partitioned into k groups, such that all of the vertices within a part have essentially the same neighbors in G . The formal definition is given in Definition 8, after we introduce notation for such partitions.

Definition 7. Given $v \in V$ and $\alpha > 0$, let $C_{v,\alpha}(u) = \{w \in \Gamma(v) : |\Gamma(w) \Delta \Gamma(u)| < \alpha \epsilon N\}$.

In other words, $C_{v,\alpha}(u)$ consists of all vertices in $\Gamma(v)$ which have essentially the same neighbors as u . When v and α are clear from context, we will omit the subscripts and write $C(u)$ for $C_{v,\alpha}(u)$.

Definition 8 ((k, α)-Partitionable). A vertex v is (k, α) -partitionable if there exist representatives $u_1, \dots, u_k \in \Gamma(v)$ such that $|\bigcup_{i=1}^k C_{v,\alpha}(u_i)| \geq |\Gamma(v)| - \alpha \epsilon N$.

Note that if $G \in \mathcal{BUC}(H)$, then every vertex is $(\deg(H), 0)$ -partitionable.

Finally, we need the idea of a minimal witness against $\mathcal{BUC}(H)$. Informally, a minimal witness is a set of vertices such that the induced subgraph proves that G is not a valid blow-up collection, while any subset does not suffice.

Definition 9 (Minimal Witness). A set $S \subseteq V$ is a minimal witness against $\mathcal{BUC}(H)$ if $G|_S \notin \mathcal{BUC}(H)$, but for any $S' \subsetneq S$, $G|_{S'} \in \mathcal{BUC}(H)$.

Note that $G|_S \notin \mathcal{BUC}(H)$ implies that $G \notin \mathcal{BUC}(H)$. Furthermore, any minimal witness must be connected, since we could otherwise replace the witness with one of its connected components.

Given H , we let $W = W(H)$ denote the maximum size of a minimal witness. It is easy to see that $W < \frac{1}{2}h^2$. When H is a simple t -cycle, $t \geq 4$, $W = |H|$.

3 Adaptively Testing $\mathcal{BUC}(H)$ Given a Promise

We begin by showing how to adaptively test whether a graph G is in $\mathcal{BUC}(H)$, given the promise that G is $O(\epsilon)$ -close to \mathcal{LD}_{ce} .

The algorithm consists of two stages. The first stage (steps 2–8) tests whether most vertices are $(\Delta, 0)$ -partitionable, where $\Delta = \deg(H)$. To do so, it repeatedly selects a vertex and then attempts to find $\Delta + 1$ neighbors of that vertex which

have mutually distinct neighborhoods (see Figure 1(a)). If it finds such a set of vertices, they act as a witness against being in $\mathcal{BUC}(H)$ and the algorithm can reject with certainty in step 7.

If the algorithm fails to find such a witness, then most vertices must be (Δ, α) -partitionable, where α is a small constant. This implies that the graph is close to being a blow-up of some base graph. In other words, almost all of the vertices can be clustered such that each pair of clusters is either almost disjoint or almost forms a complete bipartite graph. The second stage of the algorithm (steps 9–16) checks if this high-level structure is consistent with $\mathcal{BUC}(H)$ (see Figure 1(b)). It does so by performing a random search in G for $W = W(H)$ steps, where each step selects a random neighbor of the previously selected vertices. If the selected vertices form a witness against $\mathcal{BUC}(H)$, the algorithm rejects in step 15.

Algorithm 10. ADAPTIVEBLOWUPCOLLECTIONTEST_{H,c}($G; \epsilon$)

1: Let $\Delta = \deg(H)$, $W = W(H)$, and $\alpha = (16\Delta|H|^2)^{-1}$.

\\ Test whether most vertices are $(\Delta, 0)$ -partitionable.

2: **for** $O(c)$ iterations **do**

3: Select a random vertex v .

4: Select a random set S of $O(\Delta\alpha^{-1}\epsilon^{-1})$ vertices, and query $\{v\} \times S$.

5: Let \bar{S} be a random subset of $\Gamma_S(v)$ of size (at most) $c\Delta\alpha^{-1}$.

6: Select a random set T of $O(\Delta^2\alpha^{-1}\epsilon^{-1})$ vertices, and query $\bar{S} \times T$.

7: If the current view of G is inconsistent with $\mathcal{BUC}(H)$, REJECT. Specifically, reject if there exist $u_1, \dots, u_{\Delta+1} \in \bar{S}$ such that $\Gamma_T(u_i) \neq \Gamma_T(u_j)$ for all i, j .

8: **end for**

\\ If G has not yet been rejected, most vertices must be (Δ, α) -partitionable.

\\ Test that G has a high-level structure consistent with H .

9: **for** $O(Wc)^{O(W)}$ iterations **do**

10: Select v_1 at random, and let $U = \{v_1\}$.

11: **for** $j = 2$ to W **do**

12: Select a random set T_j of $O(\Delta\epsilon^{-1})$ vertices, and query $U \times T_j$.

13: If $\Gamma_{T_j}(U) = \emptyset$, **break**. Otherwise, randomly select $v_j \in \Gamma_{T_j}(U)$ and let $U = U \cup \{v_j\}$.

14: **end for**

15: If $G|_U \notin \mathcal{BUC}(H)$, REJECT.

16: **end for**

17: If the algorithm hasn't yet rejected, ACCEPT.

Algorithm 10 has query complexity $O(\epsilon^{-1})$, as desired. Furthermore, it only rejects if it finds a witness against $\mathcal{BUC}(H)$, so the algorithm accepts valid blow-up collections with probability 1. The following lemma asserts that it rejects graphs which are far from $\mathcal{BUC}(H)$ with high probability.

Lemma 11. Let G be $\frac{\epsilon}{8}$ -close to \mathcal{LD}_{ce} and ϵ -far from $\mathcal{BUC}(H)$. Then Algorithm 10 rejects with probability at least $2/3$.

The proof is the content of section 3.1.

3.1 Proof of Lemma 11

Proofs of the lemmas in this section can be found in the full version of the paper.

We first note that if G contains many vertices which are not (Δ, α) -partitionable, then G is rejected with high probability.

Lemma 12. *Let $\Delta = \deg(H)$, $W = W(H)$, and $\alpha = (16\Delta|H|^2)^{-1}$. If G is $\frac{\epsilon}{8}$ -close to \mathcal{LD}_{ce} and contains at least $\frac{1}{4c}N$ vertices which are not (Δ, α) -partitionable, then Algorithm 10 rejects in step 7 with probability at least $2/3$.*

If G has at most $\frac{1}{4c}N$ vertices which are not (Δ, α) -partitionable, then G can be partitioned into components such that, for each pair of components, either the edges between them almost form a complete bipartite graph or the components are almost disjoint.

Lemma 13. *Let $\Delta = \deg(H)$ and $\alpha = (16\Delta|H|^2)^{-1}$. Suppose that G is $\frac{\epsilon}{8}$ -close to \mathcal{LD}_{ce} and contains at most $\frac{1}{4c}N$ vertices which are not (Δ, α) -partitionable. Then G is $\frac{5\epsilon}{8}$ -close to a graph $\tilde{G} = (V, \tilde{E})$ for which the following holds: V can be partitioned into $\bigcup_{i,j} V_j^i \cup L$ such that*

1. $\tilde{\Gamma}(v) = \emptyset$ for all $v \in L$,
2. $E(V^i, V^{i'}) = \emptyset$ for all $i \neq i'$,
3. For all i, j , $\tilde{\Gamma}(u) = \tilde{\Gamma}(v)$ for all $u, v \in V_j^i$, and
4. $|V_j^i| > \frac{\epsilon}{16\Delta}N$ for all i, j ,

where $V^i = \bigcup_j V_j^i$ and $\tilde{\Gamma}(u)$ is the neighborhood of u in \tilde{G} . Furthermore,
 $|\Gamma_{V^i}(u) \Delta \tilde{\Gamma}_{V^i}(u)| < \alpha\epsilon N$ for all $u \in V^i$.

Note that the bound on $|\Gamma_{V^i}(u) \Delta \tilde{\Gamma}_{V^i}(u)|$ in the lemma implies that \tilde{G} is $\frac{\epsilon}{8}$ -close to having degree at most $2c\epsilon N$, since G is $\frac{\epsilon}{8}$ -close to having degree at most $c\epsilon N$ and the degree of each vertex increases by at most $\alpha\epsilon N$ when going from G to \tilde{G} . Therefore, conditions 3 and 4 imply that most of the components V_j^i are connected to at most $32c\Delta$ other components and have size at most $2c\epsilon N$.

Intuitively, these conditions allow us to view each cluster as a supernode and each bipartite graph as an edge. From this viewpoint, \tilde{G} becomes a bounded-degree graph, with maximum degree $32c\Delta$. Since each supernode contains $\Omega(\epsilon N)$ vertices (by condition 4 in the lemma), we can simulate neighbor queries by querying $O(\epsilon^{-1})$ random vertices and choosing a random neighbor. Furthermore, if \tilde{G} is $\Omega(\epsilon)$ -far from $\mathcal{BUC}(H)$ when viewed as a dense graph, the corresponding bounded-degree graph is $\Omega(1)$ -far from $\mathcal{BUC}(H)$.

To formalize this intuition, we use the following lemma.

Lemma 14. *Let \tilde{G} be as in Lemma 13, and suppose that \tilde{G} is $\frac{3\epsilon}{8}$ -far from $\mathcal{BUC}(H)$. Then there exist at least $(16Wc^2\epsilon)^{-1}$ distinct sets $\tilde{W}_k = \bigcup_\ell V_{j_k,\ell}^{i_k}$ such that $G|_{\tilde{W}_k}$ is a witness against $\mathcal{BUC}(H)$.*

We are now ready to show that the second half of Algorithm 10 finds a witness against G with high probability.

By Lemma 14, step 10 of the algorithm selects a vertex $v_1 \in V_{j_1}^i$ corresponding to a witness with high probability. Having chosen v_1 , we wish to bound the probability that v_2, \dots, v_W are chosen so as to form a complete witness. Recalling that any minimal witness is connected, there must exist a $v_2 \in V_{j_2}^i$ which extends the witness in \tilde{G} . By condition 3 of Lemma 13, any vertex in $V_{j_2}^i$ can be used in place of v_2 to extend the witness. However, since $|V_{j_2}^i| > \frac{\epsilon}{16\Delta}N$ and $|\Gamma_{V^i}(u) \Delta \tilde{\Gamma}_{V^i}(u)| < \alpha\epsilon N < \frac{1}{2W}|V_{j_2}^i|$, at least half of $V_{j_2}^i$ is a valid choice to extend the witness in G . Furthermore, most of these must have degree $O(\epsilon\epsilon N)$.

Iterating this procedure, we see that there are always at least $\Omega(\frac{\epsilon}{\Delta}N)$ choices for v_j which extend the witness in G . Furthermore, since each previous $v_{j'}$ was chosen to have degree $O(\epsilon\epsilon N)$, there are at most $O(W\epsilon\epsilon N)$ potential choices for v_j . Therefore, with probability $\Omega((W^2c)^{-1})$, Algorithm 10 chooses a good v_j at each step and finds a complete witness in G . \square

4 Non-adaptively Testing $\mathcal{BUC}(H)$ Given a Promise

We now show how to non-adaptively test whether a graph is in $\mathcal{BUC}(H)$, given the promise that G is $O(\epsilon)$ -close to \mathcal{LD}_{ce} .

As in the adaptive case, the first stage of the algorithm verifies that most vertices are (Δ, α) -partitionable. Assuming the graph passes the first stage, the second stage checks that the high-level structure of G is consistent with $\mathcal{BUC}(H)$.

Since we can no longer adaptively restrict our queries to neighbors of a given vertex, we instead rely on the birthday paradox to achieve sub-quadratic query complexity. Recall that the birthday paradox says that given any distribution over a discrete domain \mathcal{D} , $O(|\mathcal{D}|^{1-1/k})$ samples suffice to obtain a k -way collision with high probability.

Very informally, a collision will correspond to choosing multiple vertices from a single witness in such a way that a W -way collision corresponds to a complete witness. Assuming that the set of witnesses has size $O(\epsilon^{-1})$ and that we can sample from that set with only constant overhead, the birthday paradox implies that $O(\epsilon^{-1+1/W})$ random vertices suffice to find all W vertices corresponding to a complete witness. Querying the entire induced subgraph then yields the desired result. The primary challenge, therefore, is to show that when G is ϵ -far from $\mathcal{BUC}(H)$, the algorithm can efficiently sample from the space of witnesses.

Algorithm 15. NONADAPTIVEBLOWUPCOLLECTIONTEST _{H,c} ($G; \epsilon$)

- 1: Let $\Delta = \deg(H)$, $W = W(H)$, and $\alpha = (16\Delta|H|^2)^{-1}$.
 $\backslash\backslash$ Check that most vertices are $(\Delta, 0)$ -partitionable.
- 2: Select a set S of $O((\alpha\epsilon)^{-1+1/(\Delta+2)})$ random vertices, and query $S \times S$.
- 3: Select a set T of $O(\Delta^2\alpha^{-1}\epsilon^{-1})$ random vertices, and query $S \times T$.
- 4: If the current view of G is inconsistent with $\mathcal{BUC}(H)$, REJECT. Specifically, reject if there exist $v \in S$ and $u_1, \dots, u_{\Delta+1} \in \Gamma_S(v)$ such that $\Gamma_T(u_i) \neq \Gamma_T(u_j)$ for all i, j .

\ \ Check that G has a high-level structure consistent with H .

- 5: Select a set S of $O((\Delta c^2 \epsilon)^{-1+1/W})$ random vertices and query $S \times S$.
- 6: If $G|_S \notin \mathcal{BUC}(H)$, REJECT.
- 7: If the algorithm hasn't yet rejected, ACCEPT.

Algorithm 15 has query complexity $O(\epsilon^{-2+1/(\Delta+2)} + \epsilon^{-2+2/W})$, as desired. Since it only rejects if it finds a witness against G , it accepts all graphs in $\mathcal{BUC}(H)$ with probability 1. It remains to show that it rejects graphs which are far from $\mathcal{BUC}(H)$ with high probability.

Lemma 16. *Let $\alpha = (16\Delta|H|^{-2})^{-1}$, as in Algorithm 15, and let G be $\frac{\alpha\epsilon}{16c}$ -close to $\mathcal{LD}_{c\epsilon}$ and ϵ -far from $\mathcal{BUC}(H)$. Then Algorithm 15 rejects with probability at least $2/3$.*

The proof of lemma 16 is given in the full paper.

5 Removing the Low-Degree Promise

We now show how to test the combined property of being a valid blow-up collection and having low-degree.

In [9], the authors give an $\tilde{O}(\epsilon^{-1})$ -query algorithm for testing whether the input has maximum degree $O(\epsilon N)$.

Lemma 17 ([9], Theorem 3). *Fix $c > 1$ and $\beta > 0$. There exists a non-adaptive tester with query complexity $\tilde{O}(\epsilon^{-1})$ and two-sided error which accepts graphs with maximum degree $c\epsilon N$ with probability at least $2/3$ and rejects graphs which are $\beta\epsilon$ -far from having maximum degree $c\epsilon N$ with probability at least $2/3$.*

The other key lemma states that if G is ϵ -far from $\mathcal{BUC}(H) \cap \mathcal{LD}_{c\epsilon}$, then it must be $\Omega(\epsilon)$ -far from $\mathcal{BUC}(H)$ or $\Omega(\epsilon)$ -far from $\mathcal{LD}_{c\epsilon}$.

Lemma 18. *Suppose that G is $\frac{\epsilon}{18c\Delta^2}$ -close to \mathcal{LD} and $\frac{\epsilon}{3}$ -close to $\mathcal{BUC}(H)$. Then G is ϵ -close to $\mathcal{LD} \cap \mathcal{BUC}(H)$.*

To test whether G is in $\mathcal{BUC}(H) \cap \mathcal{LD}_{c\epsilon}$, we therefore run the tester from Lemma 17, and if it accepts, we then test whether $G \in \mathcal{BUC}(H)$.

Proof (Theorems 3 and 4). We first run the tester from Lemma 17 with $\beta = (18c\Delta(H)^2)^{-1}$. If it accepts, we then run Algorithm 10 (in the adaptive case) or Algorithm 15 (in the non-adaptive case) with tolerance $\frac{\epsilon}{3}$.

If $G \in \mathcal{LD}_{c\epsilon} \cap \mathcal{BUC}(H)$, then the low-degree tester accepts with probability at $2/3$ and blow-up collection tester accepts with probability 1. So G is accepted with probability at least $2/3$.

Suppose that G is ϵ -far from $\mathcal{LD} \cap \mathcal{BUC}(H)$. By Lemma 18, either G is $\frac{\epsilon}{18c\Delta^2}$ -far from \mathcal{LD} or G is $\frac{\epsilon}{3}$ -far from $\mathcal{BUC}(H)$. In the first case, the low-degree tester rejects with probability at least $2/3$. In the second case, the blow-up collection tester rejects with probability at least $2/3$, by Lemma 11 (in the adaptive case) or Lemma 16 (in the non-adaptive case). \square

6 Conclusions

We have shown that there exist proximity-dependent graph properties for which a non-adaptive tester must suffer an almost-quadratic increase in its query complexity over an adaptive tester. This shows that the canonical transformation is essentially optimal, in the worst case.

The primary open question is to remove the proximity-dependence from the graph properties used in Theorem 1. In particular, for any $\delta > 0$, does there exist a *single* graph property Π such that testing Π requires $Q = \Omega(q^{2-\delta})$ queries for all $\epsilon > 0$? It also remains to show whether there exists a nearly-quadratic separation when the adaptive algorithm is only allowed one-sided error. One approach to both of these questions is to prove an $\tilde{O}(\epsilon^{-1})$ upper-bound for the adaptive query complexity of $BUC(H)$ for general graphs.

Finally, we reiterate the intriguing question raised in [4] as to what relationships are possible between the adaptive and non-adaptive query complexities. Specifically, do there exist properties such that $Q = \tilde{\Theta}(q^{2-\delta})$, with $\delta \neq \frac{2}{t}$? In particular, is it true that Q must either be $\tilde{\Theta}(q)$ or $\tilde{\Omega}(q^{4/3})$?

Acknowledgments. Thank you to Oded Goldreich and Chris Umans for very helpful comments and discussions about early versions of this work.

References

1. Alon, N.: Testing subgraphs in large graphs. *Random Struct. Algorithms* 21, 359–370 (2002)
2. Alon, N., Fischer, E., Newman, I., Shapira, A.: A combinatorial characterization of the testable graph properties: it's all about regularity. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 251–260. ACM, New York (2006)
3. Alon, N., Shapira, A.: A characterization of easily testable induced subgraphs. *Comb. Probab. Comput.* 15, 791–805 (2006)
4. Avigad, L.: On the lowest level of query complexity in testing graph properties. Master's project, Weizmann Institute of Science, Department of Computer Science and Applied Mathematics (December 2009), <http://www.wisdom.weizmann.ac.il/~oded/msc-la.html>
5. Avigad, L., Goldreich, O.: Testing Graph Blow-Up. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) RANDOM 2011 and APPROX 2011. LNCS, vol. 6845, pp. 389–399. Springer, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-22935-0_33
6. Goldreich, O., Krivelevich, M., Newman, I., Rozenberg, E.: Hierarchy theorems for property testing. In: Approximation Algorithms for Combinatorial Optimization, pp. 504–519 (2009)
7. Goldreich, O., Ron, D.: Algorithmic aspects of property testing in the dense graphs model. *SIAM Journal on Computing* 40(2), 376–445 (2011)
8. Goldreich, O., Trevisan, L.: Three theorems regarding testing graph properties. *Random Struct. Algorithms* 23, 23–57 (2003)
9. Gonen, M., Ron, D.: On the benefits of adaptivity in property testing of dense graphs. *Algorithmica* 58, 811–830 (2010)

Online Linear Optimization over Permutations

Shota Yasutake, Kohei Hatano, Shuji Kijima,
Eiji Takimoto, and Masayuki Takeda

Department of Informatics, Kyushu University
`{shouta.yasutake,hatano,kijima,eiji,takeda}@inf.kyushu-u.ac.jp`

Abstract. This paper proposes an algorithm for *online linear optimization problem over permutations*; the objective of the online algorithm is to find a permutation of $\{1, \dots, n\}$ at each trial so as to minimize the “regret” for T trials. The regret of our algorithm is $O(n^2\sqrt{T \ln n})$ in expectation for any input sequence. A naive implementation requires more than exponential time. On the other hand, our algorithm uses only $O(n)$ space and runs in $O(n^2)$ time in each trial. To achieve this complexity, we devise two efficient algorithms as subroutines: One is for minimization of an entropy function over the *permutohedron* P_n , and the other is for randomized rounding over P_n .

1 Introduction

Permutation is one of fundamental concepts in discrete mathematics and computer science. Permutations can naturally represent ranking or allocation of fixed objects. So, they have been applied to ranking in machine learning, information retrieval, recommendation tasks, and scheduling tasks.

More formally, a permutation σ over the set $[n] = \{1, \dots, n\}$ of n fixed objects is a bijective function from $[n]$ to $[n]$. Another popular way of representing a permutation σ over the set $[n]$ is to describe it as the n -dimensional vector in $[n]^n$, defined as $\sigma = (\sigma(1), \dots, \sigma(n))$. For example, $(3, 4, 2, 1)$ is a representation of a permutation for $n = 4$. Let S_n be the set of all permutations over $[n]$, i.e., $S_n = \{\sigma \in [n]^n \mid \sigma \text{ is a permutation over } [n]\}$. In particular, all permutations in S_n form a convex hull P_n , which is called permutohedron. Permutahedron has been studied extensively in submodular optimization [12][4] or scheduling problems [10][9][11].

We consider the following online linear optimization problem over S_n . For each trial $t = 1, \dots, T$, (i) the player predicts a permutation $\sigma_t \in S_n$, (ii) the adversary returns a loss vector $\ell_t \in [0, 1]^n$, and (iii) the player incurs loss $\sigma_t \cdot \ell_t$. The goal of the player is to minimize the regret: $\sum_{t=1}^T \sigma_t \cdot \ell_t - \min_{\sigma \in S_n} \sum_{t=1}^T \sigma \cdot \ell_t$.

To illustrate an application of our linear optimization problem, we consider the following online job scheduling problem with a single processor. Suppose that we have a single processor and n fixed jobs to be processed sequentially. Every day, we determine a schedule represented by a permutation in S_n a priori. Then, after processing all n jobs, we are given the processing time $\ell_i \in [0, 1]$ of each

job i (assume that each processing time is normalized up to 1). As a goal, we might want to minimize *the sum of the completion time* over all jobs and T days, where the completion time of job i is the sum of processing time of jobs prior to i plus the processing time of job i . For example, suppose that we process jobs according to a permutation $\sigma = (3, 2, 1, 4)$ and each processing time is given as $\ell = (\ell_1, \ell_2, \ell_3, \ell_4)$. Here, we interpret σ so that job i is processed with priority $\sigma(i)$. In other words, jobs with higher priority are processed earlier. So, we process jobs 4, 1, 2, and 3 sequentially. The completion time of jobs $i = 4, 1, 2, 3$ are ℓ_4 , $\ell_4 + \ell_1$, $\ell_4 + \ell_1 + \ell_2$, and $\ell_4 + \ell_1 + \ell_2 + \ell_3$, respectively. So, loss $\sigma \cdot \ell$ is exactly the sum of the completion time.

In this paper, we propose a randomized prediction algorithm whose expected regret is at most $O(n^2\sqrt{\log n}\sqrt{T})$, which is the best so far. For each trial, our algorithm runs in time $O(n^2)$ using $O(n)$ space. Note that, the competitive ratio converges to 1 as T increases under the natural assumption that the cumulative losses of the best permutation is $\omega(\sqrt{T})$.

There is a related previous algorithm, PermELearn [6] proposed by Helmbold and Warmuth, that is applicable to our problem though the algorithm was developed for generalized settings. It can be shown that PermELearn has the same regret bound of ours. However, the constant factor of the regret bound of PermELearn is roughly 4/3 worse than that of ours. Further, PermELearn needs $O(n^2)$ space and $\tilde{O}(n^6)$ running time. Our preliminary experimental results show that our algorithm indeed performs better than PermELearn for some artificial data.

There are other related researches. Online convex optimization (including online linear optimization) has been extensively studied in Machine Learning these days (see, e.g., [5]). Originally, the job scheduling problem we illustrated was considered in the offline setting [7, 8].

2 Preliminaries

For any fixed positive integer n , we denote $[n]$ by the set $\{1, \dots, n\}$. *Permutahedron* P_n is the convex hull of the set of permutations S_n . It is known that P_n coincides with the set of points $\mathbf{p} \in \mathbb{R}_+^n$ satisfying $\sum_{i \in S} p_i \leq \sum_{i=1}^{|S|} (n+1-i)$ for any $S \subset [n]$, and $\sum_{i=1}^n p_i = n(n+1)/2$. For references of the permutohedron, see, e.g., [12, 4].

The *unnormalized relative entropy* $\Delta(\mathbf{p}, \mathbf{q})$ from $\mathbf{q} \in \mathbb{R}_+^n$ to $\mathbf{p} \in \mathbb{R}_+^n$ is defined as $\Delta(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^n p_i \ln \frac{p_i}{q_i} + \sum_{i=1}^n q_i - \sum_{i=1}^n p_i$. It is known that $\Delta(\mathbf{p}, \mathbf{q}) \geq 0$ and $\Delta(\mathbf{p}, \mathbf{q}) = 0$ if and only if $\mathbf{p} = \mathbf{q}$. Unnormalized relative entropy is not symmetric in general, i.e., $\Delta(\mathbf{p}, \mathbf{q}) \neq \Delta(\mathbf{q}, \mathbf{p})$ for some $\mathbf{p}, \mathbf{q} \in \mathbb{R}_+^n$. Also, unnormalized relative entropy is a special case of Bregman divergence [2, 3], which generalizes Euclid distance or natural distance measures.

In this paper, for simplicity, we assume that calculation of any elementary functions (addition, multiplication, exponential functions, and etc.) can be done in a unit time.

3 Algorithm

In this section, we propose our algorithm PermutahedLearn and prove its regret bound.

3.1 Main Structure

The main structure of PermutahedLearn is shown in Algorithm 1. The algorithm maintains a weight vector \mathbf{p}_t in \mathbb{R}_+^n , which represents a mixture of permutations in S_n . At each trial t , it decomposes \mathbf{p}_t into permutations, chooses a permutation σ_t randomly according to its coefficient, and predicts the permutation σ_t . After the loss ℓ_t is assigned, PermutahedLearn updates the weight vector \mathbf{p}_t in a multiplicative way and projects it onto the permutohedron P_n .

The main structure of our algorithm itself is built on a standard technique in online learning literature (see, e.g., [6]). Our technical contribution is to develop efficient projection and decomposition techniques specifically designed for the permutohedron.

Algorithm 1. PermutahedLearn

1. Let $\mathbf{p}_1 = ((n+1)/2, \dots, (n+1)/2) \in [0, n]^n$.
 2. For $t = 1, \dots, T$
 - (a) Run **Decomposition**(\mathbf{p}_t) and get \mathbf{p}_t as $\mathbf{p}_t = \sum_{i=1}^k \lambda_i \sigma^{(i)}$, where $k \leq n$, each $\lambda_i \geq 0$, $\sum_i \lambda_i = 1$, and each $\sigma^{(i)}$ is in S_n .
 - (b) Choose σ_t randomly from $\{\sigma^{(1)}, \dots, \sigma^{(k)}\}$ according to the distribution λ .
 - (c) Incur a loss $\sigma_t \cdot \ell_t$.
 - (d) Update $\mathbf{p}_{t+\frac{1}{2}}$ as $p_{t+\frac{1}{2}, i} = p_{t,i} e^{-\eta \ell_{t,i}} / Z$, where Z is the normalization constant such that $\sum_{i=1}^n p_{t+\frac{1}{2}, i} = n(n+1)/2$.
 - (e) Run **Projection**($\mathbf{p}_{t+\frac{1}{2}}$) and get \mathbf{p}_{t+1} , the projection of $\mathbf{p}_{t+\frac{1}{2}}$ onto the permutohedron P_n . That is, $\mathbf{p}_{t+1} = \arg \inf_{\mathbf{p} \in P_n} \Delta(\mathbf{p}, \mathbf{p}_{t+\frac{1}{2}})$.
-

First of all, we prove a cumulative regret bound of weight vectors \mathbf{p}_t .

Lemma 1. *For any $T \geq 1$ it holds that*

$$\sum_{t=1}^T \mathbf{p}_t \cdot \ell_t \leq \frac{\eta \inf_{\mathbf{p} \in P_n} \sum_{t=1}^T \mathbf{p} \cdot \ell_t + \frac{n(n+1)}{2} \ln n}{1 - e^{-\eta}}.$$

The proof is standard in the online learning literature (see, e.g., [6]) and is omitted.

To complete our analysis for our algorithm, we specify the subroutines Projection and Decomposition, respectively, in the following subsections.

3.2 Projection

We propose an efficient algorithm *Projection* for computing the projection onto the permutohedron P_n . Formally, the problem is stated as follows: $\inf_{\mathbf{p}} \Delta(\mathbf{p}, \mathbf{q})$ subject to the constraints that

$$\sum_{j \in S} p_j \leq \sum_{j=1}^{|S|} (n+1-j), \text{ for any } S \subset [n], \text{ and } \sum_{j=1}^n p_j = \frac{n(n+1)}{2}. \quad (1)$$

Here we omit the positivity constraints $\mathbf{p} \geq \mathbf{0}$ since relative entropy projection always preserves positivity.

Apparently, this problem does not seem to be tractable as it has exponentially many constraints. But, we show that relevant constraints are only linearly many.

For simplicity, we assume that elements in \mathbf{q} are sorted in descending order, i.e., $q_1 \geq q_2 \geq \dots \geq q_n$. This can be achieved in time $O(n \log n)$ by sorting \mathbf{q} . First, we show that this projection preserves the order in \mathbf{q} .

Lemma 2. *Let \mathbf{p}^* be the projection of \mathbf{q} . Then we have $p_1^* \geq p_2^* \geq \dots \geq p_n^*$.*

Proof. Assume that the claim is false. Then, there exists $i \leq j$ such that $p_i^* < p_j^*$ and $q_i \geq q_j$. Let \mathbf{r} be the vector obtained by exchanging p_i^* and p_j^* in \mathbf{p}^* . Then,

$$\Delta(\mathbf{p}^*, \mathbf{q}) - \Delta(\mathbf{r}, \mathbf{q}) = p_i^* \ln \frac{q_j}{q_i} + p_j^* \ln \frac{q_i}{q_j} = (p_j^* - p_i^*) \ln \frac{q_i}{q_j} \geq 0,$$

where, the last inequality holds since $p_j^* \geq p_i^*$ and $q_i \geq q_j$. This contradicts the assumption that \mathbf{p}^* is the projection. \square

By Lemma 2, observe that if the conditions $\sum_{j=1}^i p_j \leq \sum_{j=1}^i (n+1-j)$, for $i = 1, \dots, n-1$, are satisfied, then other inequality constraints are satisfied as well since for any $S \subset [n]$ such that $|S| = i$, $\sum_{j \in S} p_j \leq \sum_{j=1}^i p_j$.

Therefore, the problem (II) is reduced to the same minimization problem whose constraints are linearly many:

$$\sum_{j=1}^i p_j \leq \sum_{j=1}^i (n+1-j), \text{ for } i = 1, \dots, n-1, \text{ and } \sum_{j=1}^n p_j = \frac{n(n+1)}{2}. \quad (2)$$

The KKT conditions (e.g., (I)) imply that \mathbf{p}^* is the projection if and only if \mathbf{p}^* satisfies the following conditions: There exist non-negative real numbers $\alpha_1, \dots, \alpha_{n-1}$ such that

$$\begin{aligned} p_i^* &= q_i e^{-\sum_{j=i}^{n-1} \alpha_j} / Z \quad (\text{for } i = 1, \dots, n-1), \quad p_n^* = q_n / Z, \\ \sum_{j=1}^i p_j^* &\leq \sum_{j=1}^i (n+1-j) \quad (\text{for } i = 1, \dots, n-1), \quad \sum_{i=1}^n p_i^* = \frac{n(n+1)}{2}, \\ \alpha_i \left(\sum_{j=1}^i p_j^* - \sum_{j=1}^i (n+1-j) \right) &= 0 \quad (\text{for } i = 1, \dots, n-1), \end{aligned} \quad (3)$$

Algorithm 2. Projection

Input: $q \in \mathbb{R}_+^n$ satisfying that $q_1 \geq q_2 \geq \dots \geq q_n$.

Output: projection p of q onto P_n .

1. Let $i_0 = 0$.
 2. **For** $t = 1, \dots,$
 - (a) Let $C_i^t = \frac{\sum_{j=1}^i (n+1-j) - \sum_{j=1}^{i_{t-1}} p_j}{\sum_{j=i_{t-1}+1}^i q_j}$ for $i = 1, \dots, n$ and $i_t = \arg \min_{i: i_{t-1} < i \leq n} C_i^t$.
If there are multiple minimizers, choose the largest one as i_t .
 - (b) Set $p_j = q_j C_{i_t}^t$ for $j = i_{t-1} + 1, \dots, i_t$.
 - (c) **If** $i_t = n$, **then** break.
 3. **Output** p .
-

where Z is the normalization constant so that $\sum_i p_i^* = n(n+1)/2$.

Now we describe the detail of the projection algorithm in Algorithm 2.

Lemma 3. (i) Given q , the algorithm Projection outputs the projection of q onto the permutohedron P_n . (ii) The time complexity of Projection is $O(n^2)$.

Proof. We show that there exists $\alpha_1, \dots, \alpha_{n-1}$ and Z such that the output p satisfies the optimality conditions (3), which completes the proof of the first statement.

First of all, we show that $C_{i_{t-1}}^{t-1} \leq C_{i_t}^t$ for each iteration t . Because of the definition of $C_{i_{t-1}}^{t-1}$, we have $C_{i_{t-1}}^{t-1} < C_{i_t}^{t-1}$. So, it suffices to prove that $C_{i_t}^{t-1} < C_{i_t}^t$. To see this, observe that

$$\sum_{i=1}^{i_{t-2}} p_j + C_{i_t}^{t-1} \sum_{j=i_{t-2}+1}^{i_t} q_j = \sum_{j=1}^{i_t} (n+1-j) = \sum_{i=1}^{i_{t-1}} p_j + C_{i_t}^t \sum_{j=i_{t-1}+1}^{i_t} q_j$$

and

$$\sum_{i=1}^{i_{t-1}} p_j + C_{i_t}^t \sum_{j=i_{t-1}+1}^{i_t} q_j < \sum_{i=1}^{i_{t-2}} p_j + C_{i_t}^{t-1} \sum_{j=i_{t-2}+1}^{i_{t-1}} q_j + C_{i_t}^t \sum_{j=i_{t-1}+1}^{i_t} q_j,$$

where the last inequality holds since $C_{i_{t-1}}^{t-1} < C_{i_t}^{t-1}$. By rearranging the inequalities above, we have $C_{i_t}^{t-1} < C_{i_t}^t$.

Now we fix each α_{i_t} so that $e^{-\alpha_{i_t}} C_{i_{t+1}}^{t+1} = C_{i_t}^t$, i.e., $\alpha_{i_t} = \ln(C_{i_{t+1}}^{t+1}/C_{i_t}^t)$ and fix Z to be $Z = C_n^T$, where T satisfies $i_T = n$. Note that since $C_{i_{t+1}}^{t+1} > C_{i_t}^t$, each α_{i_t} is strictly positive. For other $i \notin \{i_1, \dots, i_T\}$, we set $\alpha_i = 0$. Then, each p_{i_t} can be expressed as

$$p_{i_t} = q_{i_t} C_{i_t}^t = q_i e^{-\alpha_{i_t} - \alpha_{i_{t+1}} - \dots - \alpha_{i_T}} / Z = q_i e^{-\alpha_{i_t} - \alpha_{i_{t+1}} - \dots - \alpha_{n-1}} / Z.$$

Similarly, for other i such that $i_{t-1} < i < i_t$, we have

$$p_i = q_i C_{i_t}^t = q_i e^{-\alpha_{i_t} - \alpha_{i_{t+1}} - \dots - \alpha_{n-1}} / Z = q_i e^{-\alpha_i - \alpha_{i+1} - \dots - \alpha_{n-1}} / Z.$$

To see if the specified α_i s and Z satisfies the optimality conditions (3), observe that (i) for each i_t ,

$$\sum_{j=1}^{i_t} p_j = \sum_{j=1}^{i_{t-1}} p_j + \sum_{j=i_{t-1}+1}^{i_t} q_j C_{i_t}^t = \sum_j (n+1-j)$$

and $\alpha_{i_t} > 0$, and (ii) for each i such that $i_{t-1} < i < i_t$,

$$\sum_{j=1}^i p_j = \sum_{j=1}^{i_{t-1}} p_j + \sum_{j=i_{t-1}+1}^{i_t} q_j C_{i_t}^t \leq \sum_{j=1}^{i_{t-1}} p_j + \sum_{j=i_{t-1}+1}^{i_t} q_j C_i^t = \sum_j (n+1-j)$$

and $\alpha_i = 0$.

Finally, the algorithm terminates in time $O(n^2)$ since the number of iteration is at most n and each iteration takes $O(n)$ time, which completes the second statement of the lemma. \square

3.3 Decomposition

In this subsection, we describe how to represent a point $\mathbf{p} \in P_n$ by a convex combination of permutations. For simplicity assume that $p_1 \geq \dots \geq p_n$.

To begin with, we define special points in the permutohedron, which we call *permutations with ties*. Suppose $\mathbf{q} \in P_n$ satisfies that $q_1 \geq q_2 \geq \dots \geq q_n$. A permutation with ties $\mathbf{q} \in P_n$ satisfies that if $q_i > q_{i+1}$ then $\sum_{j=1}^i q_j = \sum_{j=1}^i (n+1-i)$ hold for any $i \in [n]$. For example, if $\mathbf{q} \in P_5$ satisfies $q_1 = q_2 > q_3 = q_4 = q_5$, then \mathbf{q} is uniquely determined as $\mathbf{q} = (4.5, 4.5, 2, 2, 2)$. Note that every permutation with ties \mathbf{q} satisfying that $q_1 \geq q_2 \geq \dots \geq q_n$ is represented by a convex combination of (at most) two permutations, namely $(\boldsymbol{\sigma} + \boldsymbol{\sigma}')/2$ where $\boldsymbol{\sigma} = (n, n-1, \dots, 1)$ and $\boldsymbol{\sigma}'$ is a “partially reversed” permutation satisfying that $\sigma'(i) > \sigma'(j)$ if $q_i > q_j$ and $\sigma'(i) < \sigma'(i+1)$ if $q_i = q_{i+1}$. Note that $\boldsymbol{\sigma}'$ is uniquely determined by the permutation with ties $\mathbf{q} \in P_n$. For example, let $\mathbf{q} = (4.5, 4.5, 2, 2, 2)$ and $\boldsymbol{\sigma} = (5, 4, 3, 2, 1)$, then its partially reversed permutation $\boldsymbol{\sigma}'$ is $(4, 5, 1, 2, 3)$. Further, for any positive vector $\mathbf{p} \in \mathbb{R}_+^n$ such that $p_1 \geq \dots \geq p_n$, we say that $\boldsymbol{\sigma}'$ is the partially reversed permutation w.r.t. \mathbf{p} if there exists a permutation with ties $\mathbf{q} \in P_n$ such that $p_i = p_{i+1}$ if and only if $q_i = q_{i+1}$ for each $i = 1, \dots, n-1$, $\boldsymbol{\sigma}'$ is the partially reversed permutation w.r.t. \mathbf{q} . Note that such \mathbf{q} is unique.

Now we describe our algorithm to represent $\mathbf{p} \in P_n$ with a convex combination of permutations. Note that $\boldsymbol{\sigma}^1 = \boldsymbol{\sigma}^0$ holds if the input \mathbf{p} satisfies that $p_i > p_{i+1}$ for any $i \in [n-1]$.

We will prove the following lemma on Decomposition.

Lemma 4. *Decomposition provides a convex combination of at most $n+1$ permutations representing an arbitrarily given $\mathbf{p} \in P_n$. Its running time is $O(n^2)$.*

To show Lemma 4, we show the following Lemmas.

Algorithm 3. Decomposition

Input: $\mathbf{p} \in P_n$ satisfying that $p_1 \geq p_2 \geq \dots \geq p_n$.

Output: Permutations $\sigma^0, \dots, \sigma^K$ and $\lambda_0, \dots, \lambda_K \in \mathbb{R}_{>0}$ s.t. $\sum_{i=0}^K \lambda_i \sigma^i = \mathbf{p}$, $\sum_{i=0}^K \lambda_i = 1$.

1. Let $\sigma^0 = (n, n-1, \dots, 1)$, $\mathbf{p}^1 = \mathbf{p}$ and $\lambda = 1$.

2. **For** $t = 1, \dots$,

(a) Find a partially reversed permutation σ^t with respect to \mathbf{p}^t and σ^0 . Let $\mathbf{q}^t = (\sigma^0 + \sigma^t)/2$.

(b) Let $\lambda_t = \min \left\{ \lambda, \min_{i \in [n-1]} \left\{ \frac{p_i^t - p_{i+1}^t}{q_i^t - q_{i+1}^t} \mid q_i^t \neq q_{i+1}^t \right\} \right\}$.

(c) Let $\mathbf{p}^{t+1} = \mathbf{p}^t - \lambda_t \mathbf{q}^t$ and let $\lambda = \lambda - \lambda_t$.

(d) **If** $\lambda=0$ **then** let $K = t$ and break.

3. Set $\lambda_0 = 1/2$ and $\lambda_t = \lambda_t/2$ for $t \in [K]$.

Output permutations $\sigma^0, \dots, \sigma^K$ and $\lambda_0, \dots, \lambda_K$.

Lemma 5. At any iteration t in Decomposition, \mathbf{p}^t satisfies that $p_i^t \geq p_{i+1}^t$ for any $i \in [n-1]$.

Proof. We give an inductive proof with respect to t . In case of $t = 1$, it is clear. In case of $t > 1$, we assume $p_i^{t-1} \geq p_{i+1}^{t-1}$ holds for any $i \in [n-1]$. If $p_i^{t-1} = p_{i+1}^{t-1}$, then $q_i^{t-1} = q_{i+1}^{t-1}$ holds, from the definition of \mathbf{q}^{t-1} . Thus

$$p_{\sigma(i)}^t = p_i^{t-1} - \lambda_{t-1} q_i^{t-1} = p_{i+1}^{t-1} - \lambda_{t-1} q_{i+1}^{t-1} = p_{i+1}^t$$

and we obtain the claim. If $p_i^{t-1} > p_{i+1}^{t-1}$, then $q_i^{t-1} > q_{i+1}^{t-1}$ holds, and

$$p_i^{t+1} - p_{i+1}^{t+1} = p_i^t - p_{i+1}^t - \lambda_t (q_i^t - q_{i+1}^t) = (q_i^t - q_{i+1}^t) \left(\frac{p_i^t - p_{i+1}^t}{q_i^t - q_{i+1}^t} - \lambda_t \right) \geq 0$$

where the last inequality becomes from the definition of λ_t , followed by

$$\lambda_t \leq \min_{i \in [n-1]} \left\{ \left(p_{i+1}^t - p_i^t \right) / \left(q_{i+1}^t - q_i^t \right) \mid q_{i+1}^t \neq q_i^t \right\}.$$

□

Lemma 6. In Decomposition, \mathbf{p}^{K+1} ($= \mathbf{p}^K - \lambda^K \mathbf{q}^K$) = 0 holds.

Proof. Without loss of generality, we may assume that $p_1 \geq p_2 \geq \dots \geq p_n$, for simplicity of notations. First we show $\mathbf{p}^{K+1} \geq 0$. Since Lemma 5, if there exists $j \in [n]$ satisfying that $p_j^{K+1} < 0$, then $p_n^{K+1} < 0$ holds. Thus it is enough to show $p_n^{K+1} \geq 0$. Let $i^* = \min\{j \in [n] \mid p_j^K = p_n^K\}$. Then we have $p_{i^*}^K = p_{i^*+1}^K = \dots = p_n^K$ and $q_{i^*}^K = q_{i^*+1}^K = \dots = q_n^K$. Hence, we get $p_{i^*}^{K+1} = p_{i^*+1}^{K+1} = \dots = p_n^{K+1}$. In case of $i^* \geq 2$, $p_{i^*-1}^t > p_{i^*}^t$ holds for any $t \in [K]$, meaning that $q_{i^*-1}^t > q_{i^*}^t$ holds for any $t \in [K]$. Thus we can see that $\sum_{j=i^*}^n q_j^t = \sum_{j=i^*}^n (n+1-j)$ holds for

any $t \in [K]$, from the definition of \mathbf{q}^t . Then we obtain

$$\sum_{j=i^*}^n \sum_{t=1}^K \lambda_t q_j^t = \sum_{t=1}^K \lambda_t \sum_{j=i^*}^n q_j^t = \sum_{t=1}^K \lambda_t \sum_{j=i^*}^n (n+1-j) = \sum_{j=i^*}^n (n+1-j) \leq \sum_{j=i^*}^n p_j$$

where the last inequality is due to constraints of the permutohedron $\sum_{j=1}^{i^*-1} p_j \leq \sum_{j=1}^{i^*-1} (n+1-j)$ and $\sum_{j=1}^n p_j = \sum_{j=1}^n (n+1-j)$. Thus we obtain that $\sum_{j=i^*}^n p_j^{K+1} = \sum_{j=i^*}^n (p_j - \sum_{t=1}^K \lambda_t q_j^t) \geq 0$. As discussed above, $p_{i^*}^{K+1} = p_{i^*+1}^{K+1} = \dots = p_n^{K+1}$ holds, and we obtain $p_n^{T+1} \geq 0$. In case of $i^* = 1$, the proof is done in a similar way.

Now we show $\mathbf{p}^{K+1} = 0$. Since $\mathbf{p} \in P_n$, $\sum_{j=1}^n p_j^{K+1} = \sum_{j=1}^n (n+1-j)$ holds. In a similar way as the proof of $\mathbf{p}^{K+1} \geq 0$,

$$\sum_{j=1}^n \sum_{t=1}^K \lambda_t q_j^t = \sum_{t=1}^K \lambda_t \sum_{j=1}^n q_j^t = \sum_{t=1}^K \lambda_t \sum_{j=1}^n (n+1-j) = \sum_{j=1}^n (n+1-j).$$

Since $\mathbf{p}^{K+1} \geq 0$, $\mathbf{p}^{K+1} = \mathbf{p} - \sum_{t=1}^K \lambda_t \mathbf{q}^t = 0$. \square

Lemma 7. *The number of iterations K is at most n .*

Proof. From the definition of λ_t , there is at least one $i \in [n]$ satisfying that $p_i^t > p_{i+1}^t$ and $p_i^{t+1} = p_{i+1}^{t+1}$. If $p_i^t = p_{i+1}^t$, then $p_i^{t+1} = p_{i+1}^{t+1}$ as discussed in the proof of Lemma 5. Now the claim is clear. \square

Proof of Lemma 4. Since Lemma 6, it is clear that the output $\sum_{t=0}^K \lambda_t \boldsymbol{\sigma}^t$ by Decomposition is equal to an arbitrarily given $\mathbf{p} \in P_n$.

It is not difficult to see that every lines in Decomposition is done in $O(n)$. Hence, we obtain that the running time is $O(n^2)$, since Lemma 7. Finally, we remark that $|\{\boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1, \dots, \boldsymbol{\sigma}^K\}| \leq n+1$ holds, the existence of such representation is suggested by well-known Caratheodory's theorem. \square

Memory-Efficient Implementation of Decomposition. Now we discuss an algorithm with $O(n)$ space and in $O(n^2)$ time to obtain a random permutation $\boldsymbol{\pi} \in \{\boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1, \dots, \boldsymbol{\sigma}^K\}$ according to the probability λ_t , using a modified version of Decomposition. Firstly notice that we do not need to memorize $\boldsymbol{\sigma}^t$ in Decomposition to compute λ_s and $\boldsymbol{\sigma}^s$ for $s > t$. Thus two-paths algorithm is easily obtained; Thus generate a random number $\xi \in (0, 1]$, and find $t \in \{1, \dots, K\}$ satisfying that $1 - \sum_{i=1}^{t-1} \lambda_i > \xi \leq 1 - \sum_{i=1}^t \lambda_i$, where $1 - \sum_{i=1}^0 0\lambda_i = 1$ for convenience, then $\boldsymbol{\sigma}^t$ is the desired sample in $\{\boldsymbol{\sigma}^0, \dots, \boldsymbol{\sigma}^K\}$ with probability λ_t .

In fact, we can reduce the time complexity of to $O(n \log n)$ using a heap with $O(n)$ space, though we omit the detail here.

3.4 Main Result

Now we are ready to prove the main result. Note that by using the algorithm Decomposition, $E[\boldsymbol{\sigma}_t \cdot \boldsymbol{\ell}_t] = \mathbf{p}_t$. So, by Lemma 4, we get the following theorem immediately.

Theorem 1. $E \left[\sum_{t=1}^T \boldsymbol{\sigma}_t \cdot \boldsymbol{\ell}_t \right] \leq \frac{\eta \min_{\boldsymbol{\sigma} \in S_n} \sum_{t=1}^T \boldsymbol{\sigma} \cdot \boldsymbol{\ell}_t + \frac{n(n+1)}{2} \ln n}{1 - e^{-\eta}}.$

In particular, if we set $\eta = 2 \ln(1 + \sqrt{\ln n}/\sqrt{T})$, since, $\eta \leq e^{\frac{\eta}{2}} - e^{-\frac{\eta}{2}}$ for $\eta \geq 0$, we have $\frac{\eta}{1 - e^{-\eta}} \leq e^{\frac{\eta}{2}} = (1 + \sqrt{\ln n}/\sqrt{T})$, and $\frac{1}{1 - e^{-\eta}} = \frac{(1+1/\sqrt{T})^2}{1/T+2/\sqrt{T}} \leq 1 + \frac{1}{2}\sqrt{T/\ln n}$. Further, by using the fact that $\boldsymbol{\sigma} \cdot \boldsymbol{\ell}_t \leq n(n+1)/2$, we get the following corollary.

Corollary 2. For $\eta = 2 \ln(1 + \sqrt{\ln n}/\sqrt{T})$, the expected regret of PermutahedLearn is at most $\frac{3n(n+1)}{4} \sqrt{T \ln n}$.

It can be shown that the regret bound of PermELearn is at most $n(n + 1/2) \sqrt{T \ln n}$, which is roughly $4/3$ times worse than that of PermtahedLearn. The proof is omitted due to the page constraint.

4 Experimental Results

In this section, we show our initial experiments of our algorithms for artificial data. For our artificial data, we fix $n = 10$. To generate a loss vector at each trial t , we specify each i -the element $\ell_{t,i}$ of the loss vector $\boldsymbol{\ell}_t$ independently randomly as follows: Let $\ell_{t,i} = 1$ with probability r_i and $\ell_{t,i} = 0$, otherwise. Here, we set $r_i = i/n$ so that $E[\boldsymbol{\ell}_t] = (1/n, 2/n, \dots, 1)$. We generate $T = 600$ random loss vectors.

The algorithms we compare are PermtahedLearn, PermELearn and the best permutation in hindsight. As the parameter η , we consider $\eta \in \{0.025, 0.05, 0.1, 0.2\}$. For each setting of η , we run algorithms for 3 times and choose the one attaining the lowest average cumulative losses as the best parameters for each of them. As a result, we specify $\eta = 0.2$ for PermtahedLearn and $\eta = 0.1$ for PermELearn, respectively.

We plot the regrets of algorithms with their best parameters in Fig. 1. As can be seen, the regret of PermtahedLearn is much smaller than that of PermELearn. This may be due to the fact that the regret bound of PermtahedLearn is constant times smaller than PermELearn.

5 Conclusion

In this paper, we propose an efficient algorithm for an online linear optimization problem over permutations of n items. Our algorithm has the best regret bound so far, runs in time $O(n^2)$ and uses $O(n)$ space at each trial.

An interesting future work includes proving a matching lower bound of the regret and investigating the case where permutations to predict have to meet some partial-order constraints.

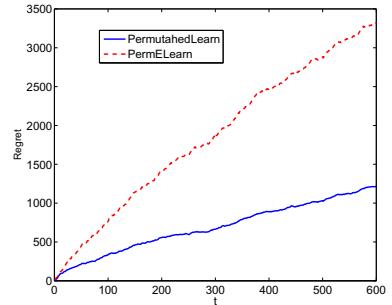


Fig. 1. Regrets of PermtahedLearn and PermELearn

Acknowledgement. We thank anonymous reviewers for their helpful comments. This work is supported in part by MEXT Grand-in-Aid for Young Scientists (B) 21700171.

References

1. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
2. Bregman, L.M.: The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR Computational Mathematics and Physics 7, 200–217 (1967)
3. Cesa-Bianchi, N., Lugosi, G.: Prediction, Learning, and Games. Cambridge University Press (2006)
4. Fujishige, S.: Submodular functions and optimization, 2nd edn. Elsevier Science (2005)
5. Hazan, E.: A survey: The convex optimization approach to regret minimization (2009), <http://www.cs.princeton.edu/~ehazan/papers/OCO-survey.pdf>
6. Helmbold, D.P., Warmuth, M.K.: Learning permutations with exponential weights. Journal of Machine Learning Research 10, 1705–1736 (2009)
7. Lawler, E.L.: On sequencing jobs to minimize weighted completion time subject to precedence constraints. Annals of Discrete Mathematics 2(2), 75–90 (1978)
8. Lenstra, J., Kan, A.R.: Complexity of scheduling under precedence constraints. Operations Research 26, 22–35 (1978)
9. Queyranne, M., Wang, Y.: Single-machine scheduling polyhedra with precedence constraints. Mathematics of Operations Research 16(1), 1–20 (1991)
10. von Arnim, A., Faigle, U., Schrader, R.: The permutohedron of series-parallel posets. Discrete Applied Mathematics 28(1), 3–9 (1990)
11. von Arnim, A., Schulz, A.S.: Facets of the generalized permutohedron of a poset. Discrete Applied Mathematics 72, 179–192 (1997)
12. Ziegler, G.M.: Lectures on Polytopes. Graduate Texts in Mathematics, vol. 152. Springer, Heidelberg (1995)

On the Best Possible Competitive Ratio for Multislope Ski Rental

Hiroshi Fujiwara, Takuma Kitano, and Toshihiro Fujito

Toyohashi University of Technology, 1-1 Tenpaku-cho, Toyohashi, 441-8580 Japan
`{h-fujiwara@,kitano@algo.,fujito@}cs.tut.ac.jp`

Abstract. The multislope ski-rental problem [11] is an extension of the classical ski-rental problem [10], where the player has several *lease* options in addition to the pure *rent* and *buy* options. In this problem an *instance*, which is the setting of the options, significantly affects the player's performance. There is an algorithm that for a given instance, computes the best possible strategy [1]. However, the output is given in numerical values and the relational nature between the instance and the best possible performance has not been known. In this paper we prove that even for the easiest instance, a competitive ratio smaller than $e/(e - 1) \approx 1.58$ cannot be achieved. More precisely, according to the number of options, tight bounds are obtained in a closed form. Furthermore, we establish a matching upper and lower bound of the competitive ratio each for the 3-slope and 4-slope problems.

1 Introduction

The *multislope ski-rental problem* [11] is an extension of the classical ski-rental problem [10], where the player has several *lease* options in addition to the pure *rent* and *buy* options. For example, a store may offer the following options for ski sets: Rent for \$50 per day, or buy for \$500, or lease for \$30 per day with initial fee of \$100, or another lease for \$15 per day with initial fee of \$250. Every time going skiing, the skier chooses one of these options, or keeps on the current choice.

We refer to a set of such options as an *instance*, denoted by (\mathbf{r}, \mathbf{b}) (the detailed definition shall appear in Section 2.) A *strategy* of the skier specifies when and to which option to switch. According to the standard definition, we say that a strategy achieves a *competitive ratio* of c if the skier along the strategy is charged at most c times the optimal offline cost, i.e., one with the number of times of skiing known in advance. For a given instance (\mathbf{r}, \mathbf{b}) , we define the *best possible competitive ratio* $\tilde{c}(\mathbf{r}, \mathbf{b})$ as the minimum value of the competitive ratios for all the possible strategies for (\mathbf{r}, \mathbf{b}) .

The numerical value of $\tilde{c}(\mathbf{r}, \mathbf{b})$ can be calculated by the algorithm of Augustine et al. [1]. However, almost nothing has been known of the dependencies between (\mathbf{r}, \mathbf{b}) and $\tilde{c}(\mathbf{r}, \mathbf{b})$. In this paper we analyze $\inf \tilde{c}(\mathbf{r}, \mathbf{b})$ and $\sup \tilde{c}(\mathbf{r}, \mathbf{b})$ over reasonable instances, revealing the easiest and the hardest instance. Notice

here that the supremum coincides with the matching upper and lower bound of the competitive ratio in the standard sense.

The analysis of the infimum is motivated by the following argument: This problem can be regarded as Dynamic Power Management [6] on a system that has multiple *energy-saving states*, for example, a Windows computer with *Stand By* state, *Hibernate* state, and so on. The objective is to minimize the energy consumption while there is no user response for an uncertain duration. A pair of a strategy and an instance that achieve the infimum make the best specification of energy-saving states and the best state-transition schedule.

Our Contribution. (I) For fixed $k \geq 2$, we prove that $\inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k)\} = (k+1)^k / ((k+1)^k - k^k)$, where $I(k)$ denotes the set of $(k+1)$ -slope instances having $k+1$ options. (For example, the above instance is in $I(3)$.) The infimum value monotonically decreases as k grows. For example, 1.80 for $k=2$, 1.73 for $k=3$, and 1.70 for $k=4$. We have a corollary: $\inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k), k \geq 2\} = e/(e-1) \approx 1.58$. The results are interpreted into the context of Dynamic Power Management as follows: The more energy-saving states are available, the better energy saving performance can be achieved. Nevertheless, there is a limit of improvement.

(II) We show $\sup\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(2)\} \approx 2.47$ and $\sup\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(3)\} \approx 2.75$. Please recall that the supremum is the matching upper and lower bound of the competitive ratio in the ordinary sense. The results establish a matching bound each for the 3-slope and 4-slope problems. Our analysis illustrates a different technique of seeking a matching bound from that in the conventional research, where an upper bound and a lower bound are separately considered. The results in (I) and (II) are graphically summarized in Figure II.

(III) We consider two subclasses of instances $I_A(k)$ and $I_I(k)$ which consist of *additive* instances and *investment* instances, respectively. (The definitions shall be given in Section 2.) We show $\sup\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I_A(k)\} = 2$ and $\inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I_I(k)\} = 2$ for any $k \geq 2$.

Related Work. We first mention studies on the deterministic model. The classical (i.e., 2-slope) ski-rental problem [10] was first introduced in the context of snoopy caching. The problem admits an optimal 2-competitive strategy. Various practical applications can be found in [7], such as context switching and virtual circuit management. In [6] the multislope ski-rental problem was discussed as Dynamic Power Management. Augustine et al. [1] developed an algorithm that for a given instance, outputs the best possible strategy and its competitive ratio. Bejerano et al. [2] provided a 4-competitive strategy for an arbitrary instance. Although their strategy was originally for an investment instance, the competitiveness straightforwardly applies to an arbitrary instance as well. Irani et al. [6] gave a 2-competitive strategy for an additive instance. For investment instances, Damaschke [3] gave a lower bound of $(5 + \sqrt{5})/2 (\approx 3.62)$. The Bahncard problem [5] is another extension of the 2-slope ski-rental problem.

Karlin et al. [9] provided an optimal randomized strategy for the 2-slope ski-rental problem with a competitive ratio of $e/(e-1)$. Karlin et al. [8] applied this

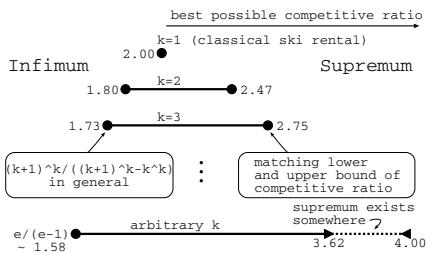


Fig. 1. Illustration of the range of the best possible competitive ratio for $(k+1)$ -slope ski-rental problem. The infimum/supremum is achieved by the easiest/hardest instance, respectively.

to several problems including TCP Acknowledgment. El-Yaniv et al. [4] studied the problem in a market with interest rates. Lotker et al. [12] analyzed a 2-slope problem with buy and lease options. For the multislope ski rental, in their other paper [11] they presented an ϵ -competitive randomized strategy for an arbitrary instance. Besides, they gave an $e/(e-1)$ -competitive randomized strategy for an additive instance, and also an algorithm that computes the best possible randomized strategy for a given additive instance.

2 Problem Statement and Preliminaries

A $(k+1)$ -slope ski-rental instance consists of a pair of two vectors (\mathbf{r}, \mathbf{b}) that specifies $k+1$ states which we have called options in Section II. Each state is associated with a per-time fee r_i . Hereafter we deal with the number of times of skiing as a nonnegative real number, so the per-time fee is regarded as a fee charged per unit time. A transition from state i to j can be done by paying $b_{i,j}$. There are two special states: State 0 with $b_{0,0} = 0$ and state k with $r_k = 0$, which correspond to *rent* and *buy*, respectively. States $1, \dots, k-1$ represent *lease* options of paying both a per-time and an initial fees. Without loss of generality we assume that the player starts from state 0 at time 0; he/she may transition to another state immediately. The problem with $k = 1$, i.e., with no lease states, is equivalent to the classical ski-rental problem.

The example of options at the beginning of Section II is now described as $(r_0, r_1, r_2, r_3) = (1, 0.6, 0.3, 0)$ and $(b_{0,1}, b_{0,2}, b_{0,3}, b_{1,2}, b_{1,3}, b_{2,3}) = (0.2, 0.5, 1, 0.3, 0.8, 0.5)$, after normalizing all the values so that (i) to buy a ski set costs 1 and (ii) to keep renting it by time 1 costs 1. Since our interest is not in the absolute cost but in the ratio of costs, these normalizations do not lose generality. In this example each transition cost between states is set as the difference of their initial costs. Figure 2 illustrates cost functions (introduced soon) based on this example.

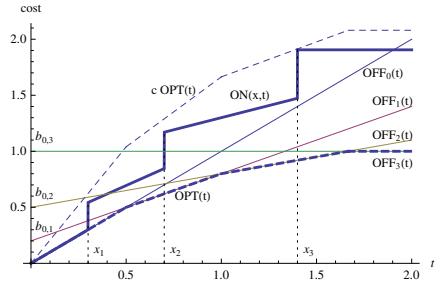


Fig. 2. Cost functions for a 4-slope ski-rental instance. $ON(x,t)$ jumps at $t = x_1, x_2$, and x_3 due to state transitions. Elsewhere it increases linearly with slope r_i . $OPT(t)$ transitions to an optimal state at the beginning.

We define the set of instances $I(k)$ to be the set of (\mathbf{r}, \mathbf{b}) such that:

$$1 = r_0 > r_1 > \cdots > r_k = 0, \quad 0 = b_{0,0} \leq b_{0,1} \leq \cdots \leq b_{0,k} = 1,$$

$$b_{l,j} - b_{l,i} \leq b_{i,j} \leq b_{l,j} \text{ for } 0 \leq l < i < j \leq k, \quad (1)$$

$$b_{0,i+1}(-r_{i-1} + r_i) + b_{0,i}(r_{i-1} - r_{i+1}) + b_{0,i-1}(-r_i + r_{i+1}) \leq 0 \text{ for } 1 \leq i \leq k-1. \quad (2)$$

$\{r_i\}$ is decreasing and $\{b_{0,i}\}$ is non-decreasing. The left inequality in (1) is a constraint that a transition cost cannot be saved by going through another state. Unless the right inequality in (1) holds, a transition from state i to j is more expensive than a transition from a state with a smaller index than i . Refer to [1] for the original motivation of these restrictions in Dynamic Power Management. We may thus assume that the player never transitions back to a state with a smaller index and so do not define $b_{j,i}$ ($0 \leq i < j \leq k$). The reason for (2) shall be clarified after the introduction of cost functions.

An *additive* instance in $I_A(k) \subset I(k)$ is such that $b_{i,j} = b_{i,l} + b_{l,j}$ holds for all $0 \leq i < l < j \leq k$: For a transition the player pays just the difference of the “potential” of state. One can confirm that the above example is in $I_A(3)$. An *investment* instance in $I_I(k) \subset I(k)$ is such that $b_{i,j} = b_{0,j}$ is satisfied for all $0 \leq i < j \leq k$: Each transition cost is independent of its origin.

A deterministic *strategy* of the player is a vector \mathbf{x} with $k+1$ entries. Each entry x_i stands for the time when the player transitions to state i . Since the player loses on a backward transition, the sequence of the entries is assumed to be non-decreasing. Also, we can fix $x_0 = 0$ even if the player goes to state $i > 0$ at time 0. The whole set of strategies is thus written as $S = \{\mathbf{x} \mid 0 = x_0 \leq x_1 \leq \cdots \leq x_k\}$. The player may transition from state i directly to j by skipping the states between. Then, we set $x_{i+1} = \cdots = x_{j-1} = x_j$ and define a relation of $i \prec j$. The online player with a strategy \mathbf{x} will have paid a cost of

$$ON(\mathbf{x}, t) := r_i(t - x_i) + \sum_{l=0}^{i-1} r_l(x_{l+1} - x_l) + \sum_{0 \leq l \prec m \leq i} b_{l,m}$$

by time t ($x_i < t < x_{i+1}$). For t being exactly a transition time x_i , we define the function as the cost immediately after the transition: $ON(\mathbf{x}, x_i) := \sum_{l=0}^{i-1} r_l(x_{l+1} - x_l) + \sum_{0 \leq l \prec m \leq i} b_{l,m}$.

The *optimal offline player* plays optimally with t known. It is observed that the optimal strategy is to transition to a state at time 0 and then keep staying there. The cost is

$$OPT(t) := \min_{0 \leq j \leq k} OFF_j(t),$$

where $OFF_j(t) := r_j t + b_{0,j}$ represents the cost of staying at state j from time 0 to t . In other words, $OPT(t)$ is the lower envelope of $\{OFF_i(t)\}$.

We now explain the reason of the condition (2) of $I(k)$. This means that every line $OFF_i(t)$ constitutes some part of $OPT(t)$, which is equivalent to the fact that there is no state that the optimal offline player never uses.

For an instance (\mathbf{r}, \mathbf{b}) , a strategy \mathbf{x} is said to be *c-competitive* if $ON(\mathbf{x}, t) - c \cdot OPT(t) \leq 0$ for all $t \geq 0$. c is called the *competitive ratio*. Competitiveness can be explained visually in Figure 2. If $ON(\mathbf{x}, t)$ is drawn below $c \cdot OPT(t)$, then strategy \mathbf{x} is *c-competitive*. We define the *best possible competitive ratio*

$$\tilde{c}(\mathbf{r}, \mathbf{b}) := \inf\{c \mid (\forall t \geq 0) ON(\mathbf{x}, t) - c \cdot OPT(t) \leq 0, \mathbf{x} \in S\}$$

for (\mathbf{r}, \mathbf{b}) . The condition can be weaken by the following two arguments. The first is a lemma that allows us to concentrate on strategies satisfying $ON(\mathbf{x}, t) = c \cdot OPT(t)$ at every transition time. Such a strategy is called *eager* in [1].

Lemma 1 ([1]). *Suppose that a strategy \mathbf{x} is c-competitive. Then, there is a c-competitive strategy \mathbf{x}' such that $ON(\mathbf{x}', x'_i) = c \cdot OPT(x'_i)$ holds for all $0 \leq i \leq k$.*

The second argument is on $OPT(t)$. The minimum operation can be eliminated by imposing $k+1$ constraints instead. Hence, we will hereafter employ

$$\tilde{c}(\mathbf{r}, \mathbf{b}) = \inf\{c \mid (0 \leq \forall i \leq k, 0 \leq \forall j \leq k) ON(\mathbf{x}, x_i) - c \cdot OFF_j(x_i) \leq 0, \mathbf{x} \in S\}.$$

The next theorem enables us to calculate the value of $\tilde{c}(\mathbf{r}, \mathbf{b})$ within arbitrary precision. However, the analytical relation between (\mathbf{r}, \mathbf{b}) and $\tilde{c}(\mathbf{r}, \mathbf{b})$ has not been known. In subsequent sections we analyze $\inf \tilde{c}(\mathbf{r}, \mathbf{b})$ and $\sup \tilde{c}(\mathbf{r}, \mathbf{b})$, revealing the easiest and the hardest instance.

Theorem 1 ([1]). *For any $\varepsilon > 0$, there exists an algorithm that for given (\mathbf{r}, \mathbf{b}) , computes a $(\tilde{c}(\mathbf{r}, \mathbf{b}) + \varepsilon)$ -competitive strategy in $O(k^2 \log k \log(1/\varepsilon))$ time.*

3 Infimum of the Best Possible Competitive Ratio

Fixed k . The infimum is written as

$$\begin{aligned} & \inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k)\} \\ &= \inf\{c \mid (0 \leq \forall i \leq k, 0 \leq \forall j \leq k) ON(\mathbf{x}, x_i) - c \cdot OFF_j(x_i) \leq 0, \mathbf{x} \in S, (\mathbf{r}, \mathbf{b}) \in I(k)\}. \end{aligned}$$

The simple lemma below implies that there is an instance in $I_A(k)$ which achieves the infimum. The proof will appear in the full version.

Lemma 2. *Let $(\mathbf{x}, \mathbf{r}, \mathbf{b}, c)$ be such that $\mathbf{x} \in S$, $(\mathbf{r}, \mathbf{b}) \in I(k)$, and $ON(\mathbf{x}, x_i) - c \cdot OFF_j(x_i) \leq 0$ for all $0 \leq i \leq k$ and $0 \leq j \leq k$. Set \mathbf{b}' as $b'_{i,j} := b_{0,j} - b_{0,i}$ for $0 < i < j \leq k$ and $b'_{0,i} := b_{0,i}$ for $0 \leq i \leq k$. Then, also for $(\mathbf{r}, \mathbf{b}') \in I_A(k)$, $ON(\mathbf{x}, x_i) - c \cdot OFF_j(x_i) \leq 0$ holds for all $0 \leq i \leq k$ and $0 \leq j \leq k$.*

For an additive instance the second term of $ON(\mathbf{x}, x_i)$ equals $b_{0,i}$, since the sum of transition costs is independent of through which states the online player has passed. So, let

$$g_{i,j}(\mathbf{x}, \mathbf{r}, \mathbf{b}, c) := ON(\mathbf{x}, x_i) - c \cdot OFF_j(x_i) = \sum_{l=0}^{i-1} r_l (x_{l+1} - x_l) + b_{0,i} - c(r_j x_i + b_{0,j}).$$

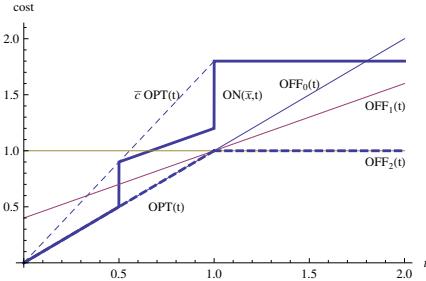


Fig. 3. Instance and strategy that achieve the infimum for $k = 2$

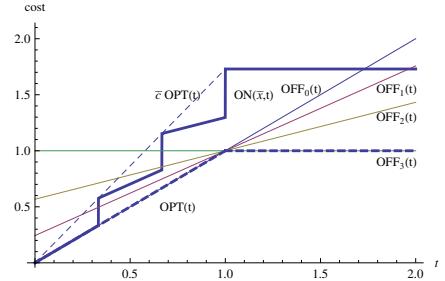


Fig. 4. Instance and strategy that achieve the infimum for $k = 3$

We thus formulate a mathematical program with variables \mathbf{x} , \mathbf{r} , \mathbf{b} , and c .

$$\begin{aligned} (\mathcal{P}) \quad & \text{minimize } c \\ & \text{subject to } g_{i,j}(\mathbf{x}, \mathbf{r}, \mathbf{b}, c) \leq 0, \quad \text{for } 0 \leq i \leq k, 0 \leq j \leq k, \\ & \mathbf{x} \in S, (\mathbf{r}, \mathbf{b}) \in I_A(k). \end{aligned}$$

In spite of its nonconvexity, we solve the problem analytically and obtain an explicit solution. The infimum is indeed the minimum.

Theorem 2. *The following $(\bar{\mathbf{x}}, \bar{\mathbf{r}}, \bar{\mathbf{b}}, \bar{c})$ is a global optimum to problem (\mathcal{P}) :*

$$\bar{x}_i = \frac{i}{k}, \quad \text{for } 0 \leq i \leq k, \quad (3)$$

$$\bar{r}_i = \bar{c} + (1 - \bar{c}) \left(1 + \frac{1}{k}\right)^i, \quad \text{for } 0 \leq i \leq k, \quad (4)$$

$$\bar{b}_{0,i} = 1 - \bar{r}_i, \quad \text{for } 0 \leq i \leq k, \quad (5)$$

$$\bar{b}_{i,j} = \bar{b}_{0,j} - \bar{b}_{0,i}, \quad \text{for } 0 < i < j \leq k, \quad (6)$$

$$\bar{c} = \frac{(k+1)^k}{(k+1)^k - k^k}. \quad (7)$$

Corollary 1. $\inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k)\} = \min\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k)\} = (k+1)^k / ((k+1)^k - k^k)$.

Before going to the proof, we give some numerical examples. For $k = 2$, we have $\bar{c} = \frac{9}{5} = 1.80$, $(\bar{x}_0, \bar{x}_1, \bar{x}_2) = (0, \frac{1}{2}, 1)$, $(\bar{r}_0, \bar{r}_1, \bar{r}_2) = (1, \frac{3}{5}, 0)$, and $(\bar{b}_{0,1}, \bar{b}_{0,2}, \bar{b}_{1,2}) = (\frac{2}{5}, 1, \frac{3}{5})$. And for $k = 3$, $\bar{c} = \frac{64}{37} \approx 1.73$, $(\bar{x}_0, \bar{x}_1, \bar{x}_2, \bar{x}_3) = (0, \frac{1}{3}, \frac{2}{3}, 1)$, $(\bar{r}_0, \bar{r}_1, \bar{r}_2, \bar{r}_3) = (1, \frac{28}{37}, \frac{16}{37}, 0)$, and $(\bar{b}_{0,1}, \bar{b}_{0,2}, \bar{b}_{0,3}, \bar{b}_{1,2}, \bar{b}_{1,3}, \bar{b}_{2,3}) = (\frac{9}{37}, \frac{21}{37}, 1, \frac{12}{37}, \frac{28}{37}, \frac{16}{37})$. Figures 3 and 4 show cost functions based on these instances and strategies. Please recall that $OPT(t)$ is the lower envelope of $\{OFF_i(t)\}$. One can see that in each figure, all other $OFF_i(t)$ than $OFF_0(t)$ and $OFF_k(t)$ degenerates to one point. This fact is interpreted that the choice

of the offline player is narrowed down to either of state 0 or k . On the other hand, the resulting strategy keeps transitioning to the next state at equal time intervals while exploiting all the states.

Our original interest was how much the online player can get advantage for the easiest instance. However, the resulting situation seems a bit different: The instance is unfavorable to the offline player, rather than easy to the online player. This would be a limit of worst-case competitive analysis.

Arbitrary k . We have

$$\bar{c} = 1 + \frac{1}{\left(1 + \frac{1}{k}\right)^k - 1} \rightarrow 1 + \frac{1}{e-1} = \frac{e}{e-1}$$

as $k \rightarrow \infty$, which provides the following corollary in a straightforward manner. The corollary is more interesting in Dynamic Power Management: Even if arbitrarily many energy-saving states can be implemented, there is a limit of improvement.

Corollary 2. $\inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k), k \geq 2\} = e/(e-1) \approx 1.58$. That is to say, for any instance, no strategy achieves a competitive ratio $\leq e/(e-1)$.

Proof (of Theorem 2). In this proof we will state some lemmas whose proofs will be given in the full version. First, by regarding \mathbf{r} in problem (\mathcal{P}) as a parameter, denoted by \mathbf{r}^* , we obtain a parametric optimization problem (\mathcal{Q}) .

$$\begin{aligned} (\mathcal{Q}) \quad & \text{minimize } c \\ & \text{subject to } g_{i,j}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}, c) \leq 0, \quad \text{for } 0 \leq i \leq k, 0 \leq j \leq k, \\ & \mathbf{x} \in S, (\mathbf{r}^*, \mathbf{b}) \in I_A(k). \end{aligned}$$

We guess a solution to problem (\mathcal{Q}) . Set $b_{0,i}^* := 1 - r_i^*$ for $0 \leq i \leq k$ and then $b_{i,j}^* := b_{0,j}^* - b_{0,i}^*$ for $0 < i < j \leq k$. We solve the equations $g_{i,0}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}^*, c) = 0$ for $0 \leq i \leq k$ and $g_{k,k}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}^*, c) = 0$. By taking the difference, we have for each $1 \leq i \leq k$,

$$g_{i+1,0}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}^*, c) - g_{i,0}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}^*, c) = -(x_{i+1} - x_i)(c - r_i^*) + r_i^* - r_{i+1}^* = 0.$$

Summing up $x_{i+1} - x_i$ yields

$$x_i = \sum_{l=0}^{i-1} \frac{r_l^* - r_{l+1}^*}{c - r_l^*}.$$

Also,

$$g_{k,k}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}^*, c) - g_{k,0}(\mathbf{x}, \mathbf{r}^*, \mathbf{b}^*, c) = -c + cx_k = 0$$

implies $x_k = 1$. Therefore c should be a root of

$$h(\mathbf{r}^*, c) := \sum_{i=0}^{k-1} \frac{r_i^* - r_{i+1}^*}{c - r_i^*} - 1 = 0.$$

Lemma 3. *The root of $h(\mathbf{r}^*, c) = 0$ uniquely exists between 1 and 2.*

We write the determined roots as \mathbf{x}^* and c^* . Set $x_0^* = 0$. Our guess turns out to be correct.

Lemma 4. *$(\mathbf{x}^*, \mathbf{b}^*, c^*)$ is a global optimum to problem (\mathcal{Q}) .*

Next, we regard c^* as the optimal value function to (\mathcal{Q}) and formulate a problem (\mathcal{R}) with \mathbf{r}^* being again a variable. Since problem (\mathcal{R}) is convex, a standard argument leads to an optimal solution.

$$\begin{aligned} (\mathcal{R}) \quad & \text{minimize } c^* \\ & \text{subject to } 1 = r_0 > r_1 > \dots > r_k = 0. \end{aligned}$$

Lemma 5. *$\bar{\mathbf{r}}$ in (4) is a global optimum to problem (\mathcal{R}) . The optimal value is \bar{c} in (7).*

We obtain $\bar{\mathbf{x}}$ and $\bar{\mathbf{b}}$ in (3), (5), and (6) by substituting $\bar{\mathbf{r}}$ in \mathbf{x}^* and \mathbf{b}^* . Lemmas 4 and 5 imply that $(\bar{\mathbf{x}}, \bar{\mathbf{r}}, \bar{\mathbf{b}}, \bar{c})$ is a global optimum to problem (\mathcal{P}) . \square

4 Supremum of the Best Possible Competitive Ratio

We first see for each of the fixed- k and arbitrary- k cases that $\sup \tilde{c}(\mathbf{r}, \mathbf{b})$ is equal to the matching upper and lower bound of the competitive ratio in the ordinary sense. In the literature of ski rental, c_u is said to be an upper bound if for all (\mathbf{r}, \mathbf{b}) , there exists a strategy \mathbf{x} which is c_u -competitive. It is observed that the set of such c_u is equivalent to the set of upper bounds of $\tilde{c}(\mathbf{r}, \mathbf{b})$ over arbitrary instances. Hence, if one identifies $\sup \tilde{c}(\mathbf{r}, \mathbf{b})$, it is equal to the least upper bound. On the other hand, c_l is called a lower bound if there exists (\mathbf{r}, \mathbf{b}) for which any \mathbf{x} is not c_l -competitive. Analogously, we have that $\sup \tilde{c}(\mathbf{r}, \mathbf{b})$ coincides with the greatest lower bound. We add that $c_l \leq \sup \tilde{c}(\mathbf{r}, \mathbf{b}) \leq c_u$ holds accordingly.

Fixed k . We present the supremum each for $k = 2$ and $k = 3$, whose proofs are left to the full version.

Theorem 3. *$\sup \{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(2)\}$ is the root of $c^3 - 4c^2 + 5c - 3 = 0$, which is approximately 2.47.*

Theorem 4. *$\sup \{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(3)\}$ is the root of $c^3 - 5c^2 + 8c - 5 = 0$, which is approximately 2.75.*

Below is the instance which asymptotically achieves each supremum: For $k = 2$, $(\bar{r}_0, \bar{r}_1, \bar{r}_2) = (1, \varepsilon_1, 0)$ and $(\bar{b}_{0,1}, \bar{b}_{0,2}, \bar{b}_{1,2}) = (0.68, 1, 1)$. For $k = 3$, $(\bar{r}_0, \bar{r}_1, \bar{r}_2, \bar{r}_3) = (1, \varepsilon_1, \varepsilon_2, 0)$ and $(\bar{b}_{0,1}, \bar{b}_{0,2}, \bar{b}_{0,3}, \bar{b}_{1,2}, \bar{b}_{1,3}, \bar{b}_{2,3}) = (0.40, 0.70, 1, 0.70, 1, 1)$. The ε 's are small positive numbers. See Figures 5 and 6. These instances are understood to be the hardest ones for the online player. Let us see the entries. The r_i 's of the intermediate states all approach zero. Also, they are investment instances. Indeed, $\bar{b}_{0,2} = \bar{b}_{1,2}$ for $k = 2$, and $\bar{b}_{0,2} = \bar{b}_{1,2}$ and

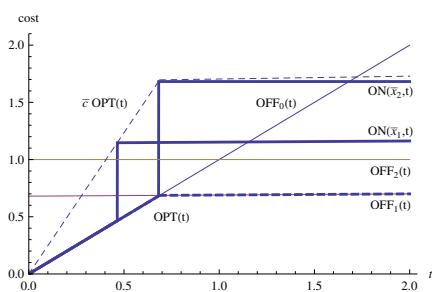


Fig. 5. Instance and strategies that asymptotically achieve the supremum for $k = 2$. \bar{r}_1 is set 0.01.

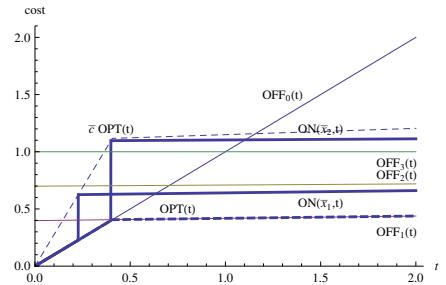


Fig. 6. Instance and strategies that asymptotically achieve the supremum for $k = 3$. \bar{r}_1 and \bar{r}_2 are set 0.02 and 0.01, respectively.

$\bar{b}_{0,3} = \bar{b}_{1,3} = \bar{b}_{2,3}$ for $k = 3$. From this observation we conjecture that for every k an investment instance achieves the supremum.

For each k , two different strategies asymptotically achieve the supremum. Note here that unlike the case of an additive instance, the choice of which states to skip affects the online player's cost. The strategies are: For $k = 2$, $\bar{x}_1 = (0, 0.68, 0.68)$ with $0 \prec 2$ (i.e., skipping state 1) and $\bar{x}_2 = (0, 0.47, 1/\delta_1)$ with $0 \prec 1 \prec 2$. And for $k = 3$, $\bar{x}_1 = (0, 0.23, 1/\delta_2, 1/\delta_2)$ with $0 \prec 1 \prec 3$ and $\bar{x}_2 = (0, 0.40, 0.40, 1/\delta_2)$ with $0 \prec 2 \prec 3$. δ 's are small positive numbers determined by ε 's in \bar{r} . Figures 5 and 6 illustrate these strategies as well.

Arbitrary k . It seems difficult to obtain a supremum for arbitrary k in the same way as we have done with fixed k . Nevertheless, the interval where the supremum can exist is implied by the known upper and lower bounds of the competitive ratio in the standard sense.

Theorem 5 ([3]). $(5 + \sqrt{5})/2 < \sup\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k), k \geq 2\}$.

Theorem 6 ([2]). $\sup\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I(k), k \geq 2\} \leq 4$.

5 Subclasses of the Multislope Ski Rental

Additive Instance. Theorem 2 and Corollary 2 provide the infimum for $I_A(k)$ as well, since Lemma 2 guarantees that an additive instance achieves the minimum.

We establish a supremum by combining with an existing result. We construct an instance for which any strategy cannot be $(2 - \varepsilon)$ -competitive. The proof will be provided in the full version. On the other hand, Irani et al. [6] provided the Lower Envelope strategy that is 2-competitive. Note that unlike the case of arbitrary instances, the supremum is constant regardless of k .

Theorem 7. Fix $k \geq 2$. Given $0 < \varepsilon < 1$, let $(\mathbf{r}, \mathbf{b}) \in I_A(k)$ be such that $1 = r_0 > r_1 > \dots > r_{k-1} = 1 - \varepsilon$, $r_k = 0$, $b_{0,i} = 1 - r_i$ for $0 \leq i \leq k$, and

$b_{i,j} = b_{0,j} - b_{0,i}$ for $0 \leq i < j \leq k$. Then, for any ε , no strategy is $(2 - \varepsilon)$ -competitive.

Theorem 8 ([6]). Fix $k \geq 2$. For any instance $(\mathbf{r}, \mathbf{b}) \in I_A(k)$, there is a 2-competitive strategy.

Corollary 3. For any $k \geq 2$, $\sup\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I_A(k)\} = 2$.

Investment Instance. We prove Theorem 9 which states that the infimum is independent of k . We will provide the proof in the full version.

As for the supremum, Theorems 3 and 4 hold true for $I_I(2)$ and $I_I(3)$, respectively, since an investment instance achieves each of the suprema. We mention that Theorem 6 was originally proved for $I_I(k)$.

Theorem 9. For any $k \geq 2$, $\inf\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I_I(k)\} = \min\{\tilde{c}(\mathbf{r}, \mathbf{b}) \mid (\mathbf{r}, \mathbf{b}) \in I_I(k)\} = 2$. The strategy and the instance that achieve the minimum are: $\bar{x}_0 = 0$ and $\bar{x}_1 = \bar{x}_2 = \dots = \bar{x}_k = 1$; $\bar{\mathbf{r}}$ is such that $1 = \bar{r}_0 > \bar{r}_1 > \dots > \bar{r}_k = 0$; $\bar{b}_{i,j} = 1 - \bar{r}_j$ for $0 \leq i < j \leq k$.

References

1. Augustine, J., Irani, S., Swamy, C.: Optimal power-down strategies. SIAM J. Comput. 37(5), 1499–1516 (2008)
2. Bejerano, Y., Cidon, I., Naor, J.: Dynamic session management for static and mobile users: a competitive on-line algorithmic approach. In: Proc. DIAL-M 2000, pp. 65–74 (2000)
3. Damaschke, P.: Nearly optimal strategies for special cases of on-line capital investment. Theor. Comput. Sci. 302(1-3), 35–44 (2003)
4. El-Yaniv, R., Kaniel, R., Linial, N.: Competitive optimal on-line leasing. Algorithmica 25(1), 116–140 (1999)
5. Fleischer, R.: On the Bahncard problem. Theor. Comput. Sci. 268(1), 161–174 (2001)
6. Irani, S., Shukla, S., Gupta, R.: Online strategies for dynamic power management in systems with multiple power-saving states. ACM Trans. Embed. Comput. Syst. 2(3), 325–346 (2003)
7. Karlin, A.R.: On the Performance of Competitive Algorithms in Practice. In: Fiat, A. (ed.) Online Algorithms 1996. LNCS, vol. 1442, pp. 373–384. Springer, Heidelberg (1998)
8. Karlin, A.R., Kenyon, C., Randall, D.: Dynamic TCP acknowledgement and other stories about $e/(e - 1)$. In: Proc. STOC 2001, pp. 502–509 (2001)
9. Karlin, A.R., Manasse, M.S., McGeough, L., Owicki, S.: Competitive randomized algorithms for nonuniform problems. Algorithmica 11(6), 542–571 (1994)
10. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 77–119 (1988)
11. Lotker, Z., Patt-Shamir, B., Rawitz, D.: Rent, lease or buy: Randomized algorithms for multislope ski rental. In: Proc. STACS 2008, pp. 503–514 (2008)
12. Lotker, Z., Patt-Shamir, B., Rawitz, D.: Ski rental with two general options. Inf. Process. Lett. 108(6), 365–368 (2008)

Input-Thrifty Extrema Testing

Kuan-Chieh Robert Tseng^{*} and David Kirkpatrick

Department of Computer Science,
University of British Columbia, Canada

Abstract. We study the complexity of one-dimensional extrema testing: given one input number, determine if it is properly contained in the interval spanned by the remaining n input numbers. We assume that each number is given as a finite stream of bits, in decreasing order of significance. Our cost measure, referred to as the *leading-input-bits-cost* (or LIB-cost for short), for an algorithm solving such a problem is the total number of bits that it needs to consume from its input streams.

An *input-thrifty algorithm* is one that performs favorably with respect to this LIB-cost measure. A fundamental goal in the design of such algorithms is to be more efficient on “easier” input instances, ideally approaching the minimum number of input bits needed to certify the solution, on all orderings of all input instances.

In this paper we present an input-thrifty algorithm for extrema-testing that is log-competitive in the following sense: if the best possible algorithm for a particular problem instance, including algorithms that are only required to be correct for presentations of this one instance, has worst-case (over all possible input presentations) LIB-cost c , then our algorithm has worst-case LIB-cost $O(c \lg \min\{c, n\})$.

In fact, our algorithm achieves something considerably stronger: if any input sequence (i.e. an arbitrary presentation of an arbitrary input set) can be tested by a monotonic algorithm (an algorithm that preferentially explores lower indexed input streams) with LIB-cost c , then our algorithm has LIB-cost $O(c \lg \min\{c, n\})$. Since, as we demonstrate, the cost profile of any algorithm can be matched by that of a monotonic algorithm, it follows that our algorithm is to within a log factor of optimality at the level of input sequences. We also argue that this log factor cannot be reduced, even for algorithms that are only required to be correct on input sequences with some fixed intrinsic monotonic LIB-cost c .

The extrema testing problem can be cast as a kind of list-searching problem, and our algorithm employs a variation of a technique called *hyperbolic sweep* that was introduced in that context. Viewed in this light, our results can be interpreted as another variant of the well-studied cow-path problem, with applications in the design of hybrid algorithms.

^{*} Many of the results appearing here appeared in the first author’s B.Sc. thesis entitled “Input Thrifty Algorithm for the Strict Containment Problem”, Computer Science Department, U.B.C., April 2009.

1 Introduction

1.1 Input-Thrifty Algorithms

In many problem settings, precise inputs can be expensive to obtain. Typically, the more precise the input is, the higher the cost incurred. For example, suppose the input is acquired from some physical measurements. Obtaining imprecise measurements may be cheap, but more precision may require the use of progressively more expensive technology. Similarly, even precise measurements of smoothly time-varying data may degrade in proportion to the delay in transmitting the data. Another possibility arises if the inputs are produced by a numerical algorithm, such as quadrature or bisection. Again, acquiring more precise input will require additional computation. In all such cases, we are motivated to solve the underlying problem using as little input precision as possible. Algorithms that minimize the total amount of input precision used to solve the problem are referred to as *input-thrifty algorithms*.

It is clear that for some problems, full input precision is required to provide a correct solution. Finding the sum or product of input numbers are two obvious examples. In such cases, all algorithms are trivially input-thrifty and are not interesting to analyze. However, for problems such as finding the maximum of a set of numbers, some but not all problem instances require full precision of all inputs. Thus, an important property of input-thrifty algorithms is that they are *adaptive*—they perform better on easier problem instances.

Consequently, we do not measure the effectiveness of input-thrifty algorithms by the total amount of precision required in the worst case over all problem instances of some fixed size. Instead, we compare the amount of precision required relative to the *intrinsic cost* of each individual problem instance - a measure of how “hard” the instance is. Intuitively, the intrinsic cost should be the cost paid by the best algorithm that solves the problem instance since every algorithm will have to incur at least this cost. Analysis of an algorithm A relative to the intrinsic cost of individual problem instances, amounts to a kind of competitive analysis, where the A is competing against the the best possible algorithm, sometimes most naturally modeled as an algorithm that exploits knowledge of the input not available to A .

1.2 The Uncertainty Model

The problem that we consider takes as input a sequence p_0, \dots, p_n of real numbers in the half-open interval $[0, 1)$. An algorithm can access each such number $p = \sum_{j>0} p^{(j)} 2^{-j}$ via its binary representation $p^{(1)}, p^{(2)}, \dots$, and this happens by examining the bits $p^{(j)}$ individually *in order of decreasing significance*, i.e. an algorithm can examine bit $p^{(j)}$ only after it has already examined bits $p^{(1)}$ through $p^{(j-1)}$.

Although the restriction of accessing input information one bit at a time in order of decreasing significance is perfectly natural, one could also adopt a more general model in which individual inputs are represented as a sequence of nested

uncertainty intervals. It turns out all of our results apply in this more general model. Nevertheless, we choose to first develop our results in the more restrictive, but less cumbersome, LIB-cost model.

1.3 Competitive Analysis

An algorithm that knows (or guesses) the numbers in the input sequence can, at least in principle, realize the *certifying* LIB-cost, defined as the minimum number of input bits needed to certify the solution for that input. In general, however, algorithms must deal not only with uncertainty concerning the *membership* of the input set (that completely determines its certifying LIB-cost) but also the *presentation* of these numbers as a sequence. For this reason, it is natural to choose, as a basis for comparison, the behavior of algorithms aggregated over all possible presentations of a given input set.

For a given presentation S_π of an input set S , the *LIB-cost* of a deterministic algorithm A on S_π refers to total number of bits that A consumes from its input streams. The *maximum-LIB-cost* (respectively, *average-LIB-cost*) of A is defined to be the maximum (respectively, average), over all input sequences S_π that are presentations of S , of the LIB-cost of A on S_π . The *intrinsic* maximum-LIB-cost (respectively, average-LIB-cost) of an input set S is defined to be the minimum, over all algorithms B that are only guaranteed to solve the problem for inputs that are presentations of S , of the maximum-LIB-cost (respectively, average-LIB-cost) of B over S .

With this notion of intrinsic cost in hand, it is possible to do a *competitive analysis*¹ with respect to the family of algorithms that are constrained to answer correctly only when the input sequence is a presentation of S (i.e. relative to what we have called the intrinsic LIB-cost of S). Algorithms that are competitive on all instances are said to be *instance-efficient*; if the competitive ratio is a constant, they are *instance-optimal* (cf. [7], [1]).

While our results can be interpreted in this light, we are able to claim something that ties more closely to the certification cost of individual input sequences. Specifically, we define a restriction on algorithms that they explore input sequences in a monotonically decreasing fashion. Such algorithms are shown to be universal in the sense that the LIB-cost profile (the set of all LIB-costs over all presentations of an input set S) of an arbitrary algorithm A is exactly matched by the LIB-cost profile of some monotonic algorithm B . We then define the monotonic-certification LIB-cost of an input sequence S_π to be the minimum, over all monotonic algorithms, of the number of input bits needed to certify the solution for that input. We demonstrate an algorithm that is log-competitive on all presentations of all inputs with respect to the monotonic-certification LIB-cost: if input sequence S_π has monotonic-certification LIB-cost c , then our

¹ We speak of competitive analysis by analogy with the analysis (cf. [14]) of the performance of on-line algorithms (whose access to input information is restricted) in relation to the performance of the most efficient off-line algorithm (which have full access to the input).

algorithm has LIB-cost $O(c \lg \min\{c, n\})$. Furthermore, we argue that this log-factor cannot be reduced, even for algorithms that are only required to be correct on input sequences with fixed monotonic-certification LIB-cost c .

1.4 Extrema Testing

Our study originated with the following question posed by Leo Guibas at a Computational Geometry Seminar held at Schloss Dagstuhl in 1993:

Given n points in the plane, does their convex hull contain the origin?

Suppose further that the points have $\lg n$ -bit coordinates. How many bits of the input do we need to examine to answer the question, in the worst case? More specifically, if there is a proof using only d bits, can we find it by examining only $O(d)$ bits?

Guibas' *origin-containment problem* is a natural example of a problem that can be addressed in our framework (where the input numbers are the coordinates of the points in question). Furthermore, the LIB-cost of algorithms, and the comparison with the certifying LIB-cost of the input, capture the essence of the associated questions.

In general, determining the LIB-complexity of certifying geometric predicates in two or more dimensions has proved to be quite difficult. We consider instead the following problem that is easily seen to be equivalent to the one-dimensional version of Guibas' origin-containment problem: suppose we are given a set S of real numbers in the interval $[0, 1)$, and another real number $a \in [0, 1)$, do there exist numbers x and y in S such that a is contained in the interval (x, y) ? Clearly, solving this problem requires either (i) certifying that a is extreme in $S \cup \{a\}$, that is either $a \leq x$, for all $x \in S$ or $a \geq x$, for all $x \in S$, or (ii) certifying that a is not extreme, by demonstrating x and y in S such that $x < a < y$.

Certifying that a is extreme has basically a *fixed cost*, dependent on S but independent of the presentation of S . This follows because the construction of a certificate that $a < x$, for every $x \in S$, must entail the exploration of at least k_x bits, for each $x \in S$, where k_x denotes the index of the first bit position at which x and a differ. Thus, in both the worst and average cases, the intrinsic LIB-cost of certifying that a is extreme is just $\sum_{x \in S} k_x$. For this reason we hereafter focus on the more interesting case where a is *not* extreme.

Note that certifying that a is not extreme entails certifying that there exists a number x in S such that $a \neq x$. This non-degeneracy problem has been explored in detail in [12]. Without loss of generality, we assume such an x exists and that $x < a$. So our problem reduces to determining whether there exists a number $y \in S$ such that $a < y$.

We begin by modeling our problem as a kind of signed list traversal problem in Section 2. As it turns out, the worst case can be addressed by a straightforward modification of techniques for standard (unsigned) list traversal developed in [12]. Thus we focus our attention, in Section 3, on lower bounds for the intrinsic cost of individual presentations of the list traversal problem, relative to list traversal algorithms of a structurally-restricted but not complexity-restricted

form. In section 4, we prove that the cost incurred by a modified hyperbolic sweep algorithm is within a logarithmic factor of this intrinsic cost on all input presentations, and that this logarithmic factor cannot be eliminated. Due to space restrictions, some of the proofs are only sketched; interested readers can find full details in an expanded version of this paper [15].

2 Preliminaries

2.1 Related Work

As we have already indicated the results of this paper build directly on those of [12] where the *LIB-cost* model, input-thrifty algorithms and the hyperbolic dovetailing technique were all introduced in the context of a basic list-searching problem: given a set of lists with unknown lengths, traverse to the end of any one of the lists with as little total exploration as possible. Hyperbolic dovetailing was shown to provide a search cost within a logarithmic factor of the optimal (intrinsic) search cost for all problem instances, in both the worst and average case over all instance presentations.

A variety of other papers have dealt with the solution of fundamental computational problems in the presence of imprecise input. For example, Khanna *et al.* [11] analyzed how to compute common database aggregate functions such as sum and mean of a set of numbers, and Feder *et al.* [8] analyzed how to compute the median of a set of numbers. Other applications include finding the MST [6] and the convex hull [4]. Although the fundamental issues are clearly related, it should be noted that the framework used in these studies differs from our model in that they assume a fixed (indivisible) cost to determine specified inputs to (typically) full precision, whereas we insist that algorithms obtain extra precision of inputs in an incremental fashion.

2.2 The List Traversal Model

For each element in $x \in S$, we can certify that $x \neq a$ by obtaining some minimum number k_x of bits from the input stream of x (i.e. the k_x^{th} bit is the most significant bit on which x and a differ); if $x = a$ we take k_x as infinity. We model the input stream for x as a list with length k_x . We mark the end of the list with a “+”, if the k_x^{th} bit of x is greater than the corresponding bit of a , and “-” otherwise. Thus, we can model S as a set of lists with unknown lengths, each marked with a positive or negative sign at the end. An algorithm is allowed to query any list at an integer position k , provided it has already queried that same list at all lower-indexed positions. If the list is queried at its last position, the query returns the corresponding sign. The goal is to find a + mark, at the end of any list, using the minimum number of queries.

We model a deterministic algorithm A as a ternary tree. At each node, the index of the list to query next is specified. Depending on the outcome, which could be either finding the end of a negative/positive list or simply discovering

that the end of the list has not yet been reached, we traverse down one of the left/right/middle edges of the tree respectively. We make the assumption that the algorithm terminates if it has encountered a positive sign, and it never queries a list again if it has confirmed that the list is a negative list. Randomized algorithms can be modeled simply as probability distributions over the set of all deterministic algorithms.

The cost incurred by a list searching algorithm depends on both the input set and its presentation. The *maximum cost* of a deterministic algorithm, for a particular input, is the maximum number of queries the algorithm makes over all possible presentations of that input. The *average cost* of a deterministic algorithm is the number of queries the algorithm makes averaged over all possible presentations. It should be clear that the maximum and average costs correspond directly to the maximum and average LIB-cost of certifying that $a < y$, for some $y \in S$.

We will denote the list traversal problem described above as the *POSITIVE_LIST_TRAVERSAL* (PLT, for short) problem. The input for the problem is described by a set $\Lambda = \Lambda^+ \cup \Lambda^-$, where Λ^+ is the set of positive lists, marked with a + sign at the end, and Λ^- is the set of negative lists, marked with a – sign at the end. Let $n = |\Lambda^+|$ and $m = |\Lambda^-|$ and denote by λ_i^+ (resp., λ_i^-) the length of the i^{th} shortest list in Λ^+ (resp., Λ^-). Thus, $\lambda_1^+ \leq \lambda_2^+ \leq \dots \leq \lambda_n^+$ and $\lambda_1^- \leq \lambda_2^- \leq \dots \leq \lambda_m^-$. Since the length of the list is the only property we are interested in, we will sometimes abuse notation, referring to the i^{th} shortest positive (resp., negative) list itself as λ_i^+ (resp., λ_i^-). When we are describing presentations of the input, it is convenient to adopt a *standard presentation* $\langle \lambda_1^+, \dots, \lambda_n^+, \lambda_1^-, \dots, \lambda_m^- \rangle$ and describe other presentations as permutations of this standard presentation. Specifically, if π is any permutation of $\langle 1, 2, \dots, n+m \rangle$ then we denote by Λ_π the presentation $\langle \lambda_{\pi(1)}, \lambda_{\pi(2)}, \dots, \lambda_{\pi(n+m)} \rangle$.

3 Intrinsic Cost

For a competitive analysis of algorithms for the PLT problem, it is desirable to have some sort of intrinsic cost associated with individual problem instances (that is, some way of distinguishing between “easy” and “hard” instances). Given a deterministic algorithm A and a fixed input instance $\Lambda = \Lambda^+ \cup \Lambda^-$, the cost of A on a particular presentation Λ_π is denoted by $c_A(\Lambda_\pi)$. With this we can define the *certification cost* of Λ_π to be minimum, over all deterministic algorithms A that behave correctly on Λ_π , of the cost $c_A(\Lambda_\pi)$. Using the certification cost as a notion of intrinsic cost for competitive analysis would be equivalent to comparing a given algorithm that knows nothing about the input with a competitor that knows not only the input instance Λ but also its presentation Λ_π . A more realistic notion that takes into account the uncertainty associated with the given presentation can be formulated by aggregating costs over all possible presentations of Λ .

An even better approach that retains some of the sensitivity of certification cost of specific presentations, without presupposing an all-knowing competitor,

is to restrict the class of algorithms to a normal form that reflects the inherent uncertainty associated with the order of inputs. For every deterministic algorithm A and every input instance Λ , we can define the associated *cost profile*, which is just the multiset $\{c_A(\Lambda_\pi) \mid \Lambda_\pi \text{ is a presentation of } \Lambda\}$. We say that two algorithms are *cost-equivalent* on Λ if their cost profiles are identical.

A deterministic algorithm A , is said to be *monotonic* with respect to Λ_π if at every stage in the algorithm, for any $i < j$, the number of queries made in list $\lambda_{\pi(i)}$ is greater or equal to the number of queries made in list $\lambda_{\pi(j)}$, unless $\lambda_{\pi(i)}$ has been confirmed to be a negative list. The algorithm is monotonic with respect to Λ if it is monotonic with respect to all possible presentations of Λ . At first glance monotonicity may seem overly restrictive. However, since our analysis is with respect to all possible input presentations, it is at least intuitively clear that, when two lists are indistinguishable, nothing is lost by exploring the lower indexed list preferentially. The following lemma makes this intuition rigorous.

Lemma 1. *Given a fixed input Λ . For any deterministic algorithm A , there exists a deterministic monotonic algorithm \hat{A} that is cost-equivalent to A on Λ .*

Proof. (Sketch) We have argued that arbitrary deterministic algorithms can be expressed as ternary trees. We show how to systematically transform the tree representation of A into a tree representation of a cost-equivalent algorithm \hat{A} . We label each tree node by the set of presentations of Λ that lead to that node and the query profile (the number of times each list has been probed in leading to that node). We then establish a bijection between the two trees that preserves (i) the parent relation, (ii) the size of the set of associated presentations, and (iii) the query profiles (up to permutations). By construction, all query profiles in the tree representing \hat{A} are monotonic. \square

We are now in a position to provide a more robust definition of intrinsic cost that coincides with the aggregated certification costs described earlier but is more realistic at the level of individual presentations. We define the *monotonic-certification cost* of Λ_π , denoted $\xi(\Lambda_\pi)$ to be minimum, over all deterministic monotonic algorithms \hat{A} that behave correctly on Λ_π , of the cost $c_{\hat{A}}(\Lambda_\pi)$.

Lemma 2. $\xi(\Lambda_\pi) = \min_{1 \leq i \leq n} \sum_{p=1}^{\pi^{-1}(i)} \min\{\lambda_{\pi(p)}, \lambda_i\}$.

Proof. Suppose that algorithm \hat{A} terminates after discovering the end of list λ_k^+ at position $\pi^{-1}(k)$ in the input presentation. Then, by the monotonicity of \hat{A} ,

$$c_{\hat{A}}(\Lambda_\pi) \geq \sum_{p=1}^{\pi^{-1}(k)} \min\{\lambda_{\pi(p)}, \lambda_k^+\} \geq \min_{1 \leq i \leq n} \sum_{p=1}^{\pi^{-1}(i)} \min\{\lambda_{\pi(p)}, \lambda_i\}$$

But if i^* satisfies $\sum_{p=1}^{\pi^{-1}(i^*)} \min\{\lambda_{\pi(p)}, \lambda_{i^*}\} = \min_{1 \leq i \leq n} \sum_{p=1}^{\pi^{-1}(i)} \min\{\lambda_{\pi(p)}, \lambda_i\}$ then the monotonic algorithm that explores lists, in order, to depth $\lambda_{i^*}^+$ realizes the minimum possible cost on Λ_π . \square

4 Hyperbolic Sweep

For the problem where every list is marked with a positive sign, Kirkpatrick [12] proposed a hyperbolic sweep algorithm whose cost is only a logarithmic factor more than the aggregated intrinsic cost. In this section, we extend the hyperbolic sweep algorithm to solve the current problem. We first recall the hyperbolic sweep algorithm. A phase of the algorithm involves iterating through every list and extending the exploration up to a depth determined by the list index and the phase number. We define the *rank* of the t^{th} position on the i^{th} list as $\sum_{j=1}^i \min\{\lambda_{\pi(j)}, t\}$. At the completion of phase $r \geq 1$ all list positions of rank at most r have been explored.

HYPERBOLIC-SWEEP(Λ_π)

```

 $r \leftarrow 1;$ 
while (end of a positive list has not been encountered)
  for ( $i = 1; i \leq m + n; ++i$ )
    continue exploration of list  $\pi(i)$  up to the last position of rank
    at most  $r$ 
   $r \leftarrow r + 1;$ 

```

Theorem 1. *Given a fixed input sequence Λ_π , the cost of the hyperbolic sweep algorithm is $O(\xi(\Lambda_\pi) \lg \min\{\xi(\Lambda_\pi), n + m\})$.*

Proof. Suppose the hyperbolic sweep terminates with $r = \alpha$. We consider the total number of queries by considering how many lists has been queried at least once, twice, etc. By the definition of rank, the number of lists that has been queried h times is at most $\min\{\alpha/h, n + m\}$. Thus, the total number of queries is:

$$\sum_{h=1}^{\alpha} \min\left\{\frac{\alpha}{h}, n + m\right\} = O(\alpha \lg \min\{\alpha, n + m\}).$$

It suffices to sum up to α since no list will be queried more than α times by the definition of rank. Finally, note that hyperbolic sweep will terminate successfully when $r = \xi(\Lambda_\pi)$. \square

Thus hyperbolic dovetailing achieves the monotonic-certification cost of *every* presentation of *every* input instance, up to at most a logarithmic factor. It is not hard to see that this logarithmic factor cannot be avoided. In particular, if there are no negative lists the problem reduces to the standard list searching problem studied in [12]. In that setting, a family of lists was described that force a logarithmic competitive ratio. In our setting, this same family shows that any deterministic monotonic algorithm that is only required to be correct on input presentations whose monotonic-certification cost is c , has cost $c \lg \min\{c, n + m\}$ for at least one such sequence.

5 Conclusion

This paper has investigated the problem of determining if a given number is extreme with respect to a given set S of numbers. We recast the problem as

a signed list-search problem. One of the main results is the specification of a notion of intrinsic cost (monotonic-certification cost) for all presentations of all instances of this problem. This is made possible by a normal-form result that shows that every algorithm that is tailored to a specific problem instance has a cost-equivalent monotonic algorithm for that instance. Our second contribution is a modified hyperbolic sweep algorithm, which can be shown to solve the signed list-search problem within a logarithmic factor of the monotonic-certification cost of every presentation of every problem instance. Finally, we argued that this log-competitiveness cannot be improved, even by algorithms that know the monotonic-certification cost of their input sequence.

The question investigated in this paper has several natural generalizations. Foremost, it still remains to settle Guibas' original origin-containment problem, the problem that motivated the investigation in this paper, in two or higher dimensions. It should be noted that although the list traversal model was developed to model the LIB-cost of certain fundamental problems defined on integers presented as a stream of bits, it can also be used to model the more general situation where inputs are presented as a sequence of nested uncertainty intervals. For our problem it suffices to interpret a positive (resp., negative) sign in the t^{th} position of list i as an assertion that the t^{th} uncertainty interval associated with the i^{th} input x_i is disjoint from and larger (resp., smaller) than the t^{th} uncertainty interval associated with the number a . Under this interpretation, all of our results extend to this more general uncertainty model.

Finally, we remark that the signed list-search problem has applications that go beyond the realm of graph search. For example, following a similar idea presented by Kao [10], suppose we wish to solve a generally intractable problem and we have at our disposal several heuristics, demonstrated to be effective for certain classes of problem instances. On a particular problem instance, some heuristics might perform well, while others will not help at all. The problem then is to determine, without knowing the nature of the problem instance, how we should dovetail among the available heuristics. This can clearly be modeled by the list-traversal problem, where each list represents the full computation associated with one heuristic, and the depth of traversal at any point in time corresponds to the resource allocated to the associated heuristic. A positive list represents a heuristic that completes successfully and a negative list represents a heuristic that completes unsuccessfully.

Acknowledgements. The authors gratefully acknowledge helpful discussions with Raimund Seidel.

References

1. Afshani, P., Barbay, J., Chan, T.M.: Instance-optimal geometric algorithms. In: 50th Annual IEEE Symposium on Foundations of Computer Science, pp. 129–138 (2009)

2. Azar, Y., Broder, A.Z., Manasse, M.S.: On-line choice of on-line algorithms. In: Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 432–440 (1993)
3. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Information and Computation 106(2), 234–252 (1993)
4. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theory of Computing Systems, 411–423 (2005)
5. Dorrigiv, R., Lopez-Ortiz, A.: A survey of performance measures for on-line algorithms. ACM SIGACT News 36(3), 67–81 (2005)
6. Erlebach, T., Hoffmann, M., Krizanc, D., Mihal'ák, M., Raman, R.: Computing minimum spanning trees with uncertainty. ArXiv e-prints (2008)
7. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proc. 20th ACM Symposium on Principles of Database Systems, pp. 102–113 (2001)
8. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. In: Proc. 32nd Annual ACM Symposium on Theory of Computing, pp. 602–607 (2000)
9. Kao, M.-Y., Littman, M.L.: for informed cows. In: AAAI 1997 Workshop on On-Line Search (1997)
10. Kao, M.-Y., Ma, Y., Sipser, M., Yin, Y.: Optimal constructions of hybrid algorithms. J. Algorithms 29(1), 142–164 (1998)
11. Khanna, S., Tan, W.-C.: On computing functions with uncertainty. In: Proc. 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 171–182 (2001)
12. Kirkpatrick, D.: Hyperbolic dovetailing. In: Proc. European Symposium on Algorithms, pp. 516–527 (2009)
13. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. In: Proc. Second Israel Symposium on Theory of Computing and Systems, Jerusalem, pp. 128–133 (June 1993)
14. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Comm. ACM 28, 202–208 (1985)
15. <http://people.cs.ubc.ca/~kirk/Papers/ISAAC2011+.pdf>

Edit Distance to Monotonicity in Sliding Windows

Ho-Leung Chan¹, Tak-Wah Lam^{1,*}, Lap-Kei Lee², Jiangwei Pan¹,
Hing-Fung Ting¹, and Qin Zhang²

¹ Department of Computer Science, University of Hong Kong, Hong Kong
`{hlchan, twlam, jwpan, hfting}@cs.hku.hk`

² MADALGO**, Department of Computer Science, Aarhus University, Denmark
`{lklee, qinzhang}@madalgo.au.dk`

Abstract. Given a stream of items each associated with a numerical value, its edit distance to monotonicity is the minimum number of items to remove so that the remaining items are non-decreasing with respect to the numerical value. The space complexity of estimating the edit distance to monotonicity of a data stream is becoming well-understood over the past few years. Motivated by applications on network quality monitoring, we extend the study to estimating the edit distance to monotonicity of a sliding window covering the w most recent items in the stream for any $w \geq 1$. We give a deterministic algorithm which can return an estimate within a factor of $(4 + \epsilon)$ using $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$ space.

We also extend the study in two directions. First, we consider a stream where each item is associated with a value from a partial ordered set. We give a randomized $(4 + \epsilon)$ -approximate algorithm using $O(\frac{1}{\epsilon^2} \log \epsilon^2 w \log w)$ space. Second, we consider an out-of-order stream where each item is associated with a creation time and a numerical value, and items may be out of order with respect to their creation times. The goal is to estimate the edit distance to monotonicity with respect to the numerical value of items arranged in the order of creation times. We show that any randomized constant-approximate algorithm requires linear space.

1 Introduction

Estimating the sortedness of a numerical sequence has found applications in, e.g., sorting algorithms, database management and webpage ranking (such as PageRank [4]). For example, sorting algorithms can take advantage of knowing the sortedness of a sequence so as to sort efficiently [9]. In relational database, many operations are best performed when the relations are sorted or nearly sorted over the relevant attributes [3]. Maintaining an estimate on the sortedness of the relations can help determining whether a given relation is sufficiently nearly-sorted or a sorting operation on the relation (which is expensive) is needed. One common measurement of sortedness of a sequence is its *edit distance to monotonicity*

* T.W. Lam was supported by the GRF Grant HKU-713909E.

** Center for Massive Data Algorithmics – a Center of the Danish National Research Foundation.

(or ED, in short) [2,7,8,10,11]: given a sequence σ of n items, each associated with a value in $[m] = \{1, 2, \dots, m\}$, the ED of σ , denoted by $\text{ed}(\sigma)$, is the minimum number of edit operations required to transform σ to the sequence obtained by sorting σ in non-decreasing order. Here, an edit operation involves removing an item and re-insert it into a new position of the sequence. Equivalently, $\text{ed}(\sigma)$ is the minimum number of items in σ to delete so that the remaining items have non-decreasing values. A closely related measurement is the *length of the longest increasing subsequence* (or LIS) of σ , denoted by $\text{lis}(\sigma)$. It is not hard to see that $\text{lis}(\sigma) = n - \text{ed}(\sigma)$.

With the rapid advance of data collection technologies, the sequences usually appear in the form of a data stream, where the stream of items is massive in size (containing possibly billions of items) and the items are rapidly arriving sequentially. This gives rise to the problem of estimating ED in the data stream model: An algorithm is only allowed to scan the sequence sequentially in one pass, and it also needs to be able to return, at any time, an estimate on ED of the items arrived so far. The main concern is the space usage and update time per item arrival, which, ideally, should both be significantly smaller than the total data size (preferably polylogarithmic).

Estimating ED of a data stream is becoming well-understood over the past few years [8,10,11]. Gopalan et al. [11] showed that computing the ED of a stream exactly requires $\Omega(n)$ space even for randomized algorithms, where n is the number of items arrived so far. They also gave a randomized $(4 + \epsilon)$ -approximate algorithm for estimating ED using space $O(\frac{1}{\epsilon^2} \log^2 n)$, where $0 < \epsilon < 1$. Later, Ergun and Jowhari [8] improved the result by giving a deterministic $(2 + \epsilon)$ -approximate algorithm using space $O(\frac{1}{\epsilon^2} \log^2(\epsilon n))$. For the closely related LIS problem, Gopalan et al. [11] also gave a deterministic $(1 + \epsilon)$ -approximate algorithm for estimating LIS using $O(\sqrt{\frac{n}{\epsilon}})$ space. This space bound is proven to be optimal in [10].

ED in sliding windows. The above results consider the sortedness of all items in the stream arrived so far, which corresponds to the *whole stream model*. Recently, it is suggested that ED can be an indicator of network quality [12]. The items of the stream correspond to the packets transmitted through a network, each associated with a sequence number. Ideally, the packets would arrive in increasing order of the sequence number. Yet network congestion would result in packet retransmission and distortion in the packet arrival order, which leads to a large ED value. One of the main causes to network congestion is that traffic is often bursty. Thus, the network quality can be measured more accurately if the measurement is based on only recent traffic. To this end, we propose studying the *sliding window model* where we estimate the ED of a window covering the latest w items in the stream. Here w is a positive integer representing the window size. The sliding window model is no easier than the whole data stream model because when w is set to be infinity, we need to estimate ED for all items arrived.

Our results. We give a deterministic $(4 + \epsilon)$ -approximate algorithm for estimating ED in a sliding window. The space usage is $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$, where w is the

window size. Our algorithm is a generalization of the algorithm by Gopalan et al. [11]. In particular, Gopalan et al. show that ED of the whole stream can be approximated by the number of “inverted” items j such that many items arrived before j has a value bigger than j . We extend this definition to the sliding window model. Yet, maintaining the number of inverted items in a sliding window is non-trivial. An item j may be inverted when it arrives, but it may become not inverted due to the expiry of items *arrived earlier*. We give an interesting algorithm to estimate the number of inverted items using existing results on basic counting and quantile estimation over sliding windows. Our algorithm also incorporates an idea in [8] to remove randomization.

We also consider two extensions of the problem.

- *Partial ordered items.* In some applications, each item arrived is associated with multiple attributes, e.g., a network packet may contain both the IP address of the sender and a sequence number. To measure the network quality, it is sometimes useful to estimate the *most congested* traffic coming from a particular sender. This corresponds to estimating the ED of packets with respect to sequence number from the same sender IP address. In this case, only sequence numbers with the same IP address can be ordered. We model such a situation by considering items each associated with a value drawn from a partial ordered universe. We are interested in estimating the minimum number of items to delete so that the remaining items are sorted with respect to the partial order. We give a randomized $(4 + \epsilon)$ -approximate algorithm using $O(\frac{1}{\epsilon^2} \log \epsilon^2 w \log w)$ space.

- *Out-of-order streams.* When a sender transmits packets to a receiver through a network, the packets will go through some intermediate routers. To measure the quality of the route between the sender and an intermediate router, it is desirable to estimate the ED of the packets received by the router from the sender. Yet in some cases, the router may not be powerful enough to deploy the algorithm for estimating the ED. We consider delegating the task of estimation to the receiver. To model the situation, whenever a packet arrives, the intermediate router marks in the packet a timestamp recording the number of packets received thus far (which can be done by maintaining a single counter). Hence, when the packets arrive at the receiver, each packet has both a sequence number assigned by the sender and a timestamp marked by the router. Note that the packets arrived at the receiver may be out-of-order with respect to the timestamp. Such stream corresponds to an *out-of-order stream*.

To measure the network quality between the sender and the router, the receiver can estimate the ED with respect to the sequence number when the items are arranged in increasing order of the timestamps. Intuitively, the problem is difficult as items can be inserted in arbitrary positions of the sequence according to the timestamp. We show strong space lower bounds even in the whole stream model. In particular, any randomized constant-approximate algorithm for estimating ED of an out-of-order stream requires $\Omega(n)$ space, where n is the number of items arrived so far. An identical lower bound holds for estimating the LIS. Like most streaming lower bounds, our lower bounds are proved based on reductions from two communication problems, namely, the INDEX problem

and the DISJ problem. Optimal communication lower bounds for randomized protocols are known for both problems [114].

Organization. Section 2 and 3 give the formal problem definitions and our main algorithm for estimating ED, respectively. Section 4 considers out-of-order streams. Due to the page limit, extension to partial ordered items is left to the full paper.

2 Formal Problem Definitions

Sortedness of a stream. Consider a stream σ of n items, $\langle \sigma(1), \sigma(2), \dots, \sigma(n) \rangle$ where each $\sigma(i)$ is drawn from $[m] = \{1, 2, \dots, m\}$. The *edit distance to monotonicity* (ED) of σ , denoted by $\text{ed}(\sigma)$, is the minimum number of items required to remove so as to obtain an increasing subsequence of σ , i.e., $\langle \sigma(i_1), \sigma(i_2), \dots, \sigma(i_k) \rangle$ such that $\sigma(i_1) \leq \sigma(i_2) \leq \dots \leq \sigma(i_k)$ for some $1 \leq i_1 < i_2 < \dots < i_k \leq n$. We use $\text{lis}(\sigma)$ to denote the *length of the longest increasing subsequence* (LIS) of σ . Note that $\text{lis}(\sigma) = n - \text{ed}(\sigma)$. The sortedness can be computed based on the *whole stream* (all items in σ received thus far) or a *sliding window* covering the most recent w items, denoted by σ_w , for $w \geq 1$. Note that the whole stream model can be viewed as a special case of the sliding window model with window size $w = \infty$. A streaming algorithm has only limited space and can only maintain an estimate on the sortedness of σ_w . For any $r \geq 1$, a r -approximate algorithm for estimating $\text{ed}(\sigma_w)$ returns, at any time, an estimate $\widehat{\text{ed}}(\sigma_w)$ such that $\text{ed}(\sigma_w) \leq \widehat{\text{ed}}(\sigma_w) \leq r \cdot \text{ed}(\sigma_w)$. We can define a r -approximate algorithm for estimating $\text{lis}(\sigma_w)$ similarly.

Partial ordered universe. We also consider a partial ordered universe with binary relation \preceq . A subsequence of σ with length ℓ , $\langle \sigma(i_1), \sigma(i_2), \dots, \sigma(i_\ell) \rangle$, is increasing if for any $k \in [\ell - 1]$, $\sigma(i_k) \preceq \sigma(i_{k+1})$. Then for any window size $w \geq 1$, $\text{ed}(\sigma_w)$ and $\text{lis}(\sigma_w)$ can be defined analogously as before.

Out-of-order stream. The data stream described above is an *in-order* stream, which assumes items arriving in the same order as their creation time. In an *out-of-order stream*, each item is associated with a distinct integral time-stamp recording its creation time, which may be different from its arrival time. Precisely, an out-of-order stream σ is a sequence of tuples $\langle t_i, v_i \rangle$ ($i \in [n]$) where t_i and v_i are the timestamp and value of the i -th item. The sortedness of σ is defined based on the permuted sequence $V(\sigma) = \langle v_{i_1}, v_{i_2}, \dots, v_{i_n} \rangle$ such that $t_{i_1} \leq t_{i_2} \leq \dots \leq t_{i_n}$, i.e., $\text{ed}(\sigma) := \text{ed}(V(\sigma))$ and $\text{lis}(\sigma) := \text{lis}(V(\sigma))$.

3 A $(4 + \epsilon)$ -Approximate Algorithm for Estimating ED

In this section, we consider a stream σ of items with values drawn from a set $[m] = \{1, 2, \dots, m\}$, and we are interested in estimating the ED of a sliding window covering the most recent w items in σ . We give a deterministic $(4 + \epsilon)$ -approximate algorithm which uses $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$ space.

Our algorithm is based on an estimator $R(i)$, which is a generalization of the estimator in [11] to the sliding window model. Let i be the index of the latest arrived item. The sliding window we consider is $\sigma_{[i-w+1,i]} = \langle \sigma(i-w+1), \sigma(i-w+2), \dots, \sigma(i) \rangle$. For any item $\sigma(j)$, let $inv(j)$ be the set of items arrived before $\sigma(j)$ but have greater values than $\sigma(j)$, i.e., $inv(j) = \{k : k < j \text{ and } \sigma(k) > \sigma(j)\}$. We define an estimator $R(i)$ for $ed(\sigma_{[i-w+1,i]})$ as follows.

Definition 1. Consider the current sliding window $\sigma_{[i-w+1,i]}$. We define $R(i)$ to be the set of indices $j \in [i-w+1, i]$ such that there exists $k \in [i-w+1, j-1]$ with $|[k, j-1] \cap inv(j)| > \frac{j-k}{2}$.

Lemma 1 ([11]). $ed(\sigma_{[i-w+1,i]})/2 \leq |R(i)| \leq 2 \cdot ed(\sigma_{[i-w+1,i]})$.

Hence, if we know $|R(i)|$, we can return $2|R(i)|$ as an estimation for $ed(\sigma_{[i-w+1,i]})$ and it gives a 4-approximation algorithm. However, maintaining $R(i)$ exactly requires space linear to the window size. In the following, we show how to approximate $R(i)$ using significantly less space.

3.1 Estimating $R(i)$

We first present our algorithm and then show that it can approximate $R(i)$. Our algorithm will make use of two data structures. Let ϵ' be a constant in $(0, 1)$ (which will be set to $\epsilon/35$ later).

ϵ' -approximate quantile data structure \mathcal{Q} : Let Q be a set of items. The rank of an item in Q is its position in the list formed by sorting Q from the smallest to the biggest. For any $\phi \in [0, 1]$, the ϵ' -approximate ϕ -quantile of Q is an item with rank in $[(\phi - \epsilon')|Q|, (\phi + \epsilon')|Q|]$. We maintain an ϵ' -approximate ϕ -quantile data structure given in [13] which can return, at any time, an ϵ' -approximate ϕ -quantile of the most recent w' items for any $w' \leq w$. This data structure takes $O(\frac{1}{(\epsilon')^2} \log^2(\epsilon'w))$ space.

ϵ' -approximate basic counting data structure \mathcal{B} : When an item $\sigma(i)$ arrives, we may associate a token with some item $\sigma(k)$ where $k < i$. The association is permanent and an item may be associated with more than one token. At any time, we are interested in the number of tokens associated with the most recent w items. We view it as a stream σ_{token} of tokens, each of which has a timestamp k if it is associated to $\sigma(k)$, and we want to return the number of tokens with timestamp in $[i-w+1, i]$. Note that the tokens may be out-of-order with respect to the timestamp, leading to the basic counting problem for out-of-order stream considered in [6]. We maintain their ϵ' -approximate basic counting data structure on σ_{token} which can return, at any time, an estimate \hat{t} such that $|\hat{t} - t| \leq \epsilon't$, where t is the number of tokens associated with the latest w items. It takes $O(\frac{1}{\epsilon'} \log w \log(\frac{\epsilon' B}{\log w}))$ space, where B is the maximum number of tokens associated within any window of w items. As we may associate one token upon any item arrival, B is at most w .

We are now ready to define our algorithm, as follows.

Algorithm 1. Estimating ED in sliding windows

Item arrival: Upon the arrival of item $\sigma(i)$, do

For $k = i - 1, i - 2, \dots, i - w + 1$

Query \mathcal{Q} for the $(\frac{1}{2} - \epsilon')$ -quantile of $\sigma_{[k, i-1]}$. Let a be the returned value.

If $a > \sigma(i)$, associate a token to $\sigma(k)$, i.e., add an item with timestamp k to the stream σ_{token} . Break the for loop.

Query: Query \mathcal{B} on the stream σ_{token} for the number of tokens associated with the last w items and let \hat{t} be the returned answer. Return $\hat{t}/(\frac{1}{2} - 2\epsilon')(1 - \epsilon')$ as the estimation $\widehat{\text{ed}}(\sigma_{[i-w+1, i]})$.

Let $R'(i)$ be the set of indices j such that when $\sigma(j)$ arrives, we associate a token to an item $\sigma(k)$ where $k \in [i - w + 1, i]$. Observe that $R'(i)$ is an approximation of $R(i)$ in the following sense.

Lemma 2. $R'(i)$ contains all indices $j \in [i - w + 1, i]$ satisfying that there exists $k \in [i - w + 1, j - 1]$ such that $|[k, j - 1] \cap \text{inv}(j)| > (\frac{1}{2} + 2\epsilon')(j - k)$. Furthermore, all indices j contained in $R'(i)$ must satisfy that there exists $k \in [i - w + 1, j - 1]$ such that $|[k, j - 1] \cap \text{inv}(j)| > \frac{j-k}{2}$.

Proof. An index j is in $R'(i)$ if $\sigma(j) < a$ when $\sigma(j)$ arrives, where a is the ϵ' -approximate $(\frac{1}{2} - \epsilon')$ -quantile for some interval $\sigma_{[k, j-1]}$. Note that the rank of a in $\sigma_{[k, j-1]}$ is at least $(\frac{1}{2} - 2\epsilon')(j - k)$. Therefore, if $|[k, j-1] \cap \text{inv}(j)| > (\frac{1}{2} + 2\epsilon')(j - k)$, the rank of $\sigma(j)$ is less than $(j - k) - (\frac{1}{2} + 2\epsilon')(j - k) = (\frac{1}{2} - 2\epsilon')(j - k)$, so $\sigma(j) < a$ and j must be included in $R'(i)$. On the other hand, the rank of a in $\sigma_{[k, j-1]}$ is at most $\frac{j-k}{2}$. Since $a > \sigma(j)$, we conclude that all indices $j \in R'(i)$ satisfy $|[k, j - 1] \cap \text{inv}(j)| > \frac{j-k}{2}$. \square

We show that $|R'(i)|$ is a good approximation for $\text{ed}(\sigma_{[i-w+1, i]})$, as follows.

Lemma 3. $(\frac{1}{2} - 2\epsilon') \cdot \text{ed}(\sigma_{[i-w+1, i]}) \leq |R'(i)| \leq 2 \cdot \text{ed}(\sigma_{[i-w+1, i]})$.

Proof. We observe that by Lemma 2, any index j in $R'(i)$ must be also in $R(i)$. Hence, $R'(i) \subseteq R(i)$ and $|R'(i)| \leq |R(i)| \leq 2 \cdot \text{ed}(\sigma_{[i-w+1, i]})$ (by Lemma 1). \square

Now, we show $(\frac{1}{2} - 2\epsilon') \cdot \text{ed}(\sigma_{[i-w+1, i]}) \leq |R'(i)|$ by giving an iterative pruning procedure to obtain an increasing subsequence (may not be the longest). First let $x = i + 1$ and $\sigma(x) = \infty$. Find the largest j such that $i - w + 1 \leq j < x$ and $j \notin R'(i) \cup \text{inv}(x)$ and delete the interval $[j + 1, x - 1]$. We then let $x = j$ and repeat the process until no such j is found. As each x is not in $R'(i)$, Lemma 2 implies that in every interval that we delete, the fraction of items of $R'(i)$ is at least $(\frac{1}{2} - 2\epsilon')$. Note that eventually all items in $R'(i)$ will be deleted. Thus, $|R'(i)| \geq (\frac{1}{2} - 2\epsilon') \cdot (\text{number of deleted items}) \geq (\frac{1}{2} - 2\epsilon') \cdot \text{ed}(\sigma_{[i-w+1, i]})$. \square

Note that $|R'(i)|$ equals the number of tokens associated with the most recent w items. Since \mathcal{B} is only an ϵ' -approximate data structure, the value \hat{t} returned only

satisfies that $(1 - \epsilon')|R'(i)| \leq \hat{t} \leq (1 + \epsilon')|R'(i)|$. Since we report $\widehat{\text{ed}}(\sigma_{[i-w+1,i]}) = \hat{t}/(\frac{1}{2} - 2\epsilon')(1 - \epsilon')$ as the estimation, we conclude with the following approximation ratio.

Lemma 4. $\text{ed}(\sigma_{[i-w+1,i]}) \leq \widehat{\text{ed}}(\sigma_{[i-w+1,i]}) \leq \frac{2(1+\epsilon')}{(1/2-2\epsilon')(1-\epsilon')} \cdot \text{ed}(\sigma_{[i-w+1,i]}).$

For any $\epsilon \leq 1$, we can set $\epsilon' = \epsilon/35$. Then, $\frac{2(1+\epsilon')}{(1/2-2\epsilon')(1-\epsilon')} \cdot \text{ed}(\sigma_{[i-w+1,i]}) \leq (4 + \epsilon) \cdot \text{ed}(\sigma_{[i-w+1,i]})$. The total space usage of the two data structures is $O(\frac{1}{\epsilon^2} \log^2(\epsilon w) + \frac{1}{\epsilon} \log w \log(\epsilon w))$. If $\epsilon > \frac{1}{w}$, $\log w = O(\frac{1}{\epsilon} \log(\epsilon w))$ and thus the total space usage is $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$. Otherwise, we can store all items in the window, which only requires $O(w) = O(\frac{1}{\epsilon})$ space.

Improving the running time. The per-item update time of the algorithm is $O(w)$ because the algorithm checks the interval $I = [k, i - 1]$ for every length $|I| \in [w - 1]$. An observation in [8] is that an $\frac{\epsilon'}{2}$ -approximate ϕ -quantile of an interval with length $|I|$ is also an ϵ' -approximate ϕ -quantile for all intervals with length $|I| + 1, \dots, (1 + \frac{\epsilon'}{2})|I|$. Hence we only need to check $O(\frac{1}{\epsilon'} \log w)$ intervals of length $1, 2, \dots, (1 + \frac{\epsilon'}{2})^i, (1 + \frac{\epsilon'}{2})^{i+1}, \dots, w$. Then we obtain an ϵ' -approximate quantile for every interval. Note that the query time for returning an approximate quantile is $O(\frac{1}{\epsilon'} \log^2 w)$, and the per-item update time of the two data structures is $O(\frac{1}{\epsilon^2} \log^3 w)$ [6, 13]. We conclude with the main result of this section.

Theorem 1. *There is a deterministic $(4 + \epsilon)$ -approximate algorithm for estimating ED in a sliding window of the latest w items. The space usage is $O(\frac{1}{\epsilon^2} \log^2(\epsilon w))$ and the per-item update time is $O(\frac{1}{\epsilon^2} \log^3 w)$.*

Remark. For the whole stream model, the state-of-the-art result is a $(2 + \epsilon)$ -approximation in [8]. They gave an improved estimator $R(i)$ as the set of indices j such that there exists $k < j$ with $|[k, j - 1] \cap \text{inv}(j)| > |[k, j - 1] \cap R(i)|$. In other words, whether an index belongs to $R(i)$ or not depends on the number of members of $R(i)$ before that index. Note that a member of $R(i)$ could become a nonmember due to window expiration. Therefore, an index j that is not a member of $R(i)$ initially, may later become a member if some of the previous $R(i)$ members become nonmembers. This makes estimating this improved $R(i)$ difficult in the sliding window model.

4 Lower Bounds for Out-of-Order Streams

In this section, we consider an out-of-order stream σ consisting of a sequence of items $\sigma(i) = \langle t_i, v_i \rangle$ for $i \in [N]$, where t_i and v_i are the timestamp and value of the i -th item, respectively. Recall that the sortedness of the stream is measured on the derived value sequence by rearranging the items in non-decreasing order of the timestamps. We show that even for the whole data stream model, any randomized constant-approximate algorithm for estimating ED or LIS requires $\Omega(N)$ space. In fact, a stronger lower bound holds for ED: any randomized algorithm that decides whether ED equals 0 uses $\Omega(N)$ space. Our proofs follow from reductions from two different communication problems.

4.1 Estimating ED in an Out-of-Order Stream

Theorem 2. Consider an out-of-order stream σ of size N . Any randomized algorithm that distinguish between the cases that $\text{ed}(\sigma) = 0$ and that $\text{ed}(\sigma) \geq 1$ must use $\Omega(N)$ bits. Therefore, for arbitrary constant $r \geq 1$, any randomized r -approximation to $\text{ed}(\sigma)$ requires $\Omega(N)$ bits.

We prove the above lower bound by showing a reduction from the classical communication problem INDEX, which has strong communication lower bound.

The problem $\text{INDEX}(x, i)$ is a two-player one-way communication game. Alice holds a binary string $x \in \{0, 1\}^n$ and Bob holds an index $i \in [n]$. In this communication game, Alice sends one message to Bob and Bob is required to output the i -th bit of x , i.e. x_i , based on the message received. A trivial protocol is for Alice to send all her input string x to Bob, which has communication complexity of n bits. It turns out that this protocol is optimal. Particularly, Alice must communicate $\Omega(n)$ bits in any randomized protocol for INDEX \blacksquare .

Proof (of Theorem 2). Given an out-of-order stream with length N , suppose there is a randomized algorithm \mathcal{A} that can determine whether its ED equals to 0 or is at least 1 using S memory bits. We define a randomized protocol \mathcal{P} for $\text{INDEX}(x, i)$ for $n = N - 1$: Alice constructs (hypothetically) an out-of-order stream σ with length n by setting

$$\sigma(j) = \begin{cases} \langle 2j-1, 3j-2 \rangle, & \text{if } x_j = 0 \\ \langle 2j-1, 3j \rangle, & \text{if } x_j = 1 \end{cases} \quad (1)$$

Alice then simulates algorithm \mathcal{A} on stream σ and sends the content of the working memory to Bob. Bob constructs another stream item $\sigma(n+1) = \langle 2i, 3i-1 \rangle$ to continue running algorithm \mathcal{A} on it and obtains the output. If the output says $\text{ed}(\sigma) = 0$, Bob outputs 0; otherwise, Bob outputs 1.

It is not hard to see that $\text{INDEX}(x, i) = x_i = 0$ implies $\text{ed}(\sigma) = 0$ and $\text{INDEX}(x, i) = 1$ implies $\text{ed}(\sigma) = 1$. Therefore, if algorithm \mathcal{A} reports the correct answer with high probability, the protocol \mathcal{P} outputs correctly with high probability, and thus is a valid randomized protocol for INDEX. In the protocol, the number of bits communicated by Alice is at most S . Combining the $\Omega(n) = \Omega(N)$ lower bound, we obtain that $S = \Omega(N)$, completing the proof. \square

4.2 Estimating LIS in an Out-of-Order Stream

Theorem 3. Consider an out-of-order stream σ with size N . Any randomized algorithm that outputs an r -approximation on $\text{lis}(\sigma)$ must use $\Omega(N/r^2)$ bits.

Proof. We prove the lower bound by considering the t -party set disjointness problem DISJ. The input to this communication game is a binary $t \times \ell$ matrix $\mathbf{x} \in \{0, 1\}^{t\ell}$, and each player P_i holds one row of \mathbf{x} , the 1-entries of which

indicate a subset A_i of $[\ell]$. The input \mathbf{x} is called *disjoint* if the t subsets are pairwise disjoint, i.e., each column of \mathbf{x} contains at most one 1-entry; and it is called *uniquely intersecting* if the subsets A_i share a unique common element y and the sets $A_i - \{y\}$ are pairwise disjoint, meaning that in \mathbf{x} , except one column with entries all equal to 1, all the other columns have at most one 1-entry. The objective of the game is to distinguish between the two types of inputs. To obtain the space lower bound, we only need to consider a restricted version of DISJ where, according to some probabilistic protocol, the first $t - 1$ players in turn send a message privately to his next neighboring player and the last player P_t outputs the answer.

An optimal lower bound of $\Omega(\ell/t)$ total communication is known for DISJ even for general randomized protocols (with constant success probability) [14], and thus the lower bound also holds for our restricted one-way private communication model. By giving a reduction and setting the parameters appropriately, we can obtain the space lower bound.

Given a randomized algorithm that outputs r -approximation to the LIS of any out-of-order stream with length N , using S memory bits, we define a simple randomized protocol for DISJ for $t = 2r = o(N)$ and $\ell = N + 1 - t = \Theta(N)$. Let \mathbf{x} be the input $t \times \ell$ matrix. The first player P_1 creates an out-of-order stream σ by going through his row of input $R_1(\mathbf{x})$ and inserting a new item $\langle(j-1)t+1, (\ell-j)t+1\rangle$ to the end of the stream whenever an entry x_{1j} equals to 1. He then runs the streaming algorithm on σ and sends the content of the memory to the second player. In general, player P_i appends a new item $\langle(j-1)t+i, (\ell-j)t+i\rangle$ to the stream for each nonzero entry x_{ij} , simulates the streaming algorithm and communicates the updated memory state to the next player. Finally, player P_t obtains the approximated LIS of stream σ . If it is at most r he reports that the input \mathbf{x} is disjoint; else, he reports it is uniquely intersecting. It's easy to verify that if the input \mathbf{x} is disjoint, the correct LIS of stream σ is 1, while if it is uniquely intersecting, the correct LIS of σ is t . Consequently, if the streaming algorithm outputs an r -approximation to $\text{lis}(\sigma)$ with probability at least $2/3$, the protocol for DISJ is correct with constant probability, using total communication at most $(t-1)S$. Following the lower bound for DISJ, this implies $(t-1)S \geq \Omega(\ell/t)$, i.e., $S = \Omega(\ell/t^2) = \Omega(N/r^2)$. Theorem 3 follows.

Remark. Actually, for deterministic algorithms, we can obtain a slightly stronger lower bound of $\Omega(N/r)$ for r -approximation, by a reduction from the HIDDEN-IS problem used in [10] to prove the $\Omega(\sqrt{N})$ lower bound for approximating LIS of an in-order stream. The reduction is similar to the above, and if we set the approximation ratio r to a constant, the lower bounds become linear in both cases. Therefore, we neglect the details here.

Acknowledgement. We thank the anonymous reviewers for helpful comments and for pointing out the randomized lower bounds to us.

References

1. Ablayev, F.: Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theoretical Computer Science* 157(2), 139–159 (1996)
2. Ajtai, M., Jayram, T.S., Kumar, R., Sivakumar, D.: Approximate counting of inversions in a data stream. In: Proc. STOC, pp. 370–379 (2002)
3. Ben-Moshe, S., Kanza, Y., Fischer, E., Matsliah, A., Fischer, M., Staelin, C.: Detecting and exploiting near-sortedness for efficient relational query evaluation. In: Proc. ICDT, pp. 256–267 (2011)
4. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. *Computer Networks* 30(1-7), 107–117 (1998)
5. Chakrabarti, A.: A note on randomized streaming space bounds for the longest increasing subsequence problem. In: ECCC, p. 100 (2010)
6. Cormode, G., Korn, F., Tirthapura, S.: Time-decaying aggregates in out-of-order streams. In: Proc. PODS, pp. 89–98 (2008)
7. Cormode, G., Muthukrishnan, S.M., Sahinalp, S.C.: Permutation Editing and Matching via Embeddings. In: Yu, Y., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 481–492. Springer, Heidelberg (2001)
8. Ergun, F., Jowhari, H.: On distance to monotonicity and longest increasing subsequence of a data stream. In: Proc. SODA, pp. 730–736 (2008)
9. Estivill-Castro, V., Wood, D.: A survey of adaptive sorting algorithms. *ACM Computing Surveys* 24, 441–476 (1992)
10. Gál, A., Gopalan, P.: Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In: Proc. FOCS, pp. 294–304 (2007)
11. Gopalan, P., Jayram, T.S., Krauthgamer, R., Kumar, R.: Estimating the sortedness of a data stream. In: Proc. SODA, pp. 318–327 (2007)
12. Gopalan, P., Krauthgamer, R., Thathachar, J.: Method of obtaining data samples from a data stream and of estimating the sortedness of the data stream based on the samples. United States Patent 7,797,326 B2 (2010)
13. Lin, X., Lu, H., Xu, J., Yu, J.X.: Continuously maintaining quantile summaries of the most recent n elements over a data stream. In: Proc. ICDE, pp. 362–374 (2004)
14. Jayram, T.S.: Hellinger Strikes Back: A Note on the Multi-party Information Complexity of AND. In: Dinur, I., Jansen, K., Naor, J., Rolim, J. (eds.) RANDOM 2009. LNCS, vol. 5687, pp. 562–573. Springer, Heidelberg (2009)

Folding Equilateral Plane Graphs

Zachary Abel¹, Erik D. Demaine², Martin L. Demaine², Sarah Eisenstat²,
Jayson Lynch², Tao B. Schardl², and Isaac Shapiro-Ellowitz³

¹ MIT Department of Mathematics, zabel@math.mit.edu

² MIT Computer Science and Artificial Intelligence Laboratory
{edemaine,mdemaine,seisenst,jaysonl,neboat}@mit.edu

³ University of Massachusetts Boston, isaac.shapiroello001@umb.edu

Abstract. We consider two types of folding applied to *equilateral* plane graph linkages. First, under continuous folding motions, we show how to reconfigure any *linear* equilateral tree (lying on a line) into a canonical configuration. By contrast, such reconfiguration is known to be impossible for linear (nonequilateral) trees and for (nonlinear) equilateral trees. Second, under instantaneous folding motions, we show that an equilateral plane graph has a noncrossing linear folded state if and only if it is bipartite. Not only is the equilateral constraint necessary for this result, but we show that it is strongly NP-complete to decide whether a (nonequilateral) plane graph has a linear folded state. Equivalently, we show strong NP-completeness of deciding whether an abstract metric polyhedral complex with one central vertex has a noncrossing flat folded state with a specified “outside region”. By contrast, the analogous problem for a polyhedral manifold with one central vertex (*single-vertex origami*) is only weakly NP-complete.

1 Introduction

This paper is motivated by two different types of problems related to folding: (1) linkage folding, specifically locked trees; and (2) paper folding, specifically single-vertex origami.

1.1 Locked Trees: Not If Equilateral and Linear

Biedl et al. [2] introduced the notion of a “locked tree” and gave the first example thereof. Here a *tree* refers to a plane tree linkage, that is, a tree graph with specified edge lengths and a preferred planar embedding. Such a linkage can *move* or *fold continuously* subject to the constraints that the edges remain straight line segments of the specified lengths, and that the edges never properly cross each other [3]. A tree is *universally foldable* [2] if it can be folded continuously from any configuration to any other. Equivalently, a tree is universally foldable if it can be folded from any configuration into a *canonical* configuration, in which the edges lie along a horizontal line and point rightward from a single root vertex. Otherwise, a tree is *locked*. The construction in [2] gives a specific tree

configuration that is *locked* in the sense that it cannot be folded to a canonical configuration.

Ballinger et al. [1] showed the existence of locked tree configurations with either of two special properties: (a) *linear*, where all edges lie along a line; and (b) *equilateral*, where all edge lengths are equal. The constructed examples of each type seem very different; for example, the locked equilateral tree has no touching bars (except at common endpoints), while the locked linear trees necessarily have many overlapping bars. Thus it is natural to wonder whether there are locked tree configurations that are simultaneously linear and equilateral.

Our results. We settle this question by showing that every linear equilateral tree configuration can be folded into a canonical configuration. As a consequence, any equilateral tree can be folded between all of its linear configurations. Our proof of this result, given in Section 3, builds up a progressively more canonical configuration by repeatedly fixing any deviations. To keep track of the overall structure of the linkage, we introduce the notion of a *plane homomorphism* to enable manipulating multiple overlapping edges as one.

1.2 Single-Vertex Origami: Generalization

A classic structure in mathematical origami is the *single-vertex crease pattern*—a circular piece of paper with creases all emanating from the center. This special case is useful because it effectively models the local behavior of a general crease pattern in a small neighborhood around a vertex. Kawasaki’s Theorem [8] describes precisely when a single-vertex crease pattern can fold flat using exactly the prescribed creases; see also [6, Thm. 12.2.1].

As described in [6, ch. 12], flat folding single-vertex crease patterns can be viewed as folding a cycle linkage into a linear configuration, subject to the constraint of bending at every vertex. Each edge of the cycle linkage corresponds to a pie wedge of the crease pattern, and the edge length equals the pie angle. Although flat pieces of paper have the extra property that the lengths/angles sum to 360° , the characterization of flat foldability has been generalized to arbitrary cycles: see [6, Thm. 12.2.2] and [9].

Compared to locked trees, a key difference here is that we are interested only in *instantaneous* motions, and thus whether a linear configuration exists.

We consider the generalized problem of instantaneous folding of plane graphs (instead of just cycles) into linear configurations. Mapped to the context of origami, this problem is equivalent to flat folding of a *single-vertex complex*, consisting of pie wedges with a common apex, sharing some edges, and bounded by great circular arcs on the surface of a sphere centered at the common apex. This situation models the local behavior of a vertex neighborhood in a polyhedral complex (3D polygons attached at edges or vertices).

Our results. For the special case of equilateral plane graphs, we prove that instantaneous folding into a linear configuration is possible if and only if the graph is bipartite. Bipartiteness is an obvious necessary condition: every cycle with a linear configuration must have an even number of edges, for the linear

configuration naturally partitions the edges into two classes, one for each direction along the line. The interesting result, shown in Section 4, is that all bipartite equilateral plane graphs have a linear configuration.

Interpreted in the context of single-vertex complex origami, this result says that a single-vertex complex in which all pie angles are equal can be folded flat if and only if every spherical region has an even number of edges. We can even require that the flat folding uses all of the given creases: any linear equilateral configuration can be folded down so that all angular regions are collocated.

Finally, we prove that these results can hold only for the equilateral situation, in a strong sense. Specifically, Section 5 shows that finding a linear configuration becomes strongly NP-complete for nonequilateral plane graphs. Our reduction is from planar monotone 3-SAT, which was recently shown to be NP-complete [4].

In the context of single-vertex complex origami, this result says that it is strongly NP-complete to determine flat foldability of an abstract metric single-vertex complex with a specified outside region. This result suggests that there is no complex analog of the efficient Kawasaki’s Theorem characterizing the case of a single cycle, though there are a few differences. First, the outside region must be specified; this technical requirement can likely be removed with additional effort. Second, the constructed complex might not be embeddable on the sphere with the given edge lengths. Third, we do not require every vertex (crease) to be folded. This change makes even the cycle problem NP-complete, though only in the weak sense, as it admits a pseudopolynomial-time algorithm. Our result shows that the problem is strongly NP-complete for general graphs.

2 Definitions

In this paper, every graph $G = (V(G), E(G))$ is assumed to be equipped with positive edge weights (lengths) $\ell : E(G) \rightarrow \mathbb{R}_{>0}$ unless otherwise specified. A **plane graph** is a (weighted) graph with a preferred combinatorial planar embedding (i.e., not necessarily respecting edge lengths).

Recall that a **linkage** is a straight-line embedding of a plane graph (known as the **underlying graph**) with compatible edge lengths. A **motion** of a linkage is a continuous deformation of the linkage that preserves the lengths of edges and does not self-intersect. Intuitively, a **self-touching linkage**, which can self-intersect, but cannot combinatorially cross itself. Connelly et al. [3] give a more formal definition of a self-touching linkage, which we use throughout this paper.

Definition 1. A **self-touching embedding** (also known as a **configuration** or **state**) of a plane graph G is a self-touching linkage L with an isomorphism identifying L ’s underlying graph with G —in particular, the planar embedding and the edge lengths agree.

In the next definition, a **chain** is a path of two unequal edges.

Definition 2. For two weighted or unweighted plane graphs G and H , a **plane homomorphism** $g : G \rightarrow H$ is a graph homomorphism (preserving edge weights

in the weighted case) together with, for each oriented edge $e \in E(H)$ from w_1 to w_2 , a linear (counterclockwise) ordering around w_1 of the edges $g^{-1}(e)$ (the set of edges in G mapping to e) satisfying certain compatibility and planarity constraints. For any vertex $w \in H$, the cyclic ordering of edges around w in H and the linear orders around w defined by g induce a cyclic order of all edges in G whose images are incident to w . The compatibility and planarity constraints may be expressed as follows:

- **Edge Ordering Compatibility:** For every oriented edge e in H , the linear orders for e and for e with opposite orientation are reversed, i.e., edges $g^{-1}(e)$ are linearly ordered.
- **Respect for Planar Embeddings:** For each vertex $v \in G$, the cyclic order of the edges incident to v around $g(v)$ induced by g matches their cyclic order around v in G .
- **Noncrossing:** For any two vertices v, v' in G with $g(v) = g(v') = w$, and any two chains (e_0, e_1) and (e'_0, e'_1) centered at v and v' respectively, the induced cyclic order of these four edges around w is not e_0, e'_0, e_1, e'_1 or e_0, e'_1, e_1, e'_0 . In other words, these chains are not made to cross at w .

A plane homomorphism is **surjective** if it is surjective on vertices and edges.

We interpret a plane homomorphism $g : G \rightarrow H$ as a “self touching embedding of G along graph H .” More explicitly, for each vertex $w \in H$, the **magnified view** of g around w is a graph inside a disk specified as follows: There is a terminal node inside the disk for each vertex in $g^{-1}(w)$, and a nonterminal node for each edge in G whose image is incident to w . The nonterminals are ordered around the boundary of the disk in the cyclic order induced by g . Finally, this graph has an edge connecting terminal-nonterminal pairs corresponding to vertex-edge incidences in G . Then the noncrossing condition can be restated simply as follows: these magnified view graphs are planar.

The following lemmas, proven in the full version, make plane homomorphisms particularly useful:

Lemma 1. *Plane homomorphisms compose: Two plane homomorphisms $g : F \rightarrow G$ and $h : G \rightarrow H$ canonically induce a plane homomorphism $h \circ g : F \rightarrow H$.*

Lemma 2. *A plane homomorphism $g : G \rightarrow H$ and a self-touching embedding M of H canonically induce a self-touching embedding of G , denoted $g^*(M)$. Furthermore, if $t \mapsto M_t$ is a valid motion of M , then $t \mapsto g^*(M_t)$ is a valid motion of $g^*(M)$.*

3 Linear Equilateral Trees

In this section, we consider the question of whether a linear equilateral tree can be “unfolded.” Recall that a **linear** (which we also refer to in this paper as **flat**) state of a graph is a state where all edges geometrically lie on a line . For

trees, a **canonical** state with root vertex v is a horizontal linear state where all simple paths in the tree starting at v proceed monotonically to the right. Note that Ballinger et al. [1] call this a “flat configuration”; we use the term “canonical configuration” instead to minimize the ambiguity of the word “flat.”

It is useful to interpret canonical states of trees as “unfolded” states, because all canonical states are equivalent: for any desired vertex v' and edge e incident to v' , there exists a continuous motion from any canonical state to the canonical state rooted at v' in which edge e is the topmost edge incident to v' [2].

Not all linear trees can be deformed into a canonical state; Ballinger et al. provide multiple such examples [1]. Likewise, not all equilateral trees are universally foldable: as shown in [1], there are configurations of equilateral trees that cannot be deformed into a canonical state. By contrast, for tree configurations that are both linear *and* equilateral, we show:

Theorem 1. *Any linear configuration of an equilateral tree can be continuously deformed (without overlap) into a canonical state.*

Our algorithm proceeds roughly as follows. The initial linear state is “partially canonical.” We search for breaks in the “boundary” of the homomorphism, and unfold G at the location of the break to make it closer to canonical. By repeating this process, we end up with a canonical state.

We will need two definitions to make this argument precise. This first definition allows us to formally discuss the “boundary” of a plane homomorphism as a set of threshold edges:

Definition 3. *Say that we have a plane graph G on n vertices, and a plane homomorphism $g : G \rightarrow H$. For each oriented edge e from w_1 to w_2 in the image of g , the edges $g^{-1}(e)$ are ordered counterclockwise around w_1 , and we define the **threshold edge** $\text{thr}(e)$ to be the first edge in that ordering. Furthermore, if $\text{thr}(e)$ has endpoints v_1, v_2 with $g(v_1) = w_1$ and $g(v_2) = w_2$, then choose orientation (v_1, v_2) for this edge.*

It will also be helpful to have a definition for “partially canonical” states to measure our progress during an induction:

Definition 4. *A configuration of a tree G is **k -canonical** if there exists a tree H_k on k nodes in a canonical state and a surjective plane homomorphism $g_k : G \rightarrow H_k$ such that the configuration of G is the one induced by g_k . Note that if a tree on n nodes is in an n -canonical state, then the tree is canonical.*

Proof (of Theorem 1). Let G be the given tree with n vertices, initially in a linear state L . This initial state of G is ℓ -canonical for some $2 \leq \ell \leq n$. We will show, by induction on $\ell \leq k \leq n$, that G can be deformed from L to a k -canonical state; the base case $k = n$ is the desired result.

Suppose that G has been deformed into the k -canonical state induced by a surjective plane homomorphism $g_k : G \rightarrow H_k$, where H_k is a tree on k vertices in a canonical state. Let $p_1, p_2, \dots, p_{2k-2}$ be the vertices of the outer face of H_k in clockwise order around this face, and define $d_1 = \text{thr}(p_1, p_2), d_2 =$

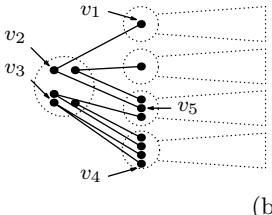
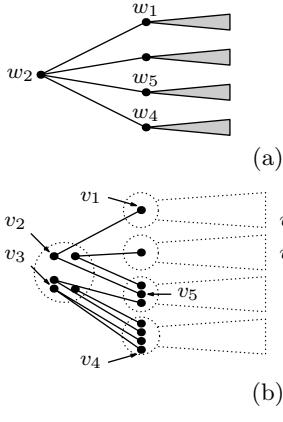


Fig. 1. An example of splitting a vertex. Figure (a) shows how the split is used to transform H_k into H_{k+1} . Figure (b) shows how the graph G is affected.

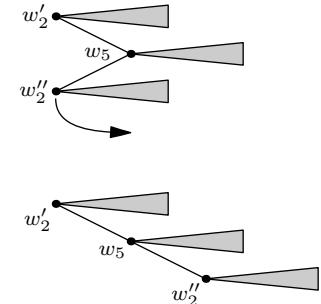


Fig. 2. Reconfiguring H_{k+1} into a canonical state

$\text{thr}(p_2, p_3), \dots, d_{2k-2} = \text{thr}(p_{2k-2}, p_1)$. Intuitively, these are the oriented edges of G that lie adjacent to H_k 's outer face according to g_k . We claim that these edges cannot form a cycle in G . Assume to the contrary that they form a cycle. Because $k < n$, there must be some oriented edge $e = (w_1, w_2)$ in H_k with $|g_k^{-1}(e)| \geq 2$. Hence the edges $\text{thr}(w_1, w_2)$ and $\text{thr}(w_2, w_1)$, when taken as unoriented edges, are not equivalent, and so each appears in the cycle exactly once. The cycle is therefore nontrivial, contradicting the fact that G is a tree.

It follows that there must be some i such that the oriented edges $d_i = (v_1, v_2)$ and $d_{i+1} = (v_3, v_4)$ have $v_2 \neq v_3$. Fix some i with this property. Let $w_1 = g_k(v_1)$, $w_2 = g_k(v_2) = g_k(v_3)$, and $w_4 = g_k(v_4)$. By [2], we may move H_k to a new canonical state with w_2 at the root, and (w_2, w_1) as the topmost edge incident to w_2 (thereby making (w_2, w_4) the bottommost edge incident to w_2). This motion of H_k to a new state N_k induces a motion of G to a new state $M_k = g_k^*(N_k)$ lying on a horizontal line. Let (v_2, v_5) be the bottommost edge of G incident to v_2 in M_k , and let $w_5 = g_k(v_5)$. Note that w_5 may equal w_1 , w_4 , or both.

We next show how to “split” the vertex w_2 along edge (w_2, w_5) to construct a surjective plane graph homomorphism $g_{k+1} : G \rightarrow H_{k+1}$, where H_{k+1} is a tree with $k+1$ vertices. We construct H_{k+1} from H_k by replacing w_2 with two vertices w'_2 and w''_2 , as depicted in Fig. 1(a). Vertex w'_2 is connected to all neighbors of w_2 between w_5 and w_1 inclusive in the counterclockwise ordering of the edges around w_2 , and likewise, vertex w''_2 is connected to all neighbors of w_2 between w_4 and w_5 inclusive in the counterclockwise ordering. Edges (w_5, w'_2) and (w_5, w''_2) replace (w_5, w_2) in the counterclockwise order of edges around w_5 , with (w_5, w'_2) coming before (w_5, w''_2) . Splitting the node in this way naturally yields a surjective plane graph homomorphism $h : H_{k+1} \rightarrow H_k$ sending w'_2 and w''_2 to w_2 , which in turn defines a planar configuration $P = h^*(N_k)$ of H_{k+1} .

This construction also yields a plane homomorphism $g_{k+1} : G \rightarrow H_{k+1}$ defined as follows: In the counterclockwise order on $g_k^{-1}(w_5, w_2)$, those edges before and

including (v_5, v_2) (equivalently, lying above (v_5, v_2) in M_k) map to (w_5, w'_2) in H_{k+1} with the same ordering, and the rest map to (w_5, w''_2) . The rest of g_{k+1} is defined to match g_k . This can be checked to be well defined. We may also prove surjectivity: Homomorphism g_{k+1} hits every edge of H_{k+1} except possibly (w''_2, w_5) , and the connected graph G cannot surject onto the disconnected graph $H_{k+1} \setminus \{(w''_2, w_5)\}$. So this edge is in the image of g_{k+1} . We also have $g_{k+1} \circ h = g_k$, and it follows that the current configuration on G , namely M_k , is induced by g_{k+1} : indeed, $M_k = g_k^*(N_k) = g_{k+1}^*(h^*(N_k)) = g_{k+1}^*(P)$.

Finally, we use plane homomorphism g_{k+1} to reconfigure G from M_k to a $(k+1)$ -canonical state. Consider (H_{k+1}, P) schematically as in Fig. 1(b) with two edges (w'_2, w_5) and (w''_2, w_5) and a canonical subtree rooted at each of these vertices. Swinging edge (w''_2, w_5) around w_5 while holding the subtree rooted at w''_2 horizontal, as in Fig. 2, reconfigures H_{k+1} into a canonical state with root w'_2 . This induces a motion on G , resulting in a $(k+1)$ -canonical state. \square

4 Flat-Foldable Planar Graphs

We now consider the more general question of instantaneously folding a plane graph into a linear state. In this section we show:

Theorem 2. *Given a plane graph G , there exists a linear equilateral linkage configuration with the same planar embedding if and only if G is bipartite.*

Proof. Suppose graph G has a linear equilateral linkage configuration. This configuration can be accordion-folded into a configuration whose geometric graph consists of a single edge of length 1. The two nodes of this graph induce a bipartite structure on G , so G is bipartite.

Conversely, consider a bipartite graph G with a planar embedding and $n = |V(G)|$ vertices. Without loss of generality, we may assume G is connected. We proceed by induction, showing roughly that we can repeatedly fold together adjacent edges on a face until only two vertices remain. Specifically, we show by downward induction on $n \geq k \geq 2$ that there exists a plane homomorphism from G to a bipartite graph H_k on k vertices. The configuration induced by the plane homomorphism $G \rightarrow H_2$ will yield a linear state of G .

The base case $k = n$ is satisfied by the identity homomorphism $G \rightarrow G$. Now suppose we have a plane homomorphism $h_k : G \rightarrow H_k$ for some $k \geq 3$. It can be verified that there must therefore exist at least one face F in H_k with at least 3 edges. Face F must contain at least two adjacent edges (u_1, u_2) and (u_2, u_3) such that u_1 , u_2 , and u_3 are all distinct. We now “fold” these two edges together: define H_{k-1} as the plane graph obtained by first inserting edge (u_1, u_3) into H_k inside face F and then contracting this edge to a vertex w . This operation defines a plane homomorphism from H_k to H_{k-1} , sending u_1 and u_3 to w , and furthermore H_{k-1} is bipartite. The composed plane homomorphism, $G \rightarrow H_k \rightarrow H_{k-1}$ proves the inductive step. Any configuration of H_2 must be linear, and therefore the configuration induced by $G \rightarrow H_2$ is also linear. \square

5 NP-Hardness of Graph Folding

Although it is possible to determine in polynomial time whether an equilateral graph has a linear state, it is hard to determine whether a weighted graph has a linear state. Consider the problem when restricted to cycles. Because the cycle need not fold at every vertex, it is possible to reduce from the integer partition problem [7] by creating a cycle whose edge lengths are the numbers to partition. Hence, it is weakly NP-hard to determine whether a weighted graph has a linear state. In this section, we show that the problem is strongly NP-hard via a reduction from planar monotone 3-SAT, which is known to be NP-hard [4].

Let $G = (U \cup (C_+ \cup C_-), E)$ be a plane graph encoding an instance of the planar monotone 3-SAT problem. Specifically, let $U = (x_1, x_2, \dots, x_n)$ denote a sequence of n variables that lie along the y -axis in order with x_1 on top. Let C_+ denote a 3-CNF formula over U containing only positive literals, and similarly let C_- denote a 3-CNF formula over U containing only negative literals. The clauses $c \in C_+$ have x -coordinate less than zero, and the clauses $c \in C_-$ have x -coordinate greater than zero. The edge set E of the graph G consists of all edges $(x, c) \in U \times (C_+ \cup C_-)$ such that clause c contains either x or \bar{x} .

We first define a new graph G' from G as follows. Each variable vertex x in G with degree k “splits” into k copies of itself in G' , thus forming a longer vertical line of variables vertices, and each clause connects to a copy of each of its literals such that each variable copy connects to at most one clause. This can be done while preserving planarity.

To perform this reduction, we represent each variable using an instance of the gadget shown in Fig. 3(a), and we represent each clause using an instance of the gadget shown in Fig. 3(c). We now discuss these gadgets in detail.

For each variable x_i with degree k in G , we construct a variable gadget with k entries, as exemplified in Fig. 3(a). These k entries correspond to the k copies of x_i in G' . On each side of the variable gadget, we consider two sets of points. We call the points $\{v_{i,0}, \dots, v_{i,k}\}$ the **(positive) spine points** of the variable gadget and the points $\{w_{i,1}, \dots, w_{i,k}\}$ the **(positive) flex points** of the variable gadget. Similarly, we define the points $\{v'_{i,0}, \dots, v'_{i,k}\}$ and $\{w'_{i,0}, \dots, w'_{i,k}\}$ to be the **(negative) spine points** and **(negative) flex points** respectively.¹ It can be shown that the resulting gadget has two linear states. The first state, illustrated in Fig. 3(a), has the flex points pointing right (i.e. positioned to the right of their spine points) and indicates $x_i = \text{true}$, while the other has the flex points pointing left and indicates $x_i = \text{false}$. We then connect the variable gadgets together to form the **variable column**, as shown in Fig. 3(b). This variable column has exactly 2^n linear states, corresponding to all possible boolean assignments to the variables U . Moreover, each of these linear states ensures that the edges between the spine and flex points have the same top-to-bottom ordering as Fig. 3(b).

We now describe the clause gadget for clause c . We describe the case in which $c \in C_+$ contains three literals; the cases of one or two literals are analogous, and the C_- clauses are symmetric. Clause c is represented using four new

¹ We omit the “positive” and “negative” specifiers when it is clear from context.

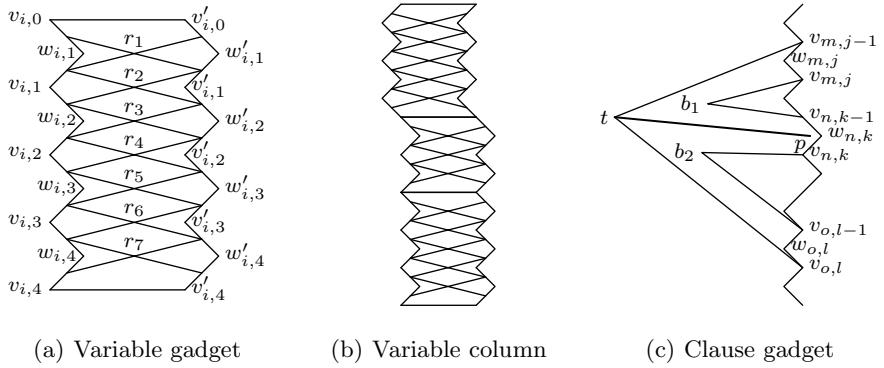


Fig. 3. Figure (a) shows a 4-entry variable gadget, set to true. The edge length between any pair of adjacent spine and truth points on one side is 2. Figure (b) shows an example variable column containing three variable gadgets. The bottom two variables represented are set to true, while the top-most is set to false. Figure (c) shows an example clause gadget. The six edges connecting t , b_1 , and b_2 to the spine points all have length 3, while the probe (edge (t, p)) has length 5.

vertices t, p, b_1, b_2 and six new edges as depicted in Fig. 3(c). The clause gadget for c is connected to the three variable entries corresponding to c 's neighbors in G' , thereby preventing the clause gadgets from interfering with each other. The **probe** (t, p) is a long edge inside the clause gadget that permits a linear state if and only if c is satisfied. The **blockers**, vertices b_1, b_2 and their incident edges, prevent the probe from accessing variable entries for variables not in c . By construction, the clause gadget has the following property.

Lemma 3. *For each clause $c \in C_+ \cup C_-$, the clause gadget for c has a linear state if and only if c is satisfied.*

Proof. If a clause is satisfied, then the clause gadget spans a distance of five and the probe gadget can be placed in the satisfied variable as in Fig. 3(c). The diagram demonstrates a linear state.

To prove necessity, consider a linear state of the clause gadget c ; by symmetry, we may assume $c \in C_+$. The structure of the variable column ensures that all of the positive spine points must be collocated at a point. Consider the magnified view of the linear configuration around this point. From the ordering of the spine points and the combinatorial embedding, the counterclockwise ordering of the edges incident to t , b_1 , and b_2 must be consistent with the top-to-bottom ordering shown in Fig. 3(c). The points b_1 , b_2 , and t must lie three units away from the spine, and because the distance from the spine to each flex point is two, b_1 and b_2 must lie at the same location as t . The probe has length five and is on the internal face of the clause gadget, so t must point away from the variable column, and at least one of the three variable entries must be set to true. Hence the clause gadget and variable column have a linear configuration only if the clause is satisfied. \square

If G is an instance of planar monotone 3-SAT and H is the graph obtained by this reduction, then by Lemma 3 and preceding discussion, linear configurations of H correspond exactly to satisfying assignments of G . We thus obtain:

Theorem 3. *Determining whether a plane graph has a linear state is strongly NP-complete.*

Acknowledgments. We thank Ilya Baran for early discussions about instantaneous graph folding, in particular conjecturing Theorem 2. We also thank Muriel Dulieu for helpful discussions on this topic. This research was continued during the open-problem sessions organized around MIT class 6.849: Geometric Folding Algorithms in Fall 2010. We thank the other participants of these sessions—Scott Kominers, Jason Ku, Thomas Morgan, Jie Qi, Tomohiro Tachi, and Andrew Winslow—for providing a conducive research environment.

References

1. Ballinger, B., Charlton, D., Demaine, E.D., Demaine, M.L., Iacono, J., Liu, C.-H., Poon, S.-H.: Minimal Locked Trees. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 61–73. Springer, Heidelberg (2009)
2. Biedl, T., Demaine, E., Demaine, M., Lazard, S., Lubiw, A., O'Rourke, J., Robbins, S., Streinu, I., Toussaint, G., Whitesides, S.: A note on reconfiguring tree linkages: Trees can lock. *Discrete Applied Mathematics* 117(1-3), 293–297 (2002)
3. Connelly, R., Demaine, E.D., Rote, G.: Infinitesimally locked self-touching linkages with applications to locked trees. In: Calvo, J., Millett, K., Rawdon, E. (eds.) *Physical Knots: Knotting, Linking, and Folding of Geometric Objects in R3*, pp. 287–311. American Mathematical Society (2002)
4. de Berg, M., Khosravi, A.: Optimal Binary Space Partitions in the Plane. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 216–225. Springer, Heidelberg (2010)
5. Demaine, E.D., Devadoss, S.L., Mitchell, J.S.B., O'Rourke, J.: Continuous foldability of polygonal paper. In: Proceedings of the 16th Canadian Conference on Computational Geometry, Montréal, Canada, pp. 64–67 (August 2004)
6. Demaine, E.D., O'Rourke, J.: *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press (2007)
7. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) *Complexity of Computer Computations*. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
8. Kawasaki, T.: On the relation between mountain-creases and valley-creases of a flat origami. In: Huzita, H. (ed.) *Proceedings of the 1st International Meeting of Origami Science and Technology*, pp. 229–237, Ferrara, Italy (December 1989); An unabridged Japanese version appeared in *Sasebo College of Technology Report* 27, 153–157 (1990)
9. Streinu, I., Whiteley, W.: Single-Vertex Origami and Spherical Expansive Motions. In: Akiyama, J., Kano, M., Tan, X. (eds.) JCDCG 2004. LNCS, vol. 3742, pp. 161–173. Springer, Heidelberg (2005)

Efficient Algorithms for the Weighted k -Center Problem on a Real Line*

Danny Z. Chen and Haitao Wang**

Department of Computer Science and Engineering,
University of Notre Dame, Notre Dame, IN 46556, USA
`{dchen, hwang6}@nd.edu`

Abstract. We present $O(\min\{n \log^{1.5} n, n \log n + k^2 \log^2 \frac{n}{k} \log^2 n\})$ time algorithms for the weighted k -center problem on a real line. Previously, the best known algorithms for this problem take $O(n \log^2 n)$ time, or $O(kn \log n)$ time, or a time linear in n but exponential in k . Our techniques involve developing efficient data structures for processing queries that find a lowest point in the common intersection of a certain subset of half-planes. This subproblem is interesting in its own right.

1 Introduction

Let $P = \{p_1, p_2, \dots, p_n\}$ be a set of n points on a real line L (e.g., the x -axis). For each i with $1 \leq i \leq n$, the point $p_i \in P$ has a weight $w(p_i) \geq 0$. For a point p on L , denote by $x(p)$ the coordinate of p on L . For two points p and q on L , let $d(p, q) = |x(p) - x(q)|$ be the *distance* between p and q . Further, for a set $F = \{f_1, f_2, \dots, f_k\}$ of (facility) points and a point q on L , define $d(q, F) = d(F, q) = \min_{1 \leq j \leq k} d(q, f_j)$. Given P and an integer $k > 0$, the *weighted k -center* problem seeks to determine a set $F = \{f_1, f_2, \dots, f_k\}$ of k points on L such that the value $\psi(P, F) = \max_{p_i \in P} (w(p_i) \cdot d(p_i, F))$ is minimized. We also refer to this problem as the *one-dimensional k -center* problem, denoted by *k Center-1D*. The points in F are called *centers* (or *facilities*), and the points in P are called *demand points* (or *customers*).

The *unweighted version* of *k Center-1D* is the case when the point weights are all equal. If $F \subseteq P$ is required, then that case is called the *discrete version*.

The k -center problem and many of its variants are NP-hard [13][15]. Frederickson [9] gave a linear time algorithm for the unweighted k -center problem on a tree. Megiddo and Tamir [15] presented an $O(n \log^2 n \log \log n)$ time algorithm for the weighted k -center problem on a tree of n nodes, and the running time can be reduced to $O(n \log^2 n)$ by applying Cole's technique in [6]. Jeger and Kariv [12] gave an $O(kn \log n)$ time algorithm for the weighted k -center problem on a tree, which is in favor of small k . For the weighted k -center problem on a real line, Bhattacharya and Shi [2] recently proposed an algorithm with a time bound

* This research was supported in part by NSF under Grant CCF-0916606.

** Corresponding author.

linear in n but exponential in k . In addition, the discrete weighted k -center problem on a tree is solvable in $O(n \log^2 n)$ time [16]. To our best knowledge, we have not found other known faster algorithms for the k Center-1D problem.

We solve k Center-1D in $O(\min\{n \log^{1.5} n, n \log n + k^2 \log^2 \frac{n}{k} \log^2 n\})$ time, which is faster than the $O(n \log^2 n)$ time result [6, 15] and the $O(kn \log n)$ time result [12] when they are applied to k Center-1D. Our result is also better than the solution in [2] for large k since the running time in [2] is exponential in k .

An Overview of Our Approach

We model k Center-1D as a problem of approximating a set of weighted points by a step function in the plane [5]. Two algorithms were given in [5] for this point approximation problem with the time bounds of $O(n \log^2 n)$ and $O(n \log n + k^2 \log^2 \frac{n}{k} \log^2 n)$, respectively. Consequently, the k Center-1D problem can be solved in $O(\min\{n \log^2 n, n \log n + k^2 \log^2 \frac{n}{k} \log^2 n\})$ time.

However, the k Center-1D problem has some special properties that allow us to develop faster solutions. Specifically, after some geometric transformations, a key component to solving the problem is the following *2-D sublist LP query* problem: Given a set of n half-planes, $H = \{h_1, h_2, \dots, h_n\}$, in the plane, design a data structure to answer any query $q(i, j)$ ($1 \leq i \leq j \leq n$) which asks for a lowest point p^* in the common intersection of all half-planes in $H_{ij} = \{h_t \mid i \leq t \leq j\}$. This LP query problem also arises in the point approximation problem [5], and based on fractional cascading [4], a data structure for it can be built in $O(n \log n)$ time which answers each query in $O(\log^2 n)$ time [5]. On the k Center-1D problem, we observe that the input half-plane set H has a special property that the intersections between the x -axis and the bounding lines of the half-planes are ordered from left to right according to the half-plane indices in H . Exploiting this special property and using compact interval trees [10], we build a new data structure in $O(n \log^{1.5} n)$ time that can answer each 2-D sublist LP query in $O(\log^{1.5} n)$ time. This new data structure allows us to solve the k Center-1D problem in $O(n \log^{1.5} n)$ time.

We present our algorithmic scheme in Section 2, and discuss the data structures in Section 3. We assume that no two different demand points in P are at the same position on L . We also assume that the weight of each point in P is positive and finite. Note that these assumptions are only for ease of exposition and our algorithm can be easily extended to the general situation.

2 The Algorithmic Scheme

In this section, we discuss our algorithmic scheme. As pointed out in [2], there may possibly be more than one optimal solution for the k Center-1D problem. Our algorithm focuses on finding one optimal solution, i.e., a center set F .

For any two points p and q on L with $x(p) \leq x(q)$, denote by $[p, q]$ the closed interval of L between p and q . We first sort all points of P from left to right on L . Without loss of generality (WLOG), let $\{p_1, p_2, \dots, p_n\}$ be the sorted order

in increasing coordinates on L . For any two points $p_i, p_j \in P$ with $i \leq j$, denote by $I(i, j)$ the interval $[p_i, p_j]$. Let ψ^* be the value of $\psi(P, F)$ for an optimal solution of k Center-1D. Suppose F is the center set in an optimal solution; for a demand point $p \in P$ and a center $f \in F$, if $(w(p) \cdot d(f, p)) \leq \psi^*$, then we say that p can be *served* by f . It is easy to see that there is an optimal solution F such that each center of F is in $[p_1, p_n]$. Further, as discussed in [2], there is an optimal solution F such that the points of P are partitioned into k intervals $I(1, i_1), I(i_1 + 1, i_2), \dots, I(i_{k-1} + 1, n)$ by integers $i_0 + 1 = 1 \leq i_1 \leq i_2 \leq \dots \leq i_{k-1} \leq n = i_k$, each interval $I(i_{j-1} + 1, i_j)$ contains exactly one center $f_j \in F$, and for each point $p \in P \cap I(i_{j-1} + 1, i_j)$, $(w(p) \cdot d(f_j, p)) \leq \psi^*$ holds. In other words, each center of F serves a subset of consecutive demand points in P .

For any two integers i and j with $1 \leq i \leq j \leq n$, denote by P_{ij} the subset of points of P in the interval $I(i, j)$, i.e., $P_{ij} = \{p_i, p_{i+1}, \dots, p_j\}$ ($P_{ij} = \{p_i\}$ for $i = j$). Consider the following *weighted 1-center* problem: Find a single center (i.e., a point) f in the interval $I[i, j]$ such that the value of $\psi(P_{ij}, f) = \max_{p_t \in P_{ij}} (w(p_t) \cdot d(p_t, f))$ is minimized. Let $\alpha(i, j)$ denote the minimum value of $\psi(P_{ij}, f)$ for this weighted 1-center problem.

For solving the k Center-1D problem, our strategy is to determine $k - 1$ integers $1 \leq i_1 \leq i_2 \leq \dots \leq i_{k-1} \leq n$ such that the value of $\max\{\alpha(1, i_1), \alpha(i_1 + 1, i_2), \dots, \alpha(i_{k-1} + 1, n)\}$ is minimized and this minimized value is ψ^* . Note that in the above formulation, for each value $\alpha(i, j)$, exact one center is determined in the interval $I(i, j)$. To solve this problem, we reduce it to a planar weighted point approximation problem [5], which is called the *weighted step function min- ϵ* problem in [5]. The problem reduction is very straightforward, and due to the space limit it is omitted and can be found in our full paper. Lemma II below follows the results in [5] provided that all points in P have been sorted on L . To apply the algorithm in Lemma II, we need a data structure for answering queries $q(i, j) = \alpha(i, j)$ with $1 \leq i \leq j \leq n$. Suppose such a data structure can be built in $O(\pi(n))$ time and can answer each query $\alpha(i, j)$ in $O(q(n))$ time; then we say the time bounds of the data structure are $O(\pi(n), q(n))$.

Lemma 1. [5] Suppose there is a data structure for the queries $\alpha(i, j)$ with time bounds $O(\pi(n), q(n))$; then the k Center-1D problem can be solved in $O(\min\{\pi(n) + n \cdot q(n), \pi(n) + q(n) \cdot k^2 \log^2 \frac{n}{k}\})$ time.

3 The Data Structure for Answering $\alpha(i, j)$ Queries

In this section, we present a data structure with time bounds $O(n \log^{1.5} n, \log^{1.5} n)$ for answering the $\alpha(i, j)$ queries. In the following, we first model the problem of computing $\alpha(i, j)$ as a problem of finding a lowest point in the common intersection of a set of half-planes. We also discuss some observations.

3.1 Problem Modeling and Observations

Consider a point subset $P_{ij} \subseteq P$ with $i \leq j$. Recall that $x(p_i) \leq x(p_{i+1}) \leq \dots \leq x(p_j)$. To compute $\alpha(i, j)$, we need to find a point f such that the value

$\psi(P_{ij}, f) = \max_{i \leq t \leq j} (w(p_t) \cdot d(f, p_t)) = \max_{i \leq t \leq j} (w(p_t) \cdot |x(f) - x(p_t)|)$ is minimized and $\alpha(i, j)$ is the minimized value. Suppose a center for P_{ij} is at a point f (f is not necessarily an optimal solution). Since $\psi(P_{ij}, f) = \max_{i \leq t \leq j} (w(p_t) \cdot |x(f) - x(p_t)|)$, each point $p_t \in P_{ij}$ defines two constraints: $w(p_t) \cdot (x(f) - x(p_t)) \leq \psi(P_{ij}, f)$ and $-w(p_t) \cdot (x(f) - x(p_t)) \leq \psi(P_{ij}, f)$.

Consider a 2-D xy -coordinate system with L as the x -axis. For each point $p_t \in P$, the inequality $w(p_t) \cdot |x - x(p_t)| \leq y$ defines two (upper) half-planes: $w(p_t) \cdot (x - x(p_t)) \leq y$ and $-w(p_t) \cdot (x - x(p_t)) \leq y$. Note that the two lines bounding the two half-planes intersect at the point p_t on L .

Based on the above discussion, if $p^* = (x^*, y^*)$ is a lowest point in the common intersection of the $2(i - j + 1)$ (upper) half-planes defined by the points in P_{ij} , then $\alpha(i, j) = y^*$ and $x(f) = x^*$ is the coordinate of an optimal center f on L for P_{ij} . Figure 11 shows an example in which each “cone” is formed by two lines (their parts below the x -axis are not needed) bounding the two upper half-planes defined by a point in P_{ij} . Clearly, this is an instance of the 2-D linear programming (LP) problem, which is solvable in $O(j - i + 1)$ time [8, 14]. However, we can make the computation faster by preprocessing. Let $H_P = \{h_1, h_2, \dots, h_{2n}\}$ be the set of $2n$ (upper) half-planes defined by the n points in P , such that for each $1 \leq i \leq n$, the demand point p_i defines h_{2i-1} and h_{2i} . Then to compute $\alpha(i, j)$, it suffices to find the lowest point p^* in the common intersection of the half-planes defined by the points in P_{ij} , i.e., the half-planes in $H_{2i-1, 2j} = \{h_{2i-1}, h_{2i}, h_{2i+1}, h_{2i+2}, \dots, h_{2j-1}, h_{2j}\}$.

We actually consider a more general problem: Given in the plane a set of n upper half-planes $H = \{h_1, h_2, \dots, h_n\}$, each query $q(i, j)$ with $1 \leq i \leq j \leq n$ asks for a lowest point p^* in the common intersection of all half-planes in $H_{ij} = \{h_t \mid i \leq t \leq j\} \subseteq H$. We call this problem the *2-D sublist LP query*. Based on our discussion above, if we solve the 2-D sublist LP query problem, then the $\alpha(i, j)$ queries for the *kCenter-1D* problem can be processed as well in the same time bound. A data structure for the 2-D sublist LP query problem with time bounds $O(n \log n, \log^2 n)$ was given in [5] based on fractional cascading [4].

Yet, the 2-D sublist LP query problem for *kCenter-1D* is special in the following sense. For each half-plane $h \in H_P$ for *kCenter-1D*, we call the x -coordinate of the intersection point between L (i.e., the x -axis) and the line bounding h the *x-intercept* of h (or its bounding line). As discussed above, for each point $p_t \in P$, the *x-intercepts* of both the half-planes h_{2t-1} and h_{2t} defined by p_t are exactly the point p_t . Since all points of P are ordered along L from left to right by their indices, a special property of H_P is that the *x-intercepts* of all half-planes in H_P are ordered from left to right on L by the indices of the half-planes. For a set H of half-planes for a 2-D sublist LP query problem instance, if H has the above special property, then we say that H is *x-intercept ordered*.

Below, we show if H is *x-intercept ordered*, then there is a data structure for the specific 2-D sublist LP query problem with time bounds $O(n \log^{1.5} n, \log^{1.5} n)$. Henceforth, we assume that H is an *x-intercept ordered* half-plane set. In the *kCenter-1D* problem, since all the point weights for P are positive finite values, the bounding line of each half-plane in H_P is neither horizontal nor vertical.

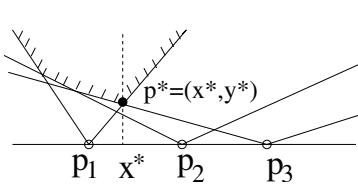


Fig. 1. Illustrating the common intersection of the half-planes defined by three points p_1 , p_2 , and p_3 . The point p^* is the lowest point in the common intersection.

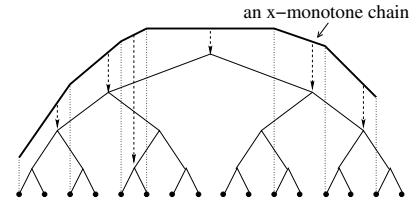


Fig. 2. A compact interval tree: The dashed arrows indicate the nodes at which the chain edges are stored

Thus, we also assume that no bounding line of any half-plane in H is horizontal or vertical. Again, this assumption is only for simplicity of discussion.

We first give some observations. For any two indices i and j , $1 \leq i \leq j \leq n$, let C_{ij} denote the common intersection of the half-planes in H_{ij} , and ∂C_{ij} be the boundary of C_{ij} . Since H contains only upper half-planes, ∂C_{ij} is an x -monotone convex chain consisting of line segments or rays bounding the region C_{ij} from below (e.g., see Fig. 1). Note that no edge of ∂C_{ij} is vertical or horizontal. Denote by ∂C_{ij}^1 (resp., ∂C_{ij}^2) the portion of ∂C_{ij} consisting of edges with negative (resp., positive) slopes. In other words, if p^* is the lowest point in C_{ij} , then ∂C_{ij}^1 (resp., ∂C_{ij}^2) is the portion of ∂C_{ij} on the left (resp., right) of p^* .

The next lemma, useful for analyzing the time bounds of our data structure, is applicable to any x -intercept ordered half-plane set but not applicable to an arbitrary half-plane set. The proof is omitted and can be found in our full paper.

Lemma 2. *For any integers i, i', j , and j' with $1 \leq i \leq i' < j \leq j' \leq n$, the two chains $\partial C_{ii'}$ and $\partial C_{jj'}$ can intersect each other at no more than four points.*

3.2 The Data Structure

We now present a data structure \mathcal{D} with time bounds $O(n \log^{1.5} n, \log^{1.5} n)$ for the 2-D sublist LP query problem with an x -intercept ordered half-plane set H . Our data structure uses the *compact interval tree* given by Guibas, Hershberger, and Snoeyink [10]; to be self-contained, we first briefly review it [10].

Let S be a (global) set of n points in the plane, sorted by the increasing x -coordinates. A *complete interval tree* \mathcal{T} for S is defined as follows. \mathcal{T} is a complete binary tree whose leaves store the points of S in the left-to-right order. Each internal node v of \mathcal{T} corresponds to the minimal interval I_v that covers the x -coordinates of the points stored in the leaves of the subtree rooted at v , and is associated with the midpoint of I_v . A line segment joining two points in S is said to *span* an internal node v of \mathcal{T} if the interval for the line segment contains the midpoint associated with v .

For an x -monotone chain whose vertices are all at (some of) the points of S (e.g., the upper hull or lower hull of a subset of S), we can represent it by a

complete interval tree \mathcal{T} by letting each node v of \mathcal{T} store the edge of the chain that spans v (thus, a chain edge may be stored in multiple nodes of \mathcal{T}). Then, with the help of \mathcal{T} , many operations can be performed in $O(\log n)$ time each, e.g., computing a lowest point on the lower hull of a subset of S .

When there are many such chains to represent, it would be inefficient to use one copy of the complete interval tree for each chain. Instead, for each chain, we build a *compact interval tree* \mathcal{T}' in which each edge of the chain is stored only once, by storing it at the highest node in \mathcal{T} that it spans (see Fig. 2). It was shown in [10] that the compact interval tree \mathcal{T}' can simulate a search from the root to any leaf in \mathcal{T} in $O(1)$ time per step; each step (on a node of \mathcal{T}) determines which chain edge spans that node of \mathcal{T} , or, if none, on which side of the node the chain lies. For an x -monotone chain of k vertices, building a compact interval tree for it takes $O(k)$ time and $O(k)$ space [10].

To present the data structure \mathcal{D} , we first give two data structures \mathcal{D}_1 and \mathcal{D}_2 whose time bounds are $O(n \log n, \log^2 n)$ and $O(n \log^2 n, \log n)$, respectively. Our final data structure \mathcal{D} then combines \mathcal{D}_1 and \mathcal{D}_2 by making a time tradeoff between the preprocessing and the query procedure.

The Data Structure \mathcal{D}_1 . We build the data structure \mathcal{D}_1 as follows. For the given (x -intercept ordered) half-plane set H , build a complete *binary tree* T_H whose i -th leaf corresponds to the i -th half-plane h_i in H . Each internal node v of T_H is associated with a data structure that represents a convex chain C_v which is the boundary of the common intersection of all half-planes corresponding to the leaves in the subtree of T_H rooted at v . Clearly, C_v is x -monotone since H contains only upper half-planes. To support efficient queries, we use a compact interval tree to represent C_v (at the corresponding node v of T_H). For this, we need to know the global point set S that consists of the vertices of the convex chains associated with all internal nodes of T_H . Thus, to build \mathcal{D}_1 , we need two procedures: (1) Compute the global point set S as well as its *complete* interval tree \mathcal{T}_S (for further queries); (2) build the binary tree T_H with every internal node in it associated with a *compact* interval tree.

For the first procedure, starting at the leaves of T_H , in a bottom-up fashion, for each internal node v of T_H , we compute C_v from the two chains stored in v 's two children L_v and R_v in T_H , in linear time. Here we simply store C_v as a linked list. Also, we maintain a point set S_v , which is the union of three sets: S_{L_v} , S_{R_v} , and the set of all vertices of C_v . Let $S_v = \emptyset$ for each leaf v of T_H . We maintain the points in S_v in the increasing order of their x -coordinates. Clearly, the set S_v can be computed in $O(|S_v|)$ time. The sorted point set S is obtained when we reach the root of T_H , i.e., $S = S_v$ if v is the root of T_H .

We first analyze the size of S . For each internal node v of T_H , let $S'_v = S_v - S_{L_v} - S_{R_v}$. Then each point of S'_v is an intersection point between the two convex chains C_{L_v} and C_{R_v} . By Lemma 2, the number of intersection points between C_{L_v} and C_{R_v} is at most four; thus, $|S'_v| \leq 4$. Hence, the size of S_v is $O(m)$, where m is the number of leaves in the subtree of T_H rooted at v . Thus, $|S| = O(n)$. Further, it is easy to see that the above procedure computes the points in S in a sorted order in $O(n \log n)$ time. The complete interval tree \mathcal{T}_S

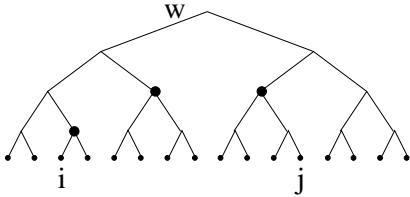


Fig. 3. Answering a query $\alpha(i, j)$: w is the LCA of i and j

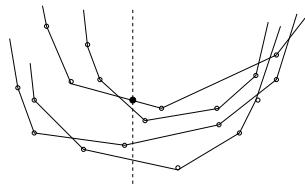


Fig. 4. The dashed vertical line is $l_{v'}$. The lowest point p^* is on the right side of $l_{v'}$.

on S can then be built in additional $O(n)$ time. Therefore, the first procedure takes totally $O(n \log n)$ time.

For the second procedure, at each internal node v of T_H , we compute the compact interval tree T_v for the chain C_v , which can be done in $O(|C_v|)$ time and $O(|C_v|)$ space [10], where $|C_v|$ is the number of vertices of C_v . Since each vertex of C_v is in the set S_v , we have $|C_v| \leq |S_v|$, and thus $|C_v| = O(|S_v|) = O(m)$, where m is the number of leaves in the subtree rooted at v . Hence, the second procedure takes $O(n \log n)$ time. This finishes the construction of the data structure \mathcal{D}_1 .

To answer a query $q(i, j)$, we need to find the lowest point p^* in the common intersection of the half-planes in H_{ij} . We first locate in T_H the lowest common ancestor (LCA) w of the two leaves i and j ; this takes $O(1)$ time [11] (with $O(n)$ time preprocessing; actually, since T_H is a complete binary tree, finding LCA in T_H is easy). By following the paths from w to i and to j , we identify $O(\log n)$ nodes of T_H such that the half-planes in H_{ij} correspond exactly to the leaves in the subtrees rooted at these nodes (in Fig. 3, such nodes are marked by black dots). Let Q be the set of these $O(\log n)$ nodes. Then, the common intersection of the common intersection regions associated with all nodes in Q is the common intersection of all half-planes in H_{ij} . Each node in Q is associated with a compact interval tree representing the boundary of the corresponding common intersection region. Denote by Q_T the set of the corresponding $O(\log n)$ compact interval trees for Q . To obtain the lowest point p^* , we search in the complete interval tree T_S and in the compact interval trees of Q_T simultaneously, as follows. We walk from the root of T_S and keep track of our position in each compact interval tree of Q_T , in the same way as described in [10]. At each node v' of T_S during the walking, we decide in $O(\log n)$ time which child of v' to proceed by the following decision procedure. For each compact interval tree T in Q_T , we can determine in $O(1)$ time the edge of the chain represented by T that spans v' (refer to [10] for more details). Hence, in $O(\log n)$ time, we find all $O(\log n)$ edges spanning the node v' in the $O(\log n)$ chains represented by the trees in Q_T . This means that we obtain all $O(\log n)$ chain edges which intersect the vertical line $l_{v'}$ passing through the midpoint of the interval associated with v' (see Fig. 4). Subsequently, by using similar techniques as in [8, 14], we can decide in $O(\log n)$ time whether p^* is at the highest intersection between $l_{v'}$ and the $O(\log n)$ chain edges, and if not, p^* lies on which side of $l_{v'}$ (see Fig. 4). Since the height of T_S is $O(\log n)$, in $O(\log^2 n)$ time, p^* can be found.

Note that Chan [3] presented an algorithm which computes the lowest point in the common intersection of m 2-D convex n -gons in $O(m \log n \log m / \log \log m)$ time (without preprocessing). In \mathcal{D}_1 , if we use an array to store each chain C_v , then for each query, after we obtain the $O(\log n)$ chains, by applying the algorithm in [3] (with $m = O(\log n)$), the query takes $O(\log^2 n \log \log n / \log \log \log n)$ time, which is a little larger than our $O(\log^2 n)$ time bound above.

The Data Structure \mathcal{D}_2 . We now discuss \mathcal{D}_2 of $O(n \log^2 n, \log n)$ time. We still build a complete binary tree T_H for the half-plane set H in the same way as for \mathcal{D}_1 . Also, as in \mathcal{D}_1 , we compute the global point set S and construct its complete interval tree T_S . However, unlike \mathcal{D}_1 , each internal node v of T_H is associated with a different data structure. Suppose the half-planes corresponding to the leaves from left to right in the subtree rooted at v are $h_{l(v)}, h_{l(v)+1}, \dots, h_{r(v)}$. We associate v with an array A_v of size m with $m = r(v) - l(v) + 1$, and each element $A_v[i]$ ($0 \leq i \leq m - 1$) is a pointer pointing to the root of a compact interval tree that represents the boundary of the common intersection of the $i+1$ half-planes $h_{l(v)}, \dots, h_{l(v)+i}$. We call the array A_v the *prefix array* of v and the m compact interval trees the *prefix compact interval trees* of v . Symmetrically, we associate with v a *suffix array* B_v and m *suffix compact interval trees*. As in [10], we can use a standard partially persistent data structure [7] to compute all prefix compact interval trees of v . Specifically, to compute the compact interval tree corresponding to $A_v[i+1]$, we insert the half-plane $h_{l(v)+i+1}$ into the compact interval tree corresponding to $A_v[i]$. Since the boundary of the common intersection of a set of upper half-planes is an x -monotone convex chain, each such insertion takes $O(\log m)$ time. Thus, building the partially persistent data structure takes $O(m \log m)$ time for an internal node v of T_H with m leaves in the subtree rooted at v . Similarly, we also use a partially persistent data structure to compute all suffix compact interval trees for the node v . Therefore, the total construction time of \mathcal{D}_2 is $O(n \log^2 n)$.

To answer a query $q(i, j)$, again, we find the lowest point p^* in the common intersection of the half-planes in H_{ij} . We first find the LCA w of the leaves i and j in T_H . Denote the left and right children of w by L_w and R_w , respectively. Locate the element corresponding to the leaf i in the suffix array B_{L_w} of the node L_w and the element corresponding to the leaf j in the prefix array A_{R_w} of the node R_w . Then, we access the two compact interval trees which represent the boundary of the common intersection C_L of the half-plane subset $H_{i,r(L_w)}$ and the boundary of the common intersection C_R of the half-plane subset $H_{l(R_w),j}$, respectively. Note that $l(R_w) = r(L_w) + 1$, and thus $H_{ij} = H_{i,r(L_w)} \cup H_{l(R_w),j}$. Thus, the common intersection of the half-planes in H_{ij} is the common intersection of C_L and C_R . As in the query procedure of the first data structure \mathcal{D}_1 , we can search in the complete interval tree T_S and these two compact interval trees for $H_{i,r(L_w)}$ and $H_{l(R_w),j}$ to find p^* . Here, at each node of T_S , we only have at most two chain edges to check (in $O(1)$ time). Thus the total query time is $O(\log n)$.

The Data Structure \mathcal{D} . Our final data structure \mathcal{D} combines \mathcal{D}_1 and \mathcal{D}_2 , by making a time tradeoff between the preprocessing and the query procedure.

After building the same complete binary tree T_H for the half-plane set H with each of its internal nodes associated with a compact interval tree as in \mathcal{D}_1 , we associate a chosen subset of the internal nodes of T_H (called the *crucial nodes*) with the same data structures as in \mathcal{D}_2 , i.e., the prefix (resp., suffix) arrays and prefix (resp., suffix) compact interval trees. The crucial nodes are chosen in the following way. We say that all internal nodes with the same depth in T_H are at the same *level*. Thus, T_H has $O(\log n)$ levels. We partition the $O(\log n)$ levels of T_H into $\sqrt{\log n}$ layers and each layer contains $\sqrt{\log n}$ successive levels. In each layer, the nodes at the topmost level are chosen as crucial nodes. Again, for each crucial node v , we build its prefix array and suffix array, as well as the corresponding prefix and suffix compact interval trees maintained by partially persistent data structures. This finishes the construction of \mathcal{D} .

The time for constructing \mathcal{D} is dominated by that of processing the crucial nodes of T_H , i.e., building the corresponding data structures for each crucial node. As in the analysis for \mathcal{D}_2 , processing the crucial nodes at the same level takes totally $O(n \log n)$ time. There are $\sqrt{\log n}$ layers and each layer has only one level of crucial nodes. Thus, \mathcal{D} can be built in $O(n \log^{1.5} n)$ time.

To answer a query $q(i, j)$, again, we find the lowest point p^* in the common intersection of the half-planes in H_{ij} . We first determine the LCA w of the leaves i and j in T_H . If the children of w happen to be crucial nodes, then we just use the same query procedure as for \mathcal{D}_2 to find p^* , in $O(\log n)$ time. Otherwise, following the path in T_H from the left child of w to i , let the first crucial node encountered (if any) be v_i ; similarly, along the path from the right child of w to j , let the first crucial node encountered (if any) be v_j . Locate the element corresponding to the leaf i in the suffix array B_{v_i} of v_i and the element corresponding to the leaf j in the prefix array A_{v_j} of v_j . Using these two elements, we access the two compact interval trees that represent the boundary of the common intersection of the half-planes in $H_{i,r(v_i)}$ and the boundary of the common intersection of the half-planes in $H_{l(v_j),j}$, respectively. In addition, as in the query procedure for \mathcal{D}_1 , by following the paths from w to v_i and from w to v_j in T_H , we find $O(\sqrt{\log n})$ compact interval trees representing the boundaries of the $O(\sqrt{\log n})$ common intersection regions whose common intersection is that of the half-planes in $H_{ij} - \{H_{i,r(v_i)} \cup H_{l(v_j),j}\}$. The fact of having $O(\sqrt{\log n})$ such compact interval trees (instead of $O(\log n)$) follows from our definition of crucial nodes in T_H and that v_i (resp., v_j) is the first encountered crucial node on the path from the left (resp., right) child of w to the leaf i (resp., j). Thus, we have totally $O(\sqrt{\log n}) + 2$ compact interval trees. By using a similar query procedure as for \mathcal{D}_1 , p^* can be found in $O(\log^{1.5} n)$ time. Note that if there is no crucial node on the path from the left (resp., right) child of w to the leaf i (resp., j), then the children of w are both in the same layer as the leaf i (resp., j). Since there are at most $\sqrt{\log n}$ levels in each layer, similarly, p^* can also be found in $O(\log^{1.5} n)$ time in this case. In summary, we have the following result.

Theorem 1. *For the 2-D sublist LP query problem, if the given half-plane set H is x -intercept ordered, then there exist data structures for it with time bounds $O(n \log n, \log^2 n)$, $O(n \log^2 n, \log n)$, and $O(n \log^{1.5} n, \log^{1.5} n)$.*

Remark: It is straightforward to extend our data structures to a more general case in which the x -intercept ordered half-plane set H can also contain lower half-planes. The same result in Theorem 1 still holds for finding a lowest point or reporting the common intersection is empty.

By combining Theorem 1 with Lemma 1, we have the following result.

Theorem 2. *The k Center-1D problem is solvable in $O(\min\{n \log^{1.5} n, n \log n + k^2 \log^2 \frac{n}{k} \log^2 n\})$ time.*

References

1. Bender, M., 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. Bhattacharya, B., Shi, Q.: Optimal Algorithms for the Weighted p -Center Problems on the Real Line for Small p . In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 529–540. Springer, Heidelberg (2007)
3. Chan, T.: Deterministic algorithms for 2-D convex programming and 3-D online linear programming. *Journal of Algorithms* 27, 147–166 (1998)
4. Chazelle, B., Guibas, L.: Fractional cascading: I. A data structuring technique. *Algorithmica* 1(1), 133–162 (1986)
5. Chen, D.Z., Wang, H.: Approximating Points by a Piecewise Linear Function: I. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 224–233. Springer, Heidelberg (2009)
6. Cole, R.: Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM* 34(1), 200–208 (1987)
7. Driscoll, J., Sarnak, N., Sleator, D., Tarjan, R.: Making data structures persistent. *Journal of Computer and System Sciences* 38(1), 86–124 (1989)
8. Dyer, M.: Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comp.* 13(1), 31–45 (1984)
9. Frederickson, G.: Parametric Search and Locating Supply Centers in Trees. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1991. LNCS, vol. 519, pp. 299–319. Springer, Heidelberg (1991)
10. Guibas, L., Hershberger, J., Snoeyink, J.: Compact interval trees: A data structure for convex hulls. *International Journal of Computational Geometry and Applications* 1(1), 1–22 (1991)
11. Harel, D., Tarjan, R.: Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing* 13, 338–355 (1984)
12. Jeger, M., Kariv, O.: Algorithms for finding P -centers on a weighted tree (for relatively small P). *Networks* 15(3), 381–389 (1985)
13. Kariv, O., Hakimi, S.: An algorithmic approach to network location problems. I: The p -centers. *SIAM J. on Applied Mathematics* 37(3), 513–538 (1979)
14. Megiddo, N.: Linear programming in linear time when the dimension is fixed. *Journal of the ACM* 31(1), 114–127 (1984)
15. Megiddo, N., Tamir, A.: New results on the complexity of p -centre problems. *SIAM J. on Computing* 12(4), 751–758 (1983)
16. Megiddo, N., Tamir, A., Zemel, E., Chandrasekaran, R.: An $O(n \log^2 n)$ algorithm for the k -th longest path in a tree with applications to location problems. *SIAM J. on Computing* 10, 328–337 (1981)

Outlier Respecting Points Approximation*

Danny Z. Chen and Haitao Wang**

Department of Computer Science and Engineering,
University of Notre Dame, Notre Dame, IN 46556, USA
`{dchen, hwang6}@nd.edu`

Abstract. In this paper, we consider a generalized problem formulation of computing a functional curve to approximate a point set in the plane with outliers. The goal is to seek a solution that not only optimizes its original objectives, but also somehow accommodates the impact of the outliers. Based on a new model of accommodating outliers, we present efficient geometric algorithms for various versions of this problem (e.g., the approximating functions are step functions or piecewise linear functions, the points are unweighted or weighted, etc). All our results are first known. Our new model and techniques for handling outliers may be useful to other applications as well.

1 Introduction

Outlier detection algorithms have been proposed for data mining applications (e.g., [1,2,17,23]). Related topics have also been considered in computational geometry (e.g., [4,5,11,18,24]). However, in all previous work, outliers are simply removed or ignored once they are detected, and the algorithms continue to achieve other goals based on the remaining data. This practice seems reasonable because outliers are usually useless data. But this view may not be correct in certain situations. First, some outliers do contain useful information. For instance, certain outliers may reflect changes in system behavior or human activities. Second, there is no single, generally accepted definition for outliers. To detect outliers, different algorithms use different criteria, implying that some data not considered as outliers in one algorithm may be taken as outliers in another algorithm. For example, in [17], a point p in a data set P is considered an outlier if no more than l points in P are at a distance at most d from p , for two parameters l and d . In [23], a point $p \in P$ is taken as an outlier if the distance from p to its l -th nearest neighbor in P is large enough for a parameter l . Therefore, some outliers detected by an algorithm may not be real “outliers”. Third, it is often difficult to know exactly how many outliers are actually in the data set for a specific problem instance, even though some algorithms (e.g., [7]) assume that an input parameter is used to specify the number of outliers involved; thus, it may not be good to rely only on this parameter when solving

* This research was supported in part by NSF under Grant CCF-0916606.

** Corresponding author.

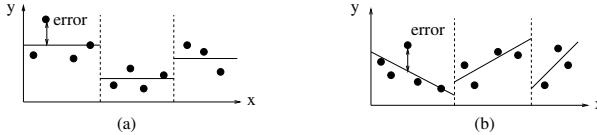


Fig. 1. (a) A step function. (b) A piecewise linear function.

the problem instance. Hence, it makes sense to design algorithms that are able to not only detect outliers but still somehow take into account the influence of the outliers when achieving their goals.

This paper studies this type of algorithms for points approximation problems, which have applications in histogram constructions [12,13,14,15]. Precisely, our solutions approximate a set of planar points by a functional curve such that: (1) the outliers are detected; (2) among all curves that minimize the approximation error or curve size for non-outlier points, the one that also minimizes the approximation error of the outliers is chosen as output.

1.1 Statements of Problems

Given a point set $P = \{p_1, \dots, p_n\}$ in the plane with $p_i = (x_i, y_i)$, we want to find a piecewise linear functional curve f to approximate P . The approximation error of each point p_i is the *vertical distance* between p_i and f , i.e., $d(p_i, f) = |y_i - f(x_i)|$. The approximation error of f , denoted by $e(P, f)$, is defined to be $\max_{p_i \in P} d(p_i, f)$, commonly known as the *uniform* or L_∞ metric. The *size* of f , $|f|$, is the number of line segments of f . Given an error tolerance $\epsilon \geq 0$, the *min-#* problem is to find an approximating function f subject to certain constraints such that $e(P, f) \leq \epsilon$ and the size $|f|$ of f is minimized. Given an integer $k > 0$, the *min- ϵ* problem is to find an approximating function f subject to certain constraints such that $|f| \leq k$ and $e(P, f)$ is minimized.

Depending on the constraints on f , there are several problem variants. (1) *Points approximation by a step function*: Given P , the sought f is a step function, represented by a sequence of horizontal line segments whose x -intervals do not overlap in their interior (see Fig. 1(a)); we denote this problem by SF. (2) *Points approximation by a piecewise linear function*: Given P , f is piecewise linear and any two consecutive line segments of f need not be jointed (see Fig. 1(b)); denote this problem by PF. (3) *Weighted versions*: Each point $p_i \in P$ has a weight $u_i \geq 0$ and $d(p_i, f)$ is defined as $u_i \cdot |y_i - f(x_i)|$; denote the weighted versions of SF and PF by WSF and WPF, respectively. (4) *Ordinary violation versions*: When approximating P , at most g points in P are allowed to violate the error tolerance and these points are called *violation points* or *outliers*. For example, if P' is the set of outliers with $|P'| \leq g$, then $e(P, f) = \max_{p_i \in P \setminus P'} d(p_i, f)$. We refer to the other (non-outlier) points in P as *normal points*. Denote the ordinary violation versions of SF, PF, WSF, and WPF by VSF, VPF, VWSF, and VWPF, respectively. These problems, which have applications in histogram constructions for database query optimization [6,7,12,13,14,15], have been studied, and the results are in Table II.

Table 1. Summary of the results ($\alpha > 0$ is an arbitrarily small constant)

	$\min\#$ (given ϵ , minimizing $ f $)	$\min\epsilon$ (given $ f \leq k$, minimizing ϵ)
SF	$O(n)$ [8]	$O(n)$ [12]
WSF	$O(n)$ [16]	$O(\min\{n \log^2 n, n \log n + k^2 \log^4 n\})$ [6]
PF	$O(n)$ [21]	$O(\min\{n \log n, n + k^2 \log^3 n \log \log n\})$ [6]
WPF	$O(n)$ [21]	$O(\min\{n \log^6 n, n \log n + k^2 \log^8 n\})$ [6]
VSF	$O(ng^2)$ [12]	$O(ng^2 \log n)$ [12], $O(ng^2 k \log(\log^* n))$ [7]
VWSF	$O(ng^2)$ [7]	$O(n^2 + ng^2 \log n)$ [7]
VPF and VWPF	$O(ng^4 \log^2 n)$ [7]	$O(ng(n \log g + g^3 n^\alpha) \min\{kg, \log n\})$ [7]
RVSF	$O(ng^3 \log \frac{n}{g})$ [this paper]	$O(ng^3 \log \frac{n}{g})$ [this paper]
RVWSF	$O(ng^3 \log \frac{n}{g} \log g)$ [this paper]	$O(n^2 + ng^3 \log \frac{n}{g} \log g)$ [this paper]
RVPF and RVWPF	$O(n^{2+\alpha} g^{\frac{7}{2} + \frac{3}{2}\alpha} \log \frac{n}{g})$ [this paper]	$O(n^{2+\alpha} g^{\frac{7}{2} + \frac{3}{2}\alpha} \log \frac{n}{g})$ [this paper]

In this paper, we study the following new problem: *Outlier-respecting points approximation*. This is a generalization of the ordinary violation versions in that the output functional curve f also minimizes the impact of the outliers. Specifically, with the (original) objective that the maximum error or the curve size of f for the normal points is minimized, the output curve f in addition also minimizes the maximum error of the outliers. Formally, given a point set P and an integer $g > 0$, we compute f and a set P' of outliers in P , with $|P'| \leq g$, such that: (1) the objective for the normal points, i.e., $e(P, f) = \max_{p_i \in P \setminus P'} d(p_i, f)$ (for $\min\epsilon$) or $|f|$ (for $\min\#$), is minimized, and (2) if there exist multiple optimal solutions for (1), then the output optimal solution is chosen to be one that also minimizes the maximum approximation error of the outliers, i.e., $\max_{p_i \in P'} d(p_i, f)$. Hence, we say that the outliers are *respected* by the output solution f in that their impact to f is held as small as possible. Denote the outlier-respecting violation versions of SF, WSF, PF, and WPF by RVSF, RVWSF, RVPF, and RVWPF, respectively. We study both the $\min\#$ and $\min\epsilon$ versions of all these outlier-respecting problems, for which we are not aware of any previous work.

Note that for the same point set, a solution for an ordinary violation problem can be very different from that for an outlier-respecting violation problem.

In the following paper, we assume no two points in P have the same x -coordinate. This assumption is only for the ease of discussion. We also assume the points in P are already sorted in increasing x -coordinates (from left to right) since otherwise we sort them in $O(n \log n)$ time. Throughout the paper, we use P_{ij} ($i \leq j$) to denote the subset of consecutive points p_i, p_{i+1}, \dots, p_j in P .

1.2 Our Results

We give algorithms for the $\min\#$ and $\min\epsilon$ versions of the four outlier-respecting problems. For the $\min\#$ versions, we develop a high-level algorithmic framework that can be applied to each of these problems. Then for each individual $\min\#$ problem, based on its specific geometric properties, we derive different data structures and procedures for carrying out the algorithmic components of the general framework. The $\min\epsilon$ versions can be solved easily after we solve the $\min\#$ versions. Our results are summarized in Table II.

Our *min-#* algorithmic framework is based on dynamic programming. Because the approximation errors of the outliers are also considered by our model, comparing with the ordinary violation problems, some new issues arise in our new problems. One major difference is that for the ordinary violation problems, an optimal solution can be obtained locally while for the outlier-respecting versions, the decisions must be made based on some global information.

In our *min-#* framework, a key algorithmic component is to compute the value of any w'_{ijq} with $1 \leq i \leq j \leq n$ and $0 \leq q \leq g$, which is the maximum approximation error of the outliers if we use one optimal line segment to approximate the points of P_{ij} with q outliers. For each individual *min-#* problem, we design a procedure for computing w'_{ijq} , which makes use of the geometric structures of the specific problem. For the problem RVSF (resp., RVWSF), we show that the value of w'_{ijq} is determined by only $O(g)$ points and can be computed in $O(g)$ (resp., $O(g \log g)$) time (with preprocessing). However, for RVPF and RVWPF, the value of w'_{ijq} can be affected by all points in P_{ij} , implying that $O(n)$ time may be needed for computing any w'_{ijq} . By using some interesting geometric techniques, we give an efficient way of computing w'_{ijq} .

In the rest of this paper, we first present the high-level algorithmic framework and then discuss the algorithmic components for each individual problem.

2 The Algorithmic Framework

For any subset P_{ij} and q with $0 \leq q \leq g$, if a line segment s (under the specific constraint on f) can be used to approximate all points in P_{ij} such that the ordinary error is at most ϵ and the number of outliers is at most q , then we say that P_{ij} is *q-approximatable* and s is a *feasible segment*. For any $1 \leq i \leq j \leq n$ and $0 \leq q \leq g$, denote by w'_{ijq} the maximum approximation error of the outliers if we use one optimal line segment to approximate the points of P_{ij} with q outliers. Denote by W' the time for computing an arbitrary w'_{ijq} .

Theorem 1. *The outlier-respecting violation min-# versions are solvable in $O(ng^2 \log \frac{n}{g} \cdot W' + R)$ time, and the min- ϵ versions are solvable in $O(ng^2 \log \frac{n}{g} \cdot W' + R + T)$ time, where $O(R)$ and $O(T)$ are the running times of the min-# and min- ϵ algorithms for the corresponding ordinary violation problem, respectively.*

Our algorithmic framework for Theorem 1 is based on dynamic programming, which is omitted due to the space limit and can be found in the full paper. Consequently, for each outlier-respecting violation problem, the remaining key task is to compute the component for W' , i.e., w'_{ijq} . In the following sections, for each specific problem, we give a corresponding algorithm for computing w'_{ijq} .

For RVSF, with $O(n \log g \log n)$ time preprocessing, we can compute any w'_{ijq} in $O(W') = O(g)$ time. Due to the space limit, the details for this are omitted and can be found in our full paper. Then, the following result follows from Theorem 1 (with $W' = O(g)$, $R = O(ng^2)$ [7], and solving the min- ϵ ordinary VSF takes $O(T) = O(ng^2 \log n)$ time [12]). Note that $n \log g \log n = O(ng \log \frac{n}{g})$.

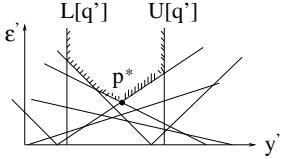


Fig. 2. The lowest point p^* in the common intersection of a set of upper half-planes

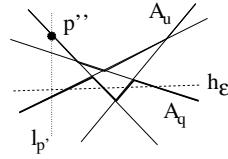


Fig. 3. Illustrating Lemma 5: p'' is the intersection of $l_{p'}$ and A_u

Theorem 2. *The min-# RVSF problem can be solved in $O(ng^3 \log \frac{n}{g})$ time. Its min- ϵ version can also be solved within the same time bound.*

3 Algorithmic Components for the Problem RVWSF

This section presents our algorithm for computing w'_{ijq} for RVWSF, which seeks a step function f as its approximation solution. Recall that in this problem, each point $p_t = (x_t, y_t)$ in P has a weight $u_t \geq 0$, and $d(p_t, f) = u_t \cdot |y_t - f(x_t)|$.

For each point $p_t \in P$, we define its *approximation interval* as $[y_t - \epsilon/u_t, y_t + \epsilon/u_t]$. A horizontal segment $s_{y'}$ with a y -coordinate y' can approximate the point p_t within an error ϵ if and only if $y' \in [y_t - \epsilon/u_t, y_t + \epsilon/u_t]$ and x_t is in the x -interval of $s_{y'}$. We call the value $y_t - \epsilon/u_t$ (resp., $y_t + \epsilon/u_t$) the *lower end* (resp., *upper end*) of the approximation interval of p_t , denoted by $le(p_t)$ (resp., $ue(p_t)$). Assume the array $LE[0 \dots q]$ (resp., $UE[0 \dots q]$) stores the $q+1$ largest lower ends (resp., smallest upper ends) of the points in P_{ij} , with $LE[0] \geq LE[1] \geq \dots \geq LE[q]$ (resp., $UE[0] \geq \dots \geq UE[q]$). We will discuss later how to obtain the above two arrays. Note that in VWSF, P_{ij} is q -approximatable if and only if there is a q' , $0 \leq q' \leq q$, such that $LE[q'] \leq UE[q']$, and if such a q' exists, then any segment with a y -coordinate in the interval $[LE[q'], UE[q']]$ is a feasible one for P_{ij} . In RVWSF, we need to find a feasible segment that minimizes the outlier error. Denote by S the set of all feasible segments for P_{ij} , ϵ , and q . Let $Q = \{q' \mid 0 \leq q' \leq q, LE[q'] \leq UE[q']\}$. The next lemma (proof omitted) characterizes the set S .

Lemma 1. *Let $S' = \cup_{q' \in Q} \{s_{y'} \mid y' \in [LE[q'], UE[q']] \}$. Then $S = S'$.*

By Lemma 1, our task is to find a segment s_{y^*} in S that minimizes the outlier error. For each $q' \in Q$, let $S_{q'}$ denote the segment set $\{s_y \mid y \in [LE[q'], UE[q']] \}$. Hence $S = \cup_{q' \in Q} S_{q'}$. If we can find the optimal segment with minimum outlier error in each $S_{q'}$, s_{y^*} is the one with the smallest outlier error. However, since we cannot enumerate all segments in $S_{q'}$ which may have infinitely many segments, computing the optimal segment in each $S_{q'}$ seems elusive at first sight.

We tackle this problem based on a geometric formulation. For each $q' \in Q$, let $V_{q'}$ be the set of points corresponding to the elements in $LE[0 \dots q'-1]$ and $UE[q'+1 \dots q]$. For any segment s_y with $LE[q'] \leq y \leq UE[q']$, its outlier set must be a subset of $V_{q'}$. Without loss of generality, we treat $V_{q'}$ as the outlier

set of s_y (the normal points in $V_{q'}$, if any, do not affect the result). Note that $|V_{q'}| \leq q$. For each point $p_t = (x_t, y_t)$ with a weight $u_t \geq 0$ in $V_{q'}$, we define two upper half-planes $u_t(y_t - y') \leq \epsilon'$ and $-u_t(y_t - y') \leq \epsilon'$ in a 2-D orthogonal coordinate system with y' as its abscissa and ϵ' as its ordinate. Let $H_{q'}$ denote the set of all upper half-planes defined by the points in $V_{q'}$ plus two more half-planes, $h_{q'1}: y' \geq LE[q']$ and $h_{q'2}: y' \leq UE[q']$. Thus $|H_{q'}| = 2|V_{q'}| + 2 = O(q)$. Let p^* be the lowest point in the common intersection of these $O(q)$ half-planes. It is not difficult to see that the y -coordinate of the optimal segment in $S_{q'}$ is the y' -coordinate of p^* and the outlier error of that optimal segment is the ϵ' -coordinate of p^* (see Fig. 2). Therefore, computing the optimal segment in each $S_{q'}$ can be modeled as a 2-D linear programming (LP) problem [1019].

Since $|Q| \leq g+1$, by using an LP algorithm to compute the optimal segment in $S_{q'}$ for each $q' \in Q$ in $O(g)$ time, computing w'_{ijq} takes $O(g^2)$ time. Next, we give an improved solution by showing that computing the optimal segment in $S_{q'}$ can be done faster by making use of some previously processed information. Let V denote the set of points corresponding to all elements in the two arrays LE and UE . Our algorithm is based on the lemma below (proof omitted).

Lemma 2. *Given $Q = \{q_1, q_2, \dots, q_m\}$ with $q_1 < q_2 < \dots < q_m$ ($m = |Q|$), suppose we generate V_{q_a} from $V_{q_{a-1}}$, for each $a = 2, 3, \dots, m$, by deleting points in $V_{q_{a-1}} \setminus V_{q_a}$ and inserting points in $V_{q_a} \setminus V_{q_{a-1}}$; then each point in V can be inserted at most once and/or deleted at most once.*

As shown above, to compute w'_{ijq} , it suffices to find the lowest point in the common intersection of the half-planes in $H_{q'}$ for each $q' \in Q$. Based on Lemma 2, an efficient algorithm works as follows. Since there is no lower half-plane in $H_{q'}$, the sought lowest point is on the upper envelope of the set of lines bounding the half-planes in $H_{q'}$. Because the upper envelope is a convex chain, the key idea is to use a dynamic data structure to maintain the upper envelope for each $H_{q'}$, to support insertions and deletions of upper half-planes as well as the lowest point queries. The data structure in [3] can process each insertion and deletion in amortized $O(\log n)$ time and a lowest point query in the worst-case $O(\log n)$ time. Since the total number of involved half-planes is $O(g)$, using a data structure in [3] and by Lemma 2 the total time is $O(g \log g)$ for computing the lowest points for all $H_{q'}$'s, $q' \in Q$. Therefore, w'_{ijq} can be computed in $W' = O(g \log g)$ time provided that the two arrays $LE[0 \dots q]$ and $UE[0 \dots q]$ are available.

We only discuss how to compute the array LE (the case for UE is similar). We compute a larger array $LE[0 \dots g]$ (recall $q \leq g$), which is to find the $g+1$ points in P_{ij} with the largest lower ends. The problem can be solved efficiently using a $(g+1)$ -range-maxima data structure [7], which, after $O(n \log g \log n)$ time preprocessing, can obtain $LE[0 \dots g]$ (in sorted order) in $O(g)$ time using a $(g+1)$ -range-maxima query for query input (i, j) . In summary, with $O(n \log g \log n)$ time preprocessing, we can compute any w'_{ijq} in $O(W') = O(g \log g)$ time.

By Theorem 1, we have the following result (with $R = O(ng^2)$ [7], and the ordinary VWSF $\min\epsilon$ version solvable in $O(T) = O(n^2 + ng^2 \log n)$ time [7]).

Theorem 3. *The min-# RVWSF problem can be solved in $O(ng^3 \log \frac{n}{g} \log g)$ time. Its min- ϵ version can be solved in $O(n^2 + ng^3 \log \frac{n}{g} \log g)$ time.*

4 Algorithmic Components for RVPF and RVWPF

This section presents our algorithm for computing w'_{ijq} for the RVPF and RVWPF. We shall focus on RVWPF since RVPF is a special case of RVWPF. The big difference here from the step function cases is that the value of w'_{ijq} can be affected by all points in P_{ij} instead of only $O(g)$ points as in the step function cases.

Our approach is based on a geometric modeling and several techniques. The q -level of the arrangement of a set of planes in 3-D is the closure of points that lie on the input planes and have exactly q planes above them [4][25]. For each point $p_t = (x_t, y_t)$ with a weight $u_t \geq 0$ in P , to model the approximation error of p_t , we define two upper half-spaces $u_t(x_t a - b - y_t) \leq \delta$ and $-u_t(x_t a - b - y_t) \leq \delta$ in a 3-D orthogonal coordinate system \mathcal{P}' with a as the x -axis, b as the y -axis, and δ as the z -axis. Denote the two planes bounding these two half-spaces by h_{t1} and h_{t2} , respectively. This geometric modeling has the following properties. Observe that h_{t1} and h_{t2} intersect at a line lying on the plane $h_0 : \delta = 0$ in \mathcal{P}' . For any line l perpendicular to h_0 , suppose l intersects h_{t1} and h_{t2} at two points with δ -coordinates δ_1 and δ_2 , respectively; then it must be $\delta_1 = -\delta_2$. A key observation is that if we use a segment on a line defined by $y = a'x - b'$ to approximate P_{ij} and suppose l is the line perpendicular to h_0 in \mathcal{P}' and passing the point $(a', b', 0)$ on h_0 , then the approximation error of any point $p_t \in P_{ij}$ is the δ -coordinate of the higher intersection point of the line l with the two planes h_{t1} and h_{t2} . Denote the set of planes bounding the $2|P_{ij}|$ half-spaces in 3-D defined above for P_{ij} by H_{ij} , the 3-D arrangement of all planes in H_{ij} by A , and the q -level of A by A_q . The lemma below (proof omitted) is critical.

Lemma 3. *Suppose a single segment s' lying on the line $y = a'x - b'$ is used to approximate P_{ij} with exactly $\min\{|P_{ij}|, q\} \geq 0$ outliers. If $|P_{ij}| \leq q$, then the ordinary error of s' is zero (all points in P_{ij} are taken as outliers). Otherwise, let $p' = (a', b', \delta')$ be the intersection point between the q -level A_q of A and the line perpendicular to h_0 in \mathcal{P}' and containing the point $(a', b', 0)$ on h_0 ; then δ' is the ordinary error of s' . Further, when $|P_{ij}| > q$, s' is a feasible segment if and only if $\delta' \leq \epsilon$.*

When there are multiple feasible segments for approximating P_{ij} with q outliers, we need to find one that minimizes the outlier error. Let S be the set of all feasible segments for P_{ij} . Denote by $s(a', b')$ the segment defined by $y = a'x - b'$. Let h_ϵ be the plane $\delta = \epsilon$ in \mathcal{P}' , R_q be the 3-D region consisting of all points no lower than A_q and no higher than h_ϵ in \mathcal{P}' , and R'_q be the projection of R_q onto the plane h_0 along the δ -axis. We have the lemma below (proof omitted).

Lemma 4. *Let $S' = \{s(a', b') \mid \text{for every point } p' = (a', b', 0) \in R'_q\}$. Then $S = S'$.*

Our objective is to find a segment in S that minimizes the outlier error. Lemma 5 (proof omitted) provides a way to determine the outlier error of any given feasible segment. Let A_u denote the upper envelope of the planes in H_{ij} .

Lemma 5. *For any point $p' = (a', b', 0) \in R'_q$, the outlier error of using the segment $s(a', b')$ to approximate P_{ij} with exactly q outliers is the δ -coordinate of the intersection point between A_u and the line $l_{p'}$ in \mathcal{P}' perpendicular to h_0 and passing through p' (see Fig. 3 for a 2-D example).*

We now determine an optimal segment in S . Let A'_u be the projection of R'_q onto A_u along the δ -axis, and $p^* = (a^*, b^*, \delta^*)$ be a lowest point on A'_u . Based on Lemmas 4 and 5, it is easy to see that $w'_{ijq} = \delta^*$ and $s(a^*, b^*)$ is a sought optimal segment. Thus, our goal is to find p^* .

Our algorithm uses the following main steps to compute p^* . (I) Compute the q -level A_q of the arrangement A for H_{ij} ; (II) compute the intersection of A_q and the plane h_ϵ , and determine R_q and R'_q ; (III) compute the upper envelope A_u of A (we store A_u by the DCEL data structure [20]); (IV) using A_u and R'_q , find a lowest point p^* on A'_u .

To analyze the running time, let $m = |P_{ij}|$ and Δ denote the size of A_q . In Step (I), we use the algorithm in [4] to compute A_q in $O(m \log \Delta + \Delta^{1+\alpha})$ time, where $\alpha > 0$ is an arbitrarily small constant. Step (II) takes $O(\Delta)$ time. Step (III) takes $O(m \log m)$ time using the (dual) 3-D convex hull algorithm [22]. For Step (IV), note that the size of A'_u is $O(|A_u| \cdot |R'_q|) = O(m\Delta)$. If computing A'_u explicitly, this step would take $O(m\Delta)$ time. Since $\Delta = O(mq^{\frac{3}{2}})$ [25], the running time of the algorithm would be dominated by Step (IV), which is $O(m\Delta)$.

Since A_u is given and is convex, we derive a faster algorithm for Step (IV) without producing A'_u explicitly in Lemma 6. As a consequence, due to $m \leq n$ and $q \leq g \leq n$, the time for computing w'_{ijq} becomes $W' = O((mq^{\frac{3}{2}})^{1+\alpha}) = O((ng^{\frac{3}{2}})^{1+\alpha})$, which is dominated by Step (I).

Lemma 6. *Given A_u and R'_q , we can find a lowest point p^* on A'_u in $O(m + \Delta \log m)$ time.*

In the rest of this section, we present our algorithm for Lemma 6.

The algorithm utilizes the convexity of A_u . For any object o in \mathcal{P}' , let $\Pi(o)$ denote the projection of o onto the plane h_0 along the δ -axis. Note that $R'_q = \Pi(R_q)$ is a set of polygonal regions (possibly unbounded, or containing holes) on h_0 . Let p_u be a lowest point on A_u . For any ray ρ originating from $\Pi(p_u)$ on the plane h_0 , let $S(\rho)$ be the set of all points on A'_u (A'_u is the projection of R'_q on A_u) such that $\Pi(S(\rho)) \subseteq \rho$. A key observation for the algorithm is as follows.

Observation 1. *For any ray ρ originating from $\Pi(p_u)$ on h_0 , if p_1 and p_2 are two arbitrary points on ρ such that p_1 is closer to $\Pi(p_u)$ than p_2 , then p'_1 is no higher than p'_2 , where p'_1 and p'_2 are the two points on A_u such that $\Pi(p'_1) = p_1$ and $\Pi(p'_2) = p_2$. Further, if p is the first point of R'_q hit by ρ , then p' is the lowest point in $S(\rho)$, where p' is the point on A_u such that $\Pi(p') = p$.*

Observation ② follows simply from the convexity of A_u and the fact that p_u is a lowest point on A_u . The next observation then follows immediately.

Observation 2. If $\Pi(p_u) \in R'_q$, $p_u \in A'_u$ and $p^* = p_u$; otherwise, $\Pi(p^*)$ lies on an edge of R'_q .

Our algorithm is based on Observation ②. We first check whether $\Pi(p_u) \in R'_q$, which can be done in $O(\Delta)$ time. If $\Pi(p_u) \in R'_q$, then we let $p^* = p_u$. Below, we discuss the case when $\Pi(p_u) \notin R'_q$.

For any point p on an edge e of R'_q , let $\rho(p)$ be the ray starting at p and going in the upward direction along the δ -axis. Denote by $\pi(p)$ the point on A_u first hit by $\rho(p)$, and let $\pi(e) = \{\pi(p) \mid p \in e\}$. We call $\pi(e)$ the *vertical projection* of e on A_u , which is a continuous curve on A_u . Denote by $\beta(e)$ a lowest point on $\pi(e)$. Thus, p^* is a lowest $\beta(e)$ point among all edges e of R'_q .

In the following Lemma ⑦ (proof omitted), by using the Dobkin-Kirkpatrick data structure [9], we show that with $O(m)$ time preprocessing we can compute $\beta(e)$ for each edge e of R'_q in $O(\log m)$ time. Consequently, since R'_q has $O(\Delta)$ edges, we find p^* in $O(m + \Delta \log m)$ time, and hence Lemma ⑥ follows.

Lemma 7. Given A_u (stored by the DCEL data structure), with $O(m)$ time preprocessing, for each edge e of R'_q , we can find $\beta(e)$ in $O(\log m)$ time.

Remark: Lemma ⑦ can be extended to the following 3-D segment dragging query problem, which may be interesting in its own right. Given a convex polyhedron G in 3-D, a query specifies a line segment e that does not intersect G and a direction $\sigma(e)$ perpendicular to the segment e , and reports the first point on G (if any) hit by e if e is translated (dragged) along the direction $\sigma(e)$. By an easy extension of Lemma ⑦, given G , after $O(|G|)$ time preprocessing, we can answer each 3-D segment dragging query in $O(\log |G|)$ time.

Based on Theorem ①, we have the following result (with $W' = O((ng^{\frac{3}{2}})^{1+\alpha})$, $R = O(ng^4 \log^2 n)$ [7], and the ordinary VWPF min- ϵ version solvable in $O(T) = O(ng(n \log g + g^3 n^\alpha) \cdot \min\{kg, \log n\})$ time [7], where k is the specified number of segments for the sought approximating functional curve).

Theorem 4. The min-# RVWPF (resp., RVPF) can be solved in $O(n^{2+\alpha} g^{\frac{7}{2}+\frac{3}{2}\alpha} \log \frac{n}{g})$ time, and its min- ϵ version can also be solved in the same time bound.

References

- Arning, A., Agrawal, R., Raghavan, P.: A linear method for deviation detection in large databases. In: Proc. of the Second International Conference on Knowledge Discovery and Data Mining, pp. 164–169 (1996)
- Barnett, V., Lewis, T.: Outliers in Statistical Data. John Wiley and Sons, New York (1994)
- Brodal, G., Jacob, R.: Dynamic planar convex hull. In: Proc. of the 43rd Symposium on Foundations of Computer Science (FOCS), pp. 617–626 (2002)
- Chan, T.: Output-sensitive results on convex hulls and extreme points, and related problems. Discrete & Computational Geometry 16(3), 369–387 (1996)

5. Chan, T.: Low-dimensional linear programming with violations. In: Proc. of 43rd Symposium on Foundations of Computer Science (FOCS), pp. 570–579 (2002)
6. Chen, D.Z., Wang, H.: Approximating Points by a Piecewise Linear Function: I. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 224–233. Springer, Heidelberg (2009)
7. Chen, D.Z., Wang, H.: Approximating Points by a Piecewise Linear Function: II. Dealing with Outliers. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 234–243. Springer, Heidelberg (2009)
8. Díaz-Báñez, J., Mesa, J.: Fitting rectilinear polygonal curves to a set of points in the plane. European Journal of Operational Research 130(1), 214–222 (2001)
9. Dobkin, D., Kirkpatrick, D.: A linear algorithm for determining the separation of convex polyhedra. Journal of Algorithms 6(3), 381–392 (1985)
10. Dyer, M.: Linear time algorithms for two- and three-variable linear programs. SIAM J. Comp. 13(1), 31–45 (1984)
11. Everett, H., Robert, J.M., van Kreveld, M.: An optimal algorithm for the ($\leq k$)-levels and with applications to separation and transversal problems. Int. J. Comput. Geometry Appl. 6(3), 247–261 (1996)
12. Fournier, H., Vigneron, A.: Fitting a Step Function to a Point Set. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 442–453. Springer, Heidelberg (2008)
13. Guha, S., Koudas, N., Shim, K.: Data streams and histograms. In: Proc. of the 33rd Annual Symposium on Theory of Computing (STOC), pp. 471–475 (2001)
14. Guha, S., Shim, K.: A note on linear time algorithms for maximum error histograms. IEEE Trans. Knowl. Data Eng. 19(7), 993–997 (2007)
15. Guha, S., Shim, K., Woo, J.: Rehist: Relative error histogram construction algorithms. In: Proc. of the 30th International Conference on Very Large Data Bases (VLDB), pp. 300–311 (2004)
16. Karras, P., Sacharidis, D., Mamoulis, N.: Exploiting duality in summarization with deterministic guarantees. In: Proc. of the 13th International Conference on Knowledge Discovery and Data Mining, pp. 380–389 (2007)
17. Knorr, E., Ng, R.: Algorithms for mining distance-based outliers in large datasets. In: Proc. of the 24th International Conference on Very Large Data Bases, pp. 382–403 (1998)
18. Matoušek, J.: On geometric optimization with few violated constraints. Discrete Comput. Geom. 14(1), 365–384 (1995)
19. Megiddo, N.: Linear programming in linear time when the dimension is fixed. Journal of the ACM 31(1), 114–127 (1984)
20. Muller, D., Preparata, F.: Finding the intersection of two convex polyhedra. Theor. Compu. Sci. 7, 217–236 (1978)
21. O'Rourke, J.: An on-line algorithm for fitting straight lines between data ranges. Commun. of ACM 24, 574–578 (1981)
22. Preparata, F., Hong, S.: Convex hulls of finite sets of points in two and three dimensions. Communications of the ACM 20(2), 87–93 (1977)
23. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. ACM SIGMOD Record 29(2), 427–438 (2000)
24. Roos, T., Widmayer, P.: k -violation linear programming. Information Processing Letters 52(2), 109–114 (1994)
25. Sharir, M., Smorodinsky, S., Tardos, G.: An improved bound for k -sets in three dimensions. Discrete & Computational Geometry 26, 195–204 (2001)

An Improved Algorithm for Reconstructing a Simple Polygon from the Visibility Angles*

Danny Z. Chen and Haitao Wang**

Department of Computer Science and Engineering,
University of Notre Dame, Notre Dame, IN 46556, USA
`{dchen, hwang6}@nd.edu`

Abstract. We study the problem of reconstructing a simple polygon: Given a cyclically ordered vertex sequence of an unknown simple polygon P of n vertices and, for each vertex v of P , the sequence of angles defined by all the visible vertices of v in P , reconstruct the polygon P (up to similarity). An $O(n^3 \log n)$ time algorithm has been proposed for this problem. We present an improved algorithm with running time $O(n^2)$, based on new observations on the geometric structures of the problem. Since the input size (i.e., the total number of input visibility angles) is $\Theta(n^2)$ in the worst case, our algorithm is worst-case optimal.

1 Introduction

In this paper, we study the problem of reconstructing a simple polygon P from the visibility angles measured at the vertices of P and from the cyclically ordered vertices of P along its boundary. Precisely, for an unknown simple polygon P of n vertices, suppose we are given (1) the vertices ordered counterclockwise (CCW) along the boundary of P , and (2) for each vertex v of P , the angles between any two adjacent rays emanating from v to the vertices of P that are visible to v such that these angles are in the CCW order as seen around v , beginning at the CCW neighboring vertex of v on the boundary of P (e.g., see Fig. 1). A vertex u of P is *visible* to a vertex v of P if the line segment connecting u and v lies in P . The objective of the problem is to reconstruct the simple polygon P (up to similarity) that fits all the given angles. We call this problem the *polygon reconstruction problem from angles*, or *PRA* for short. Figure 2 gives an example.

The PRA problem has been studied by Disser, Mihalák, and Widmayer [8] (and its preliminary version [7]), who showed that the solution polygon for the input is unique (up to similarity) and gave an $O(n^3 \log n)$ time algorithm for reconstructing such a polygon. Using the input, their algorithm first constructs the visibility graph G of P and subsequently reconstructs the polygon P . As shown by Disser et al. [8], once G is known, the polygon P can be obtained efficiently (e.g., in $O(n^2)$ time) with the help of the angle data.

* This research was supported in part by NSF under Grant CCF-0916606.

** Corresponding author.

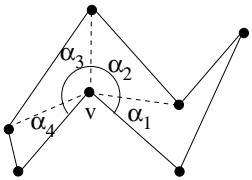


Fig. 1. The angle measurement at a vertex v : The angle sequence $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ is given as input

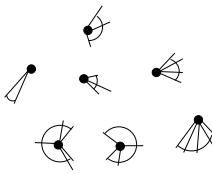
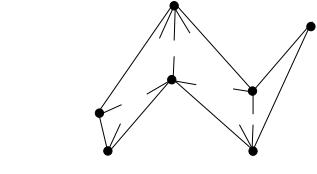


Fig. 2. The given measured angles are ordered CCW along the polygon boundary (left); reconstructing a simple polygon that fits these angles (right)



Given a visibility graph G , the problem of determining whether there is a polygon P that has G as its visibility graph is called the visibility graph *recognition* problem, and the problem of actually constructing such a polygon P is called the visibility graph *reconstruction* problem. Note that the general visibility graph recognition and reconstruction problems are long-standing open problems with only partial results known (e.g., Asano et al. provide a survey [1]). Everett [10] showed the visibility graph reconstruction problem is in PSPACE, but no better upper bound on the complexity of either problem is known (e.g., see [12]). In our problem setting, we have the angle data information and the ordered vertex list of P ; thus P can be constructed efficiently after knowing G .

Hence, the major part of the algorithm in [8] is dedicated to constructing the visibility graph G of P . As indicated in [8], the key difficulty is that the vertices in this problem setting have no recognizable labels, e.g., the angle measurement at a vertex v gives angles between visible vertices to v but does not identify these visible vertices globally. The authors in [8] also showed that some natural greedy approaches do not seem to work. An $O(n^3 \log n)$ time algorithm for constructing G is given in [8]. The algorithm, called the *triangle witness algorithm*, is based on the following observation: Suppose we wish to determine whether a vertex v_i is visible to another vertex v_j ; then v_i is visible to v_j if and only if there is a vertex v_l on the portion of the boundary of P from v_{i+1} to v_{j-1} in the CCW order such that v_l is visible to both v_i and v_j and the triangle formed by the three vertices v_i, v_j , and v_l does not intersect the boundary of P except at these three vertices (such a vertex v_l is called a *triangle witness vertex*).

In this paper, by exploiting some new geometric properties, we give an improved algorithm with running time $O(n^2)$. The improvement is due to two key observations. First, in the triangle witness algorithm [8], to determine whether a vertex v_i is visible to another vertex v_j , the algorithm needs to determine whether there exists a triangle witness vertex along the boundary of P from v_{i+1} to v_{j-1} in the CCW order; to this end, the algorithm checks every vertex in that boundary portion of P . We observe that it suffices to check only one particular vertex in that boundary portion. This removes an $O(n)$ factor from the running time of the triangle witness algorithm [8]. Second, some basic operations in the triangle witness algorithm [8] take $O(\log n)$ time each; by utilizing certain different data structures, our new algorithm can handle each of those

basic operations in constant time. This removes another $O(\log n)$ factor from the running time. Note that since the input size is $\Theta(n^2)$ in the worst case (e.g., the total number of all visibility angles), our algorithm is worst-case optimal.

Related Work

Reconstructing geometric objects based on measurement data has been studied (e.g., [24][14][16][17]). As discussed before, the general visibility graph recognition and reconstruction problems are difficult [10][12]. Yet, some results have been given for certain special polygons. For example, Everett and Corneil [11] characterized the visibility graphs of spiral polygons and gave a linear time reconstruction algorithm. Choi, Shin, and Chwa [5], and Colley, Lubiw, and Spinrad [6] characterized and recognized the visibility graphs of funnel-shaped polygons.

By adding extra information, some versions of the problems become more tractable. O'Rourke and Streinu [15] considered the *vertex-edge* visibility graph that includes edge-to-edge visibility information. Wismath [19] introduced the *stab graphs*. Snoeyink [17] proved that a unique simple polygon (up to similarity) can be determined by the interior angles at its vertices and the cross-ratios of the diagonals of any given triangulation. Jackson and Wismath [14] studied the reconstruction of orthogonal polygons from horizontal and vertical visibility information. Biedl, Durocher, and Snoeyink [2] considered the problem of reconstructing the two-dimensional floor plan of a polygonal room using different types of scanned data. Sidlesky, Barequet, and Gotsman [16] developed algorithms to reconstruct a planar polygon from its intersections with a collection of arbitrarily-oriented “cutting” lines. Given the number of visible edges and visible faces from some orthogonal projections, Biedl, Hasan and López-Ortiz [3] studied the problem of reconstructing convex polygons and polyhedra.

Reconstructing a simple polygon from angle data was first considered by Bilò *et al.* [4], who aimed to understand what kinds of sensorial capabilities are sufficient for a robot moving inside an unknown polygon to reconstruct the visibility graph of the polygon. It was shown in [4] that if the robot is equipped with a compass to measure the angle between any two vertices that are currently visible to the robot and also has the ability to know where it came from when moving from vertex to vertex, then the visibility graph of the polygon can be uniquely reconstructed. Reconstruction and exploration of environments by robots in other problem settings have also been studied (e.g., see [9][13][18]).

2 Preliminaries

In this section, we define the PRA problem in detail and introduce the needed notation and terminology, some of which follows those in [8].

Let P be a simple polygon of n vertices v_0, v_1, \dots, v_{n-1} in the CCW order along P 's boundary. Denote by $G = (V, E)$ the visibility graph of P , where V consists of all vertices of P and for any two distinct vertices v_i and v_j , E contains an edge $e(v_i, v_j)$ connecting v_i and v_j if and only if v_i is visible to v_j inside P . In this paper, the indices of all v_i 's are taken as congruent modulo n , i.e., if

$n \leq i + j \leq 2n - 1$, then v_{i+j} is the same vertex as v_l , where $l = i + j - n$ (or $l = (i + j) \bmod n$); similarly, if $-n \leq i - j < 0$, then v_{i-j} is the same vertex as v_l , where $l = i - j + n$. For each $v_i \in V$, denote by $\deg(v_i)$ its degree in the visibility graph G , and denote by $\text{vis}(v_i) = (\text{vis}_1(v_i), \text{vis}_2(v_i), \dots, \text{vis}_{\deg(v_i)}(v_i))$ the sequence of vertices in P visible to v_i from v_{i+1} to v_{i-1} ordered CCW around v_i . We refer to $\text{vis}(v_i)$ as v_i 's *visibility angle sequence*. Since both v_{i-1} and v_{i+1} are visible to v_i , $\text{vis}_1(v_i) = v_{i+1}$ and $\text{vis}_{\deg(v_i)}(v_i) = v_{i-1}$. For any two vertices v_i, v_j in V , let $\text{ch}(v_i, v_j)$ denote the sequence $(v_i, v_{i+1}, \dots, v_j)$ of the vertices ordered CCW along the boundary of P from v_i to v_j . We refer to $\text{ch}(v_i, v_j)$ as a *chain*. Let $|\text{ch}(v_i, v_j)|$ denote the number of vertices of P in the chain $\text{ch}(v_i, v_j)$.

For any two vertices $v_i, v_j \in P$, let $\rho(v_i, v_j)$ be the ray emanating from v_i and going towards v_j . For any three vertices $v, v_i, v_j \in P$, denote by $\angle_v(v_i, v_j)$ the CCW angle defined by rotating $\rho(v, v_i)$ around v to $\rho(v, v_j)$ (v_i or v_j need not be visible to v). Note that the values of all angles we use in this paper are in $[0, 2\pi]$. For any vertex $v \in P$ and $1 \leq i < j \leq \deg(v)$, let $\angle_v(i, j)$ be $\angle_v(\text{vis}_i(v), \text{vis}_j(v))$.

The PRA problem can then be re-stated as follows: Given a sequence of all vertices v_0, v_1, \dots, v_{n-1} of an unknown simple polygon P in the CCW order along P 's boundary, and the angles $\angle_v(i, i+1)$ for each vertex v of P with $1 \leq i < \deg(v)$, we seek to reconstruct P (up to similarity) to fit all the given angles. For ease of exposition, we assume that no three vertices of P are collinear.

For each $v \in V$, if we compute the angles $\angle_v(1, t)$ for all $t = 1, 2, \dots, \deg(v)$ in $O(n)$ time, then for any $1 \leq i < j \leq \deg(v)$, we can compute the angle $\angle_v(i, j) = \angle_v(1, j) - \angle_v(1, i)$ in constant time. Therefore, after $O(n^2)$ time preprocessing, for any $v \in V$ and any $1 \leq i < j \leq \deg(v)$, the angle $\angle_v(i, j)$ can be obtained in constant time. In the sequel, we assume that this preprocessing has already been done. Sometimes we (loosely) say that these angles are given as input.

The algorithm given in [8] does not construct P directly. Instead, the algorithm first computes its visibility graph $G = (V, E)$. As mentioned earlier, after knowing G , P can be reconstructed efficiently with the help of the angle data and the CCW ordered vertex sequence of P . The algorithm for constructing G in [8] is called the *triangle witness algorithm*, which will be briefly reviewed in Section 3. Since V consists of all vertices of P , the problem of constructing G is equivalent to constructing its edge set E , i.e., for any two distinct vertices $v_i, v_j \in P$, determine whether there is an edge $e(v_i, v_j)$ in E connecting v_i and v_j (in other words, determining whether v_i is visible to v_j inside P).

To discuss the details of the algorithms, we need one more definition. For any two vertices $v_i, v_j \in P$ with $|\text{ch}(v_i, v_j)| \geq 3$ (i.e., $|\text{ch}(v_{i+1}, v_{j-1})| \geq 1$), suppose a vertex $v_l \in \text{ch}(v_{i+1}, v_{j-1})$ is visible to both v_i and v_j ; then we let v'_j be the *first* visible vertex to v_i on the chain $\text{ch}(v_j, v_i)$ and let v'_i be the *last* visible vertex to v_j on the chain $\text{ch}(v_i, v_j)$ (e.g., see Fig. 3). Intuitively, imagine that we rotate a ray from $\rho(v_i, v_l)$ around v_i counterclockwise; then the first vertex on the chain $\text{ch}(v_j, v_i)$ hit by the rotating ray is v'_j . Similarly, if we rotate a ray from $\rho(v_j, v_l)$ around v_j clockwise, then the first vertex on the chain $\text{ch}(v_i, v_j)$ hit by the rotating ray is v'_i . Note that if v_i is visible to v_j , then v'_j is v_j and v'_i is v_i . We denote by $\angle_{v_i}^\uparrow(v_l, v_j)$ the angle $\angle_{v_i}(v_l, v'_j)$ and denote by $\angle_{v_j}^\uparrow(v_i, v_l)$ the

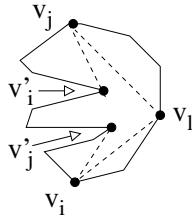


Fig. 3. Illustrating the definitions of v'_i and v'_j , and $\angle_{v_i}^\uparrow(v_l, v_j)$ and $\angle_{v_j}^\uparrow(v_i, v_l)$ which are the angles $\angle_{v_i}(v_l, v'_j)$ and $\angle_{v_j}(v'_i, v_l)$, respectively

angle $\angle_{v_j}(v'_i, v_l)$. It should be pointed out that for ease of understanding, the above statement of defining \angle^\uparrow is different from that in [8] but they refer to the same angles in the algorithm. The motivation for defining \angle^\uparrow will be clear after discussing the following lemma, which has been proved in [8].

Lemma 1. (Disser, Mihalák, and Widmayer [8]) *For any two vertices $v_i, v_j \in P$ with $|ch(v_i, v_j)| \geq 3$, v_i is visible to v_j if and only if there exists a vertex v_l on $ch(v_{i+1}, v_{j-1})$ such that v_l is visible to both v_i and v_j and $\angle_{v_i}^\uparrow(v_l, v_j) + \angle_{v_j}^\uparrow(v_i, v_l) + \angle_{v_l}(v_j, v_i) = \pi$.*

If v_i is visible to v_j , such a vertex v_l as described in Lemma 1 is called a *triangle witness* of the edge $e(v_i, v_j)$ in E .

3 The Triangle Witness Algorithm

In this section, we briefly review the triangle witness algorithm in [8] that constructs the visibility graph $G = (V, E)$ of the unknown simple polygon P . The discussion in this section is also helpful for understanding our improved algorithm given in Section 4.

The triangle witness algorithm is based on Lemma 1. The algorithm has $\lceil n/2 \rceil$ iterations. In the k -th iteration ($1 \leq k \leq \lceil n/2 \rceil$), the algorithm checks, for each $i = 0, 1, \dots, n - 1$, whether v_i is visible to v_{i+k} . After all iterations, the edge set E can be obtained. To this end, the algorithm maintains two maps F and B : $F[v_i][v_j] = t$ if v_j is identified as the t -th visible vertex to v_i in the CCW order, i.e., $v_j = vis_t(v_i)$; the definition of B is the same as F . During the algorithm, F will be filled in the CCW order and B will be filled in the clockwise (CW) order. When the algorithm finishes, for each v_i , $F[v_i]$ will have all visible vertices to v_i on the chain $ch(v_{i+1}, v_{\lceil n/2 \rceil})$ while $B[v_i]$ will have all visible vertices to v_i on the chain $ch(v_{\lceil n/2 \rceil}, v_{i-1})$. Thus, $F[v_i]$ and $B[v_i]$ together contain all visible vertices of P to v_i . For ease of description, we also treat $F[v_i]$ and $B[v_i]$ as sets, e.g., $v_l \in F[v_i]$ means that there is an entry $F[v_i][v_l]$ and $|F[v_i]|$ means the number of entries in the current $F[v_i]$.

Initially, when $k = 1$, since every vertex is visible to its two neighbors along the boundary of P , we have $F[v_i][v_{i+1}] = 1$ and $B[v_i][v_{i-1}] = deg(v_i)$ for each v_i .

In the k -th iteration, we determine for each v_i , whether v_i is visible to v_{i+k} . Below, we let $j = i+k$. Note that $|F[v_i]|+1$ is the index of the first visible vertex to v_i in the CCW order that is not yet identified; similarly, $(\deg(v_j) - |B[v_j]|)$ is the index of the first visible vertex to v_j in the CW order that is not yet identified. If v_i is visible to v_j , then we know that v_j is the $(|F[v_i]|+1)$ -th visible vertex to v_i and v_i is the $(\deg(v_j) - |B[v_j]|)$ -th visible vertex to v_j , and thus we set $F[v_i][v_j] = |F[v_i]|+1$ and $B[v_j][v_i] = \deg(v_j) - |B[v_j]|$. If v_i is not visible to v_j , then we do nothing.

It remains to discuss how to determine whether v_i is visible to v_j . According to Lemma 11, we need to determine whether there exists a triangle witness vertex v_l in the chain $ch(v_{i+1}, v_{j-1})$, i.e., v_l is visible to both v_i and v_j and $\angle_{v_i}^\uparrow(v_l, v_j) + \angle_{v_j}^\uparrow(v_i, v_l) + \angle_{v_l}(v_j, v_i) = \pi$. To this end, the algorithm checks every vertex v_l in $ch(v_{i+1}, v_{j-1})$. For each v_l , the algorithm first determines whether v_l is visible to both v_i and v_j , by checking whether there is an entry for v_l in $F[v_i]$ and in $B[v_j]$. The algorithm utilizes balanced binary search trees to represent F and B , and thus checking whether there is an entry for v_l in $F[v_i]$ and in $B[j]$ can be done in $O(\log n)$ time. If v_l is visible to both v_i and v_j , then the next step is to determine whether $\angle_{v_i}^\uparrow(v_l, v_j) + \angle_{v_j}^\uparrow(v_i, v_l) + \angle_{v_l}(v_j, v_i) = \pi$. It is easy to show that $\angle_{v_l}(v_j, v_i)$ is $\angle_{v_l}(F[v_l][v_j], B[v_l][v_i])$, which can be found readily from the input. To obtain $\angle_{v_i}^\uparrow(v_l, v_j)$, we claim that it is the angle $\angle_{v_i}(F[v_i][v_l], |F[v_i]|+1)$. Indeed, observe that all visible vertices in the chain $ch(v_{i+1}, v_{j-1})$ are in the current $F[v_i]$. As explained above, $|F[v_i]|+1$ is the index of the first visible vertex to v_i in the CCW order that has not yet been identified, which is the first visible vertex to v_i in the chain $ch(v_j, v_i)$, i.e., the vertex v'_j . Thus, $\angle_{v_i}(F[v_i][v_l], |F[v_i]|+1)$ is $\angle_{v_i}(v_l, v'_j)$, which is $\angle_{v_i}^\uparrow(v_l, v_j)$. Similarly, $\angle_{v_j}^\uparrow(v_i, v_l)$ is the angle $\angle_{v_j}(\deg(v_j) - |B[v_j]|, B[v_j][v_l])$. Recall that both angles $\angle_{v_i}(F[v_i][v_l], |F[v_i]|+1)$ and $\angle_{v_j}(\deg(v_j) - |B[v_j]|, B[v_j][v_l])$ can be obtained in constant time due to our preprocessing.

To analyze the running time of the above triangle witness algorithm, note that it has $\lceil n/2 \rceil$ iterations. In each iteration, the algorithm checks whether v_i is visible to v_{i+k} for each $0 \leq i \leq n-1$. For each v_i , the algorithm checks every v_l for $i+1 \leq l \leq i+k-1$, i.e., in the chain $ch(v_{i+1}, v_{i+k-1})$. For each such v_l , the algorithm takes $O(\log n)$ time as it uses balanced binary search trees to represent the two maps F and B . In summary, the overall running time of the triangle witness algorithm in [8] is $O(n^3 \log n)$.

4 An Improved Triangle Witness Algorithm

In this section, we present an improved solution over the triangle witness algorithm in [8] sketched in Section 3. Our improved algorithm runs in $O(n^2)$ time. Since the input size (e.g., the total number of visibility angles) is $\Theta(n^2)$ in the worst case, our improved algorithm is worst-case optimal. Our new algorithm follows the high-level scheme of the triangle witness algorithm in [8], and thus we call it the *improved triangle witness algorithm*.

As in [8], the new algorithm also has $\lceil n/2 \rceil$ iterations. In the k -th iteration ($1 \leq k \leq \lceil n/2 \rceil$), we determine whether v_i is visible to v_{i+k} for each $0 \leq i \leq n-1$.

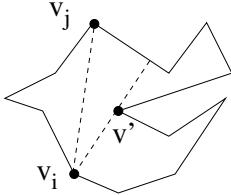


Fig. 4. Illustrating Lemma 2. v' is $\text{vis}_{|F[v_i]|}(v_i)$, the last visible vertex to v_i in $ch(v_{i+1}, v_{j-1})$; v_i is visible to v_j if and only if v' is a triangle witness vertex

For every pair of vertices v_i and v_{i+k} , let $j = i + k$. To determine whether v_i is visible to v_j , the triangle witness algorithm [8] checks each vertex v_l in the chain $ch(v_{i+1}, v_{j-1})$ to see whether there exists a triangle witness vertex. In our new algorithm, instead, we claim that we need to check only one particular vertex, $\text{vis}_{|F[v_i]|}(v_i)$, i.e., the last visible vertex to v_i in the chain $ch(v_{i+1}, v_{j-1})$ in the CCW order, as stated in the following lemma.

Lemma 2. *The vertex v_i is visible to v_j if and only if the vertex $\text{vis}_{|F[v_i]|}(v_i)$ is a triangle witness vertex for v_i and v_j .*

Proof: Let v' denote the vertex $\text{vis}_{|F[v_i]|}(v_i)$, which is the last visible vertex to v_i in the chain $ch(v_{i+1}, v_{j-1})$ in the CCW order. Recall that v' being a triangle witness vertex for v_i and v_j is equivalent to saying that v' is visible to both v_i and v_j and $\angle_{v_i}^\uparrow(v', v_j) + \angle_{v_j}^\uparrow(v_i, v') + \angle_{v'}(v_j, v_i) = \pi$.

If v_i is visible to v_j , then we prove below that v' is a triangle witness vertex for v_i and v_j , i.e., we prove that v' is visible to both v_i and v_j and $\angle_{v_i}^\uparrow(v', v_j) + \angle_{v_j}^\uparrow(v_i, v') + \angle_{v'}(v_j, v_i) = \pi$. Refer to Fig. 4 for an example. Let $P(v_i)$ denote the subpolygon of P that is visible to the vertex v_i . Usually, $P(v_i)$ is called the *visibility polygon* of v_i and it is well known that $P(v_i)$ is a star-shaped polygon with v_i as a kernel point (e.g., see [1]). Figure 5 illustrates $P(v_i)$. Since v' is the last visible vertex to v_i in $ch(v_{i+1}, v_{j-1})$ and v_j is visible to v_i , we claim that v_j is also visible to v' . Indeed, since both v' and v_j are visible to v_i , v' and v_j are both on the boundary of the visibility polygon $P(v_i)$. Let $P'(v_i)$ denote the portion of the boundary of $P(v_i)$ from v' to v_j counterclockwise (see Fig. 6). We prove below that $P'(v_i)$ does not contain any vertex of P except v' and v_j . First, $P'(v_i)$ cannot contain any vertex in the chain $ch(v_{j+1}, v_{i-1})$ (otherwise, v_j would not be visible to v_i). Let the index of the vertex v' be l , i.e., $v' = v_l$. Similarly, $P'(v_i)$ cannot contain any vertex in the chain $ch(v_{i+1}, v_{l-1})$ (otherwise, v' would not be visible to v_i). Finally, $P'(v_i)$ cannot contain any vertex in the chain $ch(v_{l+1}, v_{j-1})$, since otherwise v' would not be the last visible vertex to v_i in $ch(v_{i+1}, v_{j-1})$. Thus, $P'(v_i)$ does not contain any vertex of P except v' and v_j . Therefore, the region bounded by $P'(v_i)$, the line segment connecting v_i and v_j , and the line segment connecting v_i and v' must be convex (in fact, it is always a triangle) and this region is entirely contained in P . This implies that v_j is visible to v' . Hence, the three vertices v_i , v_j , and v' are mutually visible to each other, and we have $\angle_{v_i}^\uparrow(v_l, v_j) + \angle_{v_j}^\uparrow(v_i, v_l) + \angle_{v_l}(v_j, v_i) = \angle_{v_i}(v_l, v_j) + \angle_{v_j}(v_i, v_l) + \angle_{v_l}(v_j, v_i) = \pi$.

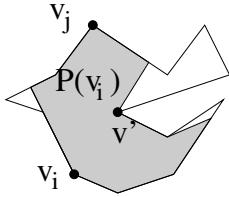


Fig. 5. Illustrating the visibility polygon $P(i)$ for the example in Fig. 4. The gray area is $P(v_i)$, which is a star-shaped polygon with the vertex v_i as a kernel point

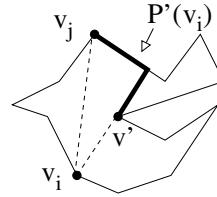


Fig. 6. Illustrating $P'(i)$ for the example in Fig. 4. The thick line segments $P'(i)$, which is a star-shaped polygon with the form $P'(v_i)$

On the other hand, if v' is a triangle witness vertex for v_i and v_j , then by Lemma 2, the vertex v_i is visible to v_j . The lemma thus follows. \square

By Lemma 2, to determine whether v_i is visible to v_j , instead of checking every vertex in $ch(v_{i+1}, v_{j-1})$, we need to consider only the vertex $vis_{|F[v_i]|}(v_i)$ in the current set $F[v_i]$. Hence, Lemma 2 immediately reduces the running time of the triangle witness algorithm by an $O(n)$ factor. The other $O(\log n)$ factor improvement is due to a new way of defining and representing the maps F and B , as elaborated below. In the following discussion, let v' be the vertex $vis_{|F[v_i]|}(v_i)$.

In our new algorithm, to determine whether v_i is visible to v_j , we check whether v' is a triangle witness vertex for v_i and v_j . To this end, we already know that v' is visible to v_i , but we still need to check whether v' is visible to v_j . In the previous triangle witness algorithm [8], this step is performed in $O(\log n)$ time by representing F and B using balanced binary search trees. In our new algorithm, we handle this step in $O(1)$ time, by redefining F and B and using a new way to represent them.

We redefine F as follows: $F[v_i][v_j] = t$ if v_j is the t -th visible vertex to v_i in the CCW order; if v_j is not visible to v_i or v_j has not yet been identified, then $F[v_i][v_j] = 0$. For convenience, we let $F[v_i] = F[v_i][v_{i+1}, v_{i+2}, \dots, v_{i-1}]$. Thus, in our new definition, the size of $F[v_i]$ is fixed throughout the algorithm, i.e., $|F[v_i]|$ is always $n - 1$. In addition, for each v_i , the new algorithm maintains two variables L_i and I_i for $F[v_i]$, where L_i is the number of non-zero entries in the current $F[v_i]$, which is also the number of visible vertices to v_i that have been identified (i.e., the number of visible vertices to v_i in the chain $ch(v_{i+1}, v_{i+k-1})$) up to the k -th iteration, and I_i is the index of the last non-zero entry in the current $F[v_i]$, i.e., I_i is the last visible vertex to v_i in the chain $ch(v_{i+1}, v_{i+k-1})$ in the CCW order. Similarly, we redefine B in the same way as F , i.e., for each v_i , $B[v_i] = B[v_i][v_{i+1}, v_{i+2}, \dots, v_{i-1}]$ and $B[v_i][v_j] = t$ if v_j is the t -th visible vertex to v_i in the CCW order. Further, for each v_i , we also maintain two variables L'_i and I'_i for $B[v_i]$, where L'_i is the number of non-zero entries in the current $B[v_i]$, which is also the number of visible vertices to v_i in the chain $ch(v_{i-k+1}, v_{i-1})$ (up to the k -th iteration), and I'_i is the first visible vertex to v_i in the chain $ch(v_{i-k+1}, v_{i-1})$ in the CCW order. During the algorithm, the array $F[v_i]$ will be filled in the CCW order, i.e., from the first entry $F[v_i][v_{i+1}]$ to the end while the array

$B[v_i]$ will be filled in the CW order, i.e., from the last entry $B[v_i][v_{i-1}]$ to the beginning. When the algorithm finishes, $F[v_i]$ will contain all the visible vertices to v_i in the chain $ch(v_{i+1}, v_{i+\lceil n/2 \rceil})$, and thus only the entries of the first half of $F[v_i]$ are possibly filled with non-zero values. Similarly, only the entries of the second half of $B[v_i]$ are possibly filled with non-zero values. Below, we discuss the implementation details of our new algorithm.

Initially, when $k = 1$, for each v_i , we set $F[v_i][v_{i+1}] = 1$ and $B[v_i][v_{i-1}] = deg(v_i)$, and set all other entries of $F[v_i]$ and $B[v_i]$ to zero. In addition, we set $L_i = 1$, $I_i = v_{i+1}$ and $L'_i = 1$, $I'_i = v_{i-1}$.

In the k -th iteration, with $1 < k \leq \lceil n/2 \rceil$, for each v_i , we check whether v_i is visible to v_j , with $j = i + k$, in the following way.

Denote by v' the last visible vertex to v_i in the chain $ch(v_{i+1}, v_{j-1})$ in the CCW order. By Lemma 2, we need to determine whether v' is a triangle witness vertex for v_i and v_j . Based on our definition of I_i , v' is the vertex I_i . After identifying v' , we then check whether v' is visible to v_j , which can be done by checking whether $B[v_j][v']$ (or $F[v'][v_j]$) is zero, in constant time. If $B[v_j][v']$ is zero, then v' is not visible to v_j and v' is not a triangle witness vertex; otherwise, v' is visible to v_j . In the following, we assume that v' is visible to v_j . The next step is to determine whether $\angle_{v_i}^\uparrow(v', v_j) + \angle_{v_j}^\uparrow(v_i, v') + \angle_{v'}(v_j, v_i) = \pi$. To this end, we must know the involved three angles. Similar to the discussion in Section 3, we have $\angle_{v'}(v_j, v_i) = \angle_{v'}(F[v'][v_j], B[v'][v_i])$, $\angle_{v_i}^\uparrow(v', v_j) = \angle_{v_i}(F[v_i][v'], L_i + 1)$, and $\angle_{v_j}^\uparrow(v_i, v') = \angle_{v_j}(deg(v_j) - L'_j, B[v_j][v'])$. Thus, all these three angles can be obtained in constant time. Hence, the step of checking whether v_i is visible to v_j can be performed in constant time, which reduces another $O(\log n)$ factor from the running time of the previous triangle witness algorithm in [8].

If v_i is not visible to v_j , then we do nothing. Otherwise, we set $F[v_i][v_j] = L_i + 1$ and increase L_i by one; similarly, we set $B[v_j][v_i] = deg(v_j) - L'_j$ and increase L'_j by one. Further, we set $I_i = v_j$ and $I'_j = v_i$.

Clearly, the running time of our new algorithm is bounded by $O(n^2)$.

Theorem 1. *Given the visibility angles and an ordered vertex sequence of a simple polygon P , the improved triangle witness algorithm can reconstruct P (up to similarity) in $O(n^2)$ time.*

As in [8], our algorithm can also be used to determine whether the input angle data are consistent. Namely, if no polygon can fit the input angle data, then the algorithm in Theorem 1 can be used to report this case as well.

References

1. Asano, T., Ghosh, S., Shermer, T.: Visibility in the plane. In: Sack, J., Urrutia, J. (eds.) *Handbook of Computational Geometry*, pp. 829–876. Elsevier, Amsterdam (2000)
2. Biedl, T., Durocher, S., Snoeyink, J.: Reconstructing polygons from scanner data. *Theoretical Computer Science* 412, 4161–4172 (2011)

3. Biedl, T., Hasan, M., López-Ortiz, A.: Reconstructing Convex Polygons and Polyhedra from Edge and Face Counts in Orthogonal Projections. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 400–411. Springer, Heidelberg (2007)
4. Bilò, D., Disser, Y., Mihalák, M., Suri, S., Vicari, E., Widmayer, P.: Reconstructing Visibility Graphs with Simple Robots. In: Kutten, S., Žerovnik, J. (eds.) SIROCCO 2009. LNCS, vol. 5869, pp. 87–99. Springer, Heidelberg (2010)
5. Choi, S.H., Shin, S., Chwa, K.Y.: Characterizing and recognizing the visibility graph of a funnel-shaped polygon. *Algorithmica* 14(1), 27–51 (1995)
6. Colley, P., Lubiwi, A., Spinrad, J.: Visibility graphs of towers. *Computational Geometry: Theory and Applications* 7(3), 161–172 (1997)
7. Disser, Y., Mihalák, M., Widmayer, P.: Reconstructing a Simple Polygon from its Angles. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 13–24. Springer, Heidelberg (2010)
8. Disser, Y., Mihalák, M., Widmayer, P.: A polygon is determined by its angles. *Computational Geometry: Theory and Applications* 44, 418–426 (2011)
9. Dudek, G., Jenkins, M., Milios, E., Wilkes, D.: Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation* 7(6), 859–865 (1991)
10. Everett, H.: Visibility Graph Recognition. Ph.D. thesis, University of Toronto, Toronto (1990)
11. Everett, H., Corneil, D.: Recognizing visibility graphs of spiral polygons. *Journal of Algorithms* 11(1), 1–26 (1990)
12. Everett, H., Corneil, D.: Negative results on characterizing visibility graphs. *Computational Geometry: Theory and Applications* 5(2), 51–63 (1995)
13. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 93–102. Springer, Heidelberg (1999)
14. Jackson, L., Wismath, S.: Orthogonal polygon reconstruction from stabbing information. *Computational Geometry: Theory and Applications* 23(1), 69–83 (2002)
15. O'Rourke, J., Streinu, I.: The vertex-edge visibility graph of a polygon. *Computational Geometry: Theory and Applications* 10(2), 105–120 (1998)
16. Sidlesky, A., Barequet, G., Gotsman, C.: Polygon reconstruction from line cross-sections. In: Proc of the 18th Canadian Conference on Computational Geometry, pp. 81–84 (2006)
17. Snoeyink, J.: Cross-ratios and angles determine a polygon. *Discrete and Computational Geometry* 22, 619–631 (1999)
18. Suri, S., Vicari, E., Widmayer, P.: Simple robots with minimal sensing: From local visibility to global geometry. *International Journal of Robotics Research* 27(9), 1055–1067 (2008)
19. Wismath, S.: Point and line segment reconstruction from visibility information. *International Journal of Computational Geometry and Applications* 10(2), 189–200 (2000)

The Parameterized Complexity of Local Search for TSP, More Refined

Jiong Guo^{1,*}, Sepp Hartung², Rolf Niedermeier², and Ondřej Suchý^{1,*}

¹ Universität des Saarlandes, Saarbrücken, Germany

{jguo, suchy}@mmci.uni-saarland.de

² Institut für Softwaretechnik und Theoretische Informatik, TU Berlin,
Berlin, Germany

{sepp.hartung, rolf.niedermeier}@tu-berlin.de

Abstract. We extend previous work on the parameterized complexity of local search for the Travelling Salesperson Problem (TSP). So far, its parameterized complexity has been investigated with respect to the distance measures (which define the local search area) “Edge Exchange” and “Max-Shift”. We perform studies with respect to the distance measures “Swap” and “ m -Swap”, “Reversal” and “ m -Reversal”, and “Edit”, achieving both fixed-parameter tractability and W[1]-hardness results. Moreover, we provide non-existence results for polynomial-size problem kernels and we show that some in general W[1]-hard problems turn fixed-parameter tractable when restricted to planar graphs.

1 Introduction

The Travelling Salesperson Problem (TSP) is probably the most studied combinatorial optimization problem. Almost all algorithm design techniques have been applied to it or were even specifically developed for it [12]. Many heuristic algorithms for TSP follow the paradigm of *local search*: Incrementally try to improve a solution by searching within its local neighborhood defined by a *distance measure*. Perhaps the most prominent and best examined distance measure for TSP is the *k-Edge Exchange* neighborhood (also called *k-Opt neighborhood* in some literature), where one is allowed to exchange at most k edges of the Hamiltonian cycle forming the tour. Implementations of this strategy for $k = 2, 3$ belong to the best performing heuristic algorithms for real-world instances [13]. However, for larger k , for which one would expect a strong increase of quality, the running time becomes infeasible since until now no algorithm is known which significantly beats the trivial $O(n^k)$ running time needed for a brute-force exploration of the local distance- k neighborhood. In an important step forward, considering the problem within the framework of *parameterized complexity* [6, 17], Marx [15] has shown that, by proving W[1]-hardness, there is no hope for an algorithm running in $f(k) \cdot n^c$ time for any function f (solely depending on k) and any

* Supported by the DFG Cluster of Excellence on Multimodal Computing and Interaction (MMCI) and DFG project DARE (GU 1023/1-2).

Table 1. Overview of our results using k as the parameter and assuming m to be a constant. The results written in italics are a direct consequence of a more general result.

	m -Swap	Swap	Edit	m -Reversal	Reversal	Edge
general graphs	FPT (Thm. ②)	W[1]-h (Thm. ⑩)	W[1]-h (Thm. ⑪)	FPT (Thm. ③)	W[1]-h (Thm. ⑪)	W[1]-h [15]
planar graphs	FPT (Thm. ⑤)	FPT (Thm. ⑤)	FPT (Thm. ⑤)	FPT (Thm. ⑤)	?	?

constant c . Note that such an algorithm is desirable since the degree of the polynomial in the input size does not depend on the parameter k . Moreover, assuming that the ETH (*exponential time hypothesis*) [5, 10] does not fail, Marx has shown that there is no algorithm running in $O(n^{o(\sqrt[3]{k})})$ time.

In this work, besides the k -Edge Exchange neighborhood (briefly, *Edge* distance measure), we consider various other distance measures such as the Reversal distance (which is also widely studied in bioinformatics in the context of genome rearrangements [4]), the Swap distance where one is allowed to exchange two vertices, and the Edit distance where one can move a vertex to an arbitrary new position. For λ being any of these distance measures, we study the following problem.

LOCALTSP(λ)

Input: An undirected graph $G = (V, E)$ with vertices labeled v_1, \dots, v_n such that the identical permutation (id) $v_1, v_2, \dots, v_n, v_1$ is a Hamiltonian cycle in G , an edge weight function $\omega : E \rightarrow \mathbb{R}_0^+$, and a positive integer k .

Question: Is there a permutation π with $\lambda(\pi, \text{id}) \leq k$ that yields a Hamiltonian cycle with $\omega(\pi) < \omega(\text{id})$, where $\omega(\pi) = \sum_{i=1}^{n-1} \omega(\{v_{\pi(i)}, v_{\pi(i+1)}\}) + \omega(\{v_{\pi(n)}, v_{\pi(1)}\})$?

Our Results. Table ⑩ summarizes our results. For $\lambda \in \{\text{Swap}, \text{Edit}, \text{Reversal}, \text{Edge}\}$, we show that the result of Marx [15] for LOCALTSP(Edge) can be extended and even strengthened, that is, we show that LOCALTSP(λ) is W[1]-hard, implying that it is probably not fixed-parameter tractable for the “locality parameter” k . Furthermore, again assuming that the ETH holds, there cannot be an algorithm with running time $O(n^{o(\sqrt{k})})$. In addition, exploring the limitations of polynomial-time preprocessing, we indicate that, unless $\text{NP} \subseteq \text{coNP/poly}$, there is no polynomial-size problem kernel for LOCALTSP(λ) for any of the considered distance measures λ .

On the positive side, for the Swap distance we show that, restricting by a parameter m the distance of two vertices that are allowed to swap, makes LOCALTSP(m -Swap) fixed-parameter tractable with respect to the combined parameter (k, m) . Furthermore, we show that an analogously restricted Reversal distance, called m -Reversal, again leads to fixed-parameter tractability. Continuing to chart the border of tractability, we show that LOCALTSP(λ) for $\lambda \in \{\text{Swap}, \text{Edit}\}$ is fixed-parameter tractable on planar graphs. Due to space limitations most details are deferred to a full version of the paper.

Related Work. The most important reference point is Marx' study of LOCALTSP(Edge) [15] (using different notation). In addition, long before Marx, Balas [1] studied LOCALTSP(Max-Shift), where Max-Shift distance k means that in order to obtain an improved Hamiltonian cycle the maximum number of positions that a vertex is allowed to shift is k . Contrasting the parameterized hardness result of Marx [15], Balas showed that LOCALTSP(Max-Shift) is fixed-parameter tractable by providing an algorithm running in $O(4^{k-1} \cdot k^{1.5} \cdot |V|)$ time.

2 Basic Notation and Distance Measures

Notation. Let \mathcal{S}_n denote the set of all bijective mappings of the set $\{1, \dots, n\}$ to itself and let $\text{id} \in \mathcal{S}_n$ be the identity. A Hamiltonian cycle through a graph $G = (V, E)$ with vertices labeled v_1, v_2, \dots, v_n is expressed by a permutation $\pi \in \mathcal{S}_n$ such that the edge set $E(\pi)$ of π , defined as $E(\pi) = \{\{v_{\pi(i)}, v_{\pi(i+1)}\} \mid 1 \leq i < n\} \cup \{\{v_{\pi(n)}, v_{\pi(1)}\}\}$, is a subset of E . For a weight function $\omega : E \rightarrow \mathbb{R}_0^+$ we define the weight of π by $\omega(\pi) = \sum_{e \in E(\pi)} \omega(e)$. The Hamiltonian cycle π is called *improved* compared to id when $\omega(\pi) < \omega(\text{id})$. In this sense, $\text{LOCALTSP}(\lambda)$ is the question whether there is an improved Hamiltonian cycle π with $\lambda(\pi, \text{id}) \leq k$.

A parameterized problem is said to be *fixed-parameter tractable* if there is an algorithm that solves every instance (I, k) (where k is the parameter) within $f(k) \cdot |I|^c$ time for a constant c and a function f which solely depends on k [7, 17]. A *kernelization algorithm* computes for a given instance (I, k) in polynomial time a new instance (I', k') (called kernel) such that (I', k') is a yes-instance iff (I, k) is a yes-instance, $k' \leq g(k)$, and $|I'| \leq g(k)$ for a function g which solely depends on k [2, 11]. The function g measures the size of the kernel.

Permissive Algorithms and Distance Measures. So far, the distance between Hamiltonian cycles was usually measured in terms of *Edge* distance, counting the number of edges used by one cycle but not used by the other. Another measure considered is the *Max-Shift* distance, which equals the maximum shift of the position of a vertex between the two permutations. We consider several further measures based on the following operations on permutations.

Definition 1. For a permutation $1, 2, \dots, n$, we define the following operations:

- reversal $\rho(i, j)$ results in $1, \dots, i - 1, j, j - 1, \dots, i + 1, i, j + 1, \dots, n$;
- swap $\sigma(i, j)$ results in $1, \dots, i - 1, j, i + 1, \dots, j - 1, i, j + 1, \dots, n$;
- edit $\epsilon(i, j)$ results in $1, \dots, i - 1, i + 1, \dots, j - 1, j, i, j + 1, \dots, n$.

We do not consider the elements 1 and n to be anyhow special and, therefore, the operations can also be applied “over them”. For a constant m and $0 < j - i \leq m - 1$ or $n + j - i \leq m - 1$ we speak about *m-swaps* and *m-reversals*.

The distance measures *Swap*, *m-Swap*, *Edit*, *Reversal*, and *m-Reversal* count the minimum number of the appropriate operations to apply to one permutation in order to obtain the other.

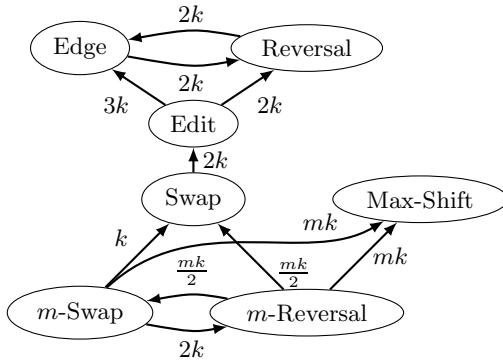


Fig. 1. Hasse diagram of the relations between the distance measures. Let $f : \mathbb{N} \rightarrow \mathbb{N}$. An arrow from a distance measure λ to a measure τ labeled “ $f(k)$ ” means that τ is λ -bounded with function $f(k)$.

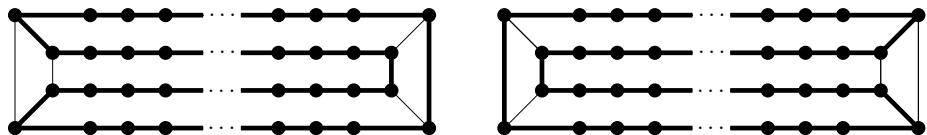


Fig. 2. A planar graph with two different Hamiltonian cycles. The cycles are only four edge modifications and four reversals from each other, while they can be made arbitrary far apart for any other of the measures by extending the horizontal lines. Furthermore, the differences between the cycles are also very far apart considering the distance in the graph and, as the graph has no other Hamiltonian cycles, there is no other solution with changes concentrated in a constant distance to one particular vertex.

Definition 2. A distance measure λ is bounded by a distance measure τ (or τ -bounded) if there is a function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that for any two permutations $\pi, \pi' \in \mathcal{S}_n$ it holds that $\lambda(\pi, \pi') \leq f(\tau(\pi, \pi'))$.

It is easy to see that the relation of boundedness is reflexive and transitive and, therefore, forms a quasi-order on the distance measures. Figure 1 depicts all relations between the measures, omitting relations that can be deduced from the transitivity, in this sense showing a “Hasse diagram” of this quasi-order. The shown relations are easy to check. It is also not hard to come up with examples showing that no further boundedness relations hold between the distance measures. See Figure 2 for an interesting case of two Hamiltonian cycles which are close for Reversal and Edge distances, but far apart for all the other distances considered.

Marx and Schlotter [16] proposed to distinguish between *strict* and *permissive* local search algorithms. Strict local search algorithms find an improved solution (or prove that it does not exist) within some limited distance from the given solution. Permissive local search algorithms find *any* improved solution (potentially, with unbounded distance to the given solution), provided that an improved solution exists within the limited distance of the given solution.

The motivation for this distinction is that finding an improved solution within a bounded distance of a given solution may be hard even for problems where an optimal solution can easily be found, e.g., MINIMUM VERTEX COVER on bipartite graphs [14]. The following lemma indicates a tight relationship between our notion of bounded distance measures and the existence of permissive FPT-algorithms.

Lemma 1. *If a distance measure λ is τ -bounded, then a (permissive) FPT-algorithm for LOCALTSP(λ) is a permissive FPT-algorithm for LOCALTSP(τ).*

3 General Graphs

We first show that LOCALTSP(λ) is W[1]-hard for $\lambda \in \{\text{Swap, Edit, Reversal, Edge}\}$. To this end, we build on the proof given by Marx [15]. In contrast to Marx, who gave a reduction from the k -CLIQUE problem, we reduce from the k -MULTICOLORED CLIQUE problem. This simplifies the construction and makes it even more powerful. Specifically, we show that, if there exists an improved Hamiltonian cycle, then there is also one that can be obtained by $O(k^2)$ swaps. Then, since the other measures are Swap-bounded and, unlike in Marx' construction, *any* improved Hamiltonian cycle in the constructed graph implies the existence of a k -multicolored clique, the hardness result holds true even for them.

Theorem 1. *LOCALTSP(λ) is W[1]-hard with respect to k for $\lambda \in \{\text{Swap, Edit, Reversal, Edge}\}$.*

It was shown that k -CLIQUE cannot be solved within $O(n^{o(k)})$ time unless the ETH fails [5, 10]. It follows from the reduction given by Fellows et al. [8] that if k -MULTICOLORED CLIQUE would be solvable in $O(n^{o(k)})$ time, then k -CLIQUE would also be solvable in $O(n^{o(k)})$ time. In the parameterized reduction given in the proof of **Theorem 1** reducing from k -MULTICOLORED CLIQUE to LOCALTSP(λ) for $\lambda \in \{\text{Swap, Edit, Reversal, Edge}\}$ the parameter for LOCALTSP is within the square of the parameter of k -MULTICOLORED CLIQUE. This implies the following corollary. For the case of Edge distance, it improves the lower bound $O(n^{o(\sqrt[3]{k})})$ given by Marx [15, Corollary 3.5].

Corollary 1. *Unless the ETH fails, LOCALTSP(λ) does not admit an algorithm with running time $O(n^{o(\sqrt{k})})$ for $\lambda \in \{\text{Swap, Edit, Reversal, Edge}\}$.*

We have shown that on general graphs there is no hope to obtain an FPT-algorithm for LOCALTSP(Swap) parameterized by k . However, restricting the distance measure to m -swaps makes the problem fixed-parameter tractable. As the exponent of the polynomial in the running time is also independent of m , we state the results with respect to the combined parameter (k, m) . The corresponding algorithm is based on the bounded search tree technique and it is mainly based on the observation that the solution can be assumed to be given by a sequence of swaps that are somehow related. For a formal description we need the following definition.

Let S be a sequence of swaps. We define an auxiliary *swap graph* G^S as follows. There is a vertex for each swap in the sequence S and two swaps $\sigma(i, j)$ and $\sigma(t, l)$ are adjacent if either t or l is contained in $\{i-1, i, i+1, j-1, j, j+1\}$. Furthermore, if a swap $\sigma(i, j)$ is applied, we call the positions i and j and the vertices at these positions *affected*.

Lemma 2. *If a LOCALTSP(λ) instance for $\lambda \in \{m\text{-Swap, Swap}\}$ admits an improved Hamiltonian cycle, it also admits an improved Hamiltonian cycle which can be obtained by swaps (or m -swaps) such that their swap graph is connected.*

Proof. Suppose that we are given a sequence S of swaps whose application to a Hamiltonian cycle $\text{id} \in \mathcal{S}_n$ creates an improved Hamiltonian cycle $\pi \in \mathcal{S}_n$. Furthermore, assume that C_1, \dots, C_p with $p \geq 2$ are the connected components of the corresponding swap graph G^S . For any of these components C , we denote by $\pi^C \in \mathcal{S}_n$ the permutation that results from applying the swaps in C to id preserving their order relative to S .

We shall show that the sets $E(\pi^{C_1}) \Delta E(\text{id}), \dots, E(\pi^{C_p}) \Delta E(\text{id})$ form a partition of the set $E(\pi) \Delta E(\text{id})$ (Δ denotes the symmetric difference). Having proved this, the rest of the argumentation is as follows. Since $\omega(\pi) < \omega(\text{id})$ or equivalently $\omega(E(\pi) \setminus E(\text{id})) < \omega(E(\text{id}) \setminus E(\pi))$, it follows that there is at least one component C of G^S with $\omega(E(\pi^C) \setminus E(\text{id})) < \omega(E(\text{id}) \setminus E(\pi^C))$. This implies that $\omega(\pi^C) < \omega(\text{id})$ and thus applying only swaps contained in C also results in an improved Hamiltonian cycle π^C .

It remains to prove that $E(\pi^{C_1}) \Delta E(\text{id}), \dots, E(\pi^{C_p}) \Delta E(\text{id})$ is a partition of $E(\pi) \Delta E(\text{id})$. First, for all $1 \leq i < j \leq p$ by definition of the swap graph, it follows that the positions, and thus also the vertices, affected by C_i are disjoint from the positions and vertices that are affected by C_j . Formally, $E(\pi^{C_i}) \Delta E(\text{id}) \cap E(\pi^{C_j}) \Delta E(\text{id}) = \emptyset$. For any component C , we next argue that $E(\pi^C) \Delta E(\text{id}) \subseteq E(\pi) \Delta E(\text{id})$. Clearly, for an edge $e = \{i, j\} \in E(\pi^C) \Delta E(\text{id})$, either vertex i or j has to be affected by at least one swap in C . Then, no swap in $S \setminus C$ also affects either i or j , because such a swap would be adjacent to at least one swap in C . Hence, $e \in E(\pi) \Delta E(\text{id})$. Finally, consider an edge $e = \{i, j\} \in E(\pi) \Delta E(\text{id})$. By the same argument as above, all swaps that affect either the vertex i or j belong to the same component of G^S . Thus, since either vertex i or j is affected by a swap, it follows that there is a component C of G^S such that $e \in E(\pi^C) \Delta E(\text{id})$. \square

Theorem 2. *LOCALTSP(m -Swap) is fixed-parameter tractable with respect to the combined parameter (k, m) . It is solvable in $O(m^{2k}(m-1)^{2k} \cdot 4^k \cdot (k^2 + n) \cdot n)$ time.*

Proof. Let (G, ω, k) be an instance of LOCALTSP(m -Swap). Furthermore, let S be a sequence of at most k m -swaps such that applying S to id results in an improved Hamiltonian cycle π . By Lemma 2 we can assume that G^S is connected. The algorithm consists of two parts. First, the algorithm guesses the positions of all swaps in S and, secondly, it finds the correct order of them.

To describe the first part, for convenience, we assume for all swaps $\sigma(i, j)$ that $j \in \{i+1, i+2, \dots, i+m-1\}$. Furthermore, we define an ordering relation \leq

on swaps with $\sigma(i, j) \leq \sigma(t, p)$ iff $i < t$ or $i = t \wedge j \leq p$. Let $\sigma_1, \sigma_2, \dots, \sigma_s$ with $s \leq k$ be the swaps of S sorted with respect to \leq in ascending order. In the first part of the algorithm, by branching into all possibilities for the positions of the swaps, the algorithm guesses all swaps in the order given above. At the beginning, the algorithm branches into all possibilities to find the position i_1 for $\sigma_1(i_1, j_1)$ and then into the $m - 1$ possibilities to find the position j_1 . Now, suppose we have already found the swap $\sigma_t(i_t, j_t)$, we next describe how to find the swap $\sigma_{t+1}(i_{t+1}, j_{t+1})$. By the ordering we know that $i_1 \leq \dots \leq i_t \leq i_{t+1}$ and, since all swaps are m -swaps, for all $1 \leq p \leq t$ with $j_p > i_t$ it holds that $j_p - i_t \leq m - 1$. From this and since G^S is connected (Lemma 2), it follows that $i_t - i_{t+1} \leq m$. Thus, we can guess the position of i_{t+1} by branching into $m + 1$ possibilities. Afterwards, by branching into $m - 1$ possibilities we find the position j_{t+1} . Overall, we can guess the positions of σ_{t+1} by branching into at most m^2 possibilities and thus the positions of all swaps can be guessed in $O(m^{2k-1} \cdot n)$ time.

In the second part, the algorithm guesses the order of the m -swaps. Clearly, the trivial way to do that is by trying all permutations of the swaps, resulting in a total running time of $O(m^{2k-1}k! \cdot n)$. This already shows that the problem is fixed-parameter tractable for (k, m) . We next describe how this can be accelerated in case that $4m^2 < k$. To this end, let $\sigma^{(1)}, \sigma^{(2)}, \dots, \sigma^{(s)}$ be all swaps in S in the order of their application resulting in π . Clearly, if there are two subsequent swaps $\sigma^{(t)}(i, j)$ and $\sigma^{(t+1)}(i', j')$ such that $\{i, j\} \cap \{i', j'\} = \emptyset$, then reversing their order in the application of the swaps also results in π . More generally, instead of finding a total order of the swaps, it is sufficient to find a partial order of the swaps that defines the order for any pair of swaps $\sigma(i, j)$ and $\sigma(t, p)$ where $|\{i, j\} \cap \{t, p\}| = 1$. Clearly, we do not have to define the order of two swaps which are of the same *type*, that is, where $\{i, j\} = \{t, p\}$. Thus, for a position i , consider all swaps which affect position i . Since all these swaps are m -swaps, there can be at most $2m - 2$ different types that affect position i . Hence, if there are k_i swaps that affect position i , then there are at most $(2m - 2)^{k_i}$ different permutations of these swaps. Combining the number of possibilities of all affected positions, since each swap affects exactly two positions, it follows that there are at most $(2m - 2)^{2k}$ permutations of all swaps yielding different Hamiltonian cycles. Once the partial orders at all relevant positions are determined, we check whether this can be obtained by some total order of the swaps, and find this order in $O(k^2)$ time, by representing the partial orders by some arcs in a directed graph on the set of swaps and finding a topological order for that graph. Then we apply the swaps in this order in $O(k)$ time and check whether we obtain an improved Hamiltonian cycle in linear time. Together with the first part, the whole algorithm runs in $O(m^{2k}(m - 1)^{2k} \cdot 4^k \cdot (k^2 + n) \cdot n)$ time. \square

Since the m -Swap distance is bounded by the m -Reversal distance, the above theorem implies also the existence of an $O(m^{mk}(m - 1)^{mk} \cdot 2^{mk} \cdot ((mk)^2 + n) \cdot n)$ -time permissive algorithm for LOCALTSP(m -Reversal), i.e., an algorithm that returns an improved Hamiltonian cycle whenever there is an improved Hamiltonian cycle in m -Reversal distance at most k from the given cycle. By modifying

the algorithm from [Theorem 2](#), we can obtain a strict local search algorithm for LOCALTSP(m -Reversal) with a better running time.

Theorem 3. LOCALTSP(m -Reversal) is fixed-parameter tractable with respect to the combined parameter (k, m) . It is solvable in $O(2^{mk} \cdot m^{2k-1} \cdot (m-1)^k \cdot (k^2 + n) \cdot n)$ time.

We remark that, for $\text{LOCAUTSP}(\lambda)$ with $\lambda \in \{m\text{-Swap}, m\text{-Reversal}\}$, by applying a standard dynamic programming approach, the algorithms given in the proofs of Theorems [2](#) and [3](#) can be extended such that not only *any* improved Hamiltonian cycle is found but also the *best* improved Hamiltonian cycle within the local neighborhood.

Further, analyzing the proofs of Theorems [2](#) and [3](#), one can show that if there is an improved Hamiltonian cycle in $\text{LOCAUTSP}(m\text{-Swap})$ or $\text{LOCAUTSP}(m\text{-Reversal})$, then there is also an improved cycle which differs from the given one only on vertices $v_i, v_{i+1}, \dots, v_{i+mk}$ for some i . Therefore, one can reduce an input instance to polynomially many instances of the same problem, each having its size bounded by a polynomial in k and m . Such a self-reduction is known as polynomial *Turing kernelization*. In contrast to this, in the next section we show that, for any of the distance measures λ considered in this work, $\text{LOCAUTSP}(\lambda)$ does not admit a polynomial kernel even when restricted to planar graphs.

4 Planar Graphs

In this section we investigate the complexity of LOCALTSP on planar graphs. Note that whether $\text{LOCAUTSP}(\text{Edge})$ on planar graphs parameterized by the locality parameter k is fixed-parameter tractable or not is the central open question stated by Marx [\[15\]](#) and it is also mentioned by Fellows et al. [\[9\]](#). We do not answer this question; however, we show that on planar graphs $\text{LOCAUTSP}(\lambda)$ for $\lambda \in \{\text{Swap}, \text{Edit}\}$ is fixed-parameter tractable for parameter k . Before that, we show that $\text{LOCAUTSP}(\lambda)$ on planar graphs does not admit a polynomial kernel for all distance measures λ considered in this work.

Bodlaender et al. [\[3\]](#) have shown that a parameterized problem does not admit a polynomial-size kernel if its unparameterized variant is NP-hard and if it is compositional. A parameterized problem is compositional if there is a polynomial time algorithm that takes as input instances $(I_1, k), \dots, (I_t, k)$ and computes a new instance (I, k') where k' is upper-bounded by a polynomial in k and (I, k') is a yes-instance iff (I_j, k) is a yes-instance for some $1 \leq j \leq t$.

To show that $\text{LOCAUTSP}(m\text{-Swap})$ on planar graphs has no polynomial kernel, we first consider a more restricted variant, namely $\text{LARGELOCAUTSP}(m\text{-Swap})$, where it is required that the underlying planar graph has more than $2mk$ vertices. We show that $\text{LARGELOCAUTSP}(m\text{-Swap})$ is NP-hard on planar graphs by a many-to-one reduction from WEIGHTED ANTIMONOTONE 2-SAT; by exploiting the properties implied by the requirement that there are more than $2mk$ vertices, we then show that it is compositional. Together with the NP-hardness, this implies that $\text{LARGELOCAUTSP}(m\text{-Swap})$ does not admit a polynomial kernel, unless $\text{NP} \subseteq \text{coNP/poly}$. Thus, the next theorem follows.

Theorem 4. Unless $NP \subseteq coNP/poly$, LOCALTSP(m -Swap) on planar graphs does not admit a polynomial kernel with respect to the parameter k for any $m \geq 2$.

Figure 1 depicts the relation between the different distance measures. Since there is a directed path from the m -Swap measure to every other measure, the m -Swap measure can be considered as the least powerful measure since all other measures are bounded by it. We thus claim here that by basically the same argumentation as for LOCALTSP(m -Swap) one can show that on planar graphs LocalTSP(λ) for $\lambda \in \{m\text{-Reversal, Swap, Edit, Reversal, Max-Shift, Edge}\}$ does not admit a polynomial kernel with respect to parameter k , unless $NP \subseteq coNP/poly$.

LOCALTSP(Edit) and LOCALTSP(Swap) on planar graphs probably do not allow for polynomial kernels; however they admit a permissive FPT-algorithm. In the following we sketch the argumentation for LOCALTSP(Swap); the result for the Edit distance can be obtained along the same lines, but it is more technical. The proof relies on the following two lemmas.

Lemma 3. If a LOCALTSP(Swap) instance with parameter k admits an improved Hamiltonian cycle, then it also admits an improved Hamiltonian cycle which differs from the given one only within the distance- $3k$ neighborhood around some vertex.

The following lemma shows that, regardless of the distance measure, on planar graphs it is fixed-parameter tractable to find the best improved Hamiltonian cycle that differs from the given one only within the neighborhood of one specific vertex.

Lemma 4. For an instance of LOCALTSP on planar graphs and a vertex v one can find in $O(2^{O(k)} \cdot n + n^3)$ time the best Hamiltonian cycle among those differing from the given one only within distance k from v .

Theorem 5. There is a permissive FPT-algorithm for LOCALTSP(λ) on planar graphs with respect to k for $\lambda \in \{\text{Swap, Edit}\}$.

Following the same approach as Fellows et al. [9], Theorem 5 can be easily generalized to any class of graphs with bounded local treewidth. As Lemma 3 does not assume anything about the graph, we only have to modify Lemma 4. The lemma is true in any class of graphs with bounded local treewidth, but the corresponding running time depends on the respective class.

5 Conclusion

We left open the central open problem due to Marx [15] whether LOCALTSP(Edge) restricted to planar graphs is fixed-parameter tractable. However, we indicated (see Section 2) that a permissive FPT-algorithm for LOCALTSP(Edge) implies a permissive FPT-algorithm for LOCALTSP(Reversal) and vice versa. Thus, the question whether the problems are fixed-parameter tractable or not, are equivalent and this might help to shed new light on this question. To this

end, it might be beneficial to explore the connections of LOCALTSP(Reversal) to the topic of SORTING BY REVERSALS as studied in bioinformatics [4].

In addition, assuming the Exponential Time Hypothesis [5, 10], we showed that there is no $O(n^{o(\sqrt{k})})$ -time algorithm for LOCALTSP(λ) for $\lambda \in \{\text{Swap}, \text{Edit}, \text{Reversal}, \text{Edge}\}$. Is there also a matching upper bound or can the lower bound still be improved?

Finally, our investigations might also be extended by moving from local neighborhoods for TSP to so-called exponential (but structured) neighborhoods as undertaken already in a non-parameterized setting [6].

References

- [1] Balas, E.: New classes of efficiently solvable generalized traveling salesman problems. *Ann. Oper. Res.* 86, 529–558 (1999)
- [2] Bodlaender, H.L.: Kernelization: New Upper and Lower Bound Techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
- [3] Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On Problems Without Polynomial Kernels. *J. Comput. System Sci.* 75(8), 423–434 (2009)
- [4] Caprara, A.: Sorting by reversals is difficult. In: Proc. 1st RECOMB, pp. 75–83. ACM Press (1997)
- [5] Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D.W., Kanj, I.A., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. *Inform. and Comput.* 201(2), 216–231 (2005)
- [6] Deineko, V.G., Woeginger, G.J.: A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Math. Program., Ser. A* 87(3), 519–542 (2000)
- [7] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
- [8] Fellows, M.R., Hermelin, D., Rosamond, F.A., Vialette, S.: On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.* 410(1), 53–61 (2009)
- [9] Fellows, M.R., Rosamond, F.A., Fomin, F.V., Lokshtanov, D., Saurabh, S., Villanger, Y.: Local Search: Is Brute-Force Avoidable? In: Proc. 21th IJCAI, pp. 486–491 (2009)
- [10] Flum, J., Grohe, M.: Parameterized complexity and subexponential time. *Bulletin of the EATCS* 84, 71–100 (2004)
- [11] Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. *SIGACT News* 38(1), 31–45 (2007)
- [12] Gutin, G., Punnen, A.: The Traveling Salesman Problem and its Variations. Combinatorial Optimization (2002)
- [13] Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: The Traveling Salesman Problem and its Variations, pp. 369–443 (2004)
- [14] Krokhin, A., Marx, D.: On the hardness of losing weight. *ACM Trans. Algorithms* (to appear, 2011) (online available)
- [15] Marx, D.: Searching the k -change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.* 36(1), 31–36 (2008)
- [16] Marx, D., Schlotter, I.: Stable assignment with couples: Parameterized complexity and local search. *Discrete Optim.* 8(1), 25–40 (2011)
- [17] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)

On the Parameterized Complexity of Consensus Clustering^{*}

Martin Dörnfelder¹, Jiong Guo¹, Christian Komusiewicz², and Mathias Weller²

¹ Universität des Saarlandes,
Campus E 1.7, D-66123 Saarbrücken, Germany
{mdoernfe,jguo}@mmci.uni-saarland.de

² Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, D-10587 Berlin, Germany
{christian.komusiewicz,mathias.weller}@tu-berlin.de

Abstract. Given a collection \mathcal{C} of partitions of a base set S , the NP-hard CONSENSUS CLUSTERING problem asks for a partition of S which has a total Mirkin distance of at most t to the partitions in \mathcal{C} , where t is a non-negative integer. We present a parameterized algorithm for CONSENSUS CLUSTERING with running time $O(4.24^k \cdot k^3 + |\mathcal{C}| \cdot |S|^2)$, where $k := t/|\mathcal{C}|$ is the average Mirkin distance of the solution partition to the partitions of \mathcal{C} . Furthermore, we strengthen previous hardness results for CONSENSUS CLUSTERING, showing that CONSENSUS CLUSTERING remains NP-hard even when all input partitions contain at most two subsets. Finally, we study a local search variant of CONSENSUS CLUSTERING, showing W[1]-hardness for the parameter ‘‘radius of the Mirkin-distance neighborhood’’. In the process, we also consider a local search variant of the related CLUSTER EDITING problem, showing W[1]-hardness for the parameter ‘‘radius of the edge modification neighborhood’’.

1 Introduction

The NP-hard CONSENSUS CLUSTERING problem aims at reconciling the information that is contained in multiple clusterings of a base set S . More precisely, the input of a CONSENSUS CLUSTERING instance is a multi-set \mathcal{C} of partitions of a base set S into subsets, also referred to as *clusters*, and the aim is to find a partition of S that is similar to \mathcal{C} . Herein, the similarity between two partitions is measured as follows. Two elements $a, b \in S$ are *co-clustered* in a partition C of S , if a and b are in the same cluster of C , and *anti-clustered*, if a and b are in different clusters of C . For two partitions C and C' of S and a pair of elements $a, b \in S$, let $\delta_{\{C,C'\}}(a, b) = 1$ if a and b are anti-clustered in C and co-clustered in C' or vice versa, and $\delta_{\{C,C'\}}(a, b) = 0$, otherwise. Then, the *Mirkin distance* $\text{dist}(C, C') := \sum_{\{a,b\} \subseteq S} \delta_{\{C,C'\}}(a, b)$ between two partitions C and C' of S is the number of pairs $a, b \in S$ that are clustered ‘‘differently’’ by C and C' .

* Supported by the DFG Excellence Cluster on Multimodal Computing and Interaction (MMCI) and DFG project DARE (NI 369/11).

The *total Mirkin distance* between a partition C and a multi-set \mathcal{C} of partitions is defined as $\text{dist}(C, \mathcal{C}) := \sum_{C' \in \mathcal{C}} \text{dist}(C, C')$. Altogether, the CONSENSUS CLUSTERING problem is defined as follows.

Input: A multi-set of partitions $\mathcal{C} = (C_1, \dots, C_n)$ of a base set $S = \{1, 2, \dots, m\}$ and an integer $t \geq 0$.

Question: Is there a partition C of S with $\text{dist}(C, \mathcal{C}) \leq t$?

CONSENSUS CLUSTERING has applications, for instance, in gene expression data analysis [13], clustering categorical data, improving clustering robustness, and preserving privacy [9]. The NP-hardness of CONSENSUS CLUSTERING was shown several times [12, 16]. For $n = 2$, that is, with two input partitions, it is solvable in polynomial time: either input partition minimizes t . In contrast, already for $n = 3$ minimizing t is APX-hard [5]. The variant of CONSENSUS CLUSTERING where the output partition is required to have at most $d \geq 2$ subsets, d being a constant, is NP-hard for every $d \geq 2$ [4] but it admits a PTAS for minimizing t [4, 6, 11]. Various heuristics for CONSENSUS CLUSTERING have been experimentally evaluated [2, 10]. CONSENSUS CLUSTERING is closely related to CLUSTER EDITING [15], also known as CORRELATION CLUSTERING [1].

Until now, the study of the parameterized complexity [7, 8, 14] of CONSENSUS CLUSTERING seems to be neglected. One reason for this might be the lack of an obvious reasonable parameter for this problem: First, the overall Mirkin distance of solutions is usually not small in practice: every element pair that is co-clustered in at least one partition and anti-clustered in at least one other partition contributes at least one to this parameter. Second, CONSENSUS CLUSTERING is trivially fixed-parameter tractable with respect to the number m of elements but m is also unlikely to take small values in real-world instances. Finally, CONSENSUS CLUSTERING is NP-hard for $n = 3$, ruling out fixed-parameter tractability with respect to n . Betzler et al. [3] considered the parameter “average Mirkin distance p between the input partitions”, that is, $p := \sum_{i \neq j} \text{dist}(C_i, C_j) / (n(n-1))$, and presented a “partial kernelization” for this parameter. More precisely, they presented a set of polynomial-time data reduction rules whose application yields an instance with $|S| = O(p)$ [3]. Then, checking all possible partitions of S gives an optimal solution, resulting in a fixed-parameter algorithm for the parameter p . Since the Mirkin distance is a metric, the average Mirkin distance of solution partitions $k := t/n$ is at least $p/2$ [3]. Hence, the above also implies fixed-parameter tractability with respect to k . However, the brute-force check of all possible partitions of S leads to an impractical running time of roughly $2^{O(k \log k)} \text{poly}(n, m)$.

Motivated by these observations, we study several parameterizations of CONSENSUS CLUSTERING. First, we complement the partial kernelization result by presenting a search tree algorithm with running time $O(4.24^k \cdot k^3 + nm^2)$. Second, we consider the parameter “maximal number of clusters in any input partition”. We show that CONSENSUS CLUSTERING remains NP-hard even if every input partition consists of at most two clusters, ruling out fixed-parameter tractability for this parameter. We also strengthen the result of Bonizzoni et al. [4] by showing that, even if all input partitions contain at most two clusters, seeking a solution partition with at most two clusters remains NP-hard. Finally, we

consider a local search variant of CONSENSUS CLUSTERING showing that, given in addition to \mathcal{C} and S a partition C of S , the problem of deciding whether there is a partition C' such that $\text{dist}(C', \mathcal{C}) < \text{dist}(C, \mathcal{C})$ and $\text{dist}(C', C) \leq d$ for some integer $d \geq 0$, is W[1]-hard with respect to d . Moreover, we also show W[1]-hardness of a local search variant of CLUSTER EDITING. Due to the lack of space, several details are deferred to a full version of this article.

Preliminaries. Given a base set S and a multi-set \mathcal{C} of partitions of S , let $n := |\mathcal{C}|$ and $m := |S|$. We use $\text{co}(a, b)$ for $a, b \in S$ to denote the number of partitions in \mathcal{C} where a and b are co-clustered and use $\text{anti}(a, b)$ to denote the number of partitions where a and b are anti-clustered. Clearly, $n = \text{co}(a, b) + \text{anti}(a, b)$. For a partition C of S and elements $a, b \in S$, the function $\text{dist}_C(a, b)$ is defined as the number of partitions in \mathcal{C} in which a, b are clustered in a different way than in C . More precisely, if a and b are co-clustered in C , then $\text{dist}_C(a, b) = \text{anti}(a, b)$; otherwise, $\text{dist}_C(a, b) = \text{co}(a, b)$. Clearly, $\text{dist}(C, \mathcal{C}) = 1/2 \cdot \sum_{\{a, b\} \subseteq S} \text{dist}_C(a, b)$.

2 A Search Tree Algorithm for the Average Mirkin Distance

In this section, we present a search tree algorithm for CONSENSUS CLUSTERING parameterized by the average Mirkin distance $k := t/n$ of a solution partition to the set of input partitions \mathcal{C} . The main idea of this search tree algorithm follows the standard paradigm of branching algorithms in parameterized algorithmics: branch into a bounded number of cases and decrease the parameter in each case. The difficulty for using this approach for the parameter k lies in the fact that for a pair of elements $a, b \in S$ the values of either $\text{co}(a, b)$ or $\text{anti}(a, b)$ can be arbitrarily small for increasing n . When branching on such element pairs, the parameter might not really decrease in some cases. We circumvent this problem by finding a way to always branch into at most two cases, decreasing k by at least $1/3$ in each case. In the following, we describe this approach in detail.

Description of the Algorithm. The algorithm consists of two phases, the first phase is a search tree algorithm and the second phase is a polynomial-time algorithm solving the remaining instances at the leaves of the search tree. Each node v of the search tree is associated with a partition C_v of S , called “temporary solution”, and a list L_v , called “separation list”, that contains pairs of subsets in C_v . These two data structures restrict the partitions we are seeking in the subtree rooted at v : The temporary solution C_v requires that the elements co-clustered by C_v will remain co-clustered in the final solution sought for. The separation list L_v requires that in the solution for each pair $\{K_1, K_2\} \in L_v$ the elements in K_1 are in different subsets than the elements in K_2 . In each search tree node, we keep track of the average Mirkin distance that is already caused by the constraints of C_v and L_v . To this end, consider the following.

Let U be the set of all unordered pairs $\{a, b\}$ of elements $a, b \in S$ with $a \neq b$. Based on C_v and L_v , we divide U into two subsets. The first subset U_v^1 contains the “resolved pairs”, that is, the pairs of elements that are either co-clustered in C_v or contained in two subsets forming a pair in L_v . The other

subset $U_v^2 := U \setminus U_v^1$ contains the ‘‘unresolved pairs’’. Then, each node carries a rational number k_v , called the ‘‘average Mirkin distance bound for unresolved pairs’’, which means that in the subtree rooted at v we seek only partitions C with $1/n \cdot \sum_{\{a,b\} \in U_v^2} \text{dist}_C(a, b) \leq k_v$.

At the root r of the search tree, we start with the partition $C_r := \{\{i\} \mid i \in S\}$ where all elements are in distinct sets, an empty separation list, and $k_r = k = t/n$. At every node v of the search tree, we branch into two cases, each performing one of the following two operations on two subsets X_i and X_j in C_v . One operation ‘‘merges’’ X_i and X_j , that is, it removes X_i and X_j from C_v and adds $X_i \cup X_j$ to C_v . The other operation ‘‘separates’’ X_i and X_j , that is, it adds the subset pair $\{X_i, X_j\}$ to the separation list L_v .

To give a formal description of the search tree we introduce the following notations. Let X and Y be two subsets of S that are contained in the temporary solution C_v of a search tree node v , and let L_v be the separation list of v . If X and Y do not form a pair in L_v , then we define $\text{co}_v(X, Y) := \sum_{a \in X} \sum_{b \in Y} \text{co}(a, b)$ and $\text{anti}_v(X, Y) := \sum_{a \in X} \sum_{b \in Y} \text{anti}(a, b)$; otherwise, we set $\text{co}_v(X, Y) := 0$ and $\text{anti}_v(X, Y) := \infty$. Moreover, we say that the predicate $(XY)_v$ is true iff $\text{anti}_v(X, Y) < n/3$, $X \leftrightarrow_v Y$ is true iff $\text{co}_v(X, Y) < n/3$, and $X \#_v Y$ is true iff $\text{co}_v(X, Y) \geq n/3$ and $\text{anti}_v(X, Y) \geq n/3$. If $X \#_v Y$ holds, we call X and Y a *dirty subset pair*. Three subsets X , Y , and Z are called a *dirty subset triple*, if $(XY)_v$, $(YZ)_v$, and $X \leftrightarrow_v Z$ are true.

The search tree algorithm uses two branching rules, the *dirty pair rule* and the *dirty triple rule*. In the following, let v denote the node of the search tree in which the rules are applied. Both rules branch into two cases, referred to as v_1 and v_2 . Furthermore, branching into case v_1 (case v_2) is only performed if $k_{v_1} \geq 0$ ($k_{v_2} \geq 0$); we refer to this as the *stop criterion*.

Branching Rule 1 (Dirty pair rule). *If C_v contains two subsets X and Y with $X \#_v Y$, then branch into the following two cases.*

- Case v_1 : merge X and Y and set $k_{v_1} := k_v - 1/n \cdot \text{anti}_v(X, Y)$.
- Case v_2 : separate X and Y and set $k_{v_2} := k_v - 1/n \cdot \text{co}_v(X, Y)$.

In case the dirty pair rule is not applicable, because there is no dirty pair, we apply the following rule.

Branching Rule 2 (Dirty triple rule). *If C_v contains three subsets X , Y , and Z such that $(XY)_v$, $(YZ)_v$, and $X \leftrightarrow_v Z$, then branch into the following two cases.*

- Case v_1 : separate X and Y and set $k_{v_1} := k_v - 1/n \cdot \text{co}_v(X, Y)$.
- Case v_2 : separate Y and Z and set $k_{v_2} := k_v - 1/n \cdot \text{co}_v(Y, Z)$.

We call a search tree node in which neither branching rule can be applied a *leaf* of the search tree. At the leaves the algorithm enters its second phase in which the temporary solution C_v is modified into a complete solution as follows.

As long as possible, merge all subset pairs X and Y for which $(XY)_v$ holds and, after each merge operation, update $k_v := k_v - 1/n \cdot \text{anti}_v(X, Y)$. Afterwards,

if $k_v \geq 0$ then output C_v (or, alternatively, answer “yes”) and terminate the algorithm. If there is no search tree leaf in which a partition is output, then answer “no”.

Correctness of the algorithm. We now show the correctness of the algorithm. In the following, a partition C of S *satisfies* the restrictions of a search tree node v if C fulfills the following three conditions:

- (C1) for every subset $X \in C_v$, there is a subset $Z \in C$ with $X \subseteq Z$,
- (C2) for every pair $\{X, Y\}$ in L_v , there are two subsets $Z, Z' \in C$ with $Z \neq Z'$,
 $X \subseteq Z$, and $Y \subseteq Z'$, and
- (C3) $1/n \cdot \sum_{\{a,b\} \in U_v^2} \text{dist}_C(a, b) \leq k_v$.

We say that a branching rule is *sound* if each partition C satisfying the restrictions of a node v , satisfies the restrictions of one of the child nodes created by applying this rule to v .

Lemma 1. *Both branching rules are sound.*

The following lemma shows the correctness of the second phase of the algorithm. More precisely, it states that the operations performed in a leaf v of the search tree yield a partition C that, of all partitions satisfying the restrictions of v , has minimum distance to the input partitions.

Lemma 2. *Let v be a leaf of the search tree, and let \mathcal{D} be the set of partitions satisfying the restrictions of v . Then, there exists a partition $C \in \mathcal{D}$ such that $\text{dist}(C, \mathcal{C}) = \min_{C' \in \mathcal{D}} \text{dist}(C', \mathcal{C})$ and for all $X, Y \in C_v$, the following holds:*

- (a) *If $(XY)_v$, then there is a subset $Z \in C$ with $X \subseteq Z$ and $Y \subseteq Z$.*
- (b) *If $X \leftrightarrow_v Y$, then there are two subsets $Z, Z' \in C$ with $Z \neq Z'$, $X \subseteq Z$, and $Y \subseteq Z'$.*

Altogether, this implies the following.

Proposition 1. *The algorithm is correct.*

Running time analysis. Next, we bound the running time of the algorithm. The exponential part of the running time clearly depends on the size of the search tree, that is, on the number of search tree nodes. A rough estimation of this size is as follows.

At each node v of the search tree, we either merge or separate two subsets $X, Y \in C_v$. Both operations cause a decrease of the average Mirkin distance bound k_v . More precisely, the dirty pair rule decreases k_v either by $1/n \cdot \text{anti}_v(X, Y)$ or $1/n \cdot \text{co}_v(X, Y)$. Since X and Y form a dirty subset pair, k_v is decreased by at least $1/3$ in both cases. Branching on a dirty triple X, Y , and Z with $(XY)_v$, $(YZ)_v$, and $X \leftrightarrow_v Z$ causes separation of X and Y in one case and separation of Y and Z in the other case. The bound k_v is decreased by $1/n \cdot \text{co}_v(X, Y)$ and $1/n \cdot \text{co}_v(Y, Z)$, respectively. Since $(XY)_v$ and $(YZ)_v$ hold, the distance bound k_v is decreased by at least $2/3$ in both cases. Since $k_v < 0$ is

a stop criterion for the search tree, the size of the tree is thus $O(2^{3k}) = O(8^k)$. Using this simple analysis, one obtains a running time bound of $8^k \cdot \text{poly}(n, m)$. In the following, we give a more detailed analysis of the search tree size. In the proof, we use $\phi := (1 + \sqrt{5})/2$ to denote the golden ratio.

Theorem 1. CONSENSUS CLUSTERING can be solved in $O(4.24^k \cdot k^3 + nm^2)$ time.

Proof. We show only the size of the search tree, the proof of the polynomial running time part is deferred to a long version of this article. More precisely, we show that the search tree has size at most $(2/\sqrt{5})\phi^{3k+2} - 1$. To this end, we consider an arbitrary node v in the tree and estimate the size of the subtree rooted at v . Clearly, $k_v \geq 0$ for every node v in the tree. We consider three cases for the value of k_v and prove the size bound for each case.

Case 1: $0 \leq k_v < 1/3$. Then, the two cases created by applying one of the two branching rules both have average Mirkin distance bounds at most $k_v - 1/3 < 0$. Hence, the search tree has only one node. Since $3k_v + 2 \geq 2$ for $k_v \geq 0$, it holds that $(2/\sqrt{5})\phi^{3k_v+2} - 1 \geq (2/\sqrt{5})\phi^2 - 1 = \frac{1+2\sqrt{5}+5}{2\sqrt{5}} - 1 = \frac{3}{\sqrt{5}} > 1$. Thus, the claimed search tree size bound holds in this case.

Case 2: $1/3 \leq k_v < 1/2$. If the dirty triple rule is applied, then k_v is decreased by at least $2/3$ in both cases. Therefore, the rule creates no child node for v , and the claimed search tree size bound holds as shown above. If the dirty pair rule is applied to a dirty subset pair $X, Y \in C_v$, then k_v is decreased by $1/n \cdot \text{co}_v(X, Y)$ in one case and by $1/n \cdot \text{anti}_v(X, Y)$ in the other case. Since $\text{co}_v(X, Y) + \text{anti}_v(X, Y) = |X| \cdot |Y| \cdot n$, at least one of $1/n \cdot \text{co}_v(X, Y)$ and $1/n \cdot \text{anti}_v(X, Y)$ is greater than $1/2$. Consequently, v has at most one child node. Since Case 1 applies to this child node, the subtree rooted at v contains at most two nodes. Since $3k_v + 2 \geq 3$ for $k \geq 1/3$ and

$$(2/\sqrt{5})\phi^3 - 1 = \frac{1 + 3\sqrt{5} + 15 + 5\sqrt{5}}{4\sqrt{5}} - 1 = \frac{4 + 2\sqrt{5}}{\sqrt{5}} - 1 = \frac{4}{\sqrt{5}} + 1 > 2,$$

the claimed search tree size bound holds for this case.

Case 3: $k_v \geq 1/2$. As argued above, the dirty pair rule creates at most two child nodes, v_1 with $k_{v_1} := k_v - 1/n \cdot \text{co}_v(X, Y)$ and v_2 with $k_{v_2} := k_v - 1/n \cdot \text{anti}_v(X, Y)$, while the dirty triple rule adds at most two child nodes, v_1 with $k_{v_1} := k_v - 1/n \cdot \text{co}_v(X, Y)$ and v_2 with $k_{v_2} := k_v - 1/n \cdot \text{co}_v(Y, Z)$. Since $\text{co}_v(X, Y) + \text{anti}_v(X, Y) = |X| \cdot |Y| \cdot n$, we have $\text{anti}_v(X, Y) \geq n/3$ and $\text{co}_v(X, Y) \geq n/3$ for a dirty subset pair X, Y . Therefore, we have $k_{v_1} \leq k_v - \alpha$ and $k_{v_2} \leq k_v - 1 + \alpha$ for some α with $1/3 \leq \alpha < 2/3$. Due to symmetry, we can assume $1/3 \leq \alpha \leq 1/2$. Moreover, for the dirty subset triple, we have $(XY)_v$ and $(YZ)_v$, implying $1/n \cdot \text{co}_v(X, Y) \geq 2/3$ and $1/n \cdot \text{co}_v(Y, Z) \geq 2/3$. As the function $(2/\sqrt{5})\phi^{3k+2} - 1$ is monotonically increasing on k , we can use $k_{v_1} = k_v - \alpha$ and $k_{v_2} = k_v - 1 + \alpha$ with $1/3 \leq \alpha \leq 1/2$ to obtain an upper bound on the size of the subtree rooted at v , that is, in our analysis the worst-case search tree size bound is obtained for the dirty pair rule.

Assume, by an inductive argument, that the search tree size bound holds for all $k' < k_v$. Clearly, the size of the subtree rooted at v is at most

$$\left[(2/\sqrt{5})\phi^{3(k_v-1+\alpha)+2} - 1 \right] + \left[(2/\sqrt{5})\phi^{3(k_v-\alpha)+2} - 1 \right] + 1.$$

We differentiate this bound with respect to α to find local extrema:

$$\begin{aligned} & \frac{d}{d\alpha} (2/\sqrt{5})\phi^{3(k_v-1+\alpha)+2} + (2/\sqrt{5})\phi^{3(k_v-\alpha)+2} - 1 \\ &= \frac{6}{\sqrt{5}} \log\left(\frac{2}{1+\sqrt{5}}\right) [\phi^{3(k_v-1+\alpha)} - \phi^{3(k_v-\alpha)}]. \end{aligned}$$

This term equals zero only if $k_v - 1 + \alpha = k_v - \alpha$, that is, if $\alpha = 1/2$. Thus, the candidates for the maximum are the critical point $\alpha = 1/2$ and the endpoints of the interval $[1/3, 1/2]$. For $\alpha = 1/2$, the search tree has size at most $(4/\sqrt{5}) \cdot \phi^{3k_v+1/2} - 1$ and for $\alpha = 1/3$ the search tree has size at most $(2/\sqrt{5}) \cdot \phi^{3k_v+2} - 1$. Hence, the claimed search tree size bound holds in this case as well.

Summarizing, the upper bound of $(2/\sqrt{5})\phi^{3k_v+2} - 1$ holds for all $k_v \geq 0$ and thus we can construct the search tree in $O(\phi^{3k+2} \cdot k^3) = O(4.24^k \cdot k^3)$ time. The overall running time bound follows. \square

3 NP-Hardness for Input Partitions with a Bounded Number of Clusters

Bonizzoni et al. [4] proved that the variation of CONSENSUS CLUSTERING in which the solution C is required to contain at most d clusters, is NP-hard for every $d \geq 2$. First, we consider—instead of *solution partitions* with a bounded number of clusters—instances (\mathcal{C}, t) in which each *input partition* has at most d' clusters, that is, $d' := \max_{C \in \mathcal{C}} |C|$. We show that CONSENSUS CLUSTERING is fixed-parameter intractable with respect to d' by proving the following.

Theorem 2. CONSENSUS CLUSTERING remains NP-hard, even if all input partitions have at most two subsets.

Next, we strengthen the hardness result of Bonizzoni et al. [4] by showing that it is NP-hard to find *solution partitions* with at most two clusters even if every input partition has at most two clusters.

CONSENSUS CLUSTERING WITH 2-PARTITIONS (CC2P)

Input: A multi-set of partitions $\mathcal{C} = (C_1, \dots, C_n)$ of a base set $S = \{1, 2, \dots, m\}$, where $|C_i| \leq 2$ for all $1 \leq i \leq n$, and an integer $t \geq 0$.

Question: Is there a partition C of S with $|C| \leq 2$ and $\text{dist}(C, \mathcal{C}) \leq t$?

Theorem 3. CC2P is NP-hard.

With introducing some dummy elements, one can then easily prove that CONSENSUS CLUSTERING is also NP-hard if the input partitions have at most $d \geq 3$ subsets and we ask for a partition with at most d subsets. Moreover, for every $d \geq 2$, a similar reduction can be used to show the NP-hardness in case the input partitions contain *exactly* d subsets and we ask for a partition with *exactly* d subsets.

4 Hardness of Local Search

In this section, we study the parameterized complexity of the following local search variant of CONSENSUS CLUSTERING:

CONSENSUS CLUSTERING WITH MIRKIN-LOCAL SEARCH (CCML)

Input: A multi-set $\mathcal{C} = (C_1, \dots, C_n)$ of partitions of a base set $S = \{1, 2, \dots, m\}$, a partition C of S , a nonnegative integer d .

Question: Is there a partition C' of S such that $\text{dist}(C', \mathcal{C}) < \text{dist}(C, \mathcal{C})$ and $\text{dist}(C, C') \leq d$?

The study of CCML is motivated as follows. For a given multi-set \mathcal{C} of input partitions, a partition that has an average Mirkin distance at most k to \mathcal{C} trivially has Mirkin distance at most k to at least one of the input partitions. Moreover, it could be that there is one input partition to which this optimal partition has a Mirkin distance $d \ll k$. Hence, a good strategy to find a partition with average distance at most k to \mathcal{C} could be to search in the local neighborhood of the input partitions. Unfortunately, as we show in the following, it is unlikely that a running time of $f(d) \cdot \text{poly}(n, m)$ can be achieved for this local search problem.

We present a parameterized reduction from the W[1]-hard CLIQUE problem [7]. More precisely, we reduce a variant of CLIQUE in which there is at least one vertex in the input graph that is adjacent to all other vertices.

CLIQUE WITH UNIVERSAL VERTEX

Input: An undirected graph $G = (V, E)$ with a vertex $u \in V$ such that $N[u] = V$, and a nonnegative integer k .

Question: Is there a clique of size k in G ?

The W[1]-hardness of CLIQUE WITH UNIVERSAL VERTEX with respect to k follows from a straightforward reduction from CLIQUE. For notational simplicity, we assume that k is an odd number in the following; the problem clearly remains W[1]-hard with this further restriction.

Given an instance $(G = (V, E), k)$ of CLIQUE WITH UNIVERSAL VERTEX, we construct an instance of CCML as follows. The base set S consists of the vertex set V and of $(|V| \cdot (k - 1)/2) - 1$ further elements. More precisely, for each vertex $v \in V \setminus \{u\}$, we create an element set $S_v := \{v_1, \dots, v_{(k-1)/2}\}$, and for the universal vertex u , we create an element set $S_u := \{u_1, \dots, u_{(k-1)/2-1}\}$. The complete element set is then $S := V \cup S_u \cup \bigcup_{v \in V \setminus \{u\}} S_v$.

We construct a CCML instance in which \mathcal{C} consists of $n := 2|E| + 1$ partitions of S . The first $|E|$ partitions consist of one cluster that completely contains S :

$$C_i := \{S\}, 1 \leq i \leq |E|.$$

For each $\{v, w\} \in E$ we create one partition containing $\{v, w\}$ as one cluster and a singleton cluster for each of the other elements. Let $E = \{e_1, \dots, e_{|E|}\}$. Then these partitions are formally defined as

$$C_{|E|+i} := \{e_i\} \cup \{\{s\} \mid s \in S \setminus e_i\}, 1 \leq i \leq |E|.$$

Finally, we create one partition that, for each $v \in V$, contains a cluster that contains v and S_v :

$$C_{2|E|+1} := \{\{v\} \cup S_v \mid v \in V\}.$$

Overall, the following can be observed for this set of partitions. Two vertices that are adjacent in G are co-clustered in $|E|+1$ partitions of the CCML instance. Two vertices that are not adjacent in G are co-clustered in $|E|$ partitions. Furthermore, each pair of elements $v \in V$ and $w \in S \setminus V$, is co-clustered in $|E|+1$ partitions if $w \in S_v$, and co-clustered in $|E|$ partitions, otherwise. Finally, each pair of elements $v, w \in S \setminus V$ is co-clustered in $|E|+1$ partitions if there is an $x \in V$ such that $v \in S_x$ and $w \in S_x$ and co-clustered in $|E|$ partitions, otherwise. Consequently, for each pair of elements $v, w \in S$ we have $|\text{co}(v, w) - \text{anti}(v, w)| = 1$.

The partition C of the instance is defined exactly as the partition $C_{2|E|+1}$, that is, for each $v \in V$, C contains the cluster $\{v\} \cup S_v$. We conclude the construction of the CCML instance by setting $d := k \cdot (k-1) - 1$.

The main idea behind the construction is that with the S_v 's and by setting $d := k \cdot (k-1) - 1$ we can enforce that in a “better” partition within distance d there is a cluster with exactly k elements from V . The elements of this cluster must then induce a clique in G since otherwise the partition is not better than C .

Theorem 4. *CCML parameterized by the radius d of the Mirkin-distance neighborhood is W[1]-hard.*

In the construction above we have $|\text{co}(v, w) - \text{anti}(v, w)| = 1$ for each element pair $v, w \in S$. Consequently, each element pair causes a Mirkin distance of at least $|E|$ and at most $|E| + 1$ in any solution. Hence, the CONSENSUS CLUSTERING instance can also be formulated as an “equivalent” instance of CLUSTER EDITING. We can modify the construction above to also show the hardness of a local search variant for CLUSTER EDITING.

CLUSTER EDITING WITH EDGE-MODIFICATION-LOCAL SEARCH

Input: An undirected graph $G = (V, E)$, a cluster graph $C = (V, E')$, and a nonnegative integer k .

Question: Is there a cluster graph $C' = (V, E'')$ such that $\text{dist}(G, C') < \text{dist}(G, C)$ and $\text{dist}(C, C') \leq k$?

Herein, a cluster graph is a disjoint union of complete graphs and $\text{dist}(G = (V, E), H = (V, E')) := |(E \setminus E') \cup (E' \setminus E)|$ denotes the number of edge modifications needed to transform a graph G into a graph H . In complete analogy to Theorem 4, we can show the following.

Theorem 5. *CLUSTER EDITING WITH EDGE-MODIFICATION-LOCAL SEARCH parameterized by the radius k of the edge-modification neighborhood is W[1]-hard.*

5 Conclusion

There are many possibilities for further research concerning CONSENSUS CLUSTERING. For instance, comparing our algorithm with known heuristics for CONSENSUS CLUSTERING would be interesting. Also, further parameters should be

considered for CONSENSUS CLUSTERING. For example, what is the complexity of CONSENSUS CLUSTERING when for each input partition, every cluster has a bounded number of elements? Also, the previously known partial kernelization implies fixed-parameter tractability [3] for the parameter “number of dirty element pairs”. Are there efficient fixed-parameter algorithms for this parameter? Finally, it would be interesting to consider further parameters for the local search variant of CONSENSUS CLUSTERING. For example, is this problem fixed-parameter tractable when the number n of input partitions is bounded?

References

1. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. *Mach. Learn.* 56(1), 89–113 (2004)
2. Bertolacci, M., Wirth, A.: Are approximation algorithms for consensus clustering worthwhile? In: Proc. 7th SDM, pp. 437–442. SIAM (2007)
3. Betzler, N., Guo, J., Komusiewicz, C., Niedermeier, R.: Average parameterization and partial kernelization for computing medians. *J. Comput. Syst. Sci.* 77(4), 774–789 (2011)
4. Bonizzoni, P., Vedova, G.D., Dondi, R.: A PTAS for the minimum consensus clustering problem with a fixed number of clusters. In: Proc. 11th ICTCS (2009)
5. Bonizzoni, P., Vedova, G.D., Dondi, R., Jiang, T.: On the approximation of correlation clustering and consensus clustering. *J. Comput. Syst. Sci.* 74(5), 671–696 (2008)
6. Coleman, T., Wirth, A.: A polynomial time approximation scheme for k -consensus clustering. In: Proc. 21st SODA, pp. 729–740. SIAM (2010)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
8. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
9. Gionis, A., Mannila, H., Tsaparas, P.: Clustering aggregation. *ACM Trans. Knowl. Discov. Data* 1(1) (2007)
10. Goder, A., Filkov, V.: Consensus clustering algorithms: Comparison and refinement. In: Proc. 10th ALENEX, pp. 109–117. SIAM (2008)
11. Karpinski, M., Schudy, W.: Linear time approximation schemes for the Gale-Berlekamp game and related minimization problems. In: Proc. 41st STOC, pp. 313–322. ACM (2009)
12. Křivánek, M., Morávek, J.: NP-hard problems in hierarchical-tree clustering. *Acta Inform.* 23(3), 311–323 (1986)
13. Monti, S., Tamayo, P., Mesirov, J.P., Golub, T.R.: Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Mach. Learn.* 52(1-2), 91–118 (2003)
14. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
15. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. *Discrete Appl. Math.* 144(1-2), 173–182 (2004)
16. Wakabayashi, Y.: The complexity of computing medians of relations. *Resenhas* 3(3), 323–350 (1998)

Two Fixed-Parameter Algorithms for the Cocompling Problem

Victor Campos¹, Sulamita Klein², Rudini Sampaio¹, and Ana Silva¹

¹ Universidade Federal do Ceará, Fortaleza, Brazil

{campos,rudini,ana.silva}@lia.ufc.br

² Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil

sula@cos.ufrj.br

Abstract. A (k, ℓ) -cocompling of a graph G is a partition of the vertex set of G into at most k independent sets and at most ℓ cliques. Given a graph G and integers k and ℓ , the *Cocompling Problem* is the problem of deciding if G has a (k, ℓ) -cocompling. It is known that determining the cochromatic number (the minimum $k + \ell$ such that G is (k, ℓ) -cocompliable) is NP-hard [24]. In 2011, Bravo et al. obtained a polynomial time algorithm for P_4 -sparse graphs [8]. In this paper, we generalize this result by obtaining a polynomial time algorithm for $(q, q - 4)$ -graphs for every fixed q , which are the graphs such that every subset of at most q vertices induces at most $q - 4$ induced P_4 's. P_4 -sparse graphs are $(5, 1)$ -graphs. Moreover, we prove that the cocompling problem is FPT when parameterized by the treewidth $tw(G)$ or by the parameter $q(G)$, defined as the minimum integer $q \geq 4$ such that G is a $(q, q - 4)$ -graph.

1 Introduction

A (k, l) -cocompling of a graph G is a partition of $V(G)$ into at most k stable sets and at most ℓ cliques. Given a graph G and positive integers k and ℓ , the *Cocompling Problem* is the problem of deciding whether G is (k, ℓ) -cocompliable. The *cochromatic number* $z(G)$ of G is the smallest integer $k + \ell$ such that G has a (k, ℓ) -cocompling. These concepts were first introduced in [23] and attracted the attention of Erdős [12–14] as a natural extension of the graph coloring problem.

An interesting application is to partition a permutation in increasing or decreasing subsequences. Unfortunately, determining the cochromatic number is NP-hard even for permutation graphs [24].

Brandstädt [6] proved that the Cocompling Problem is polynomial time solvable for $k, \ell \leq 2$. However, this problem remains NP-complete for $k \geq 3$ or $\ell \geq 3$ [6]. In fact, it is easy to see that deciding if the cochromatic number is at most 3 is NP-complete. Clearly, an $O(f(n))$ time algorithm for the Cocompling Problem implies an $O(n^2 f(n))$ time algorithm for the cochromatic number.

The matrix partition problem is a yet wider generalization of the Cocompling Problem, where the partition satisfies not only internal restrictions (such as being an independent set or a clique), but also external restrictions (e.g., being pairwise connected by all possible edges). We refer to [17, 18] for details.

In 2002, Fomin et al. [19] obtained a factor 1.71 approximation algorithm for cochromatic number on comparability (or cocomparability) graphs, and a factor $\log n$ approximation algorithm on perfect graphs. In 2010, Heggernes et al. obtained an FPT algorithm for cochromatic number on permutation graphs (and on perfect graphs) with bounded cochromatic number [20].

In 2004, Hell et al. [21] obtained a polynomial time algorithm to decide if a chordal graph is (k, ℓ) -cocolorable for any k and ℓ . This result was extended to cographs [7, 11, 16] in 2005 and to P_4 -sparse graphs [8] and to extended P_4 -laden graphs [9] in 2011. It was also investigated for perfect graphs in [15].

A graph G is said to be a $(q, q - 4)$ graph if every subset of at most q vertices of G induces at most $q - 4$ induced P_4 's. This concept was introduced in [3], where they also gave a decomposition theorem of these graphs. Cographs and P_4 -sparse graphs are $(q, q - 4)$ -graphs for $q = 4$ and $q = 5$, respectively.

In this paper, we generalize the results of [8] and [11] by obtaining polynomial time algorithms for the Cocoloring Problem and the cochromatic number on $(q, q - 4)$ -graphs, for every fixed q . Moreover, we prove that these algorithms are fixed parameter tractable on the parameter $q(G)$, which is the minimum q such that G is a $(q, q - 4)$ -graph.

We also obtain a fixed parameter algorithm on the treewidth $tw(G)$ of the graph. Notice that there exist many graphs with bounded treewidth and unbounded $q(G)$, and vice-versa. For example, $q(K_n)$ is 4 while $tw(K_n) = n$ and $tw(P_n)$ is 1 while $q(P_n) = n + 1$. This algorithm is dynamic programming, but it is not trivial and quite complicated, since we have to deal with color classes which are stable sets and with color classes which are cliques. Surprisingly, the algorithms for the Cocoloring Problem and the cochromatic number are respectively $O(n^3)$ and $O(n)$ time; an unexpected difference.

These algorithms are the first two fixed parameter algorithms for the Cocoloring Problem and the cochromatic number in the general case.

2 Background and Notation

The graph terminology used in this paper follows [5]. Let G be a simple graph. We denote by $V(G)$ and $E(G)$ the sets of vertices and edges of G , respectively.

A *subgraph* of G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Given $X \subseteq V(G)$, the *subgraph of G induced by X* is the graph $G[X] = (X, Ex)$, where $uv \in Ex$ if and only if $u, v \in X$ and $uv \in E(G)$. A *path* of G is any sequence $\langle v_1, \dots, v_q \rangle$ of vertices such that $(v_i, v_{i+1}) \in E(G)$ for $1 \leq i < q$. An *induced path* with n vertices is denoted by P_n .

A graph G is called *connected* if there is a path between every two vertices of G ; otherwise, it is called *disconnected*. A connected graph G with no cycles is called a *tree*.

We denote by \overline{G} the complement of G . A clique is a subset of vertices which induces all possible edges. A clique with n vertices is denoted by K_n . An independent set or stable set is a subset of vertices which induces no edge.

The M -partition problem was introduced by Feder et al. [17]. Let M be a symmetric $m \times m$ matrix with entries $M(i, j) \in \{0, 1, *\}$. Given a graph G , an M -partition of G is a partition of the vertex set into m parts V_1, V_2, \dots, V_m such that V_i is a clique if $M(i, i) = 1$ or a stable set if $M(i, i) = 0$; and such that parts V_i and V_j are completely adjacent if $M(i, j) = 1$ or non-adjacent if $M(i, j) = 0$. If $M(i, i) = *$ or $M(i, j) = *$, there is no restriction. Thus the diagonal entries define whether the parts are cliques or stable sets, and the off-diagonal entries define whether the parts are completely adjacent or non-adjacent (with * meaning no restriction).

Observe that if all diagonal entries of M are in $\{0, 1\}$, with k 0's and ℓ 1's, and all off-diagonal entries are *, then an M -partition of G is precisely a (k, ℓ) -cocompling of G . In 2006, Feder et al. [15] studied this case of the matrix partition problem for the class of cographs.

A cograph is a graph with no induced P_4 's. Cographs are self-complementary, since the complement of a P_4 is also a P_4 . Therefore, cographs have a nice structure.

- K_1 is a cograph;
- if G is a cograph, then \overline{G} is also a cograph;
- if G and H are cographs, then $G \cup H$ is also a cograph.

Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *union* of G_1 and G_2 is the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ and the *join* of G_1 and G_2 is the graph $G_1 \vee G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{uv : u \in V_1, v \in V_2\})$.

It follows from the decomposition of cographs that every cograph G is associated with a unique rooted tree $T(G)$, called the *cotree* of G , whose leaves are precisely the vertices of G and whose internal nodes are of two types: union (\cup) or join (\vee). That is, if G is a cograph, then

- G has at most one vertex;
- $G = G_1 \cup G_2$ is the union of two cographs G_1 and G_2 ;
- $G = G_1 \vee G_2$ is the join of two cographs G_1 and G_2 .

A graph G is P_4 -sparse if no five vertices in G induce more than one P_4 . P_4 -sparse graphs are also self-complementary and also have a nice structural decomposition in terms of unions, joins and *spiders*.

A *spider* is a graph whose vertex set has a partition (R, C, S) , where $C = \{c_1, \dots, c_k\}$ and $S = \{s_1, \dots, s_k\}$ for $k \geq 2$ are respectively a clique and a stable set; s_i is adjacent to c_j if and only if $i = j$ (a thin spider), or s_i is adjacent to c_j if and only if $i \neq j$ (a thick spider); and every vertex of R is adjacent to each vertex of C and non-adjacent to each vertex of S .

If a graph G is P_4 -sparse, then [22]

- G has at most one vertex;
- $G = G_1 \cup G_2$ is the union of two cographs G_1 and G_2 ;
- $G = G_1 \vee G_2$ is the join of two cographs G_1 and G_2 ;
- G is a spider (R, C, S) such that $G[R]$ is P_4 -sparse.

We say that a problem is *fixed parameter tractable* on the parameter $q(G)$ if it can be solved in $O(f(q) \cdot n^k)$ time, where $q(G) = q$, $f(\cdot)$ is some function on q and k is a fixed integer.

3 $(q, q - 4)$ -Graphs

A $(q, q - 4)$ -graph is a graph such that every subset of at most q vertices of G induces at most $q - 4$ induced P_4 's. The class of $(q, q - 4)$ -graphs generalizes cographs ($q = 4$) and P_4 -sparse graphs ($q = 5$). These graphs also have a structural decomposition theorem in terms of unions, joins, spiders and small *separable p-components*.

A graph is *p-connected* if, for every partition of the vertex set into two parts A and B , there is a crossing P_4 (with vertices of A and B). A *separable p-component* is a maximal p-connected subgraph with a particular bipartition (H_1, H_2) such that every crossing P_4 $wxyz$ satisfies $x, y \in H_1$ and $w, z \in H_2$. In [3], it was proved an important structural result for $(q, q - 4)$ -graphs.

Theorem 1 ([3]). *If G is a $(q, q - 4)$ -graph with at least q vertices, then*

- (a) *G is the union or the join of two $(q, q - 4)$ -graphs G_1 and G_2 ; or*
- (b) *G is a spider (R, C, S) such that $G[R]$ is a $(q, q - 4)$ -graph; or*
- (c) *G contains a separable p-component H with bipartition (H_1, H_2) , such that $|V(H)| < q$ and $G - H$ is a $(q, q - 4)$ -graph which is complete to H_1 and anti-complete to H_2 .*

This characterization suggests a graph decomposition, called *primeval decomposition*, which can be obtained in linear time [3]. It was also proved that every p-connected $(q, q - 4)$ -graph with $q \geq 8$ has at most q vertices. With this, we can obtain $q(G)$ in $O(n^7)$ time for every graph G from its primeval decomposition (observe that $q(G)$ can be greater than n and, if this is the case, $q(G)$ is the number of induced P_4 's of G plus four).

Our key technical tool to solve the *Cocompling Problem* for $(q, q - 4)$ -graphs is the introduction of the parameter $\chi_\ell(G)$. For $\ell \in \{0, \dots, |V(G)|\}$, let

$$\chi_\ell(G) = \min \left\{ k : G \text{ is } (k, \ell)\text{-cocomcolorable} \right\}.$$

Observe that $\chi_0(G)$ is the chromatic number of G , which can be computed in linear time [2]. Clearly, G is (k, ℓ) -cocomcolorable if and only if $\chi_\ell(G) \leq k$, and the cochromatic number of G is

$$z(G) = \min_{0 \leq \ell \leq |V(G)|} \left\{ \chi_\ell(G) + \ell \right\}. \quad (1)$$

Given a graph G , our main task is to calculate all these parameters $\chi_0(G)$, $\chi_1(G), \dots, \chi_n(G)$. With this, we can decide the (k, ℓ) -cocomcolorability of G for every k and ℓ (just check if $\chi_\ell(G) \leq k$). We can also determine the cochromatic number of G from (1).

We will analyse the possible structure of G according to its primeval decomposition given by Theorem 1. To each graph of this decomposition, we will calculate those parameters $\chi_0, \chi_1, \dots, \chi_n$ according to the lemmas below. We will omit the proofs throughout this text because of space restrictions.

Lemma 1. *If G is the union of two disjoint graphs G_1 and G_2 , then, for every $\ell \in \{0, \dots, |V(G)|\}$*

$$\chi_\ell(G) = \min_{0 \leq i \leq \ell} \{ \max \{ \chi_i(G_1), \chi_{\ell-i}(G_2) \} \}$$

Lemma 2. *If G is the join of two disjoint graphs G_1 and G_2 , then, for every $\ell \in \{0, \dots, |V(G)|\}$*

$$\chi_\ell(G) = \chi_\ell(G_1) + \chi_\ell(G_2)$$

Lemma 3. *If G is a spider (R, C, S) , then $\chi_0(G) = \chi_0(G[R]) + |S|$ and, for every $\ell \in \{1, \dots, |V(G)|\}$:*

$$\chi_\ell(G) = \begin{cases} \chi_\ell(G[R]), & \text{if } \chi_\ell(G[R]) > 0, \\ \min\{1, \chi_{\ell-|S|}(G[R])\}, & \text{if } \chi_\ell(G[R]) = 0 \text{ and } \ell \geq |S|, \\ 1, & \text{otherwise.} \end{cases}$$

Lemma 4. *Let G be a graph which contains a separable p -component H with bipartition (H_1, H_2) , such that $|V(H)| < q$ and $G - H$ is a $(q, q-4)$ -graph which is complete to H_1 and anti-complete to H_2 . For every cocomoring ψ of H , let $\ell_2(\psi)$, $k_1(\psi)$ and $k'(\psi)$ be respectively the number of clique colors with a vertex of H_2 , the number of stable colors with a vertex of H_1 and the number of stable colors with no vertex of H_1 . Then, for every $\ell \in \{0, \dots, |V(G)|\}$,*

$$\chi_\ell(G) = \min_{\psi \in \mathcal{C}_H} \left\{ k_1(\psi) + \max \{ \chi_{\ell-\ell_2(\psi)}(G - H), k'(\psi) \} \right\},$$

where \mathcal{C}_H is the set of all cocomolorings of H with at most ℓ clique colors.

Observe that, since H has at most q vertices, the set \mathcal{C}_H of all cocomolorings of H has size $O(2^q q^q)$, which is constant for fixed q , since we have at most q vertices and q distinct colors, and each color can be a clique color or a stable color. So, given the parameters $\chi_0, \chi_1, \dots, \chi_n$ for $G - H$, we can obtain $\chi_\ell(G)$ in constant time if q is a fixed constant.

Theorem 2. *Let G be a graph with n vertices and let q be a fixed integer. If $q(G) \leq q$, then deciding the Cocomoring Problem and determining the cochromatic number is $O(n^3 + 2^q q^{q+3} \cdot n^2)$ time solvable.*

Proof. By [3], we know that we can construct a primeval decomposition of G with $O(n)$ nodes in linear time. In this decomposition tree, the leaves are either trivial graphs, or spiders with $R = \emptyset$, or p -connected components that are not spiders and have less than q vertices. So, let H be a leaf in the obtained decomposition. If $|V(H)| = 1$, then $\chi_0(H) = 1$ and $\chi_1(H) = 0$; this takes

$O(1)$ time. If $H = (S \cup K, E)$ is a spider, then, by Lemma 3, $\chi_0(H) = |K|$, $\chi_{\ell'}(H) = 1$, for all $\ell' \in \{1, \dots, \min\{\ell, |S| - 1\}\}$, and, if $|S| \geq \ell$, $\chi_{\ell'}(H) = 0$, for all $\ell' \in \{|S|, \dots, \ell\}$. This takes $O(\ell) = O(n)$ time. Finally, if H is a p-connected component, then, by Lemma 4, $\chi_{\ell'}(H) = \min_{\psi \in \mathcal{C}_H} \{k_1(\psi) + k'(\psi)\}$, for all $\ell' \in \{0, \dots, h = \min\{|V(H)|, \ell\}\}$. Thus, computing $\chi_0(H), \dots, \chi_h(H)$ takes time $O(2^q q^{q+3})$ since $h \leq |V(H)| \leq q$, $|\mathcal{C}(H)| = O(2^q q^q)$ and we have to check the cocolorability and calculate the parameters $k_1(\psi)$, $\ell_2(\psi)$ and $k'(\psi)$, which take $O(q^2)$ time.

Now, suppose that H is an internal node. If H is the union of two other graphs, by Lemma 1, computing $\chi_{\ell'}$ takes $O(n)$ time, for each $\ell' \in \{0, \dots, \ell\}$, which gives a total of $O(n^2)$. If H is the join of two other graphs, by Lemma 2, computing $\chi_0(H), \dots, \chi_{\ell}(H)$ takes $O(n)$ time. If H is a spider $(S \cup K \cup R, E)$ where $q(H[R]) \leq q$, by Lemma 3, computing $\chi_0(H), \dots, \chi_{\ell}(H)$ takes time $O(n)$. Finally, if H has a separable p-component H' such that $q(H - H') \leq q$, by Lemma 4 and an argument analogous to the one in the previous paragraph, computing $\chi_0(H), \dots, \chi_h(H)$ takes time $O(2^q q^{q+2} n)$, where $h = \min\{\ell, |V(H)|\}$ (in this case, $h = O(n)$ since only $|V(H')|$ is bounded by q). Consequently, as the decomposition tree has $O(n)$ nodes, the total running time is $O(n^3 + 2^q q^{q+2} n^2)$.

By (II), we can determine the cochromatic number of G in the same time by searching the minimum $\chi_{\ell}(G) + \ell$, for every $\ell \in \{0, \dots, n\}$.

4 Graphs with Bounded Treewidth

A tree decomposition of a graph G is a tuple (T, \mathcal{X}) , where T is a tree, \mathcal{X} contains a subset $X_t \subseteq V(G)$ for each node $t \in T$ and:

- (i) for each edge uv of G , there exists some $X_t \in \mathcal{X}$ containing u and v ; and
- (ii) $T[\{t : v \in X_t\}]$ is connected, for all $v \in V(G)$.

Observe from (ii) that, if X_i and X_j both contain a vertex v , then all nodes X_k of the tree in the (unique) path between X_i and X_j also contain v . That is, the nodes associated with vertex v form a connected subset of T . In other words, if X_k is on the unique path of T from X_i to X_j , then $X_i \cap X_j \subseteq X_k$.

The *treewidth* of (T, \mathcal{X}) is $\max\{|X_t| - 1 : t \in T, X_t \in \mathcal{X}\}$ and the *treewidth* $tw(G)$ of G is the minimum treewidth over all tree decompositions of G .

Consider T to be rooted in r . We say that (T, \mathcal{X}) is a *nice tree decomposition* if each $t \in V(T)$ is either a leaf, or t has exactly two children t_1 and t_2 with $X_t = X_{t_1} = X_{t_2}$, or t has exactly one child t' and either $X_t = X_{t'} \setminus \{x\}$ or $X_t = X_{t'} \cup \{x\}$, for some $x \in V(G)$. It is known that, for a fixed k , we can determine if the treewidth of a given graph G is at most k , and if so, find a nice tree decomposition (T, \mathcal{X}) of G with $O(|V(G)|)$ nodes and width at most k in linear time [4].

Let G be a graph with $tw(G) \leq w - 1$ and let (T, \mathcal{X}) be a nice tree decomposition of G . Given a node t of T , we denote by G_t the subgraph of G induced by $\bigcup_{t' \in V(T_t)} X_{t'}$, where T_t is the subtree of T rooted at t . We remark that

$z(G) \leq \chi(G) \leq w$; hence, we can consider only (k, ℓ) -cocoolorings for $0 \leq k \leq w$ and $0 \leq \ell \leq n$, since G is (w, ℓ) -cocoolorable for every ℓ .

For every node t of T , we will compute a table W_t with entries “yes” or “no” for every tuple (k, ℓ, ψ, f) , where $0 \leq k \leq w$ and $0 \leq \ell \leq n$ are integers such that $k + \ell \leq n$, ψ is a coloring of the vertices in X_t (not necessarily a proper coloring) and f is a function $f : \psi(X_t) \rightarrow \{s, c_0, c_1\}$ which associates a status “ s ” (stable color), “ c_0 ” (new clique) or “ c_1 ” (old clique) for every color of ψ .

The value $W_t(k, \ell, \psi, f)$ is “yes” if and only if there exists a (k, ℓ) -cocooloring π of G_t such that ψ is π restricted to X_t and, for each color b of ψ :

- if $f(b) = s$, then b is a stable color of π ;
- if $f(b) \in \{c_0, c_1\}$, then b is a clique color of π ;
- if $f(b) = c_0$, then $\{v \in G_t : \pi(v) = b\} \subseteq X_t$ (that is, the color b only appears in X_t . In other words, b is a new clique color).

In this case, we say that the *index* (k, ℓ, ψ, f) *agrees with* π .

Let t be a node of T and $h = (k, \ell, \psi, f)$ be an index of W_t . We say that h is valid if: (i) every color of ψ induces a clique or a stable set; (ii) there are at most k stable colors and at most ℓ clique colors; and (iii) $f(b) = s$ if and only if b is a stable color. Clearly, $W_t(h)$ is “no” if h is not a valid index. Also, if h agrees with a (k, ℓ) -cocooloring of G_t , then h is a valid index. Now, we determine the value of $W_t(h)$ depending on which type of node t is.

Lemma 5. *If t is a leaf node and $h = (k, \ell, \psi, f)$ is an index of W_t , then we can compute $W_t(h)$ in $O(w^2)$ time.*

Lemma 6. *Let t be a non-leaf node with child t' such that $X_t = X_{t'} \cup \{v\}$, $h = (k, \ell, \psi, f)$ be a valid index of W_t and $b = \psi(v)$. Also, let ψ' be ψ restricted to $X_{t'}$ and f' be f restricted to $\psi'(X_{t'})$. Then, $W_t(h)$ is “yes” if and only if one of the following holds:*

1. $b \in \psi'(X_{t'})$, $f(b) = s$ and $W_{t'}(k, \ell, \psi', f) = \text{“yes”}$; or
2. $b \in \psi'(X_{t'})$, $f(b) \neq s$ and $W_{t'}(k, \ell, \psi', f^*) = \text{“yes”}$, where $f^*(b') = f(b')$, for all $b' \neq b$, and $f^*(b) = c_0$; or
3. $b \notin \psi'(X_{t'})$, $f(b) = s$ and $W_{t'}(k - 1, \ell, \psi', f') = \text{“yes”}$; or
4. $b \notin \psi'(X_{t'})$, $f(b) \neq s$ and $W_{t'}(k, \ell - 1, \psi', f') = \text{“yes”}$.

Lemma 7. *Let t be a non-leaf node with child t' such that $X_t = X_{t'} \setminus \{v\}$ and $h = (k, \ell, \psi, f)$ be a valid index of W_t . Then, $W_t(h)$ is “yes” if and only if there exist extensions ψ' of ψ that colors $X_{t'}$ and $f' : \psi'(X_{t'}) \rightarrow \{s, c_0, c_1\}$ of f such that $W_{t'}(k, \ell, \psi', f')$ is “yes”.*

In order to present the next lemma, we need the following definitions. Let $f, f' : X \rightarrow \{s, c_0, c_1\}$. We say that f' is *compatible* with f if $f'(b) = f(b)$, for all $b \in X$ such that $f(b) \in \{s, c_0\}$, and $f'(b) \in \{c_0, c_1\}$, for all $b \in X$ such that $f(b) = c_1$. Now, let t be a non-leaf node with children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$ and let $h = (k, \ell, \psi, f)$ be a valid index of W_t . We denote by \mathcal{F}_h the set of tuples

(f_1, f_2) such that f_1 and f_2 are compatible with f and at most one of $f_1(b), f_2(b)$ is c_1 , for all $b \in \psi(X_t)$. Finally, we denote by \mathcal{L}_h the set of tuples (ℓ_1, ℓ_2) such that $\ell_1 + \ell_2 - |\{b \in \psi(X_t) : f(b) \neq s\}| = \ell$.

Lemma 8. *Let t be a non-leaf node with children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$ and $h = (k, \ell, \psi, f)$ be a valid index of W_t . Then, $W_t(h)$ is “yes” if and only if $W_{t_1}(k, \ell_1, \psi, f_1) = W_{t_2}(k, \ell_2, \psi, f_2) = \text{“yes”}$, for some $(\ell_1, \ell_2) \in \mathcal{L}_h$ and some $(f_1, f_2) \in \mathcal{F}_h$.*

Theorem 3. *Let G be a graph with n vertices and treewidth $\text{tw}(G) \leq w - 1$. Then, there exists an algorithm that solves Cocoloring Problem for G and any given k, ℓ in $O(2^w 3^w w^{w+1} n^3)$ time and $O(3^w w^{w+1} n^2)$ space. Moreover, the cochromatic number can be determined in $O(2^w 3^w w^{w+3} n)$ time and $O(3^w w^{w+2} n)$ space.*

Proof. Consider a treedecomposition (T, \mathcal{X}) of G . First, we want to know the complexity of computing $W_t(h)$, for some index h of W_t , for some node $t \in V(T)$. Suppose that h is valid, otherwise we know already that $W_t(h) = \text{“no”}$ (testing the validity of h takes time $O(w^2)$). If t is a leaf, we are done. If t has a child t' and $X_t = X_{t'} \cup \{v\}$, by Lemma 6, we need only to check one position of $W_{t'}$, which takes time $O(1)$. If t has a child t' and $X_t = X_{t'} \setminus \{v\}$, by Lemma 7, we need to check $3 \cdot w$ entries of $W_{t'}$, one for each combination of an extension ψ' of ψ and an extension of f related to ψ' . Now, suppose that t has two children t_1 and t_2 . By Lemma 8, it takes $O(2^w \cdot |\mathcal{L}_h|)$ time, since $|\mathcal{F}_h| = O(2^w)$. If we are given k and ℓ as an input, $|\mathcal{L}_h| = O(n)$. If we want to compute $z(G)$, since $z(G) \leq w$, we can consider only the values $\{1, \dots, w\}$ for ℓ and, consequently, $|\mathcal{L}_h| = O(w)$.

Now, we want to compute the size of the table of a node t . We know that the number of colorings ψ of X_t is $O(w^w)$ and the number of functions f is $O(3^w)$. Also, if k and ℓ are part of the input, we need to compute $W_t(k', \ell', \psi, f)$, for all values $k' \leq k \leq w$ and $\ell' \leq \ell \leq n$; thus, there are $O(wn)$ possible pairs of values k', ℓ' . If we want to compute the cochromatic number, we only need to investigate the values $\{1, \dots, w\}$ for ℓ , which gives $O(w^2)$ possible pairs of values k', ℓ' . Thus, the size of W_t is $O(3^w w^{w+1} n)$, if the problem being treated receives k and ℓ as input, and $O(3^w w^{w+2})$, if the problem being treated is to compute $z(G)$.

Finally, we combine the complexities. As remarked before, one can find a nice treedecomposition (T, \mathcal{X}) of G in $O(n)$ time; also, T has $O(n)$ nodes. So, we get that deciding if G is (k, ℓ) -cocolorable can be done in time $O(2^w 3^w w^{w+1} n^3)$ and space $O(3^w w^{w+1} n^2)$, while computing the cochromatic number of G can be done in time $O(2^w 3^w w^{w+3} n)$ and space $O(3^w w^{w+2} n)$.

References

1. Alon, N., Krivelevich, M., Sudakov, B.: Subgraphs with a large cochromatic number. *J. Graph Theory* 25, 295–297 (1997)
2. Babel, L., Kloks, T., Kratochvl, J., Kratsch, D., Muller, H., Olariu, S.: Efficient algorithms for graphs with few P4s. *Discrete Mathematics* 235(23), 29–51 (2001)

3. Babel, L., Olariu, S.: On the structure of graphs with few P_4 's. *Discrete Applied Mathematics* 84, 1–13 (1998)
4. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* 25, 1305–1317 (1996)
5. Bondy, A., Murty, U.S.R.: *Graph Theory*. Springer (2008)
6. Brandstädt, A.: Partitions of graphs into one or two independent sets and cliques. *Discrete Mathematics* 152, 47–54 (1996)
7. Bravo, R.S.F., Klein, S., Nogueira, L.T.: Characterizing (k, ℓ) -partitionable cographs. In: 7th International Colloquium on Graph Theory, Hyeres. *Electronic Notes in Discrete Mathematics* vol. 22, pp. 277–280 (2005)
8. Bravo, R., Klein, S., Nogueira, L., Protti, F.: Characterization and recognition of P_4 -sparse graphs partitionable into k independent sets and ℓ cliques. *Discrete Applied Mathematics* 159, 165–173 (2011)
9. Bravo, R., Klein, S., Nogueira, L., Protti, F., Sampaio, R.: Partitioning extended P_4 -laden graphs into cliques and stable sets (submitted, 2011)
10. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear Time Solvable Optimization Problems on Certain Graph Families. In: Hromkovič, J., Sýkora, O. (eds.) WG 1998. LNCS, vol. 1517, pp. 1–16. Springer, Heidelberg (1999)
11. Demange, M., Ekim, T., de Werra, D.: Partitioning cographs into cliques and stable sets. *Discrete Optimization* 2, 145–153 (2005)
12. Erdős, P., Gimbel, J.: Some problems and results in cochromatic theory. In: Quo Vadis, Graph Theory?, pp. 261–264. North-Holland, Amsterdam (1993)
13. Erdős, P., Gimbel, J., Kratsch, D.: Some extremal results in cochromatic and dichromatic theory. *J. Graph Theory* 15, 579–585 (1991)
14. Erdős, P., Gimbel, J., Straight, H.J.: Chromatic number versus cochromatic number in graphs with bounded clique number. *European J. Combin.* 11, 235–240 (1990)
15. Feder, T., Hell, P.: Matrix partitions of perfect graphs. *Discrete Mathematics* 306, 2450–2460 (2006)
16. Feder, T., Hell, P., Hochsttler, W.: Generalized colourings (matrix partitions) of cographs. In: Graph Theory in Paris. Trends in Mathematics, pp. 149–167 (2007)
17. Feder, T., Hell, P., Klein, S., Motwani, R.: Complexity of list partition. In: 31st Annual ACM Symposium on Theory of Computing, pp. 464–472. Plenum Press, New York (1999)
18. Feder, T., Hell, P., Klein, S., Motwani, R.: List partitions. *SIAM Journal on Discrete Mathematics* 16, 449–478 (2003)
19. Fomin, F.V., Kratsch, D., Novelli, J.C.: Approximating minimum ccolorings. *Information Processing Letters* 84, 285–290 (2002)
20. Heggernes, P., Kratsch, D., Lokshtanov, D., Raman, V., Saurabh, S.: Fixed-Parameter Algorithms for Cochromatic Number and Disjoint Rectangle Stabbing. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 334–345. Springer, Heidelberg (2010)
21. Hell, P., Klein, S., Nogueira, L.T., Protti, F.: Partitioning chordal graphs into independent sets and cliques. *Discrete Applied Mathematics* 141, 185–194 (2004)
22. Jamison, B., Olariu, S.: A tree representation for P_4 -sparse graphs. *Discrete Applied Mathematics* 35(2), 115–129 (1992)
23. Lesniak, L., Straight, H.J.: The cochromatic number of a graph. *Ars Combinatoria* 3, 39–46 (1977)
24. Wagner, K.: Monotonic coverings of finite sets. *Elektron. Inform. Kybernet.* 20, 633–639 (1984)

Parameterized Complexity of the Firefighter Problem

Cristina Bazgan^{1,2}, Morgan Chopin¹, and Michael R. Fellows^{3,*}

¹ Université Paris-Dauphine, LAMSADE, France

{bazgan, chopin}@lamsade.dauphine.fr

² Institut Universitaire de France

³ Charles Darwin University, Australia

michael.fellows@cdtu.edu.au

Abstract. In this paper we study the parameterized complexity of the firefighter problem. More precisely, we show that SAVING k -VERTICES and its dual SAVING ALL BUT k -VERTICES are both W[1]-hard for parameter k even for bipartite graphs. We also investigate several cases for which the firefighter problem is tractable. For instance, SAVING k -VERTICES is fixed-parameter tractable on planar graphs for parameter k . Moreover, we prove a lower bound to polynomial kernelization for SAVING ALL BUT k -VERTICES.

1 Introduction

The firefighter problem was introduced in [10] and can be used to model the spread of a fire, a virus, or an idea through a network. It is a dynamic problem defined as follows. Initially, a fire breaks out at some special vertex s of a graph. At each time step, we have to choose one vertex which will be protected by a firefighter. Then the fire spreads to all unprotected neighbors of the vertices on fire. The process ends when the fire can no longer spread, and then all vertices that are not on fire are considered as saved. The objective consists of choosing, at each time step, a vertex which will be protected by a firefighter such that a maximum number of vertices in the graph is saved at the end of the process.

The firefighter problem was proved to be NP-hard even for trees of maximum degree three [8] and cubic graphs [11]. From the approximation point of view, the firefighter problem is $\frac{e}{e-1}$ -approximable on trees [3] and it is not $n^{1-\varepsilon}$ -approximable on general graphs [1], if P \neq NP. However, very little is known about the fixed parameter tractability of this problem. In [3], the authors give fixed-parameter tractable algorithms and polynomial kernels on trees for each of the following parameters: the number of saved leaves, the number of burned vertices, and the number of protected vertices.

In this paper, we consider the parameterized complexity of the firefighter problem on general graphs where, at each time step, b firefighters can be deployed. This problem, called SAVING k -VERTICES, is defined as follows. Given

* Supported by the Australian Research Council.

a graph $G = (V, E)$, an initially burned vertex $s \in V$, and two integers $b \geq 1$ and $k \geq 0$, can we protect b vertices at each time step such that at least k vertices are saved at the end of the process? The dual problem, SAVING ALL BUT k -VERTICES, is also studied and asks whether we can protect b vertices at each time step such that at most k vertices are burned at the end of the process. We show that SAVING k -VERTICES is W[1]-hard for parameter k when b is fixed. In contrast, SAVING ALL BUT k -VERTICES is fixed-parameter tractable for parameter k when b is fixed and it is W[1]-hard for parameter k when b is part of the input. SAVING k -VERTICES is proved to be fixed-parameter tractable when parameterized by k and the treewidth of the graph or when restricted to planar graphs and parameterized by k . Both problems, SAVING ALL BUT k -VERTICES and SAVING ALL BUT k -VERTICES, admit a kernel when parameterized by k and the vertex cover number. We also show that SAVING ALL BUT k -VERTICES parameterized by k and b does not admit polynomial kernels unless coNP ⊆ NP/poly. Our results are summarized in Table 1.

Our paper is organized as follows. Definitions, terminology and preliminaries are given in Section 2. In Section 3, we give several parameterized tractability results, and polynomial kernelization feasibility is studied in Section 4. Conclusions are given in Section 5. Due to the space limit, some proofs are omitted and could be found in the extended version.

Table 1. Summary of results. The vertex cover number is denoted by τ , and the treewidth by tw . Results in bold font are proved in this paper; results in italic are a direct consequence of these last results. Notice that vertex cover number is larger than treewidth.

	Parameter(s)	k	k + b	k + tw	k + τ
SAVING k -VERTICES	Tractability	<i>W[1]-hard, XP</i>	W[1]-hard, XP	FPT	FPT
	Poly Kernel?	FPT for planar graphs	<i>no</i>	<i>no</i>	<i>open</i>
SAVING ALL BUT k -VERTICES	Tractability	W[1]-hard, XP	FPT	<i>open</i>	FPT
	Poly Kernel?	<i>no</i>	no	<i>no</i>	<i>open</i>

2 Preliminaries

Graph terminology. All graphs in this paper are undirected, connected, finite and simple. Let $G = (V, E)$ be a graph. An edge in E between vertices $u, v \in V$ will be denoted by uv . The *degree* of a vertex $u \in V$, denoted by $\deg(u)$, is the number of edges incident to u . The *open* (resp. *close*) *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u \in V : uv \in E\}$ (resp. $N[v] = N(v) \cup \{v\}$). Given a subset $S \subseteq V$, the open (resp. close) neighborhood of S is the set $N(S) = \bigcup_{u \in S} N(u)$ (resp. $N[S] = N(S) \cup S$). We denote by $d(u, v)$ the minimum length of a path with endpoints $u, v \in V$. Throughout this paper, the vertex cover number will be denoted by $\tau(G)$ or τ and the treewidth by $tw(G)$ or tw .

Parameterized complexity. Here we only give the basics notions on parameterized complexity, for more background the reader is referred to [60]. The parameterized complexity is a framework which provides a new way to express the

computational complexity of problems. A problem parameterized by k is called *fixed-parameter tractable* (fpt) if there exists an algorithm (called an fpt algorithm) that solves it in time $f(k).n^{O(1)}$ (fpt-time). The function f is typically super-polynomial and only depends on k . In other words, the combinatorial explosion is confined into f . A parameterized problem P with parameter k will be denoted by (P, k) . The XP class is the set of parameterized problems (P, k) that can be solved in time $n^{g(k)}$ for a given computable function g .

One of the main tools to design such algorithms is the *kernelization*. A kernelization algorithm transforms in polynomial time an instance I of a given problem parameterized by k into an equivalent instance I' of the same problem parameterized by $k' \leq k$ such that $|I'| \leq g(k)$ for some computable function g . The instance I' is called a *kernel* of size $g(k)$ (if g is a polynomial then I' is a *polynomial kernel*). By applying any (even exponential) algorithm to a kernel of a given problem, we can derive an fpt algorithm for that problem.

Conversely to the previous approach, we can prove the parameterized intractability of a problem. To this end, we need to introduce the notion of *parameterized reduction*. An fpt-reduction is an algorithm that reduces any instance I of a problem with parameter k to an equivalent instance I' with parameter $k' = g(k)$ in fpt-time for some function g . The basic class of parameterized intractability is $W[1]$ and there is a good reason to believe that $W[1]$ -hard problems (according to the fpt-reduction) are unlikely to be FPT. We have the following inclusions $\text{FPT} \subseteq W[1] \subseteq \text{XP}$.

Very recently a new result [2] has been introduced for proving the non existence of a polynomial kernel under a reasonable complexity hypothesis. This result is based on the notion of *OR-composition* of parameterized problems. A problem P parameterized by k is OR-compositional if there exists a polynomial algorithm that receives as inputs a finite sequence $(I_1, k_1), \dots, (I_N, k_N)$ of instances of (P, k) such that $k_1 = \dots = k_N$. The algorithm is required to output an instance (I, k) of (P, k) such that $k = k_1^{O(1)}$ and (I, k) is a yes-instance if and only if there is some $i \in \{1, \dots, N\}$ such that (I_i, k_i) is a yes-instance. We have the following theorem.

Theorem 1. [2] *Let (P, k) be a parameterized problem. If P is NP-complete and (P, k) is OR-compositional then (P, k) has no polynomial kernel unless $\text{coNP} \subseteq \text{NP/poly}$.*

Problems definition. In order to define the firefighter problem, we use an undirected graph $G = (V, E)$ and notations of [1]. Each vertex in the graph can be in exactly one of the following states: *burned*, *saved* or *vulnerable*. A vertex is said to be burned if it is on fire. We call a vertex saved if it is either protected by a firefighter — that is the vertex cannot be burned in subsequent time steps — or if all paths from any burned vertex to it contains at least one protected vertex. Any vertex which is neither saved nor burned is called vulnerable. At time step $t = 0$, all vertices are vulnerable, except vertex s , which is burned. At each time $t > 0$, at most b vertices can be protected by firefighters and any vulnerable vertex v which is adjacent to a burned vertex u becomes burned at time $t + 1$,

unless it is protected at time step t . Burned and saved vertices remain burned and saved, respectively.

Given a graph $G = (V, E)$ and a vertex s initially on fire, a *protection strategy* is a set $\Phi \subseteq V \times T$ where $T = \{1, 2, \dots, |V|\}$. We say that a vertex v is protected at time $t \in T$ according to the protection strategy Φ if $(v, t) \in \Phi$. A protection strategy is *valid* with respect to a budget b , if the following two conditions are satisfied:

1. if $(v, t) \in \Phi$ then v is not burned at time t ;
2. let $\Phi_t = \{(v, t) \in \Phi\}$; then $|\Phi_t| \leq b$ for $t = 1, \dots, |V|$.

Thus at each time $t > 0$, if a vulnerable vertex v is adjacent to at least one burned vertex and $(v, t) \notin \Phi$, then v gets burned at time $t + 1$. We now define in the following the problems we study.

SAVING k -VERTICES

Input: A graph $G = (V, E)$, a burned vertex $s \in V$ and two integers k and b .

Question: Is there a valid strategy Φ with respect to budget b that saves at least k vertices?

SAVING ALL BUT k -VERTICES

Input: A graph $G = (V, E)$, a burned vertex $s \in V$ and two integers k and b .

Question: Is there a valid strategy Φ with respect to budget b where at most k vertices are burned?

Remark 1. For the SAVING k -VERTICES problem, we may only consider instances for which any valid strategy $\Phi \subseteq V \times T$ is such that $|\Phi| < k$, otherwise the answer is clearly *yes*.

Remark 2. For the SAVING ALL BUT k -VERTICES problem, we may assume that for any valid strategy $\Phi \subseteq V \times T$, if $(v, t) \in \Phi$ then $t < k$, otherwise the answer is necessarily *no*. Indeed, there is at least one newly burned vertex at each time step, then if we protect a vertex at time step $t \geq k$ there will be at least k burned vertices.

3 Parameterized Tractability

We first show that SAVING k -VERTICES and its dual SAVING ALL BUT k -VERTICES are both in XP but are fixed-parameter intractable even for bipartite graphs.

Theorem 2. SAVING k -VERTICES is solvable in time $n^{O(k)}$.

Proof. It follows from Remark 2 that the algorithm only have to try each of the $\binom{n}{k}$ possible sets of protected vertices. Notice that all of these configurations does not necessarily correspond to a valid strategy. However, one can check if a set $D \subseteq V$ corresponds to a valid strategy with the following procedure. Let r_i be the number of firefighters we did not use from time step 1 to $i - 1$ ($r_1 = 0$),

and $L_i = \{v \in V : d(s, v) = i\}$. For each time step $i = 1, \dots, k$, protect vertices in $D \cap L_i$. If $|D \cap L_i| > b + r_i$ then this strategy is not valid. Otherwise, set $r_{i+1} = r_i + (b - |D \cap L_i|)$. It follows that the running time is $n^{O(k)}$. \square

Theorem 3. SAVING ALL BUT k -VERTICES is solvable in time $n^{O(k)}$.

Proof. First of all, we need to introduce the notion of *valid burning set*. Given a graph $G = (V, E)$ and an initially burned vertex $s \in V$, a valid burning set is a subset $B \subseteq V$ with $s \in B$ such that there exists a valid strategy for which, at the end of the process, the burned vertices are exactly those in B . We have the following lemma.

Lemma 1. Let $G = (V, E)$ be a graph with an initially burned vertex $s \in V$. Verifying if a subset $B \subseteq V$ is a valid burning set can be done in linear time.

Proof. It follows from Remark 2 that we only have to consider the first k time steps. Notice that the set of protected vertices must be exactly $N(B)$. Moreover, given a vertex $v \in N(B)$ this vertex has a *due date*: it has to be protected before or at time step $d(s, v)$. Hence, at each time step $t = 1, \dots, k$, it suffices to protect all the vertices in $N(B)$ with due dates equal to t . If there are more firefighters than vertices with due date equal to t then protect vertices with due date equal to $t + 1$, then vertices with due date equal to $t + 2$ and so on until there are no more firefighters. Clearly, if all vertices in $N(B)$ are protected using the previous procedure then the answer is *yes*. However, if, at a given time step t , there are less firefighters than vertices with due date equal to t the answer is *no*. \square

The algorithm then proceeds as follows. For each subset $B \subseteq V$ with $|B| = k + 1$ if B is a valid burning set then the answer is *yes*. If no valid burning set were found then the answer is *no*. It follows from lemma 1 that the running time is $n^{O(k)}$. \square

The following result was also proved independently by Cygan et al. [5].

Theorem 4. SAVING k -VERTICES parameterized by k is W[1]-hard even for bipartite graphs with a fixed budget.

Proof. We construct an fpt-reduction from the W[1]-hard problem MULTI-COLORED CLIQUE [12] to SAVING k -VERTICES. We first recall the definition of the former problem.

MULTI-COLORED CLIQUE

Input: A graph $G = (V, E)$, an integer k , and a proper k -coloring of G (*i.e.*, every two adjacent vertices have different colors).

Parameter: k

Question: Is there a k -clique (*i.e.*, a complete subgraph on k vertices) in G ?

Let I be an instance of MULTI-COLORED CLIQUE consisting of a graph $G = (V, E)$, an integer k , and a proper k -coloring. We construct an instance I' of SAVING k -VERTICES consisting of a graph $G' = (V', E')$, a burned vertex $s \in V'$,

and two integers k' and b as follows. Set $k' = \binom{k}{2} + k$ and $b = 1$. We construct G' from G as follows: add a new vertex s ; add k copies of V denoted as V_1, \dots, V_k ; join s to every vertex of V_1 ; for $i = 1, \dots, k$, join every vertex of V_i to every vertex of V_{i+1} ; join every vertex of V_k to every vertex of V ; remove every edge $uv \in E$ and add an edge-vertex x_{uv} adjacent to u and v where $u, v \in V$.

Suppose that there is a clique $C \subseteq V$ of size k in G . Let Φ be the following valid strategy for I' : at each time step $i = 1, \dots, k$, protect a vertex $v \in V$ of G' such that $v \in C$. At time step $k+1$, Φ protects any non-burned vertex. Clearly, Φ saves at least k vertices and $\binom{k}{2}$ edge-vertices.

Conversely, suppose that a valid strategy Φ saves at least k' vertices in G' . We may assume that Φ protects no vertex in $V_c = V_1 \cup \dots \cup V_k$. Indeed, suppose that Φ protects vertices in V_c . Notice that, at time step k , all vertices in V_c are burned except the protected ones. Consider now the strategy Φ' obtained from Φ that protects vertices in $V' \setminus V_c$ instead of those in V_c . Thus, Φ' saves at least the same number of vertices. Moreover, we may assume that Φ' protects no edge-vertex x_{uv} , otherwise we could protect either u or v instead of x_{uv} in order to save at least the same number of vertices. It follows that Φ' protects exactly k vertices of V in G' . Since we save at least $\binom{k}{2} + k$, the corresponding set C is a clique in G of size k . \square

Theorem 5. SAVING ALL BUT k -VERTICES parameterized by k is W[1]-hard even for bipartite graphs.

Proof. We construct an fpt-reduction from the W[1]-hard problem REGULAR CLIQUE [12] to SAVING ALL BUT k -VERTICES. We first give the definition of the former problem.

REGULAR CLIQUE

Input: A regular graph $G = (V, E)$ and an integer k .

Parameter: k

Question: Is there a k -clique in G ?

In this reduction, a *busy* gadget denotes a $(b+k)$ -star with center c (*i.e.*, a tree with one internal vertex c and $b+k$ leaves). Attaching a busy gadget to a vertex v means to create a copy of a $(b+k)$ -star and makes c adjacent to v . Thus, if v is burning at a given time step then c has to be protected, otherwise more than k vertices would burn.

Let I be an instance of REGULAR CLIQUE consisting of a d -regular graph $G = (V, E)$ and an integer k . We construct an instance I' of SAVING ALL BUT k -VERTICES consisting of a graph $G' = (V', E')$, a burned vertex $s \in V'$, and two integers k' and b as follows. Set $k' = k$ and $b = b_1 + b_2$ where $b_1 = k(n - k)$ and $b_2 = kd - \binom{k}{2}$. We construct G' from G as follows: add a new vertex s adjacent to all vertices of G ; attach $b_1 + b_2 - (n - k)$ busy gadgets to s ; attach $n - k$ busy gadgets to every vertex in V ; remove every edge $uv \in E$ and add an edge-vertex x_{uv} adjacent to u and v (see Figure 11). Notice that, at time step 1, there are only $n - k$ firefighters that can be placed freely because of the busy gadgets.

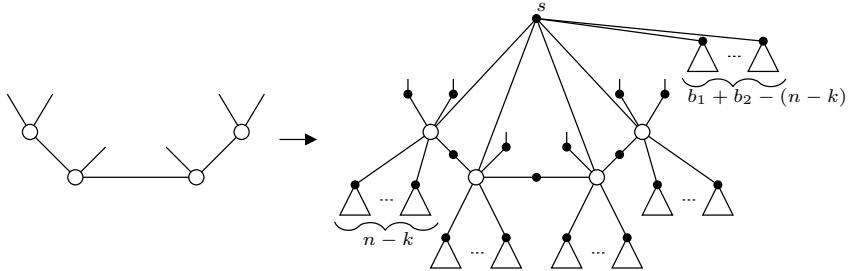


Fig. 1. The construction of G' . Added vertices are black and a triangle represents a busy gadget.

Suppose that we have a k -clique in G . Let Φ be the following valid strategy for I' : at time step 1, the strategy uses the $n - k$ remaining firefighters to protect all the original vertices V in G' except those in the k -clique. At time step 2, all the k vertices in the clique are burned. Since there are $n - k$ busy gadgets attached to each vertex in the k -clique we need to protect $b_1 = k(n - k)$ vertices. Moreover, there are $kd - \binom{k}{2}$ edge-vertices adjacent to the vertices in the clique, since it remains $b_2 = b - b_1$ firefighters we can protect them all. Hence, no more than k vertices are burned at the end of the process.

Conversely, suppose that there is no k -clique in G . A time step 1, any valid strategy has to place the $n - k$ remaining firefighters on vertices that are not edge-vertices otherwise at least $k' + 1$ vertices will burn. At time step 2, since there is no k -clique, there will be at least $kd - \binom{k}{2} + 1$ edge-vertices adjacent to the k burned vertices. For the same reason as before, it remains $b_2 = b - b_1$ firefighters which is not enough to protect these edge-vertices. Therefore, given any valid strategy there will be at least k' burned vertices. \square

We note that the parameters in the reduction used in Theorem 5 are linearly related. Since REGULAR CLIQUE cannot be solved in time $n^{o(k)}$ unless FPT = M[1] [12], we obtain the following lower bound that shows that the algorithm given in Theorem 3 is optimal.

Corollary 1. SAVING ALL BUT k -VERTICES *cannot be solved in time $n^{o(k)}$ unless FPT = M[1]*.

The following results show that SAVING k -VERTICES and SAVING ALL BUT k -VERTICES are fixed-parameter tractable in several cases.

Theorem 6. SAVING ALL BUT k -VERTICES parameterized by k and the budget b is FPT.

Proof. We describe a recursive algorithm that solves SAVING ALL BUT k -VERTICES in time $O^*(2^{k^2(b+1)+kb})$. Let $G = (V, E)$ be a graph with an initially burned vertex $s_0 \in V$. We may assume that $|N(s_0)| < b + k$ otherwise the answer is no. Since a solution of the problem will protect none or up to b vertices

of $N(s_0)$, the algorithm must decide which of the $\sum_{i=0}^b \binom{b+k}{i}$ possible subsets of $N(s_0)$ to protect at time step 1. At time step 2, the newly burned vertices are exactly those in $N(s_0)$ that were not protected. Notice that we can merge every burned vertex into a single burned vertex s_1 without changing the answer of the instance. We may also assume that $|N(s_1)| < b + k + r$ where r is the number of firefighters we did not use in the previous time step. The algorithm has now $\sum_{i=0}^{b+r} \binom{b+k+r}{i}$ possible subsets to protect at time step 2. Clearly, we may apply the previous procedure recursively. Moreover, it follows from Remark 2 that there are at most k recursive calls. The value of r is then at most bk , and the running time of the algorithm is $O^*(2^{k(b+k+r)}) = O^*(2^{k^2(b+1)+kb})$. \square

We use the Monadic Second Order Logic formulation of SAVING k -VERTICES and theorems from [4] and [7] to prove the following theorems. These were also proved independently in [5].

Theorem 7. SAVING k -VERTICES parameterized by k is FPT for planar graphs.

Theorem 8. SAVING k -VERTICES parameterized by the treewidth and k is FPT.

4 Kernelization Feasibility

In this section, we provide a kernelization for SAVING k -VERTICES (resp. SAVING ALL k -VERTICES) when parameterized by τ and k . Moreover, we show that SAVING ALL BUT k -VERTICES parameterized by k and b does not admit a polynomial kernel unless $\text{coNP} \subseteq \text{NP/poly}$.

Theorem 9. SAVING k -VERTICES admits a kernel of size at most $O(2^\tau k)$.

Proof. Let $G = (V, E)$ be a graph where the fire breaks out at vertex $s \in V$ and there are b firefighters available at each time step. A set $S \subseteq V$ is called a *twins set* if for every $v, u \in S$, $v \neq u$, we have $N(u) = N(v)$ and $uv \notin E$. Consider the following reduction rule.

Rule: If there exists a twins set S such that $|S| \geq k + 1$ then delete $|S| - k$ vertices of S .

Let $G' = (V', E')$ be the graph obtained by iteratively applying the above rule to every twins set in G . Notice that the procedure runs in polynomial time. Let $C \subseteq V'$ be a minimum vertex cover and let $D = V' \setminus C$ be an independent set. The number of distinct twins set in D is at most 2^τ (one for each subset in C). Moreover, each twins set in G' has at most k vertices. Therefore, the size of the reduced instance is at most $O(2^\tau k)$.

Correctness of the rule: Suppose that there exists a valid strategy Φ that saves at least k vertices in G . We have the following observation.

Observation 1. Let $G = (V, E)$ be a graph, $S \subseteq V$ be a twins set, and Φ be a valid strategy with respect to budget b that saves at least k vertices. If Φ protects a subset $S_1 \subseteq S$ then protecting any subset $S_2 \subseteq S$ instead of S_1 such that $|S_2| = |S_1|$ leads to a valid strategy Φ' that saves exactly the same number of vertices.

It follows from Observation 10 that if Φ protects a vertex in a twins set that has been deleted by the reduction rule then we can protect instead any other non-deleted vertex in the same twins set. Moreover, it follows from Remark 11 that Φ protects no more than k vertices in G . Since there are k vertices in any twins set in G' , we can always apply Observation 11. Hence there is a strategy Φ' for the reduced instance that saves at least k vertices in G' . Conversely, if a strategy saves at least k vertices in G' then this strategy clearly saves at least k vertices in G . \square

Theorem 10. SAVING ALL BUT k -VERTICES admits a kernel of size at most $O(2^\tau k\tau)$.

Proof. First we may assume that $b < \tau$, otherwise it suffices to protect all the vertices in the vertex cover at time step 1 to stop the fire. The reduction is the same as the one describes in Theorem 9 but we use the following slightly different reduction rule.

Rule: Let $S \subseteq V$ be a twins set. If $|S| \geq kb$ then delete $|S| - kb$ vertices of S . Similarly to the proof of Theorem 9, the size of the kernel is $O(2^\tau kb) = O(2^\tau k\tau)$.
Correctness of the rule: It follows from Remark 2 that there are at most kb protected vertices in any twins set at the end of the process. Using the same argument as in Theorem 9, the result follows. \square

Theorem 11. SAVING ALL BUT k -VERTICES parameterized by k and the budget b has no polynomial kernel unless $\text{coNP} \subseteq \text{NP/poly}$.

Proof. We show that SAVING ALL BUT k -VERTICES is OR-compositional. Let $I_1 = (G_1, s_1, k_1, b_1), \dots, I_N = (G_N, s_N, k_N, b_N)$ be a sequence of SAVING ALL BUT k -VERTICES instances with $k_1 = \dots = k_N$ and $b_1 = \dots = b_N$. First we may assume that $N < 2^{p(b_1, k_1)}$ where $p(b_1, k_1) = k_1^2(b_1 + 1) + k_1 b_1$, otherwise we could apply the fpt algorithm of Theorem 6 to each input instance. The output instance is the first input instance for which the fpt algorithm return *yes* if such instance exists, and the last instance otherwise. Clearly, the output instance is *yes* if and only if there exists a *yes* instance in the input sequence. Moreover, the procedure runs in time $O(N \cdot 2^{p(b_1, k_1)} n) = O(N^2 n)$.

When $N < 2^{p(b_1, k_1)}$ we construct the output instance $I = (G, s, k, b)$ as follows. Set $k = k_1 + \lceil \log_2 N \rceil$ and $b = b_1$. Build a perfect binary tree (*i.e.*, a tree in which every vertex other than the leaves has two children and all leaves are at the same distance from the root) of height $\lceil \log_2 N \rceil$. For each $i = 1, \dots, N$, identify a leave of the tree with vertex s_i . Identify remaining leaves with a copy of s_N . Attach $b - 1$ busy gadgets to every non-leaf vertex of the tree. By construction, the only burned vertices from time step 1 to $\lceil \log_2 N \rceil$ are on a path from s to some s_i . It follows that there exists a yes-instance I_i for some i if and only if there exists a valid strategy for I such that no more than $k = k_i + \lceil \log_2 N \rceil$ vertices are burned at the end of the process. Since $k = O(k_1 + p(b_1, k_1))$ and $b = b_1$ it follows that SAVING ALL BUT k -VERTICES is OR-compositional. Using Theorem 1 and the NP-completeness of SAVING ALL BUT k -VERTICES the result follows. \square

5 Conclusion

In this paper, we study the parameterized complexity of the firefighter problem on general graphs when more than one firefighter is available at each time step. We establish some tractable and intractable cases and study the existence of a polynomial kernel. Several interesting questions remain open. Does SAVING k -VERTICES or SAVING ALL BUT k -VERTICES admit a polynomial kernel for parameters τ and k ? Is SAVING ALL BUT k -VERTICES fixed-parameter tractable when parameterized by tw and k ?

References

1. Anshelevich, E., Chakrabarty, D., Hate, A., Swamy, C.: Approximation Algorithms for the Firefighter Problem: Cuts Over Time and Submodularity. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 974–983. Springer, Heidelberg (2009)
2. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On Problems without Polynomial Kernels. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 563–574. Springer, Heidelberg (2008)
3. Cai, L., Verbin, E., Yang, L.: Firefighting on Trees $(1 - 1/e)$ -Approximation, Fixed Parameter Tractability and a Subexponential Algorithm. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 258–269. Springer, Heidelberg (2008)
4. Courcelle, B.: The monadic second-order logic of graphs. III. tree-decompositions, minors and complexity issues. RAIRO Informatique Théorique Appliquée 26(3), 257–286 (1992)
5. Cygan, M., Fomin, F., Van Leeuwen, E.J.: Parameterized complexity of firefighting revisited. In: Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC 2011) (to appear, 2011)
6. Downey, R.G., Fellows, M.R.: Parameterized complexity. Springer, Heidelberg (1999)
7. Fellows, M.R., Szeider, S., Wrightson, G.: On finding short resolution refutations and small unsatisfiable subsets. Theoretical Computer Science 351, 351–359 (2006)
8. Finbow, S., King, A., MacGillivray, G., Rizzi, R.: The firefighter problem for graphs of maximum degree three. Discrete Mathematics 307(16), 2094–2105 (2007)
9. Flum, J., Grohe, M.: Parameterized complexity theory. Springer, Heidelberg (2006)
10. Hartnell, B.: Firefighter! an application of domination, Presentation. In: 10th Conference on Numerical Mathematics and Computing. University of Manitoba in Winnipeg, Canada (1995)
11. King, A., MacGillivray, G.: The firefighter problem for cubic graphs. Discrete Mathematics 310(3), 614–621 (2010)
12. Mathieson, L., Szeider, S.: Parameterized Graph Editing with Chosen Vertex Degrees. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 13–22. Springer, Heidelberg (2008)

Faster Approximate Pattern Matching in Compressed Repetitive Texts

Travis Gagie¹, Paweł Gawrychowski², and Simon J. Puglisi^{3,*}

¹ Department of Computer Science,
Aalto University,
Espoo, Finland
`travis.gagie@aalto.fi`

² Department of Computer Science,
University of Wrocław,
Wrocław, Poland
`gawry@cs.uni.wroc.pl`

³ Department of Informatics,
King's College London,
London, United Kingdom
`simon.puglisi@kcl.ac.uk`

Abstract. Motivated by the imminent growth of massive, highly redundant genomic databases we study the problem of compressing a string database while simultaneously supporting fast random access, substring extraction and pattern matching to the underlying string(s).

Bille et al. (2011) recently showed how, given a straight-line program with r rules for a string s of length n , we can build an $\mathcal{O}(r)$ -word data structure that allows us to extract any substring $s[i..j]$ in $\mathcal{O}(\log n + j - i)$ time. They also showed how, given a pattern p of length m and an edit distance $k \leq m$, their data structure supports finding all occ approximate matches to p in s in $\mathcal{O}(r(\min(mk, k^4 + m) + \log n) + \text{occ})$ time. Rytter (2003) and Charikar et al. (2005) showed that r is always at least the number z of phrases in the LZ77 parse of s , and gave algorithms for building straight-line programs with $\mathcal{O}(z \log n)$ rules. In this paper we give a simple $\mathcal{O}(z \log n)$ -word data structure that takes the same time for substring extraction but only $\mathcal{O}(z(\min(mk, k^4 + m)) + \text{occ})$ time for approximate pattern matching.

1 Introduction

The recent revolution in high-throughput sequencing technology has made the acquisition of large genomic sequences drastically cheaper and faster. As the new technology takes hold, ambitious sequencing projects such as the 1,000 Human Genomes [14] and the 10,000 Vertebrate Genomes [7] projects are set to create large databases of strings (genomes) that vary only slightly from each other,

* Supported by a Newton Fellowship.

and so will contain large numbers of long repetitions. Efficient storage of these collections is not enough: fast access to enable search and sequence alignment is paramount. The utility of such a data structure is not limited to treatment of DNA collections. Ferragina and Manzini's recent study of the compressibility of web pages reveals enormous redundancy in web crawls [5]. Exploiting this redundancy to reduce space while simultaneously enabling fast access and search over crawled pages (for snippet generation or cached page retrieval) is a significant challenge.

The problem of compressing and indexing such highly repetitive strings (or string collections) was introduced in [6] (see also [12]). Given a repetitive string s of length n with an LZ78- or BWT-based data structure [14], we can store s in space bounded in terms of the t th-order empirical entropy [13], for any $t = o(\log_\sigma n)$, and later extract any substring of length m in $\mathcal{O}(m/\log_\sigma n)$ time. For very repetitive texts, however, compression based on the LZ77 [18] can use significantly fewer than $nH_t(s)$ bits [6].

For example, Rytter [15] showed that the number z of phrases in the LZ77 parse of s is at most proportional to the number of rules in the smallest straight-line program (SLP) for s ¹. He then showed how the LZ77 parse can be turned into an SLP for s with $\mathcal{O}(z \log n)$ rules whose parse-tree has height $\mathcal{O}(\log n)$. This SLP can be viewed as a data structure that stores s in $\mathcal{O}(z \log^2 n)$ bits and supports substring extraction in $\mathcal{O}(\log n + m)$ time. Bille, Landau, Raman, Rao, Sadakane and Weimann [2] recently showed how, given an SLP for s with r rules, we can build a data structure that takes $\mathcal{O}(r \log n)$ bits and supports substring extraction in $\mathcal{O}(\log n + m)$ time. Unfortunately, since no polynomial-time algorithm is known to produce an SLP for s with $o(z \log n)$ rules, we still do not know how to efficiently build a data structure that has better bounds than Rytter's. Bille et al. [2] also show how, given a pattern p of length m and an edit distance $k \leq m$, their data structure supports finding all occ approximate matches to p in s in $\mathcal{O}(r(\min(mk, k^4 + m) + \log n) + \text{occ})$ time. Unfortunately, neither Rytter's nor Bille et al.'s data structures are practical.

In another strand of recent work Kreft and Navarro [9][10] introduced a variant of LZ77 called LZ-End and gave a data structure based on it with which we can store s in $\mathcal{O}(z' \log n) + o(n)$ bits, where z' is the number of phrases in the LZ-End parse of s , and later extract any *phrase* (not arbitrary substring) in time proportional to its length. The $o(n)$ term can be removed at the cost of slowing extraction down by an $\mathcal{O}(\log n)$ factor. Extracting arbitrary substrings is fast in practice but could be slow in the worst case. Also, although the LZ-End encoding is small in practice for very repetitive strings, it is not clear whether z' can be bounded in terms of z .

Our Contribution. In this paper we describe a simple $\mathcal{O}(z \log n)$ -word data structure, which we call the *block graph* for s , that takes $\mathcal{O}(\log n + \ell - f)$ time to extract a substring $s[f..l]$, and allows us to find all occ approximate matches

¹ In this paper we consider only the version of LZ77 without self-referencing, sometimes called LZSS [17].

of a pattern of length m in $\mathcal{O}(z(\min(mk + m, k^4 + m)) + \text{occ})$ time. Our space bound and substring extraction time are essentially the same as Bille et al. [2], and our approximate pattern matching time is faster by a factor of $\log n$; importantly, however, our results require much simpler machinery. We believe the block graph is the first practical data structure with solid theoretical guarantees for compression and retrieval of highly repetitive collections.

In the next section we describe the block graph, a data structure built from a string, on which our compressed representation is based. Then, in Section 3 we relate the size of the block graph to the size of the LZ77 parsing of its underlying string. We show that a block graph naturally compresses the string while allowing efficient random access and extraction of substrings. In Section 4 we show how to augment the block graph to support fast approximate pattern matching. We close by outlining some avenues future work might take.

2 Block Graphs

For the moment, assume $n = 2^h$ for some integer h . We start building the block graph of s with node $\langle 1..n \rangle$, which we call the root and consider to be at depth 0. For $0 \leq d < t$, for each node $v = \langle i..i + b - 1 \rangle$ at depth d , where $b = 2^{t-d}$ is the block size at depth d , we add pointers from v to nodes $\langle i..i + b/2 - 1 \rangle$, $\langle i + b/4..i + 3b/4 - 1 \rangle$ and $\langle i + b/2..i + b - 1 \rangle$, creating those nodes if necessary. We call these three nodes the children of v and each other's siblings, and we call v their parent. Notice that a node can have two parents. We associate with each node $\langle i..j \rangle$ the block $s[i..j]$ of characters in s . If n is not a power of 2, then we append blanks to s until it is. After building the block graph, we remove any nodes whose blocks contain only blanks or blanks and characters in another block at the same depth, and replace any node $\langle i..j \rangle$ with $j > n$ by $\langle i..n \rangle$. We delete all pointers to any such nodes.

We can reduce the size of the block graph by truncating it such that we keep only the nodes at depths where storing three pointers takes less space than storing a block of characters explicitly. We mark as an internal node each node whose block is the first occurrence of that substring in s . At the deepest internal nodes, instead of storing a pointer, we store the nodes' blocks explicitly. We mark as a leaf all nodes whose block is not unique and whose parents are internal nodes. We then remove any node that is not marked as an internal node or a leaf. Figure 1 shows the block graph for the eighth Fibonacci string, abaababaabaababa, truncated at depth 3. Oval nodes are internal nodes and rectangular nodes are leaves. Notice that the root has only two children, because the block for node $\langle 17..32 \rangle$ would contain only blanks and characters in $s[9..21]$, so $\langle 17..32 \rangle$ is removed; similarly, $\langle 21..24 \rangle$ is removed.

The key phase in building the block graph is updating the leaves' pointers, shown in Figure 1 as the arrows below rectangular nodes. Suppose a leaf u at depth d had a child $\langle i..j \rangle$, which was been removed because it was neither an internal node nor a leaf. Consider the first occurrence $s[i'..j']$ in s of the substring $s[i..j]$. Notice that $s[i'..j']$ is completely contained within some

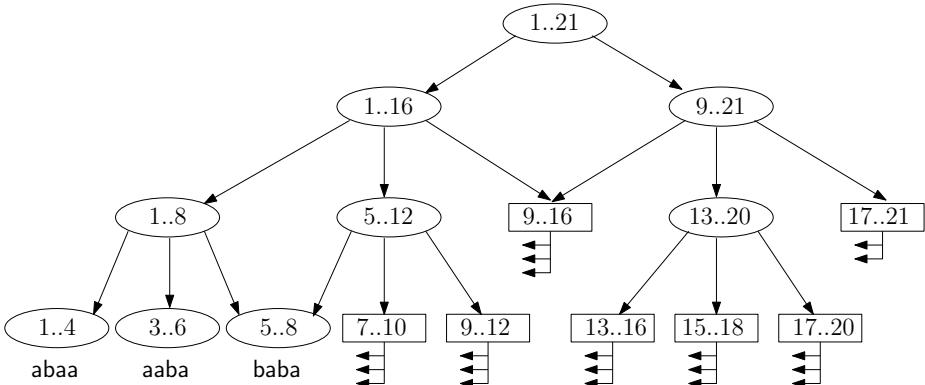


Fig. 1. The block graph for the eighth Fibonacci string, abaababaabaababaababa, truncated at depth 3

block at depth d — this is one reason why we use overlapping blocks — and, since $s[i'..j']$ is the first occurrence of that substring in s , that block is associated with an internal node v . We replace the pointer from u to $\langle i..j \rangle$ by a pointer to v and the offset of i' in v 's block. For the example shown in Figure 1, $\langle 17..21 \rangle$ previously had children $\langle 17..20 \rangle$ and $\langle 19..21 \rangle$. The blocks $s[17..20] = abab$ and $s[19..21] = aba$, which first occur in positions 4 and 1, respectively. Therefore, we replace $\langle 17..21 \rangle$'s pointer to $\langle 17..20 \rangle$ by a pointer to $\langle 1..8 \rangle$ and the offset 3; we replace its pointer to $\langle 19..21 \rangle$ by another pointer to $\langle 1..8 \rangle$ and the offset 0.

Extracting a single character $s[i]$ in $\mathcal{O}(\log n)$ time is fairly straightforward: we start at the root and repeatedly descend to any child whose block contains $s[i]$; if we come to a leaf u such that $s[i]$ is the j th character in u 's block but, instead of pointing to a child whose block contains $s[i]$, u stores a pointer to internal node v and offset c , then we follow u 's pointer to v and extract the $(j+c)$ th character in v 's block; finally, when we arrive at an internal node with maximum depth, we report the appropriate character of its block, which is stored there explicitly. By definition the maximum depth of the block graph is $\log n$ and at each depth, we either descend immediately in $\mathcal{O}(1)$ time, or follow a pointer from a leaf to an internal node in $\mathcal{O}(1)$ time and then descend. Therefore, we use a total of $\mathcal{O}(\log n)$ time.

For example, suppose we want to extract the 11th character from $s =$ abaababaabaababaababa using the block graph shown in Figure 1. Starting at the root, we can descend to either child, since both their blocks contain $s[11]$; suppose we descend to the left child, $\langle 1..16 \rangle$. From $\langle 1..16 \rangle$ we can descend to either the middle or right children; suppose we descend to the right child, $\langle 9..16 \rangle$. Since $\langle 9..16 \rangle$ is a leaf, the pointer to child $\langle 9..12 \rangle$ has been replaced by a pointer to $\langle 1..8 \rangle$ and offset 0, while the pointer to child $\langle 11..14 \rangle$ has been replaced by another pointer to $\langle 1..8 \rangle$ and offset 2. This is because the first occurrence of $s[9..12] = abaa$ is $s[1..4]$ and the first occurrence of $s[11..14] = aaba$ is $s[3..6]$.

Suppose we follow the second pointer. Since we would have extracted the first character from $\langle 11..14 \rangle$'s block, we are now to extract the third character from $\langle 1..8 \rangle$'s block. We can descend to either $\langle 1..4 \rangle$ and extract the third character of its block, or descend to $\langle 3..6 \rangle$ and extract the first character of its block.

Extracting longer substrings is similar, but complicated by the fact that we want to avoid breaking the substring into too many pieces as we descend. In the next section we will show how to extract any substring of length m in $\mathcal{O}(\log n + m)$ time; however, we first prove an upper bound on the block graph's size.

3 Fast Random Access in Compressed Space

In this section we show that block graphs achieve compression while simultaneously allowing easy access to the underlying string. Our space result relies on the following easily proved lemma.

Lemma 1 (§6). *The first occurrence of any substring in s must touch at least one boundary between phrases in the LZ77 parse.*

Lemma §1 allows us to relate the size of the block graph to the LZ77 parsing of the underlying string, as summarized below.

Theorem 1. *The block graph for s takes $\mathcal{O}(z \log^2 n)$ bits.*

Proof. Each internal node's block is the first occurrence of that substring in s so, by Proposition §1, it must touch at least one boundary between phrases in the LZ77 parse. Since each such boundary can touch at most three blocks in the same level, there are at most $3z$ internal nodes in each level. It follows that there are $\mathcal{O}(z \log n)$ nodes in all. Since each node stores $\mathcal{O}(\log n)$ bits, the whole block graph takes $\mathcal{O}(z \log^2 n)$ bits.

We define the query $\text{extract}(u, i, j)$ to return the i th through j th characters in u 's block. Notice that, if u is the root, then these characters are $s[i..j]$. We now show how to implement extract queries in such a way that extracting a substring of s with length m takes $\mathcal{O}(\log n + m)$ time.

There are three cases to consider when performing $\text{extract}(u, i, j)$: u could be an internal node at maximum depth, in which case we simply return the i th through j th characters of its block, which are stored explicitly; u could be an internal node with children; or u could be a leaf. First suppose that u is an internal node with children. Let d be u 's depth and $b = 2^{\lceil \log_2 n \rceil - d}$; notice b is the length of u 's block unless the block is a suffix of s , in which case the block might be shorter. If the interval $[i..j]$ is completely contained in one of the intervals $[1..b/2]$, $[b/4 + 1..3b/4]$ or $[b/2 + 1..b]$, then we set v to be the left, middle or right child of u , respectively (choosing arbitrarily if two intervals each completely contain $[i..j]$), and implement $\text{extract}(u, i, j)$ as either $\text{extract}(v, i, j)$, $\text{extract}(v, i - b/4, j - b/4)$ or $\text{extract}(v, i - b/2, j - b/2)$. Otherwise, $[i..j]$ must be more than a quarter of $[1..b]$ and we can split $[i..j]$ into 2 or 3 subintervals, each

of length at least $b/8$ but completely contained in one of $[1..b/2]$, $[b/4+1..3b/4]$ or $[b/2+1..b]$; this is the other reason why we use overlapping blocks. We implement $\text{extract}(u, i, j)$ with an extract query for each subinterval.

Now suppose that u is a leaf. Again, let d be u 's depth and $b = 2^{\lceil \log_2 n \rceil - d}$. If the interval $[i..j]$ is completely contained in one of the intervals $[1..b/2]$, $[b/4+1..3b/4]$ or $[b/2+1..b]$, then we set v to be the first, second or third internal node at the same depth to which u points, respectively, and implement $\text{extract}(u, i, j)$ as $\text{extract}(v, i', j')$, where i' and j' are i and j plus the appropriate offset. Otherwise, $[i..j]$ must be more than a quarter of $[1..b]$; we split $[i..j]$ into subintervals and implement $\text{extract}(u, i, j)$ with an extract query for each subinterval, as before.

Theorem 2. *Extracting a substring $s[f..ℓ]$ from the block graph of s takes $\mathcal{O}(\log n + ℓ - f)$ time.*

Proof. Consider the query $\text{extract}(\text{root}, f, ℓ)$ and let d be first depth at which we split the interval. Descending to depth d takes a total of $\mathcal{O}(d)$ time. By induction, if we perform a query $\text{extract}(v, i, j)$ on a node v at depth $d' > d$, then $j - i + 1$ is more than a quarter of the block size $2^{\lceil \log_2 n \rceil - d'}$ at that level. It follows that we make $\mathcal{O}\left((ℓ - f + 1)/2^{\log n - d'}\right)$ calls to extract at depth d' , each of which takes $\mathcal{O}(1)$ time. Summing over the depths, we use a total of $\mathcal{O}(\log n + ℓ - f)$ time. \square

One interesting property of our block graph structure is that, at the cost of storing a node for every possible block of size $n/2^d$ — i.e., storing $\mathcal{O}(2^d \log n)$ extra bits — we can remove the top d levels and, thus, change the overall space bound to $\mathcal{O}(z(\log n - d) \log n + 2^d \log n)$ bits and reduce the access time to $\mathcal{O}(\log n - d)$. For example, if $d = \log z$, then we store a total of $\mathcal{O}(z \log n \log(n/z))$ bits and need only $\mathcal{O}(\log(n/z))$ time for access. If $d = \log(n/\log^2 n)$, then we store a total of $\mathcal{O}(z \log n \log \log n + n/\log n)$ bits and reduce the access time to $\mathcal{O}(\log \log n)$.

González and Navarro [8] showed how, by applying grammar-based compression to a difference-coded suffix array (SA), we can build a new kind of compressed suffix array that supports access to $\text{SA}[i..j]$ in $\mathcal{O}(\log n + ℓ - f)$ time. It seems likely that, by using a modified block graph of the difference-coded suffix array instead of a grammar, we can improve their access time to $\mathcal{O}(\log \log n + ℓ - f)$ at the cost of only slightly increasing their space bound. Once we have an implementation of our data structure, we plan to test whether the resulting compressed suffix array is indeed practical.

4 Accelerated Approximate Pattern Matching

Suppose we are given an uncompressed string s of length n , the LZ77 parse [18] of s , a pattern p of length $m \leq n$ and an edit distance $k \leq m$. The primary matches of p are the substrings of s within edit distance k of p whose characters are all within distance $(m + k)$ of phrase boundaries in the parse. It is not difficult to find all p 's primary matches in $\mathcal{O}(z \min(mk + m, k^4 + m))$ time, where z is the number of phrases. To do this, we extract the substrings all of whose characters are within distance $(m + k)$ of phrase boundaries and apply to them either the

sequential approximate pattern-matching algorithm by Landau and Vishkin [11] or the one by Cole and Hariharan [3].

Once we have found p 's primary matches, we can use them to find recursively the approximate matches not within distance $(m + k)$ of any phrase boundary, which are called p 's secondary matches. To do this, we process the phrases from left to right, maintaining a sorted list of the approximate matches we have already found. For each phrase copied from a previous substring $s[i..j]$, we search in the list to see if there are any approximate matches in $s[i..j]$ that are not completely contained in $s[i..i + m + k - 1]$ or $s[j - m - k + 1..j]$. If so, we insert the corresponding secondary matches in our list. Processing all the phrases takes $\mathcal{O}(z + \text{occ})$ time, where occ is the number of approximate matches to p in s ; we give more details in Appendix A. Notice that finding p 's secondary matches does not require access to s .

Bille et al. [2] recently showed how, given a straight-line program for s with r rules, we can build an $\mathcal{O}(r)$ -word data structure that allows us to extract any substring $s[f..l]$ in $\mathcal{O}(\log n + l - f)$ time. When the straight-line program is built with the best known algorithm for approximately minimizing the number of rules, $r = \mathcal{O}(z \log n)$ [15]. It follows that we can store s in $\mathcal{O}(z \log n)$ words such that, given p and k , in $\mathcal{O}(z(\log n + m))$ time we can extract all the characters within distance $(m + k)$ of phrase boundaries and, therefore, find all p 's approximate matches in $\mathcal{O}(z(\min(mk + m, k^4 + m) + \log n) + \text{occ})$ time. (Bille et al. themselves gave a bound of $\mathcal{O}(r(\min(mk + m, k^4 + m) + \log n) + \text{occ})$ but, since even the smallest straight-line program for s has at least z rules [15], the one we state is slightly stronger.)

The key to supporting approximate pattern matching in the block graph is the addition of *bookmarks*, which will allow us to quickly extract certain regions of the underlying string. To add a bookmark to a character $s[i]$, for each block size b in the block graph, we store pointers to the two nodes whose blocks of size $2b$ completely contain the first occurrence of the substrings $s[i - b + 1..i]$ and $s[i..i + b - 1]$, and those occurrences' offsets in the blocks. Thus, storing a bookmark takes $\mathcal{O}(\log n)$ words. To extract a substring that touches $s[i]$, we extract separately the parts of the substring to the left and right of $s[i]$. Without loss of generality, we assume the part $s[i..j]$ to the right is longer and consider only how to extract it. We first find the smallest block size $b \geq j - i + 1$, then follow the pointer to the node whose block of size $2b$ contains the first occurrence $s[i..i + b - 1]$. Since that node has height $\mathcal{O}(\log(j - i + 1))$, we can extract $s[i..j]$ in $\mathcal{O}(j - i + 1)$ time.

Lemma 2. *Extracting a substring $s[f..l]$ that touches a bookmark takes $\mathcal{O}(\ell - f)$ time.*

Inserting a bookmark to each phrase boundary in the LZ77 parse takes $\mathcal{O}(z \log n)$ words and allows us, given m and k , to extract the characters within distance $(m + k)$ of phrase boundaries in a total of $\mathcal{O}(zm)$ time. Combined with the approach described above for finding secondary matches, we have our main result.

Theorem 3. Let s be a string of length n whose LZ77 parse consists of z phrases. We can store s in $\mathcal{O}(z \log n)$ words such that, given a pattern p of length $m \leq n$ and an edit distance $k \leq m$, we can find all occ substrings of s within edit distance k of p in $\mathcal{O}(z \min(mk + m, k^4 + m) + \text{occ})$ time.

Note that in the above theorem the time to find all p 's approximate matches is the same as if we were keeping s uncompressed, in the approach described at the start of this section.

5 Conclusions and Future Work

We have described a data structure for effectively compressing highly repetitive strings (or collections of strings) that supports efficient random access, substring extraction and approximate pattern matching. We believe the real advantage of our data structure lies in its simplicity: while we only offer slight theoretical improvements over the results of Bille et al. [2], we do so using far simpler, less exotic algorithmic machinery. We are currently pursuing a practical implementation.

An important direction for future work is the development of a scalable construction algorithm, particularly one that avoids the use of a suffix array (or any compressed full-text index). We feel that on Terabyte-scale collections our data structure has a particular advantage over other methods, because the positions of potential factors and their lengths are known *a priori*. This is in contrast to LZ77 and LZEND where the starting position of a given phrase depends on all those which come before it in the parsing. Along this line, we are currently pursuing efficient external memory construction algorithms (based on the doubling algorithm).

Acknowledgments. Many thanks to Francisco Claude, Juha Kärkkäinen, Sebastian Kreft, Gonzalo Navarro and Jorma Tarhio, for helpful discussions.

References

1. Arroyuelo, D., Navarro, G., Sadakane, K.: Stronger Lempel-Ziv based compressed text indexing. Algorithmica (to appear)
2. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Satti, S.R., Weimann, O.: Random access to grammar-compressed strings. In: Proceedings of the 22nd Symposium on Discrete Algorithms, SODA (2011)
3. Cole, R., Hariharan, R.: Approximate string matching: A simpler faster algorithm. SIAM Journal on Computing 31(6), 1761–1782 (2002)
4. Ferragina, P., Venturini, R.: A simple storage scheme for strings achieving entropy bounds. Theoretical Computer Science 372(1), 115–121 (2007)
5. Ferragina, P., Manzini, G.: On compressing the textual web. In: WSDM 2010: Proceedings of the Third ACM International Conference on Web Search and Data Mining, pp. 391–400. ACM, New York (2010)

6. Gagie, T., Gawrychowski, P.: Grammar-Based Compression in a Streaming Model. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 273–284. Springer, Heidelberg (2010)
7. Genome 10K Community of Scientists: A proposal to obtain whole-genome sequence for 10,000 vertebrate species. *Journal of Heredity* 100, 659–674 (2009)
8. González, R., Navarro, G.: Compressed Text Indexes with Fast Locate. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 216–227. Springer, Heidelberg (2007)
9. Kreft, S., Navarro, G.: LZ77-like compression with fast random access. In: Proceedings of the Data Compression Conference, DCC (2010)
10. Kreft, S., Navarro, G.: Self-Indexing Based on LZ77. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 41–54. Springer, Heidelberg (2011)
11. Landau, G.M., Vishkin, U.: Fast parallel and serial approximate string matching. *Journal of Algorithms* 10(2), 157–169 (1989)
12. Mäkinen, V., Navarro, G., Sirén, J., Välimäki, N.: Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology* 17(3), 281–308 (2010)
13. Manzini, G.: An analysis of the Burrows-Wheeler transform. *Journal of the ACM* 48(3), 407–430 (2001)
14. Durbin, R., et al.: 1000 genomes (2010), <http://www.1000genomes.org/>
15. Rytter, W.: Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science* 302(1-3), 211–222 (2003)
16. Sirén, J., Välimäki, N., Mäkinen, V., Navarro, G.: Run-Length Compressed Indexes are Superior for Highly Repetitive Sequence Collections. In: Amir, A., Turpin, A., Moffat, A. (eds.) SPIRE 2008. LNCS, vol. 5280, pp. 164–175. Springer, Heidelberg (2008)
17. Storer, J.A., Szymanski, T.G.: Data compression via textual substitution. *Journal of the ACM* 29(4), 928–951 (1982)
18. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23(3), 337–343 (1977)

A Finding Secondary Matches

We have claimed that, once we have found the primary matches (those cross phrase boundaries in the LZ77 parse), we can recursively find the secondary matches from them. Suppose we have a list of all the phrase sources, which we call red intervals, and all the primary matches, which we call blue intervals. Moreover, assume we have permutations that sort these intervals by their start points and by their end points; for the red intervals, these permutations can be computed in advance and, for the blue intervals, these permutations can be computed as we are searching for primary matches. For each red interval, we create a red opening node storing the interval's start point as a key, and a red closing node storing the interval's end point as a key; these nodes store pointers to each other and to a list storing secondary matches in the phrase. Similarly, for each blue interval, we create a blue opening and closing nodes, which store the interval's start and end points and pointers to each other. We use the permutations to sort the nodes by their keys and start merging them into a list. Whenever we would insert a red closing node, however, we follow its pointer to the corresponding red opening node a discard both. Whenever we would insert a blue closing node, we follow its pointer to the corresponding blue opening node and walk from the opening node to the head of the list, passing the opening nodes for all the red intervals that started earlier and have not ended yet; for each of these red opening nodes, we add a secondary match to the phrase's list, copied from the blue interval; we then discard both blue nodes. When we reach the start of a phrase — the phrase itself, not its source — we create blue nodes for all the secondary matches in it and merge them with the blue nodes for the primary matches. In total, we use $\mathcal{O}(z + \text{occ})$ time. By induction, when we reach the end of the i th phrase, for $j > i$, the list for the j th phrase contains all the secondary matches copied from before the end of the i th phrase. It follows that we recursively find all secondary matches.

A New Algorithm for the Characteristic String Problem under Loose Similarity Criteria

Yoshifumi Sakai

Graduate School of Agricultural Science, Tohoku University,
1-1 Amamiya-machi, Tsutsumidori, Aoba-ku, Sendai 981-8555, Japan
sakai@biochem.tohoku.ac.jp

Abstract. Given two strings S and T , together with an integer representing the similarity bound, the characteristic string problem consists in finding the shortest substrings of T such that S has no substrings similar to them, in the sense that one string is similar to another if the amount of ‘dissimilarities’ between them is less than or equal to the similarity bound. Under the similarity criterion that uses Levenshtain distance to measure the amount of dissimilarities between two strings, this problem is known to be solvable in cubic time and linear space. The present article proposes a new algorithm for this problem that performs in almost quadratic time and almost linear space, under a certain class of similarity criteria, including the similarity criterion based on Levenshtain distance.

1 Introduction

Given two sets \mathcal{S} and \mathcal{T} of strings, together with an integer κ representing the similarity bound, the characteristic string problem consists in finding the shortest substrings common to all strings in \mathcal{T} such that all strings in \mathcal{S} have no substrings similar to them. Here, one string is similar to another if and only if the amount of ‘dissimilarities,’ such as the edit distance (also called Levenshtain distance) [6] or Hamming distance [3], between them is less than or equal to κ . For simplicity, the present article focuses only on the case where \mathcal{S} and \mathcal{T} are singletons consisting of string S of length m and string T of length n , respectively, because it is not difficult to modify the algorithms to be proposed in this article for this special case so as to work well for the general case, using the method [7] to find all substrings common to all strings in \mathcal{T} . The characteristic string problem has applications in molecular biology [2]. In polymerase chain reaction (PCR) [10], for example, an exponentially many number of copies of the interval of a DNA between the positions where two primers hybridize are made. To strictly specify the target interval to be copied, it is necessary to select the primers so that they never hybridize to other places in the DNA. Setting T to the region just outside of the target interval, and S to the remaining part of the DNA, the solution of the problem can be used as the candidates of the primer.

The characteristic string problem in the exact setting, which finds the shortest substrings of T that never occur in S as substrings, can be solved in $O(m + n)$ time and $O(m + n)$ space [9] by traversing the suffix tree [15] generalized for

the strings S and T . In inexact setting, on the other hand, this problem is generally more difficult to solve. Under the similarity criterion that uses edit distance to measure the amount of dissimilarities between two strings, Ito et al. [4] showed that this problem can be solved in $O(\kappa mn)$ time and $O(m + n)$ space. The key idea is to use the diagonal method of dynamic programming [5,8], which provides the shortest prefix of an arbitrary suffix of T such that S has no substrings similar to it in $O(\kappa m)$ time. In contrast, under the similarity criterion that uses Hamming distance, Maaß [7] presented an $O(mn)$ -time, $O(m + n)$ -space algorithm for this problem. The execution time under this similarity criterion is hence independent of the similarity bound κ . Algorithms with this property are suitable to use when, for example, seeking the shortest substrings T' of T such that S has no substrings similar to them with the similarity bound $\kappa = \lfloor p|T'| \rfloor$ that is proportional to the length of T' for a given parameter p with $p \geq 0$. In PCR mentioned earlier, the solution of this variant of the characteristic string problem can be treated as more natural candidates of the primer than that of the original problem, because the longer the primer is, the more likely it is that the primer and a region of the DNA similar to it with a fixed amount of dissimilarities hybridize undesirably.

The present article proposes a new algorithm for the characteristic string problem under broad classes of similarity criteria, including the ones based on edit distance and Hamming distance, the execution time of which is independent of the similarity bound κ . The main idea is to use $O(m + |T'|)$ -size data structures for substrings T' of T , each of which allows the algorithm to examine whether S has no substrings similar to T' in $O(m)$ time. Consider two classes of similarity criteria respectively having the following properties: any one of the data structures for T' , T'' , or the concatenation $T'T''$ can be constructed in $t(m, n)$ time from the other two; and the data structures for $T'T''$ can be constructed in $t(m, n)$ time from the data structures for T' and T'' . Using these properties, the proposed algorithm executes in $O((m + t(m, n))n)$ time and $O(m + n)$ space under any similarity criterion in the first class. The algorithm can also be modified to execute in $O((m + t(m, n))n \log n)$ time and $O(m \log n + n)$ space, under any similarity criterion in the second class. The difference between the performance of them comes from how to reuse the data structures constructed so far.

As examples of the similarity criteria included in the above classes, three types of similarity criteria are introduced: the Hamming-distance-like criterion in the first class with $t(m, n) = O(m)$, the extra-length-like criterion (closely related to the episode matching problem [1]) in the second class with $t(m, n) = O(m)$, and the edit-distance-like criterion in the second class with $t(m, n) = O(m \log n)$. While the data structures for T' under the first two types of similarity criteria are rather straightforward, sophisticated techniques developed by Tiskin [13,14] for semi-local string comparison are used to design the data structure for T' under the last type of similarity criterion.

The remainder of the present article is organized as follows. The preliminary section gives a formal definition of the problem, together with some notations. Section 3 introduces the classes of the similarity criteria considered in this article.

Section 4 proposes a new algorithm that solves the characteristic string problem. Finally, the conclusion is presented in Sect. 5.

2 Preliminaries

Let Σ be a finite alphabet set. The concatenation of string A followed by string B is denoted by AB . For any string A , $|A|$ denotes the length of A , and $A[i]$ denotes the symbol of A at position i . A *substring* of string A is the string $A[i+1]A[i+2]\cdots A[j]$, which will be denoted by $A[i, j]$, for some index pair (i, j) with $0 \leq i \leq j \leq |A|$. Note that $|A[i, j]| = j - i$. Substrings $A[0, j]$ and $A[i, |A|]$ are a *prefix* of A and a *suffix* of A , respectively. A *subsequence* of A is the string $A[i_1]A[i_2]\cdots A[i_\ell]$ for some indices i_1, i_2, \dots, i_ℓ with $0 \leq \ell \leq |A|$ and $1 \leq i_1 < i_2 < \cdots < i_\ell \leq |A|$. If string B is a subsequence of A , then A *contains* B . Let ϵ denote the empty string.

Let Δ be a special symbol not belonging to Σ , which will be referred to as the *gap symbol*. An alignment of two strings A and B over Σ is any pair of strings A' and B' over $\Sigma \cup \{\Delta\}$ with $|A'| = |B'|$, where A' and B' are obtained from A and B by inserting any number of the gap symbols at any position so that $A'[i] \neq \Delta$ if $B'[i] = \Delta$ for any index i with $1 \leq i \leq |A'|$. For any ordered pair (a, b) of symbols in $\Sigma \cup \{\Delta\}$, let $d(a, b)$ be either a nonnegative integer or ∞ , which represents how much dissimilar a is to b . For any strings A and B over Σ , let $d(A, B)$ denote the minimum value of $\sum_{1 \leq z \leq |A'|} d(A'[z], B'[z])$, where (A', B') ranges over all alignments of A and B . Any *similarity criterion* considered in the present article is specified by the values $d(a, b)$, together with a nonnegative integer κ , which will be referred to as the *similarity bound*. This criterion decides that string A is *similar* to string B if and only if $d(A, B) \leq \kappa$. String B *occurs* in string A , if A has a substring that is similar to B .

Given strings S of length m and T of length n over Σ and a nonnegative integer κ as the similarity bound, the *characteristic string problem* under a similarity criterion consists in finding all pairs (i, j) such that $T[i, j]$ is the shortest substring of T that never occurs in S under the similarity criterion.

Before closing this section, three dissimilarity measures are introduced. For any strings A and B over Σ , the *edit distance* between A and B is the minimum number of symbol deletions, insertions, and substitutions needed to transfer A into B . Setting $d(a, b)$ to one, if $a \neq b$, and zero, otherwise, for any symbols a and b in $\Sigma \cup \{\Delta\}$, $d(A, B)$ represents the edit distance between A and B . The *Hamming distance* between A and B is the minimum number of symbol substitutions needed to transfer A into B , or ∞ if it is impossible to do that (that is, if $|A| \neq |B|$). Setting $d(a, b)$ to ∞ , if either a or b is the gap symbol, one, if $a \neq b$ and neither a nor b is the gap symbol, and zero, otherwise, $d(A, B)$ represents the Hamming distance between A and B . Let the *extra length* of A with respect to B be the minimum number of symbol deletions needed to transfer A into B , which is equal to $|A| - |B|$, or ∞ if it is impossible to do that (that is, if A does not contain B). This dissimilarity measure is closely related to the episode matching problem **I** for given two strings S and P , the objective of

which is to find all substrings of S having the minimal extra length with respect to P , if S contains P . Setting $d(a, b)$ to ∞ , if $a \neq b$ and b is not the gap symbol, one, if b is the gap symbol, and zero, otherwise, $d(A, B)$ represents the extra length of A with respect to B .

3 Similarity Criteria

This article considers the characteristic string problem under any similarity criteria having the following properties.

A similarity criterion has an *exclusion identifier* if, for any substring $T[i, j]$ of T , there exists a data structure $D[i, j]$ of size $O(m + (j - i))$ that can be used to examine whether $T[i, j]$ never occurs in S in $O(m)$ time (and further, that can be constructed in $O(m)$ time from scratch by scanning S , if $i = j - 1$). The exclusion identifier is

- $t(m, n)$ -**time mergeable**, if $D[i, j]$ can be constructed from $D[i, k]$ and $D[k, j]$ in $t(m, n)$ time, and
- $t(m, n)$ -**time removable**, if any one of $D[i, j]$, $D[i, k]$, or $D[k, j]$ can be constructed from the other two in $t(m, n)$ time.

The requirement that the similarity criterion has an (almost) linear-time mergeable exclusion identifier is not too restrictive. The similarity criteria having the following setting of the values $d(a, b)$, for example, satisfy at least this requirement.

The Hamming-distance-like criterion

$d(a, b) = \infty$ if either a or b is the gap symbol.

The extra-length-like criterion

$d(a, b) = \infty$ if $a \neq b$ and b is not the gap symbol.

The edit-distance-like criterion

$d(a, b) < \infty$ if either a or b is the gap symbol.

From this definition, the similarity criteria such that A is similar to B if and only if the Hamming distance between A and B , the extra length of A with respect to B , and the edit distance between A and B are less than or equal to κ are one of the Hamming-distance-like criteria, the extra-length-like criteria, and the edit-distance-like criteria, respectively.

Theorem 1. *The Hamming-distance-like criterion has an $O(m)$ -time removable exclusion identifier.*

Proof. For any indices g , h , i , and j with $h - g = j - i$, let $d_*(S[g, h], T[i, j])$ denote the sum of the values $d(S[g + z], T[i + z])$ with z ranging over all indices from 1 to $j - i$ such that $d(S[g + z], T[i + z]) < \infty$, and let $d_\infty(S[g, h], T[i, j])$ denote the number of indices z from 1 to $j - i$ with $d(S[g + z], T[i + z]) = \infty$. Let $\hat{d}(S[g, h], T[i, j])$ denote the pair $(d_*(S[g, h], T[i, j]), d_\infty(S[g, h], T[i, j]))$.

Due to the setting of the values $d(a, b)$, for any substrings $S[g, h]$ and $T[i, j]$, $d(S[g, h], T[i, j])$ is equal to $\sum_{1 \leq z \leq j-i} d(S[g + z], T[i + z])$, if $h - g = j - i$, or

∞ , otherwise. Based on this observation, for any substring $T]i, j]$, let $D]i, j]$ be the array of the pairs $(\hat{d}(S]g, h], T]k, l])$ indexed by $g + (j - k)$ for all pairs of a substring $S]g, h]$ of S and a substring $T]k, l]$ of $T]i, j]$ of the maximal length such that $h - g = l - k \geq 1$. Hence, for any index x with $1 \leq x < m + (j - i)$, the x th element of $D]i, j]$, denoted $D]i, j](x) = (D_*]i, j](x), D_\infty]i, j](x))$, is equal to

$$\hat{d}(S] \max(0, x - (j - i)), \min(x, m)], T] \max(i, j - x), \min(j - x + m, j)]).$$

It follows that whether $T]i, j]$ never occurs in S can be determined in $O(m)$ time by scanning $D]i, j]$ to examine whether $D_*]i, j](g + (j - i)) > \kappa$ or $D_\infty]i, j](g + (j - i)) \geq 1$ for all indices g from 0 to $m - (j - i)$, because $D]i, j](g + (j - i)) = \hat{d}(S]g, g + (j - i)], T]i, j]).$ For any index j with $1 \leq j \leq n$, $D]j - 1, j]$ can be constructed from scratch in $O(m)$ time by scanning S . Furthermore, any one of $D]i, j]$, $D]i, k]$, or $D]k, j]$ can be constructed in $O(m)$ time if the other two are available, according to the recurrence that

$$D]i, j](x) = \begin{cases} D]k, j](x) & \text{if } 1 \leq x \leq j - k, \\ D]i, k](x - (j - k)) + D]k, j](x) & \text{if } j - k < x < m + (j - k), \\ D]i, k](x - (j - k)) & \text{otherwise,} \end{cases}$$

where $(p, q) + (p', q')$ denotes the pair $(p + p', q + q')$ for pairs (p, q) and (p', q') . Thus, the theorem holds. \square

Theorem 2. *The extra-length-like criterion has an $O(m)$ -time mergeable exclusion identifier.*

Proof. It follows from the setting of the values $d(a, b)$ that, for any substrings $S]g, h]$ and $T]i, j]$, $d(S]g, h], T]i, j]) = \infty$ unless $S]g, h]$ contains $T]i, j]$. Furthermore, for any substring $S]e, f]$ of $S]g, h]$, if $d(S]e, f], T]i, j]) < \infty$, then $d(S]e, f], T]i, j]) \leq d(S]g, h], T]i, j])$. Based on these observations, let $D]i, j]$ be the list of all triples $(g, h, d(S]g, h], T]i, j]))$ such that $S]g, h]$ contains $T]i, j]$ but no substrings of $S]g, h]$ except itself do not contain $T]i, j]$ in ascending order with respect to g . Because of the minimality of $S]g, h]$, the number of elements in this list is at most m . Whether $T]i, j]$ never occurs in S can be determined in $O(m)$ time by scanning $D]i, j]$ to check whether $v > \kappa$ for all elements (g, h, v) . The list $D]j - 1, j]$ can be constructed from scratch in $O(m)$ time by scanning S , because this list consists of the index pairs $(h - 1, h, d(S[h], T[j]))$ for all indices h from 1 to m such that $S[h] = T[j]$. On the other hand, (e, h, v) is an element of $D]i, j]$ if and only if $D]i, k]$ and $D]k, j]$ have elements (e, f, u) and (g, h, w) with $f \leq g$, respectively, $D]i, k]$ has no elements (e', f', u') with $f < f' \leq g$, $D]k, j]$ has no elements (g', h', w') with $f \leq g' < g$, and $v = u + d(S[f, g], \epsilon) + w$. Furthermore, for any elements (e, h, v) and (e', h', v') in $D]i, j]$ with $e < e'$, if (e, h, v) is obtained from (e, f, u) in $D]i, k]$ and (g, h, w) in $D]k, j]$, and (e', h', v') is obtained from (e', f', u') in $D]i, k]$ and (g', h', w') in $D]k, j]$, then $g \leq f'$ holds. Therefore, $D]i, k]$ and $D]k, j]$ can be merged into $D]i, j]$ in $O(m)$ time in a straightforward manner. Thus, the theorem holds. \square

Theorem 3. *The edit-distance-like criterion has an $O(m \log n)$ -time mergeable exclusion identifier.*

Proof. Let δ be the maximum value of $d(\Delta, b) + d(a, \Delta) - d(a, b)$, where (a, b) ranges over all ordered pairs of symbols in Σ . Note that this value is constant. It follows from [I3], together with [II], that, for any strings A and B , there exists a set of $(|A| + |B|)\delta$ index pairs (u, v) with $|B|\delta < u \leq |A|\delta$ and $0 < v \leq (|A| + |B|)\delta$ such that, for any substring $A[g, h]$, $d(A[g, h], B)$ is equal to $d(\epsilon, B) + d(A[g, h], \epsilon) - (h - g)\delta$ plus the number of elements (u, v) in the set such that both $g\delta < u$ and $v \leq h\delta$. Let $P_{A,B}$ denote this set of index pairs. Using this set, together with the value $d(\epsilon, B)$, the values $\min_{0 \leq g \leq h} d(A[g, h], B)$ for all indices h from 0 to $|A|$ can be calculated in $O(|A|)$ time. For any string B of length one, $P_{A,B}$ can be constructed in $O(|A|)$ time. Furthermore, for any strings B and C , if $P_{A,B}$ and $P_{A,C}$ are available, then the set $P_{A,BC}$ can be constructed in $O(|A| \log \min(|A|, |B|, |C|))$ time, using the method [I4][I2] that executes a certain kind of matrix multiplication. Thus, letting $D[i, j]$ be the pair of the set $P_{S,T}[i, j]$ and the value $d(\epsilon, T[i, j])$, the theorem can be proven. \square

4 Algorithm

This section proposes algorithms that solve the characteristic string problems under any similarity criterion having a $t(m, n)$ -time removal exclusion identifier, or a $t(m, n)$ -time mergeable exclusion identifier.

The proposed algorithms follow a common outline. The key observation underlying this outline is stated in the following lemma.

Lemma 1. *Under any similarity criterion, if $T[i, j]$ occurs in S , then any substring of $T[i, j]$ also occurs in S .*

Proof. Assume that $S[g, h]$ is similar to $T[i, j]$, and let $T[k, l]$ be any substring of $T[i, j]$. Let (S', T') be an arbitrary alignment of $S[g, h]$ and $T[i, j]$ such that $\sum_{1 \leq z \leq |T'|} d(S'[z], T'[z]) \leq \kappa$. There exist indices $z_{i+1}, z_{i+2}, \dots, z_j$ with $1 \leq z_{i+1} < z_{i+2} < \dots < z_j \leq |T'|$ such that $T'[z_{i+1}]T'[z_{i+2}] \cdots T'[z_j] = T[i, j]$. Let $S[e, f]$ be the substring of $S[g, h]$ obtained from $S'[z_k, z_l]$ by deleting all of the gap symbols occurring in it. Recall that $d(a, b) \geq 0$ for any symbols a and b in $\Sigma \cup \{\Delta\}$. It follows that $\sum_{z_k < z \leq z_l} d(S'[z], T'[z]) \leq \kappa$, where $z_i = 0$. Since $(S'[z_k, z_l], T'[z_k, z_l])$ is an alignment of $S[e, f]$ and $T[k, l]$, $d(S[e, f], T[k, l]) \leq \kappa$ holds, and hence, $S[e, f]$ is similar to $T[k, l]$. \square

The above lemma implies that, if $T[i, j]$ is the shortest substring of $T[0, j]$ that never occurs in S , then the shortest suffix of $T[h - (j - i), h]$ that never occurs in S is the shortest substring of $T[0, h]$ never occurring in S , where h is the least index greater than j such that $T[h - (j - i), h]$ never occurs in S . Furthermore, no substrings $T[k, l]$ with $j < k < l < h$ are the shortest substrings of T that never occurs in S . Similarly, in the initial case, if $T[0, j]$ is the shortest prefix of T that never occur in S , then the shortest suffix of $T[0, j]$ that never occurs in S is

```

1: Let  $i = 0$ , and let  $j = 1$ ;
2: while  $T[i, j]$  occurs in  $S$ ,
3:   increase  $j$  by one;
4: let  $L$  be the list consisting of the index pair  $(i, j)$  alone;
5: while  $j \leq n$ ,
6:   increase  $i$  by one;
7:   if  $T[i, j]$  never occurs in  $S$ , then
8:     let  $L$  be the list consisting of the index pair  $(i, j)$  alone,
9:   otherwise,
10:    increase  $j$  by one;
11:    if  $T[i, j]$  never occurs in  $S$ , then
12:      append  $(i, j)$  to  $L$ ;
13: output  $L$ .

```

Fig. 1. Conceptual algorithm indicating the outline common to the proposed algorithms

the shortest substring of $T[0, j]$ that never occurs in S . Moreover, no substrings of $T[0, j - 1]$ are the shortest substrings of T that never occur in S . Based on this recurrence, the outline common to the proposed algorithms is described in Fig. II as an conceptual algorithm without explicitly presenting how to obtain the data structure $D[i, j]$ before examining whether $T[i, j]$ never occurs in S . It is easy to verify that the list L just before executing statement 10 consists of all index pairs (g, h) such that $T[g, h]$ is the shortest substring of $T[0, j]$ that never occurs in S . Thus, this conceptual algorithm successfully outputs the solution of the characteristic string problem.

Before presenting the proposed algorithms, it is useful to observe the following properties belonging to the conceptual algorithm. Let (i_s, j_s) be the index pair (i, j) such that the s th examination of whether $T[i, j]$ (never) occurs in S is performed by the conceptual algorithm, and let r be the total number of these examinations. It then follows from the description of the conceptual algorithm that (i_1, j_1) is equal to $(0, 1)$, while (i_s, j_s) with $2 \leq s \leq r$ is equal to either $(i_{s-1} + 1, j_{s-1})$ or $(i_{s-1}, j_{s-1} + 1)$. Furthermore, since the empty substring of T occurs in S , $j_s - i_s \geq 0$ holds for any index s with $2 \leq s \leq r$. Thus, $r \leq 2n$.

In the proof of the following two theorems, the proposed algorithms are presented by describing how to construct the data structure $D[i_s, j_s]$ for each index s from 1 to r in this order.

Theorem 4. *The characteristic string problem under any similarity criterion having a $t(m, n)$ -time removable exclusion identifier can be solved in $O((m + t(m, n))n)$ time and $O(m + n)$ space.*

Proof. Since (i_s, j_s) with $s \geq 2$ is equal to either $(i_{s-1} + 1, j_{s-1})$ or $(i_{s-1}, j_{s-1} + 1)$, the algorithm can update $D[i_{s-1}, j_{s-1}]$ to $D[i_s, j_s]$ in $O(m) + t(m, n)$ time for each index s from 2 to r , after constructing the initial data structure $D[i_1, j_1]$ from scratch in $O(m)$ time. \square

- 1: Let X be the empty list;
- 2: for each index s from 1 to r ,
- 3: if $s = 1$ or $j_s = j_{s-1} + 1$, then
- 4: construct $D[j_s - 1, j_s]$ from scratch;
- 5: append $D[j_s - 1, j_s]$ to X ;
- 6: while the last two elements of X are $D[i, i + 2^p]$ and $D[i + 2^p, (i + 2^p) + 2^p]$
 for some nonnegative integer p ,
- 7: merge $D[i, i + 2^p]$ and $D[i + 2^p, (i + 2^p) + 2^p]$ into $D[i, i + 2^{p+1}]$;
- 8: delete the last two elements from X ;
- 9: append $D[i, i + 2^{p+1}]$ to X ;
- 10: if $i_s = i_{s-1} + 1$, then
- 11: delete the first element $D[i, i + 2^p]$ from X ;
- 12: for each integer q from $p - 1$ to 0 in this order,
- 13: construct $D[i + 2^q, i + 2^q + 1]$ from scratch, and then update $D[i + 2^q, (i + 2^q) + x - 1]$ to $D[i + 2^q, (i + 2^q) + x]$ incrementally (by merging $D[i + 2^q, (i + 2^q) + x - 1]$ and $D[(i + 2^q) + x - 1, (i + 2^q) + x]$ after constructing $D[(i + 2^q) + x - 1, (i + 2^q) + x]$ from scratch)
 for each index x from 2 to 2^q ;
- 14: prepend $D[i + 2^q, (i + 2^q) + 2^q]$ to X ;
- 15: construct $D[i_s, j_s]$ by merging all elements in X from the first to the last.

Fig. 2. Algorithm for constructing $D[i_s, j_s]$ for each index s from 1 to r , under the similarity criterion having a $t(m)$ -time mergeable exclusion identifier

Theorem 5. *The characteristic string problem under any similarity criterion having a $t(m, n)$ -time mergeable exclusion identifier can be solved in $O((m + t(m, n))n \log n)$ time and $O(m \log n + n)$ space.*

Proof. The algorithm maintains a list as the value of variable X so that it consists of the data structures $D[i, i + 2^p]$ for at most $2 \log_2 n$ pairs (i, p) that can be merged into $D[i_s, j_s]$ whenever constructing $D[i_s, j_s]$ with $s \geq 2$. Fig. 2 presents the description of this algorithm, in which, although not explicitly described, it is assumed that the pair (i, p) is attached to each element $D[i, i + 2^p]$ in X , so as to give the algorithm access to the values i and p .

Let X be valid with respect to s , if $D[i_s, j_s]$ can be obtained by merging all elements in X in the order from the first to the last. Note that X is valid with respect to 1 just before the first execution of statement 15. For any index s with $2 \leq s \leq r$, if $j_s = j_{s-1} + 1$ (and hence, $i_s = i_{s-1}$), then the algorithm appends $D[j_s - 1, j_s]$ to X by executing statements 4 and 5. Therefore, if X just before the execution of statement 5 is valid with respect to $s - 1$, then it becomes valid with respect to s after executing this statement. This situation remains unchanged even after executing the subsequent statements 6–9, because these statements only iteratively replace the two consecutive elements $D[i, i + 2^p]$ and $D[i + 2^p, (i + 2^p) + 2^p]$ at the last position of X by $D[i, i + 2^{p+1}]$ as many times as possible. Similarly, if $i_s = i_{s-1} + 1$ (hence, $j_s = j_{s-1}$), and X is valid with respect to $s - 1$ just after the execution of statement 10, then X becomes valid with respect to s by the execution of statements 11–14, because these statements

replace the first element $D[i_{s-1}, i_{s-1} + 2^p]$ of X by the list of the data structures $D[i_{s-1} + 1, i_{s-1} + 2], D[i_{s-1} + 2, i_{s-1} + 4], \dots, D[i_{s-1} + 2^{p-1}, i_{s-1} + 2^p]$. Thus, by induction on s , it can be verified that X is valid with respect to s just before the s th execution of statement 15 for any index s with $1 \leq s \leq r$, which proves the correctness of the algorithm.

The list X can be split into the prefix consisting of the elements prepended by statement 14, and the suffix consisting of those appended by statements 5 and 9. It is easy to verify that $p < q$ for any consecutive elements $D[i, i + 2^p]$ and $D[k, k + 2^q]$ in the prefix, and also that $p > q$ for any consecutive elements $D[i, i + 2^p]$ and $D[k, k + 2^q]$ in the suffix. Hence, X consists of at most $2 \log_2 n$ elements. Recall that the size of each element $D[i, i + 2^p]$ in X is $O(m + 2^p)$. Therefore, this algorithm requires $O(m \log n + n)$ space.

It can also be verified, by induction on s , that i is divisible by 2^p for any element $D[i, i + 2^p]$ in X . Based on this observation, the execution time of this algorithm is analyzed as follows. For any index i with $i_1 \leq i \leq i_r$, there exists exactly one integer q with $0 \leq q \leq \log_2 n$ such that $D[i, i + 2^q]$ is prepended to X by statement 14. Recall that i is divisible by 2^q . Hence, for any integer q with $0 \leq q \leq \log_2 n$, the data structures $D[i, i + 2^q]$ for at most $\frac{n}{2^q}$ indices i are prepended to X in the entire algorithm. Note that statement 13 constructs $D[i, i + 2^q]$ in $O((m + t(m, n)) \cdot 2^q)$ time. Therefore, the total execution time of statements 10–14 in the entire algorithm is $O((m + t(m, n))n \log n)$. On the other hand, for any index i with $i_1 \leq i \leq i_r$, if $D[i, i + 2^{p+1}]$ is appended to X by statement 9, then i is divisible by 2^{p+1} . Statement 7 constructs $D[i, i + 2^{p+1}]$ in $t(m, n)$ time by merging $D[i, i + 2^p]$ and $D[i + 2^p, i + 2^{p+1}]$. Furthermore, for any index p with $1 \leq p + 1 \leq \log_2 n$, there exist at most $\frac{n}{2^{p+1}}$ indices i with $i_1 \leq i \leq i_r$ such that i is divisible by 2^{p+1} . Consequently, the total execution time of statements 3–9 is $O((m + t(m, n))n)$, including the time for constructing $D[j_s - 1, j_s]$ in statement 4. Recall that X consists of at most $2 \log_2 m$ elements. This implies that statements 15 constructs $D[i_s, j_s]$ in $O(t(m, n) \log n)$ time, and hence, the execution time of statement 15 in the entire algorithm is $O(t(m, n) n \log n)$. Thus, the algorithm executes in $O((m + t(m, n))n \log n)$ time. \square

The above two theorems, together with Theorems 1, 2, and 3, immediately yield the following corollary.

Corollary 1. *The characteristic string problem can be solved in $O(mn)$ time and $O(m + n)$ space under the Hamming-distance-like criterion, can be solved in $O(mn \log n)$ time and $O(m \log n + n)$ space under the extra-length-like criterion, and can be solved in $O(mn \log^2 n)$ time and $O(m \log n + n)$ space under the edit-distance-like criterion.*

5 Conclusions

A new algorithm for the characteristic string problem for strings S of length m and T of length n and the similarity bound κ that runs in time independent of κ was proposed. This algorithm executes in $O((m + t(m, n))n \log n)$ time and in

$O(m \log n + n)$ space under the class of similarity criteria having a $t(m, n)$ -time mergeable exclusion identifier. This class with $t(m, n) = O(m \log n)$ includes the similarity criterion based on edit distance (by which the amount of dissimilarities between two strings is measured), and the class with $t(m, n) = O(m)$ includes the one based on how much longer a string is than a prescribed subsequence. The algorithm also executes in $O((m + t(m, n))n)$ time and $O(m + n)$ space under any similarity criterion having a $t(m, n)$ -time removable exclusion identifier. This class with $t(m, n) = O(m)$ includes the similarity criterion based on Hamming distance. One interesting and important open question is whether the execution time under the Hamming-distance-like criterion can be considerably improved using, for example, the fast Fourier transform.

References

1. Dan, G., Fleischer, R., Gąsieniec, L., Gunopulos, D., Kärkkäinen, J.: Episode Matching. In: Hein, J., Apostolico, A. (eds.) CPM 1997. LNCS, vol. 1264, pp. 12–27. Springer, Heidelberg (1997)
2. Gusfield, D.: Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology. Cambridge University Press (1997)
3. Hamming, R.W.: Error detecting and error correcting codes. Bell Syst. Tech. J. 29, 147–160 (1950)
4. Ito, M., Shimizu, K., Nakanishi, M., Hashimoto, A.: Polynomial-Time Algorithms for Computing Characteristic Strings. In: Crochemore, M., Gusfield, D. (eds.) CPM 1994. LNCS, vol. 807, pp. 274–288. Springer, Heidelberg (1994)
5. Landau, G.M., Vishkin, U.: Introducing efficient parallelism into approximate string matching and a new serial algorithm. In: 18th Annual ACM Symposium on Theory of Computing, pp. 220–230 (1986)
6. Levenshtain, V.I.: Binary codes capable of correcting insertions and reversals. Sov. Phys. Dokl. 10, 707–710 (1966)
7. Maafß, M.G.: A fast algorithm for the inexact characteristic string problem. Technical Report TUM-I0312, Fakultät für Informatik, TU München (2003)
8. Myers, E.W.: An $o(nd)$ difference algorithm and its variations. Algorithmica 1, 251–266 (1986)
9. Nakanishi, M., Hasidume, M., Ito, M., Hashimoto, A.: A Linear-Time Algorithm for Computing Characteristic Strings. In: Du, D.-Z., Zhang, X.-S. (eds.) ISAAC 1994. LNCS, vol. 834, pp. 315–323. Springer, Heidelberg (1994)
10. Saiki, R., Gelfand, D., Stoffel, S., Scharf, S., Higuchi, R., Horn, G., Mullis, K., Erlich, H.: Primer-directed enzymatic amplification of DNA with a thermostable DNA polymerase. Sci. 239, 487–491 (1988)
11. Sakai, Y.: An almost quadratic time algorithm for sparse spliced alignment. Theory Comput. Syst. 48, 189–210 (2011)
12. Sakai, Y.: A fast algorithm for multiplying min-sum permutations. Discrete Appl. Math. 159, 2175–2183 (2011)
13. Tiskin, A.: Semi-local string comparison: Algorithmic techniques and applications. Math. Comput. Sci. 1, 571–603 (2008)
14. Tiskin, A.: Fast distance multiplication of unit-Monge matrices. In: 21st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1287–1295 (2010)
15. Ukkonen, E.: On-line construction of suffix-trees. Algorithmica 14, 249–260 (1995)

Succinct Indexes for Circular Patterns

Wing-Kai Hon¹, Chen-Hua Lu², Rahul Shah³, and Sharma V. Thankachan³

¹ National Tsing Hua University, Taiwan

wkhon@cs.nthu.edu.tw

² Academia Sinica, Taiwan

walchl@iis.sinica.edu.tw

³ Louisiana State University, USA

{rahul,thanks}@csc.lsu.edu

Abstract. Circular patterns are those patterns whose circular permutations are also valid patterns. These patterns arise naturally in bioinformatics and computational geometry. In this paper, we consider succinct indexing schemes for a set of d circular patterns of total length n , with each character drawn from an alphabet of size σ . Our method is by defining the popular Burrows-Wheeler transform (BWT) on circular patterns, based on which we achieve succinct indexes with space $n \log \sigma(1 + o(1)) + O(n) + O(d \log n)$ bits, while pattern matching or dictionary matching queries can be supported efficiently.

1 Introduction

Circular patterns are those patterns whose circular permutations are also valid patterns. For example, for the pattern `abcd`, its circular permutations include `abcd`, `bcda`, `cdab`, and `dabc`. Some of the practical fields where we need to handle circular patterns include bioinformatics and computational geometry [14]. For instance, the genomes of many viruses, such as herpes simplex virus (HSV-1), exist as circular strings [21]. Also, with the launch of next-generation sequencing technology, the biologists in the metagenomics area are now collecting and sequencing microbial samples (which have circular chromosomes and plasmid DNA) from an environment directly, without isolating and culturing the samples [6][20]. To process the above data efficiently thus requires a suitable index for possibly circular patterns. In computational geometry, a polygon may be stored by listing the co-ordinates of its vertices in clockwise order. As applications in these two areas usually involve huge data sets, finding a compact indexing scheme for the circular patterns, which can support online pattern matching queries efficiently, is therefore important.

Compact indexing schemes for sequential texts (or patterns) have been studied actively in the past decades [9][7][10][19][5][13]. One of the major techniques is to apply the Burrows-Wheeler transform (BWT) [4] to compress the indexed texts, and augment the BWT-compressed texts with sublinear-sized auxiliary data structures to support the desired pattern matching queries in an efficient manner. Indeed, BWT is essentially “circular” in nature, in a sense that we can

start with any character in the text, and make use of BWT to repeatedly retrieve the preceding characters and recover the original text. Recently, Ferragina and Venturini [8] made use of such a property to achieve a compressed version of the Permuterm index, where the latter is an important tool in the information retrieval area. In this paper, we exploit this property further to give a definition of the BWT for a set of circular texts, and show that with this definition, a set of d circular texts with n characters over an alphabet of size σ can be encoded succinctly in $(1 + o(1))n \log \sigma + O(n) + O(d \log n)$ bits, while the basic pattern matching queries can be supported efficiently.

Given a set \mathcal{D} of d patterns, the *dictionary matching problem* is to index \mathcal{D} such that for any online query text T , we can locate the occurrences of any valid pattern within T efficiently. Such a problem with a set of sequential patterns has been well-studied in the literature [1, T2, 2, 11]. When dealing with circular patterns, this problem can be naively reduced to the sequential case, where we maintain a dictionary matching index on a set \mathcal{D}' that consists of all the circular permutations of all patterns in \mathcal{D} . Clearly, this is not a space-efficient approach as the total length of all patterns in \mathcal{D}' can be $\Omega(n^2)$. In this paper, we consider a special case where the *aspect ratio* α between the lengths of any two patterns is bounded by a constant (that is, $\alpha = \max_i\{|P_i|\} / \min_j\{|P_j|\} = O(1)$), and show a succinct index of space $(1 + o(1))n \log \sigma + O(n) + O(d \log n)$ bits that can answer each online query in $O((|T| + occ) \log^{1+\epsilon} n)$ time, for any fixed $\epsilon > 0$; here, occ denotes the total number of occurrences in the output.¹ The major idea is to augment our BWT-based index for circular patterns with a non-trivial adaptation of Sadakane's compressed suffix tree [19].

The organization of the paper is as follows. Section 2 gives the preliminaries, while Section 3 gives the formal definition of the BWT for a set of circular texts (or patterns). In Section 4, we give the framework of the dictionary matching algorithm using our index. In Section 5, we show how to adapt Sadakane's compressed suffix tree so that it can be used to index circular patterns. Finally, in Section 6, we show how to combine the algorithm in Section 4 with the index in Section 5 so that dictionary matching queries can be answered efficiently.

2 Preliminaries

In this section, we give notations to ease the discussion on circular strings, and give analogous definitions of the well-known suffix tree [22] and suffix array [16] with respect to a set of circular patterns.

Definition 1. Let $P = P[1..p]$ be a circular pattern. Intuitively, if we move from one character to the next character repeatedly in P , we will end up with

¹ Note that the aspect ratio assumption can be removed if we are willing to create $O(\log n)$ indexes, where patterns of similar lengths (within a ratio 2) are maintained by the same index. Consequently, the total index space will still be succinct, but the online dictionary matching query will be slowed down due to the search in multiple indexes, taking $O((|T| \log n + occ) \log^{1+\epsilon} n)$ time.

an infinite string $P[1..p]P[1..p]P[1..p]\dots$. We call such an infinite length string the infinite form of P , and it is denoted by P^∞ .

Definition 2. The lexicographic ordering of two circular patterns P and Q is determined by comparing the lexicographic ordering of the strings P^∞ and Q^∞ . For example, $\text{baa} < \text{ba} < \text{bac}$ because $\text{baabaa}\dots < \text{babaa}\dots < \text{bacbac}\dots$. This definition is crucial in designing our succinct index in later sections.

Let $\mathcal{D} = \{P_1, P_2, \dots, P_d\}$ denote a set of d circular patterns of total length n , where P_i^∞ is lexicographically less than P_{i+1}^∞ ². Let $S = P_1 P_2 \dots P_d$ be a text of length n obtained by concatenating all the patterns in \mathcal{D} . We shall consider $\mathcal{D} = \{\text{aba}, \text{bacba}, \text{daba}\}$ as our running example, then $S = \text{ababacbadaba}$.

Definition 3. A circular suffix of S is a circular permutation of a pattern in \mathcal{D} . In particular, the circular suffix S_i , which starts at position i in S , is referred to as the i th circular suffix of S . The following are the circular suffixes in our example: $S_1 = \text{aba}$, $S_2 = \text{baa}$, $S_3 = \text{aab}$, $S_4 = \text{bacba}$, $S_5 = \text{acbba}$, $S_6 = \text{cbaba}$, $S_7 = \text{babac}$, $S_8 = \text{abacb}$, $S_9 = \text{daba}$, $S_{10} = \text{abad}$, $S_{11} = \text{bada}$, and $S_{12} = \text{adab}$.

Note that a circular suffix is considered as a circular pattern.

Definition 4. We define the circular suffix array $SA_c[1..n]$ of S , such that $SA_c[i] = j$ if the lexicographically i th smallest circular suffix of S starts at position j in S . In our example, $SA_c[1..12] = [3, 1, 8, 10, 5, 12, 2, 7, 4, 11, 6, 9]$ ³

Definition 5. A compact trie containing the infinite form of each circular suffix of S is called a circular suffix tree ST_c of S . For any given node u in ST_c , $\text{path}(u)$ represents the concatenation of all the edge labels along the path from root to u and $|\text{path}(u)|$ represents the length of $\text{path}(u)$ which is also called as string depth of u . For any string Q , the locus of Q in ST_c is defined to be the highest node v (i.e., nearest to the root) such that Q is a prefix of $\text{path}(v)$ (the existence of the locus node is not necessary for all strings).

3 Circular Burrows-Wheeler Transform

In this section, we define formally a variant of the BWT, called *circular BWT* (*cbwt*), on the set \mathcal{D} of circular patterns. The transformation gives a string $L[1..n]$ of length n , such that the i th character $L[i]$ is the last character of the lexicographically i th smallest circular suffix $S_{SA_c[i]}$. The *cbwt* for our running example is $L[1..12] = \text{babdbbacaaaa}$ (See Figure 1).

² For ease of discussion, we assume that each pattern in \mathcal{D} is not *periodic*. That is, each P_i cannot be written as P^k for some period string P and some integer $k > 1$. Under this definition, the lexicographical ordering of the patterns will be distinct. Handling the general case involves minor adaptation to the indexing scheme, where the main idea is to represent a periodic string by its shortest period and its length. We defer the details in the full paper.

³ Note that under the assumption that each pattern in \mathcal{D} is not periodic, the lexicographical order of the circular suffixes of S will be distinct.

i	S_i	$SA_c[i]$	$S_{SA_c[i]}$	$L[i]$
1	aba	3	aab	b
2	baa	1	aba	a
3	aab	8	abacb	b
4	bacba	10	abad	d
5	acbab	5	acbab	b
6	cbaba	12	adab	b
7	babac	2	baa	a
8	abacb	7	babac	c
9	daba	4	bacba	a
10	abad	11	bada	a
11	bada	6	cbaba	a
12	adab	9	daba	a

Fig. 1. The circular Burrows-Wheeler transform on \mathcal{D} in our running example

Let $C_r(P)$ represents the circular permutation of a pattern P by moving its last character to the front (for instance, if $P = \text{abcd}$, then $C_r(P) = \text{dabc}$). Then we have the following observation.

Lemma 1. Suppose that $S_{SA_c[j]} = C_r(S_{SA_c[i]})$. Then we have

$$j = \text{Count}(L[i]) + \text{rank}_{L[i]}(L, i),$$

where $\text{Count}(c)$ gives the number of characters in L that are lexicographically smaller than c , and $\text{rank}_c(L, i)$ is the number of c 's in $L[1..i]$.

We call the mapping between i and j in the above observation the *circular mapping*. This is analogous to the *LF* mapping in the Burrows-Wheeler transform [7]. For example, when $i = 6$, $S_{SA_c[i]} = \text{adab}$. Then $\text{Count}(L[6]) = \text{Count}(\text{b}) = 6$ and $\text{rank}_{\text{b}}(L, 6) = \text{rank}_{\text{b}}(L, 6) = 4$, hence $j = 10$. Therefore $C_r(S_{SA_c[6]}) = \text{bada} = S_{SA_c[10]}$.

By maintaining a wavelet tree structure [10] of $L[1..n]$ in $n \log \sigma(1+o(1))$ bits, the *rank* query on L can be answered in $O(\log \sigma)$ time. For the *Count* query, each can be answered in $O(1)$ time by maintaining a bit-vector of size $n + \sigma$ bits, together with the auxiliary data structure of [18]. Thus, on given any i , the circular mapping that computes the corresponding j (with $S_{SA_c[j]} = C_r(S_{SA_c[i]})$) can be performed in $O(\log \sigma)$ time. In addition, we have the following lemma.

Lemma 2. By maintaining an $o(n \log \sigma) + O(d \log n)$ -bit auxiliary data structure, the circular suffix array entry $SA_c[i]$ for any i can be computed in $O(\log^{1+\epsilon} n)$ time, where ϵ is any fixed value with $\epsilon > 0$. After that, the length of the corresponding circular suffix can be reported in $O(1)$ time.

Let P be a string. The circular suffix range $[\ell, r]$ of P is defined to be the maximal range in the circular suffix array such that P is the prefix of the infinite form of each of the circular suffixes $S_{SA_c[\ell]}, S_{SA_c[\ell+1]}, \dots, S_{SA_c[r]}$. In case such a range

does not exist, ℓ and r can be any arbitrary values with $\ell > r$. The following lemma gives a hint on the pattern matching power of $cbwt$, which is analogous to the one demonstrated in the original BWT [7].

Lemma 3. *Let $[\ell, r]$ be the circular suffix range of a string P , and c be a character. Let $[\ell', r']$ be the circular suffix range of the string cP (that is, the string formed by adding c before P). Then we have*

$$\ell' = \text{Count}(c) + \text{rank}_c(L, \ell - 1) + 1; \quad r' = \text{Count}(c) + \text{rank}_c(L, r).$$

Given a text $Q[1..|Q|]$, we can repeatedly apply the above lemma to obtain the circular suffix range $[First, Last]$ of Q . This is exactly the same as performing the backward search algorithm of [7] (See Algorithm 1). With a further post-processing step as shown in the theorem below, we can refine the results to find out all those circular suffixes (instead of their infinite forms) having Q as their prefix.

Algorithm 1. The backward search algorithm of [7]

Input: A text $Q[1..|Q|]$

Output: Circular suffix range for all suffixes with Q as their prefixes

```

1:  $i = |Q|$ ,  $c = Q[i]$ ,  $First = \text{Count}(c) + 1$ ,  $Last = \text{Count}(c + 1)$ ;
2: while (( $First \leq Last$ ) and ( $i \geq 2$ )) do
3:    $c = Q[i - 1]$ ;
4:    $First = \text{Count}(c) + \text{rank}_c(L, First - 1) + 1$ ;
5:    $Last = \text{Count}(c) + \text{rank}_c(L, Last)$ ;
6:    $i = i - 1$ ;
7: end while
8: if ( $Last < First$ ) then
9:   return NIL;
10: else
11:   return [ $First, Last$ ];
12: end if
```

Theorem 1. *A given set of d circular patterns of total length n can be indexed in $n \log \sigma(1+o(n)) + O(n) + O(d \log n)$ bits, such that on given any query text Q , we can report all those circular suffixes having Q as their prefix in $O(|Q| \log \sigma + (1 + occ) \log^{1+\epsilon} n)$ time, where occ is the number of such circular suffixes.*

Proof. Let $Length[1..n]$ be an array such that $Length[i] = |S_{SA_c[i]}|$. We maintain a range maximum query structure [3,19] of $2n + o(n)$ bits over the $Length$ array, so that for any range $[x, y]$, we can report the location z of the maximum element in $Length[x..y]$ using constant time. After that, we can obtain the value $Length[z]$ in $O(\log^{1+\epsilon} n)$ time by Lemma 2.

Using backward search, the circular suffix range ($[First, Last]$) of Q can be computed in $O(|Q| \log \sigma)$ time. Now, all those circular suffixes $S_{SA_c[i]}$ such that

$i \in [First, Last]$ and $|S_{SA_c[i]}| \geq |Q|$ can be retrieved in $O((occ + 1) \log^{1+\epsilon} n)$ time by repeatedly performing the above range maximum queries on the *Length* array. \square

Remark. Although *cbwt* is defined on circular patterns, it can indeed be considered as a generalized version of the original BWT. Precisely, given a string T , its original BWT will be equivalent to the *cbwt* on a circular string $T\#$, where $\#$ is a unique character not in the alphabet Σ . Consequently, one may have a BWT index containing both sequential texts and circular texts. Similarly, the main component of the compressed Permuterm index of [Q] for a set of patterns $\{P_1, P_2, \dots, P_d\}$, which is based on the original BWT, will be a special case of the *cbwt*. Precisely, it will be equivalent to the *cbwt* on the set of circular patterns $\{P_1\#, P_2\#, \dots, P_d\#\}$.

4 Framework for Circular Dictionary Matching

Given an online query text T , the circular dictionary matching query can be answered by reporting all those circular suffixes which comes as a prefix of $T[i\dots|T|]$ for $i = 1, 2, 3, \dots, |T|$. To support the query efficiently, our main structure is a circular suffix tree ST_c of S along with an array $Length[1\dots n]$, such that $Length[j] = |S_{SA_c[j]}|$, and a range minimum query structure of size $n(2 + o(1))$ bits [B19] over *Length*. In addition, we maintain the following marked node information, such that a node u is marked if there exists a circular suffix S_i such that S_i is a prefix of $path(u)$, $path(parent(u))$ is a prefix of S_i^∞ , but $path(u)$ is not a prefix of S_i^∞ ; here, $parent(u)$ denotes the parent node of u . Note that each S_i may cause more than one node to be marked, and we assume that repeated marking of the same node is the equivalent to marking the node once. Moreover, each node in ST_c maintains a pointer to its lowest marked ancestor (If no such marked ancestor exists, the pointer points to the root).

Now, we explain the framework of answering dictionary matching query using the above ST_c with marked nodes. Firstly, given a string Q , the longest prefix of Q that has a locus node in ST_c is denoted by $lcp(ST_c, Q)$. Next, for an online query text T , we let v_i denote the locus node of $lcp(ST_c, T[i\dots|T|])$ in ST_c . Then, the circular dictionary matching of T can be performed as follow:

For $i = 1, 2, 3, \dots, |T|$,

1. Compute the locus node v_i .
2. Retrieve all the circular suffixes S_j such that S_j is a prefix of $lcp(ST_c, T[i\dots|T|])$ and $lcp(ST_c, T[i\dots|T|])$ is a prefix of S_j^∞ .
3. Retrieve all the circular suffixes S_j such that S_j is a prefix of $lcp(ST_c, T[i\dots|T|])$ but $lcp(ST_c, T[i\dots|T|])$ is not a prefix of S_j^∞ .

Step 2 is equivalent to retrieving all the circular suffixes S_j in the subtree of v_i , such that $|S_j| \leq |lcp(ST_c, T[i\dots|T|])|$. This can be done by repeatedly performing range minimum queries on the *Length* array. Next, for each S_j that should be reported in Step 3, there exists a unique ancestor w of v_i that is

marked by S_j in the definition. It is easy to see that one of the sibling of w must contain S_j^∞ in its subtree. Thus, to retrieve all the S_j s in Step 3, we visit the siblings of each marked ancestor w of v_i , and repeatedly perform range minimum queries on the $Length$ array to report those S_j in the subtree of the siblings with $|S_j| \leq |path(parent(w))|$. Note that the two repeated sequences of range minimum queries (one on all the left siblings as a batch, and one on all the right siblings as a batch) corresponding to a marked ancestor is guaranteed to give at least one output.

In the next section, we give a space-efficient encoding for ST_c and its marked nodes, so that Step 2 and Step 3 can be performed efficiently. In Section 6, we explain how to compute the locus nodes v_i s for an online query text, with the help of the $cbwt$, in an efficient manner. Combining both will then allow us to achieve the desired result for the dictionary matching problem.

5 Succinct Encoding of Circular Suffix Tree

In this section, we show how ST_c can be encoded succinctly, whose main components include the $cbwt$ on \mathcal{D} in Section 3 (and its auxiliary structures), and the following two structures.

5.1 Parentheses Encoding of ST_c

We maintain a parentheses encoding of the tree structure of ST_c , along with the marked nodes information (by maintaining an additional bit vector over the parentheses encoding) in $O(n)$ bits. Assume that each node in ST_c is represented by its pre-order rank in the tree. Then for any two nodes u and v in ST_c , the following tree operations can be performed in constant time [3, 17, 19]:

- $parent(u)$, which returns the parent of node u ;
- $lma(u)$, which returns the lowest marked ancestor of u ;
- $lca(u, v)$, which returns the lowest common ancestor of nodes u and v ;
- $leftmost(u)$ and $rightmost(u)$, which respectively returns the leftmost leaf and the rightmost leaf in the subtree rooted at node u .

We also maintain a bit-vector $B_{leaf}[1\dots 2n - 1]$ such that $B[i] = 1$ if and only if the node with pre-order rank i is a leaf (assume that pre-order rank of the root is 1). An auxiliary data structure of $o(n)$ bits is augmented so that $rank$ and $select$ on B_{leaf} can be answered in constant time; here, $rank(i)$ counts the number of 1s in $B_{leaf}[1..i]$, and $select(j)$ returns the position of the j th 1 in B_{leaf} . Then we have the following result.

Lemma 4. *Suppose that a pattern P has locus u in ST_c . Let $[\ell_u, r_u]$ denote the circular suffix range of P . Then we have*

$$\ell_u = rank_{B_{leaf}}(leftmost(u)) \quad \text{and} \quad r_u = rank_{B_{leaf}}(rightmost(u)).$$

On the other hand, the locus node u corresponding to the circular suffix range $[\ell_u, r_u]$ is equal to $lca(select_{B_{leaf}}(\ell_u), select_{B_{leaf}}(r_u))$.

5.2 Height Array

Kasai et al. [15] showed that a bottom-up traversal of the suffix tree can be simulated by using only the suffix array and an array Hgt storing a set of the lengths of the longest common prefixes between two adjacent-rank suffixes, called *height array*, where $Hgt[1..n - 1]$, where $Hgt[i] = |\text{lcp}(S_{SA[i]}, S_{SA[i+1]})|$ for $1 \leq i \leq n - 1$. Sadakane [19] observed that this Hgt array can be encoded succinctly in $O(n)$ bits based on its following property $Hgt[i] - 1 \leq Hgt[\text{SA}^{-1}[\text{SA}[i] + 1]]$. We re-define the height array in the context of a circular suffix tree as follows: $Hgt_c[i] = |\text{lcp}(S_{SA_c[i]}^\infty, S_{SA_c[i+1]}^\infty)|$ for $1 \leq i \leq n - 1$. Note that $Hgt_c[i] \leq \max(|S_{SA_c[i]}|, |S_{SA_c[i+1]}|)$, under our assumption that no circular pattern (and thus any of its circular permutation) is periodic.

Let $B_{\ell e}[1..n]$ is a bit vector marking the starting position of all patterns $P_j \in \mathcal{D}$ in the concatenated text S . That is, $B_{\ell e}[i] = 1$ if and only if $S_i = P_j \in \mathcal{D}$, where $1 \leq j \leq d$. Thus, $B_{\ell e}[i + 1] = 0$ implies that S_i and S_{i+1} are circular permutations of the same pattern.

Lemma 5. *For $1 \leq i \leq n - 1$, if $B_{\ell e}[i + 1] = 0$, then*

$$Hgt_c[\text{SA}_c^{-1}[i]] - 1 \leq Hgt_c[\text{SA}_c^{-1}[i + 1]].$$

Let $H[i] = Hgt_c[\text{SA}_c^{-1}[i]]$ and let $x_j = \text{select}_{B_{\ell e}}(j)$. From the above lemma,

$$H[x_1] \leq H[x_1 + 1] + 1 \leq H[x_1 + 2] + 2 \leq \dots \leq H[x_1 + |P_1| - 1] + |P_1| - 1$$

$$H[x_2] \leq H[x_2 + 1] + 1 \leq H[x_2 + 2] + 2 \leq \dots \leq H[x_2 + |P_2| - 1] + |P_2| - 1$$

⋮

$$H[x_d] \leq H[x_d + 1] + 1 \leq H[x_d + 2] + 2 \leq \dots \leq H[x_d + |P_d| - 1] + |P_d| - 1$$

Now, if the largest value in the i th row for each i can be bounded by $O(|P_i|)$, then these non-decreasing sequences can be encoded with bit-vectors of total length $O(\sum_{j=1}^d |P_j|) = O(n)$, such that any $H[i]$ can be retrieved in constant time.⁴ Such a bound of $O(|P_i|)$ is not true in general, but can be guaranteed when the aspect ratio $\alpha = \max_i \{|P_i|\} / \min_j \{|P_j|\}$ is bounded by a constant. This gives the following result.

Lemma 6. *Suppose that the aspect ratio of the patterns in \mathcal{D} is bounded by a constant. The path length $|\text{path}(u)|$ of any given node u in ST_c can be computed in $O(\log^{1+\epsilon} n)$ time by maintaining an additional structure of size $2n + o(n)$ bits.*

Proof. For any given node u , $|\text{path}(u)| = Hgt_c[i_{\min}]$, such that i_{\min} is the largest j satisfying $Hgt_c[j] \leq Hgt_c[i]$ for $\ell_u \leq i < r_u$, where $[\ell_u, r_u]$ is the circular suffix range corresponding to the node u . Here i_{\min} can be computed in constant time by maintaining a range minimum query structure ($2n + o(n)$ bits) [3, 19] of Hgt_c array and $Hgt_c[i_{\min}] = H[\text{SA}_c[i_{\min}]]$. Thus, the main bottleneck is the time for computing $\text{SA}_c[i_{\min}]$, which is $O(\log^{1+\epsilon} n)$. □

⁴ Consider the following sequence as an example: $a_1 + 1 \leq a_2 + 2 \leq s_3 + 3 \leq \dots \leq a_n + n$, then let $b_i = a_i + 2i$, then $b_1 < b_2 < b_3 \dots < b_n$ and it can be encoded using a bit-vector $B[1..b_n]$ such that $B[b_i] = 1$ for all i , else 0. Thus $a_i = \text{select}_B(i) - 2i$.

Lemma 7. Suppose that the aspect ratio of the patterns in \mathcal{D} is bounded by a constant. Given the locus node u of $Q[1..|Q|]$ in ST_c , the locus u' of $Q[1..(|Q|-1)]$ in ST_c can be computed in $O(\log^{1+\epsilon} n)$ time and the locus u'' of cQ can be computed in $O(\log \sigma)$ time.

Proof. If $|path(parent(u))| = |Q| - 1$, then $u' = parent(u)$, else $u' = u$. In either case, computing path length takes $O(\log^{1+\epsilon} n)$ time (Lemma 6). Then, we get the locus u'' by a backward search step followed by an *lca* operation in ST_c . \square

6 Circular Dictionary Matching with Our Index

In the previous section, we have described our encoding of the circular suffix tree ST_c , in which the *cbwt* on \mathcal{D} is already included as its component. The total index size can be bounded by $n \log \sigma(1 + o(1)) + O(n) + O(d \log n)$ bits. In this section, we show how each of the three steps in our dictionary matching query algorithm (see Section 4) can be performed efficiently based on ST_c and *cbwt*.

For Step 1, we shall find the locus nodes v_i in a backward manner, where we obtain $v_{|T|}$ initially, and for each subsequent step, we obtain v_{j-1} from v_j . In particular, suppose that $lcp(ST_c, T[i..|T|]) = T[i..j]$ and its locus node v_i in ST_c are known. Now, to compute $lcp(ST_c, T[i-1..n]) = T[i-1..j']$ (the longest prefix of $T[i-1..j]$ that has a locus in ST_c) and its locus node v_{i-1} , we perform the following steps. First, observe that $j' \leq j$, and j' must be the largest k where the locus node of $T[i-1..k]$ exists. Hence, this can be obtained by Lemma 7 in $O((j - j' + 1) \log^{1+\epsilon} n)$ time. Thus the time for computing all locus nodes $v_{|T|}, v_{|T|-1}, \dots, v_2, v_1$ can be bounded by $O(|T| \log^{1+\epsilon} n)$ time.

For Steps 2 and 3, since navigational operations and the lowest marked ancestor in ST_c can each be supported in $O(1)$ time, the bottleneck comes from performing the range minimum queries on *Length* and retrieving the corresponding minimum entries of *Length* (See the proof of Theorem 1 for the time complexity of a similar operation). The total number of such queries and retrievals can be bounded by $O(|T| + occ)$, and each requires $O(\log^{1+\epsilon} n)$ time, assuming that the aspect ratio of the patterns in \mathcal{D} is bounded by a constant. Thus, the total time of Steps 2 and 3 can be bounded by $O((|T| + occ) \log^{1+\epsilon} n)$, and we have:

Theorem 2. Suppose that the aspect ratio of the patterns in \mathcal{D} is bounded by a constant. By maintaining an index of size $n \log \sigma(1 + o(1)) + O(n) + O(d \log n)$ bits, the circular dictionary matching query for any online text T can be answered in $O((|T| + occ) \log^{1+\epsilon} n)$ time. In case the aspect ratio is unbounded, we can maintain $O(\log n)$ indexes of the same total space, so that the circular dictionary matching query is answered in $O((|T| \log n + occ) \log^{1+\epsilon} n)$ time.

References

1. Aho, A., Corasick, M.: Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM 18(6), 333–340 (1975)

2. Belazzougui, D.: Succinct Dictionary Matching With No Slowdown. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 88–100. Springer, Heidelberg (2010)
3. Bender, M.A., Farach-Colton, M.: The Level Ancestor Problem Simplified. *Theoretical Computer Science* 321(1), 5–12 (2004)
4. Burrows, M., Wheeler, D.J.: A Block-Sorting Lossless Data Compression Algorithm, Technical Report 124, Digital Equipment Corporation, USA (1994)
5. Chan, H.L., Hon, W.K., Lam, T.W., Sadakane, K.: Compressed Indexes for Dynamic Text Collections. *ACM Transactions on Algorithms* 3(2) (2007)
6. Eisen, J.A.: Environmental Shotgun Sequencing: Its Potential and Challenges for Studying the Hidden World of Microbes. *PLoS Biology* 5(3), e82 (2007)
7. Ferragina, P., Manzini, G.: Indexing Compressed Text. *Journal of the ACM* 52(4), 552–581 (2005)
8. Ferragina, P., Venturini, R.: The Compressed Permuterm Index. *ACM Transactions on Algorithms* 7(1) (2010)
9. Grossi, R., Vitter, J.S.: Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. *SIAM Journal on Computing* 35(2), 378–407 (2005)
10. Grossi, R., Gupta, A., Vitter, J.S.: High-Order Entropy-Compressed Text Indexes. In: SODA, pp. 841–850 (2003)
11. Hon, W.K., Ku, T.H., Shah, R., Thankachan, S.V., Vitter, J.S.: Faster Compressed Dictionary Matching. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 191–200. Springer, Heidelberg (2010)
12. Hon, W.K., Lam, T.W., Shah, R., Tam, S.L., Vitter, J.S.: Compressed Index for Dictionary Matching. In: DCC, pp. 23–32 (2008)
13. Hon, W.K., Shah, R., Vitter, J.S.: Compression, Indexing, and Retrieval for Massive String Data. In: Amir, A., Parida, L. (eds.) CPM 2010. LNCS, vol. 6129, pp. 260–274. Springer, Heidelberg (2010)
14. Iliopoulos, C.S., Rahman, M.S.: Indexing Circular Patterns. In: Nakano, S.-i., Rahman, M. S. (eds.) WALCOM 2008. LNCS, vol. 4921, pp. 46–57. Springer, Heidelberg (2008)
15. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
16. Manber, U., Myers, G.: Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing* 22(5), 935–948 (1993)
17. Munro, J.I., Raman, V.: Succinct Representation of Balanced Parentheses and Static Trees. *SIAM Journal on Computing* 31(3), 762–776 (2001)
18. Raman, R., Raman, V., Rao, S.S.: Succinct Indexable Dictionaries with Applications to Encoding k -ary Trees, Prefix Sums and Multisets. *ACM Transactions on Algorithms* 3(4) (2007)
19. Sadakane, K.: Compressed Suffix Trees with Full Functionality. *Theory of Computing Systems*, pp. 589–607 (2007)
20. Simon, C., Daniel, R.: Metagenomic Analyses: Past and Future Trends. *Applied and Environmental Microbiology* 77(4), 1153–1161 (2011)
21. Strang, B.L., Stow, N.D.: Circularization of the Herpes Simplex Virus Type 1 Genome upon Lytic Infection. *Journal of Virology* 79(19), 12487–12494 (2005)
22. Weiner, P.: Linear Pattern Matching Algorithms. In: Proceedings of Symposium on Switching and Automata Theory, pp. 1–11 (1973)

Range LCP

Amihood Amir^{1,2,*}, Alberto Apostolico^{3,4,**} Gad M. Landau^{5,6,***},
Avivit Levy^{7,8}, Moshe Lewenstein¹, and Ely Porat^{1,†}

¹ Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
{amir,porately}@cs.biu.ac.il

² Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218

³ College of Computing, Georgia Institute of Technology, 801 Atlantic Drive,
Atlanta, GA 30318, USA
axa@cc.gatech.edu

⁴ Dipartimento di Ingegneria dell' Informazione, Università di Padova, Via Gradenigo
6/A, 35131 Padova, Italy

⁵ Department of Computer Science, University of Haifa, Mount Carmel, Haifa 31905,
Israel
landau@cs.haifa.ac.il

⁶ Department of Computer Science and Engineering,
Polytechnic Institute of New York University, 6 Metrotech Center, Brooklyn,
NY 11201

⁷ Department of Software Engineering, Shenkar College, 12 Anna Frank,
Ramat-Gan, Israel
avivitlevy@shenkar.ac.il

⁸ CRI, Haifa University, Mount Carmel, Haifa 31905, Israel

Abstract. In this paper, we define the *Range LCP* problem as follows. Preprocess a string S , of length n , to enable efficient solutions of the following query:

Given $[i, j]$, $0 < i \leq j \leq n$, compute $\max_{\ell, k \in \{i, \dots, j\}} LCP(S_\ell, S_k)$, where $LCP(S_\ell, S_k)$ is the length of the longest common prefix of the suffixes of S starting at locations ℓ and k . This is a natural generalization of the classical LCP problem.

Surprisingly, while it is known how to preprocess a string in linear time to enable LCP computation of two suffixes in constant time, this seems quite difficult in the *Range LCP* problem. It is trivial to answer such queries in time $O(|j - i|^2)$ after a linear-time preprocessing and easy to show an $O(1)$ query algorithm after an $O(|S|^2)$ time preprocessing. We provide algorithms that solve the problem with the following complexities:

* Partly supported by NSF grant CCR-09-04581, ISF grant 347/09, and BSF grant 2008217.

** Partly supported by BSF grant 2008217.

*** Partly supported by the National Science Foundation Award 0904246, Israel Science Foundation grant 347/09, Yahoo, Grant No. 2008217 from the United States-Israel Binational Science Foundation (BSF) and DFG.

† Partly supported by BSF grant 2006334, ISF grant 1484/08, and Google award.

1. Preprocessing Time: $O(|S|)$, Space: $O(|S|)$, Query Time: $O(|j-i| \log \log n)$.
2. Preprocessing Time: no preprocessing, Space: $O(|j-i| \log |j-i|)$, Query Time: $O(|j-i| \log |j-i|)$. However, the query just gives the pairs with the longest LCP, *not* the LCP itself.
3. Preprocessing Time: $O(|S| \log^2 |S|)$, Space: $O(|S| \log^{1+\varepsilon} |S|)$ for arbitrary small constant ε , Query Time: $O(\log \log |S|)$.

1 Introduction

The Longest Common Prefix (LCP) has been historically an important tool in Combinatorial Pattern Matching:

1. The connection between Edit Distance and Longest Common Prefix (LCP) calculation has been shown and exploited in the classic Landau-Vishkin paper in 1989 [11]. It was shown in that paper that computing mismatches and LCPs is sufficient for computing the Edit Distance.
2. The LCP is the main tool in various Bioinformatics algorithms for finding maximal repeats in a genomic sequence.
3. The LCP plays an important role in compression. Its computation is required in order to compute the Ziv-Lempel compression, for example [12].

Therefore, the LCP has been amply studied and generalized versions of the problem are of interest. A first natural generalization of the problem is a “range” version. Indeed, a “range” version of the LCP problem was considered by Cormode and Muthukrishnan [4]. They called it the *Interval Longest Common Prefix (ILCP) Problem*. In that version, the maximum LCP between a given suffix and all suffixes in a given interval, is sought. They provide an algorithm whose preprocessing time is $O(|S| \log^2 |S| \log \log |S|)$, and whose query time is $O(\log |S| \log \log |S|)$. This result was then improved by [10] to $O(|S| \log |S|)$ preprocessing time and $O(\log |S|)$ query time.

This paper provides efficient algorithms for a more general version of the Range LCP problem.

Problem Definition. The formal definitions of LCP and the range LCP problem are given below.

Definition 1. Let $A = A[1], \dots, A[n]$ and $B = B[1], \dots, B[n]$ be strings over alphabet Σ . The Longest Common Prefix (LCP) of A and B is the empty string if $A[1] \neq B[1]$. Otherwise it is the string a_1, \dots, a_k , $a_i \in \Sigma$, $i = 1, \dots, k$, $k \leq n$, where $A[i] = B[i] = a_i$, $i = 1, \dots, k$, and $A[k+1] \neq B[k+1]$ or $k = n$.

We abuse notations and sometimes refer to the length of the LCP, k , as the LCP. We, thus, denote the length of the LCP of A and B by $LCP(A, B)$.

Given a constant c and string $S = S[1], \dots, S[n]$ over alphabet $\Sigma = \{1, \dots, n^c\}$, one can preprocess S in linear time in a manner that allows subsequent LCP queries in constant time. I.e., any query of the form $LCP(S_i, S_j)$, where S_i and S_j are the suffixes of S starting at locations i, j of S , respectively, $1 \leq i, j, n$, can

be computed in constant time. For general alphabets, the preprocessing takes time $O(n \log n)$.

This can be done either via Suffix tree construction [16,13,14,15] and Lowest Common Ancestor (LCA) queries [6,2], or suffix array construction [8] and LCP queries [9].

Our problem is formally defined as follows.

Definition 2. *The Range LCP problem is the following:*

INPUT: String $S = S[1], \dots, S[n]$ over alphabet Σ .

Preprocess S in a manner allowing efficient solutions to queries of the form:

QUERY: $i, j, \quad 1 \leq i \leq j \leq n$.

Compute $\text{RangeLCP}(i, j) = \max_{\ell, k \in \{i, \dots, j\}} \text{LCP}(S_\ell, S_k)$, where S_k is the suffix of S starting at location k , i.e. $S_k = S[k], S[k+1], \dots, S[n]$.

Simple Baseline Solutions. For the sake of completeness, we describe two straight-forward algorithms, which we henceforth call Algorithm B1 and Algorithm B2, to solve the Range LCP problem. They are the baseline to improve. Algorithm B1 has a linear-time preprocessing but a quadratic Range LCS query time. The algorithm preprocess the input string S for LCP queries. Then, given an interval $[i, j]$, it computes $\text{LCP}(k, \ell)$ for every pair $k, \ell, \quad i \leq k \leq \ell \leq j$, and chooses the pair with the maximum LCP. The preprocessing can be accomplished in linear time for alphabets that are fixed polynomials of n and in time $O(n \log n)$ for general alphabets. The query processing has $|j - i|^2$ LCP calls, each one taking a constant time for a total time $O(|j - i|^2)$. Algorithm B2 uses two-dimensional range-maximum queries, defined as follows.

Definition 3. *Let M be an $n \times n$ matrix of natural numbers. The Two-dimensional Range-Maximum Query (2dRMQ) problem is the problem of preprocessing the matrix M so that subsequent 2dRM queries can be answered efficiently.* ¹ A 2dRM query is of the form:

Query: $2dRM([i_1, j_1], [i_2, j_2]) = \max\{M[k, \ell] \mid i_1 \leq k \leq j_1; i_2 \leq \ell \leq j_2\}$, where $1 \leq i_1 \leq j_1 \leq n$ and $1 \leq i_2 \leq j_2 \leq n$.

Yuan and Atallah [17] showed that the 2dRMQ problem can be solved with preprocessing time linear in the matrix size and subsequent constant-time queries. The LCP preprocessing can be accomplished in linear time for alphabets that are fixed polynomials of n and in time $O(n \log n)$ for general alphabets. Constructing table L requires n^2 LCP calls, each one taking a constant time for a total time $O(n^2)$. This is also the time required by the Yuan-Atallah [17] preprocessing algorithm as we have $O(n^2)$ -size matrix. The subsequent 2dRM queries are done in constant time and provide the requested Range LCP query results, since it is precisely the maximum of all LCPs in the interval $[i, j]$. Therefore, algorithm B2 has a quadratic preprocessing time that enables subsequent constant-time Range LCS queries.

¹ Note that in this definition the matrix is not sparse. In Section 4 we need a different version of the problem where the matrix is sparse and therefore use other tools.

Paper Contributions. The contributions of the paper are three-fold:

- A formalization of a natural generalization of the classical LCP problem.
- An efficient algorithm for the generalized Range LCP version.
- Introducing the notion of *bridges* and *optimal bridges* to the computation of LCP which enabled our efficient Range LCP algorithm.

The rest of the paper is organized as follows. In Section 2 we present a simple algorithm that solves the Range LCP problem with linear preprocessing and query-time linear in the range size. In Section 3 we show that, if we relax the query to only ask for a *pair* of indices in the range that provide the LCP (and not require the length of the LCP), then there is a solution whose time is linear in the range size with *no preprocessing at all*. Finally, in Section 4 we provide our main result, which is an algorithm whose preprocessing time is $O(n \log^2 n)$ that can answer Range LCP queries in time $O(\log \log n)$. Note that our preprocessing algorithm has an additional factor of $\log n$ compared to the algorithm for the *limited* version of [10] and our query time is faster.

The results are summarized in Theorem 1.

Theorem 1. *Given a string S , the range LCP problem can be solved in:*

1. Preprocessing Time: $O(|S|)$, Space: $O(|S|)$, Query Time: $O(|j - i| \log \log n)$.
2. Preprocessing Time: *no preprocessing*, Space: $O(|j - i| \log |j - i|)$, Query Time: $O(|j - i| \log |j - i|)$. However, the query just gives the pairs with the longest LCP, not the LCP itself.
3. Preprocessing Time: $O(|S| \log^2 |S|)$, Space: $O(|S| \log^{1+\varepsilon} |S|)$ for arbitrary small constant ε , Query Time: $O(\log \log |S|)$.

2 An Algorithm Linear in the Range Length

The quadratic time of algorithm B1 comes from choosing the maximum LCP of all pairs. We will show that it is sufficient to choose the maximum LCP of a judiciously chosen linear number of pairs that will guarantee that the maximum LCP of *all* pairs is indeed chosen. The following lemma provides the necessary key property required for the choice of pairs.

Lemma 1. *Let A be the suffix array of S . For any i, k, j such that $1 \leq i \leq k \leq j \leq n$ we have that $LCP(S_{A[i]}, S_{A[j]}) \leq LCP(S_{A[i]}, S_{A[k]})$ and $LCP(S_{A[i]}, S_{A[j]}) \leq LCP(S_{A[k]}, S_{A[j]})$.*

Lemma 1 means that for any set X of suffixes, the largest LCP must be between a pair of suffixes S' and S'' in X such that there is no suffix $S''' \in X$ that is lexicographically between S' and S'' . Formally,

Corollary 1. *Let $X = \{S_{A[i_1]}, \dots, S_{A[i_k]}\}$ be a set of suffixes where $i_1 \leq i_2 \leq \dots \leq i_k$, in other words, the suffixes are sorted lexicographically. Let $mlcp = \max\{LCP(S', S'') | S', S'' \in X\}$. Then,*

$$mlcp \in \{LCP(S_{A[i_1]}, S_{A[i_2]}), LCP(S_{A[i_2]}, S_{A[i_3]}), \dots, LCP(S_{A[i_{k-1}]}, S_{A[i_k]})\}.$$

The corollary means that if the suffixes are lexicographically sorted, then only the LCP of pairs of adjacent suffixes needs to be tested, and there is only a linear number of such pairs, not a quadratic number. Algorithm LinearRange is an adaptation of Algorithm B1 that exploits Corollary 10 to achieve time linear in the range size.

Algorithm LinearRange's Time: As seen in the analysis of the preprocessing time of Algorithm B1, the preprocessing can be accomplished in linear time for alphabets that are fixed polynomials of n and in time $O(n \log n)$ for general alphabets. The query processing has $|j - i|$ LCP calls, each one taking a constant time for a total time $O(|j - i|)$. Sorting X can be done in time $O(|j - i| \log \log n)$ by noting that the indices to be sorted are from the range $\{1, \dots, n\}$ and thus a data structure such as 15 can be used.

We have, therefore, proven the first part of Theorem 11.

3 Finding the Pair with the Longest Common Prefix

The liner-time preprocessing algorithm of Section 2 is, nevertheless, quite heavy in that it requires constructing a suffix array of the text. If we relax the query to require only a pair of indices whose LCP is the longest in the range, it is possible to eliminate the need for the preprocessing and solve the query in time and space proportional to the range length. The idea of this algorithm leads to the efficient algorithm of Section 4.

Algorithm Pair's Idea: Until now, we have considered the LCP of all pairs in the range and chose the largest. We reverse our outlook. We now consider the largest common prefix in the range, and then choose a pair of indices that have that LCP. The suffix tree provides all common prefixes, and can be constructed in linear time. The problem is that we do not want to construct the suffix tree from the range till the end, since that construction time will not be linear in the range size.

Let the query be $\text{RangeLCP}(i, j)$, and let $d = |j - i| + 1$. Construct the suffix tree of $S' = S[i]S[i+1] \dots S[i+3d]$, and mark on that tree the leaves representing suffixes that start in the range $[i, j]$. Consider the lowest suffix tree node that has at least two marked descendants. Call that node x . There are two cases:

Case a: If x has no marked child whose edge is the special *end-of-text* symbol $\$$, then the substring ending at node x is the $\text{RangeLCP}(i, j)$, and any pair of marked leaves in the subtree rooted at x is a pair whose LCP is longest in the range.

Case b: If x has a marked child whose edge is the symbol $\$$, then it could be that the LCP extends beyond the end of the substring of length $3d$ for which we have constructed the suffix tree. In this case, we do not know how far the LCP extends. However, we chose the pair of marked leaves of x to be the marked child whose edge is labeled $\$$ (assume that it represents the suffix starting at index ℓ), and the marked leaf representing the suffix that is closest to ℓ .

Lemma 2 guarantees that the pair of indices we choose in the second case has the largest common prefix in the range.

Lemma 2. *Let x be a lowest node in the suffix tree of S' that has more than one marked leaf. Assume also that the substring on the path from the root to x is the suffix $S[\ell], S[\ell + 1], \dots, S[i + 3d]$, and that one of the marked children of x has edge labelled $\$$. Let S_t be the marked suffix of S' that is in x 's subtree and that is closest to ℓ , i.e., every marked suffix S'_p in the subtree of x has $t \geq p$. Then ℓ and t are the indices between i and j whose common prefix is largest.*

Proof. Note that $|\ell - t| < d$ yet the length of the common prefix of the two suffixes S_ℓ and S_t is at least $2d$. The situation is, therefore, that the suffix S_ℓ is periodic, and its period's length is $|\ell - t|$. Clearly, the LCP will extend as long as the period continues in S . Also, the substring $S[i + d], \dots, S[i + 3d]$ is periodic with the same period as S_ℓ and S_t . Therefore, it is impossible that within the range $[i, j]$ there is a pair with a longer common prefix, because such a pair either implies the existence of another different period and contradicts the well-known periodicity lemma or contradicts the choice of x as the lowest node having at least two marked nodes. \square

The pseudo-code for Algorithm Pair is presented in Fig. 1.

Algorithm Pair

Query: Given interval $[i, j]$.

Construct suffix tree for $S' = S[i], S[i + 1], \dots, S[i + 3|j - i|]$

Mark all leaves of the suffix tree of S' that are the suffixes S'_i, \dots, S'_j .

Let x_1, \dots, x_k be the nodes with the longest path from the root, which have at least two marked leaves in their subtrees.

If there is no $x \in \{x_1, \dots, x_k\}$ that has a marked child where the label on the edge to that child is $\$$:

Then any pair of leaves in any of the x_1, \dots, x_k subtrees has the LCP.

Otherwise, let $x \in \{x_1, \dots, x_k\}$ be such that the substring on the path from the root to x

is the suffix $S[\ell], S[\ell + 1], \dots, S[i + 3|j - i|]$, and that one of the children of x has an edge labelled $\$$.

Let S_t be the marked suffix of S' that is in x 's subtree and that is closest to ℓ ,

i.e., every marked suffix S'_p in the subtree of x has $t \geq p$.

Choose ℓ and t as the pair with the LCP.

Fig. 1. The no-preprocessing linear-time query algorithm

Algorithm Pair's Time: Algorithm Pair constructs the suffix tree of $S[i], \dots, S[i + 3d]$, thus its time is $O(|j - i|)$ for fixed finite alphabets and $O(|j - i| \log |j - i|)$ for general alphabets. All other manipulations of the suffix tree are linear in the size of the tree, which is $O(|j - i|)$.

We have, therefore, proven the second part of Theorem 1.

4 An Efficient Range LCP Algorithm

The algorithm presented in this section is based on efficiently calculating and using *optimal bridges*. We define this concept below².

Definition 4. A bridge of height ℓ over a string S of length n is a pair of indices $\langle i, j \rangle$, $1 \leq i < j \leq n$, where ℓ is the length of $LCP(S_i, S_j)$. We say that $|j - i|$ is the length of the bridge.

Bridge $\langle i', j' \rangle$ is said to be nested in bridge $\langle i, j \rangle$, if $i' \geq i$ and $j' \leq j$.

Bridge $\langle i', j' \rangle$ is said to be interleaved with bridge $\langle i, j \rangle$, if one of the following two conditions hold: (1) $i' \geq i$ and $j' > j$, or (2) $i' < i$ and $j' \leq j$.

An optimal bridge is a bridge $\langle i, j \rangle$ of height ℓ such that there is no bridge $\langle i', j' \rangle$ of height ℓ' , $\ell' \geq \ell$, where $\langle i', j' \rangle$ is nested in $\langle i, j \rangle$, and there is no interleaving bridge $\langle i', j' \rangle$ where $LCP(S_i, S_j) = LCP(S_{i'}, S_{j'})$.

Algorithm Bridges' Idea: The algorithm exploits the following property:

Observation 1. (The Bridge-LCP Property) $\text{RangeLCP}(a, b)$ is the maximum height of all the bridges $\langle i, j \rangle$ contained in interval $[a, b]$, i.e. where $a \leq i < j \leq b$.

The problem is that there is a quadratic number of bridges. The idea of the efficient *RangeLCP* algorithm is to use optimal bridges, rather than bridges. It is easy to verify that the Bridge-LCP property holds when using only optimal bridges, rather than all bridges, however, it is also clear that there are fewer optimal bridges than bridges. In the sequel, we will prove that there are $O(n \log n)$ optimal bridges, and show how to construct them in time $O(n \log n)$. In addition, we will show a suitable reduction to two dimensional points where an orthogonal range query provides the maximum length bridge within a given interval.

To prove the bound on the number of maximal bridges, we need the following concept.

Definition 5. A substring S' of string S is a maximal LCP, if there exist indices i, j such that $S' = LCP(S_i, S_j)$ and there are no indices k, ℓ where S' is a proper prefix of $LCP(S_k, S_\ell)$.

Example: In the string $S = ABABCDE$, the maximal LCPs are: B , E , AB , DE and CDE . C and CD are not maximal, since they are prefixes of CDE which is $LCP(S_5, S_8)$, and A is not maximal since it is a prefix of AB .

Lemma 3. There are $O(n \log n)$ optimal bridges in a string of length n .

A Lower Bound on the Number of Optimal Bridges. As noted, our analysis gives an upper bound on the bridges, but cleaning out the non-optimal bridges may reduce that number. Unfortunately, the Fibonacci word provides a

² The methods used in this section do not use assumptions on the alphabet and, therefore, apply to general alphabets.

lower bound of $\Omega(n \log n)$ on the number of optimal bridges. The Fibonacci words are defined as follows.

$S_0 = A$, $S_1 = AB$, ..., $S_n = S_{n-2}S_{n-1}$ (the concatenation of the previous two Fibonacci words.)

A *repetition* is a string of the form $U^k U'$ where $k \geq 2$ and U' is a prefix of U . Clearly, every such U is an optimal bridge. A Fibonacci word of length n has $O(n \log n)$ repetitions, providing a lower bound of $\Omega(n \log n)$ optimal bridges.

4.1 Constructing the Optimal Bridges

We present an algorithm to construct the succinct representation of the optimal bridges. Our algorithm constructs the lists from the highest bridge down in time linear in their number. Lemma 3 assures that the number of bridges is $O(n \log n)$.

Algorithm *Construct-Optimal-Bridges* Idea: Let T_S be a suffix tree of string S . Consider a node x in T_S such that the length of the substring from the root to x is the maximum. Let the substring from the root to x be P . All the children of x are leaves (suffixes) and thus all pairs of children of x define bridges. However, if x_1, \dots, x_k are the children of x (leaves in x 's subtree, i.e., suffixes of S) sorted by their index, then they provide us with the succinct representation of the bridges of LCP P .

In the initial stage of the algorithm all the maximal LCPs have their succinct bridge representations. We inductively climb up T_S , merging the succinct lists of the lower nodes. The merge is done in a fashion whereby the smaller list is merged into the larger one, to create the succinct representation of bridges of a smaller size. When the root is reached we have the succinct representation of all bridges.

Algorithm *Construct-Optimal-Bridges* Time: Because of the fact that the smaller lists are merged into the larger ones, this algorithm can be trivially implemented in time $O(n \log^2 n)$. However, Apostolico and Preparata [1] show how this process can be done in time $O(n \log n)$. Alternatively, it can be done using the mergeable dictionaries of [7] with the same complexity bounds.

4.2 Finding the Smallest Bridge in an Interval

Assuming that we have a list of the optimal bridges. We consider every bridge $\langle x, y \rangle$ as a point on the plane with coordinates (x, y) . Every such point has weight $\ell = LCP(S_x, S_y)$. For any interval $[i, j]$, $\text{RangeLCP}(i, j)$ is the point with the maximum value in the orthogonal range $[i, j], [i, j]$.

Orthogonal range queries on the plane, where k points are assigned weights can be answered using algorithms whose preprocessing time is $O(k \log k)$ giving a data structure of $O(k \log^\varepsilon k)$ -space, for arbitrary small ε , and whose query time is $O(\log \log k)$ [3].

Algorithm's Complexity: Our preprocessing algorithm's time is $O(n \log n)$ to produce the optimal bridges, and $O(k \log k)$ for the orthogonal range query preprocessing, where k is the number of optimal bridges. This number may be as large as $O(n \log n)$. Thus the total preprocessing algorithm time is $O(n \log^2 n)$ in the worst case, with $O(\log \log n)$ query time. The space is dominated by the space needed for the orthogonal range query data structure of [3] for the $O(n \log n)$ points, which is $O(n \log^{1+\varepsilon} n)$.

We have, therefore, proven the third part of Theorem \square

References

1. Apostolico, A., Preparata, F.P.: Optimal off-line detection of repetitions in a string. *Theoretical Computer Science* 22, 297–315 (1983)
2. Berkman, O., Breslauer, D., Galil, Z., Schieber, B., Vishkin, U.: Highly parallelizable problems. In: Proc. 21st ACM Symposium on Theory of Computation, pp. 309–319 (1989)
3. Chan, T.M., Larsen, K.G., Pătrașcu, M.: Orthogonal range searching on the ram, revisited. In: Proc. 27th ACM Symposium on Computational Geometry (SoCG), pp. 1–10 (2011)
4. Cormode, G., Muthukrishnan, S.: Substring compression problems. In: Proc. 16th annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 321–330 (2005)
5. Farach, M.: Optimal suffix tree construction with large alphabets. In: Proc. 38th IEEE Symposium on Foundations of Computer Science, pp. 137–143 (1997)
6. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestor. *Journal of Computer and System Science* 13, 338–355 (1984)
7. Iacono, J., Özkan, Ö.: Mergeable Dictionaries. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, Part I, vol. 6198, pp. 164–175. Springer, Heidelberg (2010)
8. Kärkkäinen, J., Sanders, P.: Simple Linear Work Suffix Array Construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 943–955. Springer, Heidelberg (2003)
9. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
10. Keller, O., Kopelowitz, T., Landau, S., Lewenstein, M.: Generalized Substring Compression. In: Kucherov, G., Ukkonen, E. (eds.) CPM 2009 Lille. LNCS, vol. 5577, pp. 26–38. Springer, Heidelberg (2009)
11. Landau, G.M., Vishkin, U.: Fast parallel and serial approximate string matching. *Journal of Algorithms* 10(2), 157–169 (1989)
12. Lempel, A., Ziv, J.: On the complexity of finite sequences. *IEEE Transactions on Information Theory* 22, 75–81 (1976)
13. McCreight, E.M.: A space-economical suffix tree construction algorithm. *J. of the ACM* 23, 262–272 (1976)

14. Ukkonen, E.: On-line construction of suffix trees. *Algorithmica* 14, 249–260 (1995)
15. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and implementation of an efficient priority queue. *Mathematical Systems Theory* 10, 99–127 (1977)
16. Weiner, P.: Linear pattern matching algorithm. In: Proc. 14 IEEE Symposium on Switching and Automata Theory, pp. 1–11 (1973)
17. Yuan, H., Atallah, M.J.: Data structures for range minimum queries in multidimensional arrays. In: Proc. 21st ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 150–160 (2010)

Computing Knapsack Solutions with Cardinality Robustness

Naonori Kakimura*, Kazuhisa Makino*, and Kento Seimi

Department of Mathematical Informatics,
University of Tokyo, Tokyo 113-8656, Japan

{kakimura,makino,kento_seimi}@mist.i.u-tokyo.ac.jp

Abstract. In this paper, we study the robustness over the cardinality variation for the knapsack problem. For the knapsack problem and a positive number $\alpha \leq 1$, we say that a feasible solution is α -robust if, for any positive integer k , it includes an α -approximation of the maximum k -knapsack solution, where a k -knapsack solution is a feasible solution that consists of at most k items.

In this paper, we show that, for any $\varepsilon > 0$, the problem of deciding whether the knapsack problem admits a $(\nu + \varepsilon)$ -robust solution is weakly NP-hard, where ν denotes the rank quotient of the corresponding knapsack system. Since the knapsack problem always admits a ν -robust knapsack solution [7], this result provides a sharp border for the complexity of the robust knapsack problem. On the positive side, we show that a max-robust knapsack solution can be computed in pseudo-polynomial time, and present a fully polynomial time approximation scheme (FPTAS) for computing a max-robust knapsack solution.

1 Introduction

The classical *knapsack problem* (KP) is defined as follows: We are given a set of *items* $E = \{1, \dots, n\}$ with *price* p_i and *weight* w_i for items $i \in E$, and the capacity $C \in \mathbb{Z}_+$. The aim is to find a subset X of E that maximizes the total price of X subject to the knapsack constraint, i.e., the total weight of X is bounded by C . We denote an instance of the knapsack problem by a quadruple $I = (E, p, w, C)$.

The knapsack problem is one of the most well-studied combinatorial optimization problems. It is known that the knapsack problem is weakly NP-hard [6], but pseudo-polynomially solvable through dynamic programming [2][14]. In addition, KP admits a *fully polynomial time approximation scheme* (FPTAS), i.e., for any $\varepsilon > 0$, a $(1 - \varepsilon)$ -approximate solution can be computed in time polynomial in the input length and $1/\varepsilon$ [9][13][16][18]. See e.g., [14][20] for surveys on the knapsack problems.

* This work was partially supported by Grant-in-Aid for Scientific Research and by Global COE Program “The research and training center for new development in mathematics,” MEXT, Japan.

A natural extension of KP is the *k-item knapsack problem* (*k-KP*) [11], which is a KP in which we are additionally given a cardinality constraint that bounds the number of selected items to be at most k . That is, given a knapsack instance $I = (E, p, w, C)$ and a positive integer k , the *k-KP* is the following problem:

$$\begin{aligned} k\text{-KP} : & \text{maximize } p(X) \\ & \text{subject to } w(X) \leq C, \\ & |X| \leq k, \end{aligned}$$

where for a vector $u \in \mathbb{R}^E$, we define $u(X) = \sum_{i \in X} u_i$ for $X \subseteq E$. A subset X with $w(X) \leq C$ is called a *knapsack solution*, and a *k-knapsack solution* if the size is at most k in addition. If k is sufficiently large, e.g., $k = n$, then *k-KP* coincides with KP. Also, *k-KP* can be seen as a special case of the two-dimensional knapsack problem. The *k-item knapsack problem* is also NP-hard, but the classical dynamic programming, PTAS, and FPTAS for KP are all adapted to *k-KP* [11].

For *k-KP*, imagine the case where we do not know an integer k in advance. In such an uncertain situation, it is natural to keep a knapsack solution which has a good approximation for *all k-KPs*. Such solution is called *robust*, where we will give more precise definition as below. For a knapsack instance I , let O_k denote an optimal *k-knapsack solution*, and for a knapsack solution $X = \{e_1, \dots, e_{|X|}\}$ with $p_{e_1} \geq \dots \geq p_{e_{|X|}}$, define

$$p_{\leq k}(X) = \sum_{i \leq k} p_{e_i}, \quad k = 1, 2, \dots, |E|.$$

For a real number α with $0 < \alpha \leq 1$, we say that a knapsack solution X is α -robust if $p_{\leq k}(X) \geq \alpha \cdot p(O_k)$ for any size k . We also say that a knapsack instance I is α -robust if it has an α -robust knapsack solution. The main purpose of this paper is to find the maximum α so that a given knapsack instance I is α -robust. A knapsack solution with the maximum α is called *max-robust*.

The robustness was first introduced by Hassin and Rubinstein [7] for the *maximum weight independent problem*, which is the problem of maximizing a linear function over an independence system. Here, an *independence system* is a family \mathcal{F} of subsets in E with $\emptyset \in \mathcal{F}$ and the *hereditary property*, that is, if $X \subseteq Y \in \mathcal{F}$ then $X \in \mathcal{F}$, and it includes as a special case a variety of combinatorial objects in graphs and hypergraphs such as matchings, stable sets, and matroids. The α -robustness for independence systems is a natural generalization of the greedy property of matroids, since the greedy algorithm for the maximum weight independent problem finds a 1-robust solution if \mathcal{F} is a matroid [3][21]. For a general independence system \mathcal{F} , Hassin and Rubinstein [7] proved that a greedy solution is $\nu(\mathcal{F})$ -robust, where $\nu(\mathcal{F})$ is the *rank quotient* defined in Section 2. Moreover, they showed that the maximum matching problem, i.e., when \mathcal{F} arises from the family of matchings in a graph, admits a $1/\sqrt{2}$ -robust solution, and that $1/\sqrt{2}$ -robustness is the best possible for the maximum matching problem. Fujita, Kobayashi, and Makino [4] extended their matching result to the matroid intersection problem. Recently, Kakimura and Makino [11] showed the existence of a highly robust solution for general independence systems, which includes the

results of both matchings and matroid intersections. The robustness was also studied for several combinatorial optimization problems such as coloring and subgraph problems [5, 8]. Another concept similar to the robustness, called the *incremental problems*, has been investigated for covering problems in connection with online algorithms [17, 19].

Given an instance I of the knapsack problem, the family of knapsack solutions forms an independence system, called the *knapsack system*. By Hassin and Rubinstein [7], we can compute a $\nu(I)$ -robust solution in polynomial time, where $\nu(I)$ denotes the rank quotient of the knapsack system that corresponds to I . It is, however, *not* known whether we can efficiently compute a knapsack solution with maximum robustness, or even, with *better* robustness.

In this paper, we show that for any constant $\varepsilon > 0$, it is weakly NP-hard to decide whether a given instance I of the knapsack problem has a $(\nu(I) + \varepsilon)$ -robust solution. By [7], this gives us a sharp border for the complexity of the robust knapsack problem.

We then design two kinds of pseudo-polynomial time algorithms for finding a max-robust knapsack solution. The first one is a simple extension of dynamic programming algorithms for the knapsack problem, and requires $O(n^2CU)$ time, where $U = \sum_i p_i$. We note that the time complexity depends on the knapsack capacity C , and hence it seems difficult to design an FPTAS based on this algorithm. The second one is based on a procedure to find an α -robust solution for a given α . This procedure, together with a binary search for α , leads to an $O(n^2U \log U)$ algorithm for finding a max-robust knapsack solution. In addition, the second algorithm leads to an FPTAS by using a cost rounding technique. That is, for a knapsack instance I and $\varepsilon > 0$, it returns a $(1 - \varepsilon)\alpha(I)$ -robust solution in time polynomial in the input length and $1/\varepsilon$, where $\alpha(I)$ denotes the maximum robustness for I . We note that the max-robust knapsack problem has complexity properties similar to the classical knapsack problem, where it is *not* generally true, for example, the max-robust matching problem is strongly NP-hard [4], while the matching problem is polynomially solvable. We can also say that our result for FPTAS is a first positive result for the max-robust problems, except for matroids.

This paper is organized as follows. In Section 2, we present the NP-hardness result for the robust knapsack problem. Sections 3 and 4 describe two kinds of pseudo-polynomial time algorithms for finding a max-robust knapsack solution, respectively, and in Section 5, we design an FPTAS based on the algorithm in Section 4. Due to the space limitation, we omit proofs of some lemmas and theorems, which can be found in the full version of this paper [12].

2 NP-Hardness for the Max-Robust Knapsack Problem

In this section, we prove the NP-hardness for finding max-robust knapsack solutions. We first provide some notations. Let \mathcal{F} be an independence system. For $J \subseteq E$, let $\rho(J)$ and $\gamma(J)$ be the minimum and maximum sizes of maximal independent sets in \mathcal{F} contained in J , respectively. Define the *rank quotient* $\nu(\mathcal{F})$ to be

$$\nu(\mathcal{F}) = \min_{J \subseteq E} \frac{\rho(J)}{\gamma(J)}.$$

For a knapsack instance $I = (E, p, w, C)$, let $\nu(I)$ denote the rank quotient of the knapsack system \mathcal{F} corresponding to I , i.e., $\mathcal{F} = \{X \subseteq E \mid w(X) \leq C\}$. Jenkyns [10] and Korte and Hausmann [15] showed the greedy algorithm finds a $\nu(\mathcal{F})$ -approximate solution for the maximum weight independent problem. Hassin and Rubinstein [7] proved a greedy solution is in fact $\nu(\mathcal{F})$ -robust.

Theorem 1 (Hassin and Rubinstein [7]). *Every knapsack problem admits a ν -robust solution, where ν is the rank quotient.*

It, however, turns out to be NP-hard to decide whether or not a given knapsack instance has better than a greedy solution.

Theorem 2. *Let $\varepsilon < 1$ be a fixed positive constant. Problem ε -ROBUSTKNAPSACK defined as below is NP-hard.*

Problem: ε -ROBUSTKNAPSACK

Instance: A knapsack instance $I = (E, p, w, C)$.

Question: Is there a knapsack solution X whose robustness is $\nu(I) + \varepsilon$?

We here give the proof outline of Theorem 2. The details can be found in [12]. We will show that the well-known NP-complete problem PARTITION [6] can be reduced to the ε -robust knapsack problem. The problem PARTITION is as follows: Given n nonnegative integers $a_1 \geq \dots \geq a_n$, find $X \subseteq \{1, \dots, n\}$ such that $\sum_{i \in X} a_i = A/2$, where $A = \sum_{i=1}^n a_i$. The following problem is also NP-complete for a fixed positive constant δ , which easily follows from PARTITION.

Problem: δ -PARTITION'

Instance: n nonnegative integers $a_1 \geq \dots \geq a_n$. Let $A = \sum_{i=1}^n a_i$.

Question: Is there any X such that $\sum_{i \in X} a_i = \delta A$ and $|X| = \delta n$?

Define $\delta = (1 - \varepsilon)/2$. Let $a_1 \geq \dots \geq a_n$ with $A = \sum_{i=1}^n a_i$ be an instance I of δ -PARTITION'. Note that we can take a sufficiently large n so that n is larger than a constant $g(\varepsilon)$ for some function g . Define $b_i = a_i + na_1$ for $i = 1, \dots, n$ and $B = \sum b_i = A + n^2 a_1$, and the obtained instance $b_1 \geq \dots \geq b_n$ of δ -PARTITION' is denoted by I' . Then I has a solution of δ -PARTITION' if and only if so does I' . We construct an instance $J = (E, p, w, C)$ of ε -ROBUSTKNAPSACK as follows: Let $E = \{0, 1, \dots, n\}$, and define

$$\begin{aligned} p_0 &= \varepsilon B, \quad w_0 = (1 - \delta)B, \\ p_i &= w_i = b_i \quad (i = 1, \dots, n), \\ C &= B, \text{ and } \alpha = \delta + \varepsilon. \end{aligned}$$

Note that $p_0 > \alpha^{-1} p_1 > p_1 \geq \dots \geq p_n$ and $w_0 \geq w_1 \geq \dots \geq w_n$, and that $\nu(J) = \delta$. This implies that if X is α -robust then X has to contain the item 0. In addition, we can show that X is an α -robust solution of J if and only if $X \setminus \{0\}$ is a solution of I' , which implies Theorem 2.

3 Simple Pseudo-Polynomial Time Algorithm

In this section we propose a pseudo-polynomial time algorithm for finding a max-robust knapsack solution. Given a knapsack instance $I = (E, p, w, C)$, let O_k be an optimal solution of the k -knapsack problem with respect to I . We may assume that p_i 's are sorted in nonincreasing order, and $p_i \leq C$ for any $i \in E$.

For $X \subseteq E$ with $X \neq \emptyset$, define $\alpha(X)$ and $\beta(X)$ as follows:

$$\alpha(X) = \min_{1 \leq k \leq n} \frac{p_{\leq k}(X)}{p(O_k)} \quad \text{and} \quad \beta(X) = \min_{1 \leq k \leq |X|} \frac{p_{\leq k}(X)}{p(O_k)}.$$

Note that $\alpha(X)$ means that X is $\alpha(X)$ -robust, but not $(\alpha(X) + \varepsilon)$ -robust for any $\varepsilon > 0$. The function β is related to the robustness as follows.

Proposition 1. A knapsack solution $X (\neq \emptyset)$ satisfies that $\alpha(X) = \min \left\{ \beta(X), \frac{p(X)}{p(O_n)} \right\}$.

Proof. By the definition of α we obtain

$$\alpha(X) = \min \left\{ \beta(X), \min_{|X|+1 \leq k \leq n} \frac{p(X)}{p(O_k)} \right\} = \min \left\{ \beta(X), \frac{p(X)}{p(O_n)} \right\},$$

where the last equation holds because of $p(O_{|X|+1}) \leq \dots \leq p(O_n)$. \square

For $1 \leq i, \ell \leq n$, $0 \leq P \leq U$, and $0 \leq W \leq C$, define

$$A(i, \ell, P, W) = \max \{ \beta(X) \mid X \in \mathcal{S}(i, \ell, P, W) \}, \text{ where}$$

$$\mathcal{S}(i, \ell, P, W) = \{ X \mid X \subseteq \{1, \dots, i\}, |X| = \ell, p(X) = P, w(X) \leq W \}.$$

Then $A(i, \ell, P, W)$ is represented recursively as follows.

Lemma 1. We can express $A(i, \ell, P, W)$ to be as follows:

(i) For $i = 1$,

$$A(1, \ell, P, W) = \begin{cases} 1 & \text{if } \ell = 1, P = p_1, \text{ and } W \geq w_1, \\ -\infty & \text{otherwise.} \end{cases}$$

(ii) For $2 \leq i \leq n$ and $\ell = 1$,

$$A(i, 1, P, W) = \begin{cases} \max \left\{ A(i-1, 1, P, W), \frac{p_i}{p_1} \right\} & \text{if } P = p_i \text{ and } W \geq w_i, \\ A(i-1, 1, P, W) & \text{otherwise.} \end{cases}$$

(iii) For $2 \leq i, \ell \leq n$, if $P \geq p_i$ and $W \geq w_i$ then

$$A(i, \ell, P, W) = \max \left\{ \begin{array}{l} A(i-1, \ell, P, W), \\ \min \left\{ A(i-1, \ell-1, P-p_i, W-w_i), \frac{P}{p(O_\ell)} \right\} \end{array} \right\},$$

and otherwise, $A(i, \ell, P, W) = A(i-1, \ell, P, W)$.

With the recursive equations in Lemma 1, we can design a dynamic programming algorithm to obtain $A(i, j, P, W)$ for all (i, j, P, W) 's. This can be done in $O(n^2CU)$ time, where $U = \sum_i p_i$. Let me remark that this dynamic programming needs the optimal value of k -KP for all sizes k , which can be obtained in $O(n^2U)$ time by standard dynamic programming [1]. It follows from Proposition 1 that the maximum robustness $\alpha(I)$ is equal to

$$\alpha(I) = \max_{\ell, P} \min \left\{ A(n, \ell, P, C), \frac{P}{p(O_n)} \right\}. \quad (1)$$

Thus one can obtain $\alpha(I)$ from (1) in pseudo-polynomial time.

Let me notice that a max-robust knapsack solution can also be obtained by the above dynamic programming. Indeed, in each step of the above recursion, $A(i, \ell, P, W)$ is obtained from $A(i - 1, \ell, P, W)$ or $A(i - 1, \ell - 1, P - p_i, W - w_i)$ with a new item i . Hence, associated with $A(i, \ell, P, W)$ we store a pointer to the previous entry and a pointer to the item i if we add i . Then, by traversing these pointers from $A(n, \ell, P, C)$ with $\alpha(I) = \min\{A(n, \ell, P, C), \frac{P}{p(O_n)}\}$, one can obtain an $\alpha(I)$ -robust knapsack solution in linear time.

Theorem 3. *A dynamic programming algorithm based on Lemma 1 finds a max-robust knapsack solution in $O(n^2CU)$ time, where $n = |E|$ and $U = \sum_i p_i$.*

4 Improved Pseudo-Polynomial Time Algorithm

In this section, we present another pseudo-polynomial time algorithm, whose time complexity does not depend on the item weights. We first develop a procedure EXIST so that, for a given knapsack instance $I = (E, p, w, C)$ and a given α , it decides whether I is α -robust or not and returns an α -robust solution if exists. Using this procedure, we design an algorithm for finding a max-robust solution with the aid of a binary search for α . Recall that O_k denotes a maximum k -knapsack solution for a positive integer k , and that p_i 's are sorted in nonincreasing order.

For $1 \leq i, \ell \leq n$ and $1 \leq P \leq U$, let

$$\mathcal{T}(i, \ell, P) = \{X \mid X \subseteq \{1, \dots, i\}, |X| = \ell, p(X) = P, \beta(X) \geq \alpha\}.$$

We first observe the following proposition.

Proposition 2. *Let $2 \leq i, \ell \leq n$. If $P < \alpha P(O_\ell)$ then $\mathcal{T}(i, \ell, P) = \emptyset$, and, otherwise, i.e., if $P \geq \alpha P(O_\ell)$, it holds that*

$$\mathcal{T}(i, \ell, P) = \begin{cases} \mathcal{T}(i - 1, \ell, P) \cup \{Z \cup \{i\} \mid Z \in \mathcal{T}(i - 1, \ell - 1, P - p_i)\} & \text{if } P \geq p_i, \\ \mathcal{T}(i - 1, \ell, P) & \text{if } P < p_i, \end{cases}$$

Proof. If $P < \alpha P(O_\ell)$ then $\mathcal{T}(i, \ell, P) = \emptyset$ by the definition of \mathcal{T} . Suppose that $P \geq \alpha P(O_\ell)$. If $P < p_i$ then $i \notin X$ for any $X \in \mathcal{T}(i, \ell, P)$, and hence $\mathcal{T}(i, \ell, P) =$

$\mathcal{T}(i-1, \ell, P)$ holds. Next assume that $P \geq p_i$. For $X \in \mathcal{T}(i, \ell, P)$, if $i \notin X$ then $X \in \mathcal{T}(i-1, \ell, P)$, and otherwise, we have

$$\beta(X) = \min \left\{ \beta(X \setminus \{i\}), \frac{P}{P(O_\ell)} \right\},$$

which follows from that p_i 's are nonincreasing order. Since $P \geq \alpha p(O_\ell)$, this implies that $\beta(X) \geq \alpha$ if and only if $X \setminus \{i\} \in \mathcal{T}(i-1, \ell-1, P-p_i)$. Thus the statement holds. \square

For $1 \leq i, \ell \leq n$ and $0 \leq P \leq U$, let us define

$$B(i, \ell, P) = \min \{w(X) \mid X \in \mathcal{T}(i, \ell, P)\}.$$

We show the following recursive equation by Proposition 2.

Lemma 2. *We can express $B(i, \ell, P)$ as follows:*

(i) *For $i = 1$,*

$$B(1, \ell, P) = \begin{cases} w_1 & \text{if } \ell = 1 \text{ and } P = p_1 \\ \infty & \text{otherwise.} \end{cases}$$

(ii) *For $2 \leq i \leq n$ and $\ell = 1$,*

$$B(i, 1, P) = \begin{cases} \min\{B(i-1, 1, P), w_i\} & \text{if } p_i \geq \alpha p_1 \text{ and } p_i = P \\ B(i-1, 1, P) & \text{otherwise.} \end{cases}$$

(iii) *For $2 \leq i, \ell \leq n$, if $P < \alpha P(O_\ell)$ then $B(i, \ell, P) = \infty$, and otherwise, i.e., if $P \geq \alpha P(O_\ell)$,*

$$B(i, \ell, P) = \begin{cases} \min \{B(i-1, \ell, P), B(i-1, \ell-1, P-p_i) + w_i\} & \text{if } P \geq p_i \\ B(i-1, \ell, P) & \text{if } P < p_i \end{cases}$$

Proof. By the definition of B , (i) and (ii) hold. Let $2 \leq i, \ell \leq n$. By Proposition 2, it is not difficult to see the cases where $P < \alpha P(O_\ell)$ and where $P \geq \alpha p(O_\ell)$ and $P < p_i$. If $P \geq \alpha p(O_\ell)$ and $P \geq p_i$, Proposition 2 implies that $B(i, \ell, P)$ is the minimum of $\min\{w(Y) \mid Y \in \mathcal{T}(i-1, \ell, P)\}$ and $\min\{w(Z \cup \{i\}) \mid Z \in \mathcal{T}(i-1, \ell-1, P-p_i)\}$. The former is equal to $B(i-1, \ell, P)$, while the latter is $B(i-1, \ell-1, P-p_i) + w_i$. Thus the lemma holds. \square

Procedure EXIST can be designed as follows. In the first step, we compute the optimal values of k -KP for all $k = 1, \dots, n$ in $O(n^2U)$ time [1], where $U = \sum_i p_i$. With the recursive equation in Lemma 2, we can design a dynamic programming algorithm to obtain $B(i, \ell, P)$ for all (i, ℓ, P) 's. By Proposition 1, there exists an α -robust solution X with $|X| = \ell$ and $p(X) = P$ if and only if $B(n, \ell, P) \leq C$ and $P \geq \alpha p(O_n)$. Thus we can decide whether there exists an α -robust solution or not from all $B(i, j, P)$'s. Moreover, an α -robust knapsack solution can be found in a similar way to the algorithm of Theorem 3.

Theorem 4. Procedure EXIST decides whether there exists an α -robust knapsack solution or not and returns one if exists. Its time complexity is $O(n^2U)$, where $n = |E|$ and $U = \sum_i p_i$.

Using this procedure, we can find a max-robust knapsack solution with the aid of a binary search.

Theorem 5. We can find a max-robust knapsack solution in $O(n^2U \log U)$ time, where $n = |E|$ and $U = \sum_i p_i$.

Proof. The algorithm is described as follows. Let $I = (E, p, w, C)$ be a given knapsack instance. First initialize $\alpha = 1/2$. We repeat the following for $r = 1, 2, \dots$: Apply Procedure EXIST to find an α -robust knapsack solution for I . If such a solution exists, set $\alpha = \alpha + 2^{-r-1}$, and otherwise set $\alpha = \alpha - 2^{-r-1}$.

We claim that the number of the repetition is at most $\lceil 2 \log_2 U \rceil$. Indeed, for two knapsack solutions X, Y with $\alpha(X) > \alpha(Y)$, the difference is

$$\alpha(X) - \alpha(Y) = \min_k \frac{p_{\leq k}(X)}{p(O_k)} - \min_k \frac{p_{\leq k}(Y)}{p(O_k)} > \frac{1}{U^2}.$$

Hence it suffices to repeat $\lceil 2 \log_2 U \rceil$ times to distinguish any two knapsack solutions with different robustness. Since EXIST requires $O(n^2U)$ time by Theorem 4, the total time complexity is $O(n^2U \log U)$. \square

5 Fully Polynomial Time Approximation Scheme

In this section, we propose an FPTAS for the max-robust knapsack problem. This scheme is obtained from the second pseudo-polynomial time algorithm presented in Section 4 by rounding prices.

We first describe the outline of the scheme, denoted by **Approx**, as below.

Approximation scheme for the max-robust knapsack problem: **Approx**
Input: A knapsack instance $I = (E, p, w, C)$ and $0 < \varepsilon < 1$.

Step 1: (Rounding) Define $p'_i = \left\lfloor \frac{p_i}{N} \right\rfloor$ for $i = 1, \dots, n$, where

$$N = \frac{p_1 \delta}{n} \quad \text{and} \quad \delta = 1 - \sqrt{1 - \varepsilon}. \quad (2)$$

Let $I' = (E, p', w, C)$ be the rounded instance.

Step 2: (Pseudo-polynomial time algorithm) Execute the algorithm in Theorem 5 for I' , and return a knapsack solution X which is max-robust with respect to I' .

The main purpose of this section is to show **Approx** is an FPTAS.

Theorem 6. Scheme **Approx** is an FPTAS for finding a max-robust knapsack solution. That is, it returns a $(1 - \varepsilon)\alpha(I)$ -robust knapsack solution in time polynomial in the input length and $1/\varepsilon$.

To prove Theorem 6, we first discuss the time complexity.

Lemma 3. *Scheme Approx runs in $O(n^4 \frac{1}{\varepsilon} \log(\frac{n}{\varepsilon}))$ time, where n is the number of items.*

Proof. For $i = 1, \dots, n$, it holds

$$p'_i \leq \frac{p_i}{N} = \frac{p_i}{p_1} \frac{n}{\delta} \leq \frac{n}{\delta},$$

where the last inequality follows from $p_1 \geq p_i$ for all i 's. Note that $1/\delta \leq 2/\varepsilon$, and hence $p'_i \leq 2n/\varepsilon$ holds. Therefore, it follows from Theorem 5 that Step 2 requires $O(n^4 \frac{1}{\varepsilon} \log(\frac{n}{\varepsilon}))$, and thus so is the total time complexity. \square

We next estimate the approximation factor of the output. For that purpose, we show the following lemma. For the rounded instance I' , let O'_k be an optimal k -knapsack solution with respect to I' . For a knapsack solution X with $X \neq \emptyset$, we denote by $\alpha'(X)$ the robustness of X with respect to the rounded instance I' , that is,

$$\alpha'(X) = \min_{1 \leq k \leq n} \frac{p'_{\leq k}(X)}{p'(O'_k)}.$$

Lemma 4. *The following statements hold.*

- (a) *Let Y be a max-robust solution of I' . Then $\alpha(Y) \geq \sqrt{1 - \varepsilon} \cdot \alpha'(Y)$.*
- (b) *Let Z be a max-robust solution of I . Then $\alpha'(Z) \geq \sqrt{1 - \varepsilon} \cdot \alpha(Z)$.*

Lemma 4 implies the following lemma, which, together with Lemma 3, completes the proof of Theorem 6.

Lemma 5. *Scheme Approx returns a $(1 - \varepsilon)\alpha(I)$ -robust knapsack solution.*

Proof. Scheme Approx returns a max-robust solution X with respect to the rounded instance I' . Let Z be a max-robust solution with respect to I . Then Lemma 4 (b) implies that $\sqrt{1 - \varepsilon} \cdot \alpha(Z) \leq \alpha'(Z) \leq \alpha'(X)$ by the maximum robustness of X . Therefore, it follows from Lemma 4 (a) that

$$\alpha(X) \geq \sqrt{1 - \varepsilon} \cdot \alpha'(X) \geq (1 - \varepsilon)\alpha(Z).$$

Thus X is $(1 - \varepsilon)\alpha(I)$ -robust. \square

References

1. Caprara, A., Kellerer, H., Pferschy, U., Pisinger, D.: Approximation algorithms for knapsack problems with cardinality constraints. European J. Oper. Res. 123, 333–345 (2000)
2. Dantzig, G.B.: Discrete-variable extremum problems. Ope. Res. 5, 266–277 (1957)
3. Edmonds, J.: Matroids and the greedy algorithm. Math. Program. 1, 127–136 (1971)

4. Fujita, R., Kobayashi, Y., Makino, K.: Robust Matchings and Matroid Intersections. In: de Berg, M., Meyer, U. (eds.) *ESA 2010. LNCS*, vol. 6347, pp. 123–134. Springer, Heidelberg (2010)
5. Fukunaga, T., Halldórsson, M., Nagamochi, H.: Robust cost colorings. In: Proc. SODA 2008, pp. 1204–1212 (2008)
6. Garey, M.R., Johnson, D.E.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York (1979)
7. Hassin, R., Rubinstein, S.: Robust matchings. *SIAM J. Discrete Math.* 15, 530–537 (2002)
8. Hassin, R., Segev, D.: Robust subgraphs for trees and paths. *ACM Trans. Algorithms* 2, 263–281 (2006)
9. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. *ACM* 22(4), 463–468 (1975)
10. Jenkyns, T.A.: The efficacy of the “greedy” algorithm. In: Proc. 7th Southeastern Conference on Combinatorics, Graph Theory and Computing, pp. 341–350 (1976)
11. Kakimura, N., Makino, K.: Robust Independence Systems. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) *ICALP 2011. LNCS*, vol. 6755, pp. 367–378. Springer, Heidelberg (2011)
12. Kakimura, N., Makino, K., Seimi, K.: Computing Knapsack Solutions with Cardinality Robustness, Mathematical Engineering Technical Reports METR 2011-31, University of Tokyo (2011)
13. Kellerer, H., Pferschy, U.: A new fully polynomial approximation scheme for the knapsack problem. *J. Comb. Optim.* 3, 59–71 (1999)
14. Kellerer, H., Pferschy, U., Pisinger, D.: *Knapsack Problems*. Springer, Berlin (2004)
15. Korte, B., Hausmann, D.: An analysis of the greedy heuristic for independence systems. *Ann. Discrete Math.* 2, 65–74 (1978)
16. Lawler, E.G.: Fast approximation algorithms for knapsack problems. *Math. Oper. Res.* 4, 339–356 (1979)
17. Lin, G., Nagarajan, C., Rajarama, R., Williamson, D.: A general approach for incremental approximation and hierarchical clustering. *SIAM J. Comput.* 39, 3633–3669 (2010)
18. Magazine, M.J., Oguz, O.: A fully polynomial approximation algorithm for the 0 – 1 knapsack problem. *European J. Oper. Res.* 8, 270–273 (1981)
19. Mettu, R.R., Plaxton, C.G.: The online median problem. *SIAM J. Comput.* 32, 816–832 (2003)
20. Pisinger, D., Toth, P.: *Handbook of Combinatorial Optimization*, pp. 1–89. Kluwer, Norwell (1998)
21. Rado, R.: Note on independence relations. *Proc. Lond. Math. Soc.* 7, 300–320 (1957)

Max-Throughput for (Conservative) k -of- n Testing

Lisa Hellerstein*, Özgür Özkan**, and Linda Sellie***

Department of Computer Science and Engineering, Polytechnic Institute of NYU,
Six MetroTech Center, Brooklyn, NY 11201-3840 USA

Abstract. We define a variant of k -of- n testing that we call *conservative* k -of- n testing. We present a polynomial-time, combinatorial algorithm for maximizing the throughput of conservative k -of- n testing, in a parallel setting. This extends previous work of Kodialam and Condon et al. on the parallel pipelined filter ordering problem, which is the special case where $k = 1$ (or $k = n$) [123]. We also consider the problem of maximizing throughput for *standard* k -of- n testing, and describe a polynomial-time algorithm for it based on the ellipsoid method.

1 Introduction

In *standard* k -of- n testing, there are n binary tests, that can be applied to an “item” x . We use x_i to denote the value of the i^{th} test on x , with $x_i = 0$ denoting failure. We treat x as an element of $\{0, 1\}^n$. With probability p_i , $x_i = 1$, and with probability $1 - p_i$, $x_i = 0$. The tests are independent, and we are given p_1, \dots, p_n . We need to determine whether at least k of the n tests on x have a value of 0, by applying the tests sequentially to x . Once we have enough information to determine whether this is the case, that is, *once we have observed k tests with value 0, or $n - k + 1$ tests with value 1*, we do not need to perform further tests. ^[1]

We define *conservative* k -of- n testing the same way, except that we perform tests until we have either observed k tests with value 0, or have performed all n tests. In particular, we do not stop after observing $n - k + 1$ tests with value 1.

There are many applications where k -of- n testing problems arise, including quality testing, medical diagnosis, and database query optimization.

For $k = 1$, standard and conservative k -of- n testing are the same. For $k > 1$, the conservative variant is relevant in a setting where, for items failing fewer than k tests, we need to know *which* tests they failed. For example, in quality testing, we may want to know which tests were failed by items failing fewer than k tests in order to repair the associated defects.

Our focus is on the MAXTHROUGHPUT problem for k -of- n testing. Here the objective is to maximize the throughput of a system for k -of- n testing in a parallel setting where each test is performed by a separate “processor”. In this problem, in addition to

* This research is supported by the NSF Grant CCF-0917153.

** This research supported by US Department of Education grant P200A090157.

*** This research is supported by a CIFellows Project postdoc, sponsored by NSF and the CRA.

¹ In an alternative definition of k -of- n testing, the task is to determine whether at least k of the n tests have a value of 1. Symmetric results hold for this definition.

the probabilities p_i , there is a *rate limit* r_i associated with the processor that performs test i , indicating that the processor can only perform tests on r_i items per unit time.

MAXTHROUGHPUT problems are closely related to MINCOST problems [4][5]. In the MINCOST problem for k -of- n testing, in addition to the probabilities p_i , there is a cost c_i associated with performing the i^{th} test. The goal is to find a testing strategy (i.e. decision tree) that minimizes the expected cost of testing an individual item. There are polynomial-time algorithms for solving the MINCOST problem for standard k -of- n testing [6][7][8][9].

Kodialam was the first to study the MAXTHROUGHPUT k -of- n testing problem, in the special case where $k = 1$ [1]. He gave a $\mathcal{O}(n^3 \log n)$ algorithm for the problem that is combinatorial, but its correctness proof relies on polymatroid theory. Later, Condon et. al. also studied the problem for $k = 1$, calling it “parallel pipelined filter ordering”. They gave an $\mathcal{O}(n^2)$ combinatorial algorithm with a direct correctness proof [3].

In this paper, we extend the previous work by giving a combinatorial algorithm for the MAXTHROUGHPUT problem for conservative k -of- n testing. Our algorithm can be implemented using simple dynamic programming to run in time $\mathcal{O}(kn^2)$, which is $\mathcal{O}(n^2)$ for constant k . We leave for future work the problem of reducing the runtime for non-constant k .

The MAXTHROUGHPUT problem for standard k -of- n testing appears to be fundamentally different from its conservative variant. We leave as an open problem the task of developing a polynomial time *combinatorial* algorithm for this problem. We show that previous techniques can be used to obtain a polynomial-time algorithm based on the ellipsoid method. This approach could also be used to yield an algorithm, based on the ellipsoid method, for the conservative variant.

A more complete version of this paper can be found in [10]. It includes pseudocode, additional examples, and missing proofs.

2 Related Work

Deshpande and Hellerstein studied the MAXTHROUGHPUT problem for $k = 1$, when there are precedence constraints between tests [5]. They also showed a close relationship between the exact MINCOST and MAXTHROUGHPUT problems for k -of- n testing, when $k = 1$. Their results can be generalized to apply to testing of other functions.

Liu et al. [4] presented a generic, linear-programming based, method for converting an approximation algorithm for a MINCOST problem, into an approximation algorithm for a MAXTHROUGHPUT problem. Their results are not applicable to this paper, where we consider only exact algorithms.

Polynomial-time algorithms for the MINCOST problem for standard k -of- n testing were given by by Salloum, Breuer, Ben-Dov, and Chang et al. [6][7][8][9][11].

The problem of how to best order a sequence of tests has been studied in many different contexts and models. See e.g. [4] and [3] for a discussion of related work on the filter-ordering problem (i.e. the MINCOST problem for $k = 1$) and its variants, and [12] for a general survey of sequential testing of functions.

3 Problem Definitions

A k -of- n testing strategy for tests $1, \dots, n$ is a binary decision tree T that computes the k -of- n function, $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where $f(x) = 1$ if and only if x contains fewer than k 0's. Each node of T is labeled by a variable x_i . The left child of a node labeled with x_i is associated with $x_i = 0$ (i.e., failing test i), and the right child with $x_i = 1$ (i.e., passing test i). Each $x \in \{0, 1\}^n$ corresponds to a root-to-leaf path in the usual way, and the label at the leaf is $f(x)$.

A k -of- n testing strategy T is *conservative* if, for each root-to-leaf path leading to a leaf labeled 1, the path contains exactly n non-leaf nodes, each labeled with a distinct variable x_i .

Given a permutation π of the n tests, define $T_k^c(\pi)$ to be the conservative strategy described by the following procedure: *Perform the tests in order of permutation π until at least k 0's have been observed, or all tests have been performed, whichever comes first. Output 0 in the first case, and 1 in the second.*

Similarly, define $T_k^s(\pi)$ to be the following standard k -of- n testing strategy: *Perform the tests in order of permutation π until at least k 0's have been observed, or until $n - k + 1$ 1's have been observed, whichever comes first. Output 0 in the first case, and 1 in the second.*

Each test i has an associated probability p_i , where $0 < p_i < 1$. Let D_p denote the product distribution on $\{0, 1\}^n$ defined by the p_i 's; that is, if x is drawn from D_p , then $\forall i, \Pr[x_i = 1] = p_i$ and the x_i are independent. We use $x \sim D_p$ to denote a random x drawn from D_p . In what follows, when we use an expression of the form $\text{Prob}[\dots]$ involving x , we mean the probability with respect to D_p .

The MINCOST problem. In the MINCOST problem for standard k -of- n testing, we are given n probabilities p_i and costs $c_i > 0$, for $i \in \{1, \dots, n\}$, associated with the tests. The goal is to find a k -of- n testing strategy T that minimizes the expected cost of applying T to a random item $x \sim D_p$. The cost of applying a testing strategy T to an item x is the sum of the costs of the tests along the root-to-leaf path for x in T . In the MINCOST problem for conservative k -of- n testing, the goal is the same, except that we are restricted to finding a *conservative* testing strategy.

For example, consider the MINCOST 2-of-3 problem with probabilities $p_1 = p_2 = 1/2$, $p_3 = 1/3$ and costs $c_1 = 1$, $c_2 = c_3 = 2$. A standard testing strategy for this problem can be described procedurally as follows: *Given item x , begin by performing test 1. If $x_1 = 1$, follow strategy $T_2^s(\pi_1)$, where $\pi_1 = (2, 3)$. Else if $x_1 = 0$, follow strategy $T_1^s(\pi_2)$, where $\pi_2 = (3, 2)$.* Under the above strategy, which can be shown to be optimal, evaluating $x = (0, 0, 1)$ costs 5, and evaluating $x' = (1, 1, 0)$ costs 3. The expected cost of applying this strategy to a random item $x \sim D_p$ is $3\frac{5}{6}$.

Because the optimal testing strategy may be a tree of exponential size, algorithms for the MINCOST problem may output it in a compact form.

The MAXTHROUGHPUT problem. The MAXTHROUGHPUT problem for k -of- n testing is a natural generalization of the MAXTHROUGHPUT problem for 1-of- n testing, first studied by Kodialam [1]. We give basic definitions and motivation here. For further information, and for discussion of associated practical problems and issues, see [1][2][3].

In the MAXTHROUGHPUT problem for k -of- n testing, as in the MINCOST problem, we are given the probabilities p_i associated with the tests. Instead of costs c_i for the tests, we are given *rate limits* $r_i > 0$. The MAXTHROUGHPUT problem arises in the following context. There is an (effectively infinite) stream of items x that need to be tested. Every item x must be assigned a strategy T that will determine which tests are performed on it. Different items may be assigned to different strategies. Each test is performed by a separate “processor”, and the processors operate in parallel. (Imagine a factory testing setting.) Item x is sent from processor to processor for testing, according to its strategy T . Each processor can only test one item at a time. We view the problem of assigning items to strategies as a flow-routing problem.

Processor O_i performs test i . It has rate limit (capacity) r_i , indicating that it can only process r_i items x per unit time.

The goal is to determine how many items should be assigned to each strategy T , per unit time, in order to maximize the throughput of the system, i.e., the number of items that can be processed by the system per unit time. The solution must respect the rate limits of the processors, in that the expected number of items that need to be tested by processor O_i per unit time must not exceed r_i . We assume that tests behave according to expectation: if m items are tested by processor O_i per unit time, then mp_i of them will have the value 1, and $m(1 - p_i)$ will have the value 0.

Let \mathcal{T} denote the set of all k -of- n testing strategies and \mathcal{T}_c denote the set of all conservative k -of- n testing strategies. Formally, the MAXTHROUGHPUT problem for standard k -of- n testing is defined by the linear program (LP) below. The LP defining the MAXTHROUGHPUT problem for conservative k -of- n testing is obtained by replacing the set of k -of- n testing strategies \mathcal{T} by the set of conservative k -of- n testing strategies \mathcal{T}_c .

MAXTHROUGHPUT LP:

Given $r_1, \dots, r_n > 0$ and $p_1, \dots, p_n \in (0, 1)$, find an assignment to the variables z_T , for all $T \in \mathcal{T}$, that maximizes

$$F = \sum_{T \in \mathcal{T}} z_T$$

subject to the constraints:

- (1) $\sum_{T \in \mathcal{T}} g(T, i) z_T \leq r_i$ for all $i \in \{1, \dots, n\}$ and
- (2) $z_T \geq 0$ for all $T \in \mathcal{T}$

where $g(T, i)$ denotes the probability that test i will be performed on an item x that is tested using strategy T , when $x \sim D_p$.

We refer to a feasible assignment to the variables z_T in the above LP as a *routing*. We call constraints of type (1) *rate constraints*. The value of F is the *throughput* of the routing. For $i \in \{1, \dots, n\}$, if $\sum_{T \in \mathcal{T}} g(T, i) z_T = r_i$, we say that the routing *saturates* processor O_i .

We will refer to the MAXTHROUGHPUT problems for standard and conservative k -of- n testing as the “SMT(k) Problem” and “CMT(k) Problem”, respectively.

As a simple example, consider the following CMT(k) Problem (equivalently, SMT(k) Problem) instance, where $k = 1$ and $n = 2$: $r_1 = 1$, $r_2 = 2$, $p_1 = 1/2$, $p_2 = 1/4$. There are only two possible strategies, $T_1(\pi_1)$, where $\pi_1 = (1, 2)$, and $T_1(\pi_2)$, where $\pi_2 = (2, 1)$. Since all flow assigned to $T_1(\pi_1)$ is tested by O_1 , $g(T_1(\pi_1), 1) = 1$; this flow continues on to O_2 only if it passes test 1, which happens with probability $p_1 = 1/2$, so $g(T_1(\pi_1), 2) = 1/2$. Similarly, $g(T_1(\pi_2), 2) = 1$ while $g(T_1(\pi_2), 1) = 1/4$, since $p_2 = 1/4$. Consider the routing that assigns $F_1 = 4/7$ units of flow to strategy $T_1(\pi_1)$, and $F_2 = 12/7$ units to strategy $T_1(\pi_2)$. Then the amount of flow reaching O_1 is $4/7 \times g(T_1(\pi_1), 1) + 12/7 \times g(T_1(\pi_2), 1) = 1$, and the amount of flow reaching O_2 is $4/7 \times g(T_1(\pi_1), 2) + 12/7 \times g(T_1(\pi_2), 2) = 2$. Since $r_1 = 1$ and $r_2 = 2$, this routing saturates both processors. By the results of Condon et al. [3], it is optimal.

4 The Algorithm for the MINCOST Problem

The following well-known, simple algorithm solves the MINCOST problem for standard 1-of- n testing (see e.g. [?]): First, sort the tests in increasing order of the ratio $c_i/(1 - p_i)$. Next, renumber the tests, so that $c_1/(1 - p_1) < c_2/(1 - p_2) < \dots < c_n/(1 - p_n)$. Finally, output the sorted list $\pi = (1, \dots, n)$ of tests, which is a compact representation of the strategy $T_1^s(\pi)$ (which is the same as $T_1^c(\pi)$).

The above algorithm can be applied to the MINCOST problem for conservative k -of- n testing, simply by treating π as a compact representation of the conservative strategy $T_k^c(\pi)$. In fact, that strategy is optimal [14]. It has minimum expected cost among all conservative strategies.

5 The Algorithm for the CMT(k) Problem

We begin with some useful lemmas. The algorithms of Condon et al. for maximizing throughput of 1-of- n testing rely crucially on the fact that saturation of all processors implies optimality. We show that the same holds for conservative k -of- n testing.

Lemma 1. *Let R be a routing for an instance of the conservative k -of- n testing problem. If R saturates all processors, then it is optimal.*

Proof. Each processor O_i can test at most r_i items per unit time. Thus at processor O_i , there are at most $r_i(1 - p_i)$ tests performed that have the value 0. Let f denote the k -of- n function.

Suppose R is a routing achieving throughput F . Since F items enter the system per unit time, F items must also leave the system. An item x such that $f(x) = 0$ does not leave the system until it fails k tests. An item x such that $f(x) = 1$ does not leave the system until it has had all tests performed on it. Thus, per unit time, in the entire system, the number of tests performed that have the value 0 must be $F \times M$, where $M = (k \cdot \text{Prob}[x \text{ has at least } k \text{ 0's}] + \sum_{j=0}^{k-1} j \cdot \text{Prob}[x \text{ has exactly } j \text{ 0's}])$.

Since at most $r_i(1 - p_i)$ tests with the value 0 can occur per unit time at processor O_i , $F \times M \leq \sum_{i=1}^n r_i(1 - p_i)$. Solving for F , this gives an upper bound of $F \leq \sum_{i=1}^n r_i(1 - p_i)/M$ on the maximum throughput. This bound is tight if all processors are saturated, and hence a routing saturating all processors achieves the maximum throughput. \square

In the above proof, we rely on the fact that every routing with throughput F results in the same number of 0 test values being generated in the system per unit time. Note that this is not the case for *standard* testing, where the number of 0 test values generated can depend on the routing itself, and not just on the throughput of that routing. In fact, it is easy to show that for the SMT(k) Problem, saturation does not always imply optimality.

The routing produced by our algorithm for the CMT(k) Problem uses only strategies of the form $T_k^c(\pi)$, for some permutation π of the tests (in terms of the LP, this means $z_T > 0$ only if $T = T_k^c(\pi)$ for some π). We call such a routing a *permutation routing*. We say that it has a *saturated suffix* if for some subset Q of the processors (1) R saturates all processors in Q , and (2) for every strategy $T_k^c(\pi)$ used by R , the processors in Q appear together in a suffix of π .

With this definition, and the above lemma, we are now able to generalize a key lemma of Condon et al. to apply to conservative k -of- n testing, using essentially the same proof.

Lemma 2. (*Saturated Suffix Lemma*) *Let R be a permutation routing for an instance of the CMT(k) Problem. If R has a saturated suffix, then R is optimal.*

5.1 The Equal Rates Case

We begin by considering the CMT(k) Problem in the special case where the rate limits r_i are equal to some constant value r for all processors. Condon et al. presented a closed-form solution for this case when $k = 1$ [3]. The solution is a permutation routing that uses n strategies of the form $T_1(\pi)$. Each permutation π is one of the n left cyclic shifts of the permutation $(1, \dots, n)$. More specifically, for $i \in \{1, \dots, n\}$, let $\pi_i = (i, i+1, \dots, n, 1, 2, \dots, i-1)$, and let $T_i = T_1^c(\pi_i)$. The solution assigns $r(1 - p_{i-1})/(1 - p_1 \cdots p_n)$ units of flow to each T_i (where $p_0 = p_n$). By simple algebra, Condon et al. verified that the solution saturates all processors. Hence it is optimal.

The solution of Condon et al. is based on the fact that for the 1-of- n problem, assigning $(1 - p_{i-1})$ flow to each T_i equalizes the load on the processors. Surprisingly, this same assignment equalizes the load for the k -of- n problem as well. Using this fact, we obtain a closed-form solution to the CMT(k) Problem.

Lemma 3. *Consider an instance of the CMT(k) Problem where all processors have the same rate limit r . For $i \in \{1, \dots, n\}$, let T_i be as defined above. Let $X_{a,b} = \sum_{\ell=a}^b (1 - x_\ell)$. The routing that assigns $r(1 - p_{i-1})/\alpha$ items per unit time to strategy T_i saturates all processors, where $\alpha = \sum_{t=1}^k \text{Prob}[X_{1,n} \geq t]$.*

Proof. We begin by considering the routing in which $(1 - p_{i-1})$ units of flow are assigned to each T_i . Consider the question of how much flow arrives per unit time at processor 1, under this routing. For simplicity, assume now that $k = 2$. Thus as soon as an item has failed 2 tests, it is discarded. Let $q_i = (1 - p_i)$.

Of the q_n units assigned to strategy T_1 , all q_n arrive at processor 1. Of the q_{n-1} units assigned to strategy T_n , all q_{n-1} arrive at processor 1, since they can fail either 0 or 1 test (namely test n) beforehand.

Of the q_{n-2} units assigned to strategy T_{n-1} , the number reaching processor O_1 is $q_{n-2}\beta_{n-1}$, where β_{n-1} is the probability that an item fails either 0 or 1 of tests $n - 1$

and n . Therefore, $\beta_{n-1} = 1 - q_{n-1}q_n$. More generally, for $i \in \{1, \dots, n\}$, of the q_{i-1} units assigned to T_i , the number reaching processor 1 is $q_{i-1}\beta_i$, where β_i is the probability that a random item fails a total of 0 or 1 of tests $i, i+1, \dots, n$. Thus, $\beta_i = \text{Prob}[X_{i,n} = 0] + \text{Prob}[X_{i,n} = 1]$. It follows that the total flow arriving at Processor 1 is $\sum_{i=1}^n q_{i-1}(\text{Prob}[X_{i,n} = 0]) + \sum_{i=1}^n q_{i-1}(\text{Prob}[X_{i,n} = 1])$.

Consider the second summation, $\sum_{i=1}^n (q_{i-1}\text{Prob}[X_{i,n} = 1])$. We claim that this summation is equal to $\text{Prob}[X_{1,n} \geq 2]$, which is the probability that x has *at least* two x_i 's that are 0. To see this, consider a process where we observe the value of x_n , then the value of x_{n-1} and so on down towards x_1 , stopping if and when we have observed exactly two 0's. The probability that we will stop at some point, having observed two 0's, is equal to the probability that x has *at least* two x_i 's set to 0. The condition $\sum_{j=i}^n (1-x_j) = 1$ is satisfied when exactly 1 of x_n, x_{n-1}, \dots, x_i has the value 0. Thus $q_{i-1}(\text{Prob}[X_{i,n} = 1])$ is the probability that we observe exactly one 0 in x_n, \dots, x_i , and then we observe a second 0 at x_{i-1} . That is, it is the probability that we stop after observing x_{i-1} . Since the second summation takes the sum of $q_{i-1}\text{Prob}[X_{i,n} = 1]$ over all i , the summation is equal to the probability of stopping at some point in the above process, having seen two 0's. This proves the claim. An analogous argument shows that the first summation, $\sum_{i=1}^n (q_{i-1}\text{Prob}[X_{i,n} = 0])$, is equal to $\text{Prob}[X_{1,n} \geq 1]$.

It follows that the amount of flow reaching Processor 1 is $\text{Prob}[X_{1,n} \geq 1] + \text{Prob}[X_{1,n} \geq 2]$. This expression is independent of i , so the amount of flow reaching every O_i is equal to this value. Thus the above routing causes all processors to receive the same amount of flow. Hence, if all processors have the same rate limit r , a scaling of this routing will saturate all processors. Specifically, assigning $rq_{i-1}/(\text{Prob}[X_{1,n} \geq 1] + \text{Prob}[X_{1,n} \geq 2])$ units to each strategy T_i will saturate all processors.

The above argument for $k = 2$ can easily be extended to arbitrary k . The resulting saturating routing for arbitrary k , when all processors have rate limit r , assigns $rq_{i-1}/(\sum_{t=1}^k \text{Prob}[X_{1,n} \geq t])$ items per unit time to strategy T_i . \square

5.2 The Equalizing Algorithm of Condon et al.

Our algorithm for the CMT(k) Problem is an adaptation of one of the two MAX-THROUGHPUT algorithms, for the special case where $k = 1$, given by Condon et al. [3]. We begin by reviewing that algorithm, which we will call the *Equalizing Algorithm*. Note that when $k = 1$, it only makes sense to consider strategies that are permutation routings, since an item can be discarded as soon as it fails a single test.

Consider the CMT(k) Problem for $k = 1$. View the problem as one of constructing a flow of items through the processors. The capacity of each processor is its rate limit, and the amount of flow sent along a permutation π (i.e., assigned to strategy $T_1^c(\pi)$) equals the number of items sent along that path per unit time. Sort the tests by their rate limits, and re-number them so that $r_n \geq r_{n-1} \geq \dots \geq r_1$. Assume for the moment that all r_i are distinct.

The Equalizing Algorithm constructs a flow incrementally as follows. Imagine pushing flow along the single permutation $(n, \dots, 1)$. Suppose we continuously increase the amount of flow being pushed, beginning from zero, while monitoring the “residual capacity” of each processor, i.e., the difference between its rate limit and the amount of

flow it is already receiving. (For the moment, let us not worry about exceeding the rate limit of an processor.)

Consider two adjacent processors, i and $i - 1$. As we increase the amount of flow, the residual capacity of each decreases continuously. Initially, at zero flow, the residual capacity of i is greater than the residual capacity of $i - 1$. It follows by continuity that the residual capacity of i cannot become less than the residual capacity of $i - 1$ without the two residual capacities first becoming equal. We now impose the following stopping condition: increase the flow sent along permutation $(n, \dots, 1)$ until either (1) some processor becomes saturated, or (2) the residual capacities of at least two of the processors become equal. The second stopping condition ensures that when the flow increase is halted, permutation $(n, \dots, 1)$ still orders the processors in decreasing order of their residual capacities. (Algorithmically, we do not increase the flow continuously, but instead directly calculate the amount of flow which triggers the stopping condition.)

If stopping condition (1) above holds when the flow increase is stopped, then the routing can be shown to have a saturated suffix, and hence it is optimal.

If stopping condition (2) holds, we keep the current flow, and then augment it by solving a new MAXTHROUGHPUT problem in which we set the rate limits of the processors to be equal to their residual capacities under the current flow (their p_i 's remain the same).

We solve the new MAXTHROUGHPUT problem as follows. We group the processors into equivalence classes according to their rate limits. We then replace each equivalence class with a single mega-processor, with a rate limit equal to the rate limit of the constituent processors, and probability p_i equal to the product of their probabilities. We then essentially apply the procedure for the case of distinct rate limits to the mega processors.

The one twist is the way in which we translate flow sent through a mega-processor into flow sent through the constituent processors of that mega-processor; we route the flow through the constituent processors so as to equalize their load. We accomplish this by dividing the flow proportionally between the cyclic shifts of a permutation of the processors, using the same proportional allocation as used in the saturating routing of Lemma 3. We thus ensure that the processors in each equivalence class continue to have equal residual capacity. Note that, under this scheme, the residual capacity of a processor in a mega-processor may decrease more slowly than it would if all flow were sent directly to that processor (because some flow may first be filtered through other processors in the mega-processor) and this needs to be taken into account in determining when the stopping condition is reached.

The Equalizing Algorithm, implemented in a straightforward way, produces a routing that may use an exponential number of different permutations. Condon et al. describe methods for reducing the number of permutations used to be linear in n [3].

5.3 An Equalizing Algorithm for the CMT(k) Problem

We prove the following theorem.

Theorem 4. *There is a $O(kn^2)$ combinatorial algorithm for solving the CMT(k) Problem.*

Proof. We extend the Equalizing Algorithm of Condon et al., to apply to arbitrary values of k . Again, we will push flow along the permutation of the processors $(n, \dots, 1)$ (where $r_n \geq r_{n-1} \geq \dots \geq r_1$) until one of the two stopping conditions is reached: (1) a processor is saturated, or (2) two processors have equal residual capacity. Here, however, we do not discard an item until it has failed k tests, rather than discarding it as soon as it fails one test. To reflect this, we divide the flow into k different types, numbered 0 through $k - 1$, depending on how many tests its component items have failed. Flow entering the system is all of type 0.

When m flow of type τ enters a processor O_i , $p_i m$ units pass test O_i , and $(1 - p_i)m$ units fail it. So, if $\tau < k - 1$, then of the m incoming units of type τ , $(1 - p_i)m$ units will exit processor O_i as type $\tau + 1$ flow, and $p_i m$ will exit as type τ flow. Both types will be passed on to the next processor in the permutation, if any. If $\tau = k - 1$, then $p_i m$ units will exit as type τ flow and be passed on to the next processor, and the remaining $(1 - p_i)m$ will be discarded.

Algorithmically, we need to calculate the minimum amount of flow that triggers a stopping condition. This computation is only slightly more complicated for general k than it is for $k = 1$. The key is to compute, for each processor O_i , what fraction of the flow that is pushed into the permutation will actually reach processor O_i (i.e. we need to compute the quantity $g(T_k^c(\pi), i)$ in the LP).

If stopping condition (2) holds, we keep the current flow, and augment it by solving a new MAXTHROUGHPUT problem in which we set the rate limits of the processors to be equal to their residual capacities under the current flow (their p_i 's remain the same.) To solve the new MAXTHROUGHPUT problem, we again group the processors into equivalence classes according to their rate limits, and replace each equivalence class with a single mega-processor, with a rate limit equal to the rate limit of the constituent processors, and probability p_i equal to the product of their probabilities.

We then want to apply the procedure for the case of distinct rate limits to the mega processors. To do this, we need to translate flow sent into a mega-processor into flow sent through the constituent processors of that mega-processor, so as to equalize their load. We do this translation separately for each type of flow entering the mega-processor. Flow of type τ entering the mega-processor is sent into the constituent processors of the mega-processor according to the closed-form solution for $(k - \tau)$ -of- n' testing, where n' is the number of constituent processors of the mega-processor, because it will be discarded if it fails $k - \tau$ more tests. We also need to compute how much flow of each type ends up leaving the mega-processor (some of the incoming flow of type τ entering the mega-processor may, for example, become outgoing flow of type $\tau + n'$), and how much its residual capacity is reduced by the incoming flow. The algorithm can be implemented to run in time $O(kn^2)$. \square

6 An Ellipsoid-Based Algorithm for the SMT(k) Problem

There is a simple and elegant algorithm that solves the MINCOST problem for standard k -of- n testing, due to Salloum, Breuer, and (independently) Ben-Dov [678]. It outputs a strategy compactly represented by two permutations, one ordering the operators in increasing order of the ratio $c_i/(1 - p_i)$, and the other in increasing order of the ratio

c_i/p_i . Chang et al. and Salloum and Breuer later gave modified versions of this algorithm that output a less compact, but more efficiently evalutable representation of the same strategy [119].

We now describe how to combine previous techniques to obtain a polynomial-time algorithm for the SMT(k) Problem based on the ellipsoid method. The algorithm uses a technique of Deshpande and Hellerstein [5]. They showed that, for 1-of- n testing, an algorithm solving the MINCOST problem can be combined with the ellipsoid method to yield an algorithm for the MAXTHROUGHPUT problem. In fact, their approach is actually a generic one, and can be applied to testing of other functions.

Theorem 5. *There is a polynomial-time algorithm, based on the ellipsoid method, for solving the SMT(k) Problem.*

The following is a sketch of the proof of the above theorem. Deshpande and Hellerstein's approach for the MAXTHROUGHPUT 1-of- n testing problem works by running the ellipsoid algorithm on the dual of the MAXTHROUGHPUT LP. To do this efficiently, they rely on the existence of two polynomial-time algorithms: one for solving the MINCOST problem, and one for computing the $g(T, i)$ values (as defined in the MAXTHROUGHPUT LP) for the optimal T output by the MINCOST algorithm. The same approach can be applied to MAXTHROUGHPUT for k -of- n testing (or testing any function), provided we have these two algorithms. For k -of- n testing, we can get them using the MINCOST algorithm of Chang et al. [Q], which can easily be modified to output the $g(T, i)$ values for the output strategy T .

References

1. Kodialam, M.S.: The Throughput of Sequential Testing. In: Aardal, K., Gerards, B. (eds.) IPCO 2001. LNCS, vol. 2081, pp. 280–292. Springer, Heidelberg (2001)
2. Condon, A., Deshpande, A., Hellerstein, L., Wu, N.: Flow algorithms for two pipelined filter ordering problems. In: PODS, ACM, pp. 193–202 (2006)
3. Condon, A., Deshpande, A., Hellerstein, L., Wu, N.: Algorithms for distributional and adversarial pipelined filter ordering problems. ACM Transactions on Algorithms 5 (2009)
4. Liu, Z., Parthasarathy, S., Ranganathan, A., Yang, H.: A generic flow algorithm for shared filter ordering problems. In: Lenzerini, M., Lembo, D. (eds.) PODS, ACM, pp. 79–88 (2008)
5. Deshpande, A., Hellerstein, L.: Parallel pipelined filter ordering with precedence constraints. Pre-publication version (To appear in ACM Transactions on Algorithms), <http://cis.poly.edu/string~hstein/pubs/filterwithconstraint.pdf>
6. Salloum, S.: Optimal testing algorithms for symmetric coherent systems. PhD thesis, Univ. of Southern California (1979)
7. Salloum, S., Breuer, M.A.: An optimum testing algorithm for some symmetric coherent systems. J. Mathematical Analysis and Applications 101, 170–194 (1984)
8. Ben-Dov, Y.: Optimal testing procedure for special structures of coherent systems. Management Science 27, 1410–1420 (1981)
9. Chang, M.F., Shi, W., Fuchs, W.K.: Optimal diagnosis procedures for k-out-of-n structures. IEEE Trans. Computers 39, 559–564 (1990)
10. Hellerstein, L., Özkan, Ö., Sellie, L.: Max-throughput for (conservative) k-of-n testing. CoRR abs/1109.3401 (2011)

11. Salloum, S., Breuer, M.: Fast optimal diagnosis procedures for k-out-of-n:g systems. *IEEE Transactions on Reliability* 46, 283–290 (1997)
12. Ünlüyurt, T.: Sequential testing of complex systems: a review. *Discrete Applied Mathematics* 142, 189–205 (2004); Boolean and Pseudo-Boolean Functions
13. Garey, M.R.: Optimal task sequencing with precedence constraints. *Discrete Math.* 4, 37–56 (1973)
14. Boros, E., Ünlüyurt, T.: Diagnosing double regular systems. *Ann. Math. Artif. Intell.* 26, 171–191 (1999)

Closest Periodic Vectors in L_p Spaces

Amihood Amir^{1,2,*}, Estrella Eisenberg¹, Avivit Levy^{3,4}, and Noa Lewenstein⁵

¹ Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
amir@cs.biu.ac.il

² Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218
³ Department of Software Engineering, Shenkar College, 12 Anna Frank,
Ramat-Gan, Israel

avivitlevy@shenkar.ac.il
⁴ CRI, Haifa University, Mount Carmel, Haifa 31905, Israel

⁵ Netanya College, Netanya, Israel
noa.lewenstein@gmail.com

Abstract. The problem of finding the period of a vector V is central to many applications. Let V' be a periodic vector closest to V under some metric. We seek this V' , or more precisely we seek the smallest period that generates V' . In this paper we consider the problem of finding the closest periodic vector in L_p spaces. The measures of “closeness” that we consider are the metrics in the different L_p spaces. Specifically, we consider the L_1 , L_2 and L_∞ metrics. In particular, for a given n -dimensional vector V , we develop $O(n^2)$ time algorithms (a different algorithm for each metric) that construct the smallest period that defines such a periodic n -dimensional vector V' . We call that vector the *closest periodic vector* of V under the appropriate metric. We also show (three) $O(n \log n)$ time constant approximation algorithms for the (appropriate) period of the closest periodic vector.

1 Introduction

Exact data periodicity has been amply researched over the years [15]. Linear time algorithms for exploring the periodic nature of data represented as strings were suggested (e.g. [8]). Multidimensional periodicity [1,11,17] and periodicity in parameterized strings [6] was also explored. In addition, periodicity has played a role in efficient parallel string algorithms [10,2,3,7,9]. Many phenomena in the real world have a particular type of event that repeats periodically during a certain period of time. The ubiquity of cyclic phenomena in nature, in such diverse areas as Astronomy, Geology, Earth Science, Oceanography, Meteorology, Biological Systems, the Genome, Economics, and more, has led to a recent interest in periodicity. Examples of highly periodic events include road traffic peaks, load peaks on web servers, monitoring events in computer networks and many others. Finding periodicity in real-world data often leads to useful insights,

* Partly supported by NSF grant CCR-09-04581 and ISF grant 347/09.

because it sheds light on the structure of the data, and gives a basis to predict future events. Moreover, in some applications periodic patterns can point out a problem: In a computer network, for example, repeating error messages can indicate a misconfiguration, or even a security intrusion such as a port scan [16].

However, real data generally contain errors, either because they are inherent in the data, or because they are introduced by the data gathering process. Nevertheless, it is still valuable to detect and utilize the underlying periodicity. This calls for the notion of *approximate periodicity*. Given a data vector, we may not be confident of the measurement or suspect the periodic process to be inexact. We may, therefore, be interested in finding the *closest periodic* nature of the data, i.e., what is the smallest period of the closest periodic vector to the given input vector, where the term closest means that it has the smallest number of errors. It is natural to ask if such a period can be found efficiently. The error cause varies with the different phenomena. This fact is formalized by considering different metrics to define the error.

Different aspects of approximate periodicity, inspired by various applications, were studied in the literature. In some applications, such as computational biology, the search is for periodic patterns (or, as coined in the literature, *approximate multiple tandem repeats*) and not necessarily full periodicity (e.g. [13]). Another example of such applications is monitoring events in computer networks which inspired [12] to study the problem of finding approximate arithmetic progressions in a sequence of numbers. In many other applications such as Astronomy, Geology, Earth Science, Oceanography and Meteorology, the studied phenomena have a full (inexact) periodic nature that should be discovered. The term *approximate periodicity* is typically used only for such applications, where terms such as *approximate periodic patterns* are used otherwise. In [4] approximate periodicity was studied under two metrics: the Hamming distance and the swap distance. Both these metrics are *pseudo-local*, which intuitively means that any error causes at most a constant number of mismatches (see [5]). In [5] it was shown that for a guaranteed small distance in pseudo-local metrics, the original cycle can be detected and corrected in the corrupted data.

The focus of this paper is on vector spaces. The common and natural metrics for vector spaces are L_1 , L_2 and L_∞ . These metrics are not pseudo-local and, therefore, the methods of [5] do not apply. In this paper we tackle the problem of finding the period of the closest periodic vector under the L_1 , L_2 , and L_∞ metrics. Specifically, given a vector $V \in \mathbb{R}^n$, and a metric L_1 , L_2 , or L_∞ , we seek a natural number $p \leq \frac{n}{2}$, and a period P of length p , such that the distance between $P^{\lfloor n/p \rfloor} P'$ and V is smallest, where P^i denotes P concatenated to itself i times, P' is the prefix of P of length $n - \lfloor \frac{n}{p} \rfloor$, and the metric is L_1 , L_2 , or L_∞ . We prove the following theorems:

Theorem 1. *Given a vector $V \in \mathbb{R}^n$, then for each of the metrics L_1 , L_2 or L_∞ , a vector P which is the period of the closest periodic vector under the metric can be found in $O(n^2)$ time.*

Theorem 2. Given a vector $V \in \mathbb{R}^n$, then:

1. the period P of the closest periodic vector of V under the L_2 metric can be approximated to a factor of $\sqrt{3}$ in $O(n \log n)$ time.
2. for any $\epsilon > 0$, the period P of the closest periodic vector of V under the L_1 metric can be approximated to a factor of $3 + \epsilon$ in $O(\frac{1}{\epsilon} n \log n)$ time.
3. for any $\epsilon > 0$, the period P of the closest periodic vector under the L_∞ metric can be approximated to a factor of $3 + \epsilon$ in $O(\frac{1}{\epsilon} n \log n)$ time.

The proof of Theorem 1 is described in Sect. 3 and the proof of Theorem 2 is described in Sect. 4.

2 Preliminaries

In this section we give basic definitions of periodicity and related issues as well as a formal definition of the problem.

Definition 1. Let $V = \langle V[1], V[2], \dots, V[n] \rangle$ be an n -dimensional vector. A sub-vector of V is a vector $T = \langle V[i], \dots, V[j] \rangle$, where $1 \leq i \leq j \leq n$. Clearly, the dimension of T is $j - i + 1$.

The sub-vector $T = \langle V[1], \dots, V[j] \rangle$ is called a prefix of V and the sub-vector $T = \langle V[i], \dots, V[n] \rangle$ is called a suffix of V .

For two vectors $V = \langle V[1], V[2], \dots, V[n] \rangle$, $T = \langle T[1], T[2], \dots, T[m] \rangle$ we say that the vector $R = \langle V[1], V[2], \dots, V[n], T[1], T[2], \dots, T[m] \rangle$ is the concatenation of V and T .

Definition 2. Let V be a vector. Denote by $|V|$ the dimension of V and let $|V| = n$. V is called periodic if $V = P^i \text{pref}(P)$, where $i \in \mathbb{N}$, $i \geq 2$, P is a sub-vector of V such that $1 \leq |P| \leq n/2$, P^i is the concatenation of P to itself i times, and $\text{pref}(P)$ is a prefix of P . The smallest such sub-vector P is called the period of V . If V is not periodic it is called aperiodic.

Notation: Throughout the paper we use p to denote a period dimension and P the period vector, i.e., $|P| = p$.

Definition 3. Let P be a vector of dimension p . Let $n \in \mathbb{N}$ such that $2 \cdot p \leq n$. The vector V_P is defined to be a periodic vector of dimension n with period P , i.e., $V_P = P^{\lfloor \frac{n}{p} \rfloor} \text{pref}(P)$, where $\text{pref}(P)$ is the prefix of P of dimension $n - \lfloor \frac{n}{p} \rfloor \cdot p$.

Example: Given the vector $P = \langle A, B, C \rangle$ of dimension $p = 3$, let $n = 10$ then $V_P = \langle A, B, C, A, B, C, A, B, C, A \rangle$.

Definition 4. Let V be an n -dimensional vector over \mathbb{R} . Let d be a metric defined on \mathbb{R}^n . V is called α -close to periodic under the metric d , if there exists a vector P over \mathbb{R}^p , $p \in \mathbb{N}$, $p \leq n/2$, such that $d(V_P, V) \leq \alpha$. The vector P is called an α -close period of V .

The Problem Definition. The problem is formally defined below.

Definition 5. Given a metric d , the Closest Periodic Vector Problem under the metric d , is the following:

INPUT: Vector V of dimension n over \mathbb{R} .

OUTPUT: The period P of the closest periodic vector of V under the metric d , and α such that P is an α -close period of V under d .

We will also need the following definition.

Definition 6. Given a metric d , let P be the vector of dimension p such that $d(V, V_P)$ is minimal over all possible vectors of dimension p . We call P the p -dimension close period under d .

3 Closest Periodic Vectors in L_1 , L_2 and L_∞ Spaces

In this section we study the problem of finding the closest periodic vector in L_1 , L_2 and L_∞ spaces. We begin with the simpler cases of L_2 and L_∞ and then study the L_1 case.

3.1 Closest Periodic Vector in L_2 Space

We first examine the case where the metric d is L_2 . Given a vector $V \in \mathbb{R}^n$, we seek another vector P , $1 \leq |P| \leq \frac{n}{2}$, which minimizes the L_2 distance between V and V_P . Formally,

INPUT: Vector $V \in \mathbb{R}^n$.

OUTPUT: $P \in \mathbb{R}^p$, $1 \leq p \leq \frac{n}{2}$ minimizing $d_{L_2}(V_P, V) = \sqrt{\sum_{i=1}^n (V_P[i] - V[i])^2}$.

We begin by studying the case of a monochromatic vector and showing that it is easy to find a monochromatic vector closest to a given vector under the L_2 metric. A *monochromatic vector* refers to a vector containing the same scalar in all the coordinates, namely for a scalar x the monochromatic vector is $\langle x, x, \dots, x \rangle$. We denote by x^k the k -dimensional vector $\langle x, x, \dots, x \rangle$. Lemma 1 is the cornerstone of our method.

Lemma 1. Let $V \in \mathbb{R}^k$. Then, the scalar x such that $d_{L_2}(V, x^k)$ is minimal can be found in time $O(k)$. Moreover, x equals the average of $V[1], \dots, V[k]$.

Using Lemma 1, we can now show that given a dimension p the p -dimension close period of V under the L_2 metric can be computed in linear time.

Lemma 2. Let $V \in \mathbb{R}^n$ and let p be a dimension $p \leq \frac{n}{2}$. Then, $P \in \mathbb{R}^p$ such that $d_{L_2}(V_P, V)$ is minimal over all vectors of dimension p can be found in $O(n)$ time.

We can now use the above in order to find the period of the closest periodic vector under the L_2 metric. According to Lemma 2, the p -dimension close period for any given dimension p can be found in $O(n)$ time. Hence, we can perform a

brute-force search over every dimension between 1 and $\frac{n}{2}$ and check for the best over all p -dimension periods yielding our desired period P . Since finding the p -dimension close period for a given dimension takes $O(n)$ time and there are $O(n)$ different potential dimensions, the time for this algorithm is $O(n^2)$. This proves Theorem 1 for the L_2 metric.

3.2 Closest Periodic Vector in L_∞ Space

We now examine the case where the metric is L_∞ . Given a vector $V \in \mathbb{R}^n$, we seek another vector P , $1 \leq p \leq \frac{n}{2}$, which minimizes the L_∞ distance between V and V_P . Formally,

INPUT: Vector $V \in \mathbb{R}^n$.

OUTPUT: $P \in \mathbb{R}^p$, $1 \leq p \leq \frac{n}{2}$ minimizing $d_{L_\infty}(V_P, V) = \max_{i=1}^n |V_P[i] - V[i]|$.

As in Subsect. 3.1, we begin by showing that the closest monochromatic vector under the L_∞ -metric can be easily found.

Lemma 3. *Let $V \in \mathbb{R}^k$. The scalar x such that $d_{L_\infty}(V, x^k)$ is minimal can be found in time $O(k)$. Moreover, x equals the average between the maximal and minimal values among $V[1], \dots, V[k]$.*

From Lemma 3 we get that given a dimension p , finding the p -dimension period of V under the L_∞ metric can be implemented efficiently.

Lemma 4. *Let $V \in \mathbb{R}^n$ and let p be a given dimension, $p \leq \frac{n}{2}$. Then a vector $P \in \mathbb{R}^p$ such that $d_{L_\infty}(V_P, V)$ is minimal over all vectors of dimension p can be found in $O(n)$ time.*

From Lemma 4, we deduce similar to the proof of Theorem 1 for L_2 that finding the closest periodic vector to V under the L_∞ metric can be implemented efficiently. This concludes the proof of Theorem 1 for the L_∞ metric.

3.3 Closest Periodic Vector in L_1 Space

We now turn to the case where the metric is L_1 . Given a vector $V \in \mathbb{R}^n$, we seek another vector P , $1 \leq |P| \leq \frac{n}{2}$, which minimizes the L_1 distance between V and V_P . Formally,

INPUT: Vector $V \in \mathbb{R}^n$.

OUTPUT: $P \in \mathbb{R}^p$, $1 \leq p \leq \frac{n}{2}$ minimizing $d_{L_1}(V_P, V) = \sum_{i=1}^n |V_P[i] - V[i]|$.

Once again we focus on finding a monochromatic vector, $x^k = \langle x, x, \dots, x \rangle$, closest to a given vector in \mathbb{R}^k . For the L_1 metric, x is the median value of the input vector values, as the following well known lemma shows.

Lemma 5. *Let $V \in \mathbb{R}^k$. The scalar x such that $d_{L_1}(V, x^k)$ is minimal is the median of $V[1], \dots, V[k]$.*

Given a dimension p , the p -dimension period of V under the L_1 metric can be implemented efficiently, using Lemma 5.

Lemma 6. *Let $V \in \mathbb{R}^n$ and let p be a dimension $p \leq \frac{n}{2}$. Then, $P \in \mathbb{R}^p$ such that $d_{L_1}(V_P, V)$ is minimal over all vectors of dimension p can be found in time $p \times T(\frac{n}{p})$, where $T(\frac{n}{p})$ is the time to construct the $\frac{n}{p}$ -dimension monochromatic vector closest to a given $\frac{n}{p}$ -dimension vector.*

Given Lemma 6, we need to compute the closest monochromatic vector x^k from a vector $V \in \mathbb{R}^k$. Since by Lemma 5, x is the median of $V[1], \dots, V[k]$, we need to sort these values in $O(k \log k)$ time and choose the median. This gives an $O(n \log \frac{n}{p})$ computation for a given period dimension p (by Lemma 6). Using Stirling's approximation it is not difficult to see that $\sum_{p=1}^{n/2} \log \frac{n}{p} = O(n)$, giving an overall $O(n^2)$ time algorithm for finding the closest periodic vector in the L_1 distance.

This concludes the proof of Theorem 1 for the L_1 metric.

4 Fast Approximations of the Closest Periodic Vector in L_1 , L_2 and L_∞ Spaces

We have seen that the closest periodic vector, under the L_1 , L_2 and L_∞ metrics, of an n -dimensional vector can be found in time $O(n^2)$. In this section we show that the closest periodic vector can be *approximated* in almost linear time (with logarithmic factors).

4.1 Approximation in L_2 Space

We begin by showing that the closest periodic vector under the L_2 metric can be *approximated* in time $O(n \log n)$. Our algorithm will find a period length p_{\min} for which the p_{\min} -dimension close period will be no larger than $\sqrt{3}$ times the distance of the closest periodic vector. We need the following definition:

Definition 7. *Let V be an n -dimensional vector. The s -shift vector of V , denoted by V^s , is the concatenation of the s -length prefix of V and the $(n-s)$ -length suffix of V , i.e., V^s is the n -dimensional vector whose elements are: $V^s[i] = V[i]$, for $i = 1, \dots, s$, and $V^s[i] = V[i-s]$, for $i = s+1, \dots, n$.*

Example: Let $V = <1, 2, 3, 4, 5, 6, 7>$. Then $V^3 = <1, 2, 3, 1, 2, 3, 4>$.

Important Property: The interesting property of the shift vector V^s is the following. If V^{pre} is the prefix of V of length $n-s$ and V^{suf} is the suffix of V of length $n-s$, then: $d_{L_t}(V, V^s) = d_{L_t}(V^{pre}, V^{suf})$, for $t = 1, 2, \infty$.

Example: For the above V , V^3 , $d_{L_1}(V, V^3) = |1-1| + |2-2| + |3-3| + |4-1| + |5-2| + |6-3| + |7-4| = d_{L_1}(<1, 2, 3, 4>, <4, 5, 6, 7>)$.

Let $A_p = \sum_{i=1}^{\lceil \frac{n}{p} \rceil - 1} d_{L_2}^2(V, V^{i \cdot p})$, where $1 \leq p \leq n/2$. Lemma 7 below uses A_p to find the period length that approximates the closest periodic vector of V .

Lemma 7. Let V be a vector of dimension n , and let $1 \leq p \leq n/2$. Let V_P be the n -dimensional vector for which the p -dimensional vector P is the p -dimension close period of V under the L_2 metric. Then

$$(a) \frac{1}{\sqrt{\lceil \frac{n}{p} \rceil - 1}} \sqrt{A_p} \leq d_{L_2}(V, V_P)$$

$$(b) d_{L_2}(V, V_P) \leq \sqrt{\frac{2}{\lceil \frac{n}{p} \rceil}} \sqrt{A_p}$$

Lemma 8 is needed for the proof of Lemma 9.

Lemma 8. [Concatenation Lemma] Let $V \in \mathbb{R}^n$. Let P be the p -dimensional close period of V , $p \leq \frac{n}{4}$, and let V_P be the n -dimensional vector for which P is the p -dimensional period. Let P' be a $2p$ -dimensional close period of V and $V_{P'}$ the n -dimensional vector for which P' is a $2p$ -dimensional period. Then $d_{L_t}(V, V_{P'}) \leq d_{L_t}(V, V_P)$, for $t = 1, 2, \infty$.

Lemma 9. Given a vector $V \in \mathbb{R}^n$, and given p_{min} , $\frac{n}{4} < p_{min} \leq \frac{n}{2}$ for which $d_{L_2}(V, V^{p_{min}})$ is smallest. Then the p_{min} -dimension close period P_{min} approximates the period P of the closest periodic vector of V to within a factor of $\sqrt{3}$.

Proof. Lemma 8 allows us to consider only vectors whose length is in the range $(\frac{n}{4}, \frac{n}{2}]$. Let P_{min} be the p_{min} -dimensional close period of V , and let P of length p be the period of the closest periodic vector of V . By the definition of the closest periodic vector we have: $d_{L_2}(V, V_P) \leq d_{L_2}(V, V_{P_{min}})$. By Lemma 7(b) and the fact that $p_{min} \leq \frac{n}{2}$ we know that: $d_{L_2}(V, V_{P_{min}}) \leq \sqrt{A_{p_{min}}}$. Now, because p_{min} was chosen as the one giving the minimum A_p , then: $\sqrt{A_{p_{min}}} \leq \sqrt{A_p}$. Also, because of Lemma 7(a) and the fact that $p_{min} > \frac{n}{4}$, which implies that $\sqrt{\lceil \frac{n}{p} \rceil - 1} \leq 3$, we get: $\frac{\sqrt{A_p}}{\sqrt{3}} \leq d_{L_2}(V, V_P)$. Conclude that $d_{L_2}(V, V_P) \leq d_{L_2}(V, V_{P_{min}}) \leq \sqrt{3}d_{L_2}(V, V_P)$. \square

We are now ready to present the approximation algorithm.

Approximation Algorithm for L_2 :

1. Compute A_p for all $p = \lceil \frac{n}{4} \rceil, \dots, \lfloor \frac{n}{2} \rfloor$
2. Choose p_{min} for which $A_{p_{min}}$ is smallest.
3. Compute the p_{min} -dimension close period of V .

Time: By [14], all A_p can be computed in time $O(n \log n)$, choosing the minimum is done in linear time, as is constructing the p_{min} -dimension close period. Thus the total time is $O(n \log n)$.

We have, therefore, proven the first part of Theorem 2.

4.2 Approximation in L_1 Space

We now describe a fast approximation of the closest periodic vector in L_1 space. We first need a bounding lemma, similar to Lemma 7. The proof is also similar to

the proof of Lemma 7, but $d(V, W)$ is taken to be $d_{L_1}(V, W)$ and thus no square root is taken at the end. Let $B_p = \sum_{i=1}^{\lceil \frac{n}{p} \rceil - 1} d_{L_1}(V, V^{i \cdot p})$, where $1 \leq p \leq n/2$.

Lemma 10. *Let V be a vector of dimension n , and let $1 \leq p \leq n/2$. Let V_P be the n -dimensional vector for which the p -dimensional vector P is the p -dimension close period of V under the L_1 metric. Then*

- (a) $\frac{1}{\lceil \frac{n}{p} \rceil - 1} B_p \leq d_{L_1}(V, V_P)$
- (b) $d_{L_2}(V, V_P) \leq \frac{2}{\lfloor \frac{n}{p} \rfloor} B_p$

The concatenation lemma gives:

Lemma 11. *Given a vector $V \in \mathbb{R}^n$, and given p_{min} , $\frac{n}{4} < p_{min} \leq \frac{n}{2}$ for which $d_{L_1}(V, V^{p_{min}})$ is smallest. Then the p_{min} -dimension close period P approximates the period of the closest periodic vector of V to within a factor of 3.*

Approximation Algorithm for L_1

1. Compute B_p for all $p = \lceil \frac{n}{4} \rceil, \dots, \lfloor \frac{n}{2} \rfloor$
2. Choose p_{min} for which $B_{p_{min}}$ is smallest.
3. Compute the p_{min} -dimension close period of V .

Time: By [14], for any $\epsilon > 0$, B_p can be approximated to within a factor of $(1 + \epsilon)$ in time $O(\frac{1}{\epsilon} n \log n)$, choosing the minimum is done in linear time, as is constructing the p_{min} -dimension close period. Thus the total time is $O(\frac{1}{\epsilon} n \log n)$. It should be noted, however, that the approximation factor is $3 + \epsilon$ because of the approximation of the shift vectors.

We have, therefore, proven the second part of Theorem 2.

4.3 Approximation in L_∞ Space

Finally, we describe a fast approximation of the closest periodic vector in L_∞ space. As in the previous cases, we need a bounding lemma for L_∞ .

Lemma 12. *Let V be a vector of dimension n , and let $1 \leq p \leq n/2$. Let V_P be the n -dimensional vector for which the p -dimensional vector P is the p -dimension close period of V under the L_∞ metric. Then*

- (a) $\frac{1}{2} d_{L_\infty}(V, V^p) \leq d_{L_\infty}(V, V_P)$.
- (b) $d_{L_\infty}(V, V_P) \leq \frac{\lceil n/p \rceil - 1}{2} d_{L_\infty}(V, V^p)$.

The concatenation lemma gives:

Lemma 13. *Given a vector $V \in \mathbb{R}^n$, and given p_{min} , $\frac{n}{4} < p_{min} \leq \frac{n}{2}$ for which $d_{L_\infty}(V, V^{p_{min}})$ is smallest. Then the p_{min} -dimension close period P approximates the period of the closest periodic vector of V to within a factor of 3.*

Approximation Algorithm for L_∞

1. Compute $d_{L_\infty}(V, V^p)$ for all $p = \lceil \frac{n}{4} \rceil, \dots, \lfloor \frac{n}{2} \rfloor$
2. Choose p_{min} for which $d_{L_\infty}(V, V^{p_{min}})$ is smallest.
3. Compute the p_{min} -dimension close period of V .

Time: By [14], for any $\epsilon > 0$, all $d_{L_\infty}(V, V^p)$ can be approximated to within a factor of $(1 + \epsilon)$ in time $O(\frac{1}{\epsilon}n \log n)$, choosing the minimum is done in linear time, as is constructing the p_{min} -dimension close period. Thus the total time is $O(\frac{1}{\epsilon}n \log n)$. It should be noted, however, that the approximation factor is $3 + \epsilon$ because of the approximation of the shift vectors.

We have, therefore, proven the third part of Theorem 2.

References

1. Amir, A., Benson, G.: Two-dimensional periodicity and its application. SIAM J. Comp. 27(1), 90–106 (1998)
2. Amir, A., Benson, G., Farach, M.: Optimal parallel two dimensional pattern matching. In: Proc. of the 5th ACM Symp. on Parallel Algorithms and Architectures, pp. 79–85 (1993)
3. Amir, A., Benson, G., Farach, M.: Optimal parallel two dimensional text searching on a crew pram. Information and Computation 144(1), 1–17 (1998)
4. Amir, A., Eisenberg, E., Levy, A.: Approximate Periodicity. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 25–36. Springer, Heidelberg (2010)
5. Amir, A., Eisenberg, E., Levy, A., Porat, E., Shapira, N.: Cycle Detection and Correction. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 43–54. Springer, Heidelberg (2010)
6. Apostolico, A., Giancarlo, R.: Periodicity and repetitions in parameterized strings. Discrete Appl. Math. 156(9), 1389–1398 (2008)
7. Cole, R., Crochemore, M., Galil, Z., Gasieniec, L., Harihan, R., Muthukrishnan, S., Park, K., Rytter, W.: Optimally fast parallel algorithms for preprocessing and pattern matching in one and two dimensions. In: Proc. 34th IEEE FOCS, pp. 248–258 (1993)
8. Crochemore, M.: An optimal algorithm for computing the repetitions in a word. Information Processing Letters 12(5), 244–250 (1981)
9. Crochemore, M., Galil, Z., Gasieniec, L., Hariharan, R., Muthukrishnan, S., Park, K., Ramesh, H., Rytter, W.: Parallel two-dimensional pattern matching. In: Proc. 34th Annual IEEE FOCS, pp. 248–258 (1993)
10. Galil, Z.: Optimal parallel algorithms for string matching. In: Proc. 16th ACM Symposium on Theory of Computing, vol. 67, pp. 144–157 (1984)
11. Galil, Z., Park, K.: Alphabet-independent two-dimensional witness computation. SIAM J. Comp. 25(5), 907–935 (1996)
12. Gfeller, B.: Finding longest approximate periodic patterns. In: WADS (2011)
13. Landau, G.M., Schmidt, J.P., Sokol, D.: An algorithm for approximate tandem repeats. Journal of Computational Biology 8(1), 1–18 (2001)

14. Lipsky, O., Porat, E.: Approximated Pattern Matching with the ℓ_1 , ℓ_2 and ℓ_∞ Metrics. In: Amir, A., Turpin, A., Moffat, A. (eds.) SPIRE 2008. LNCS, vol. 5280, pp. 212–223. Springer, Heidelberg (2008)
15. Lothaire, M.: Combinatorics on Words. Addison-Wesley, Reading (1983)
16. Ma, S., Hellerstein, J.L.: Mining partially periodic event patterns with unknown periods. In: The 17th International Conference on Data Engineering (ICDE), pp. 205–214. IEEE Computer Society (2001)
17. Régnier, M., Rostami, L.: A unifying look at d-dimensional periodicities and space coverings. In: Proc. 4th Symp. on Combinatorial Pattern Matching, vol. 15, pp. 215–227 (1993)

Maximum Weight Digital Regions Decomposable into Digital Star-Shaped Regions*

Matt Gibson, Dongfeng Han, Milan Sonka, and Xiaodong Wu

Dept. of Electrical and Computer Engineering,
University of Iowa,
Iowa City, IA 52242, USA

Abstract. We consider an optimization version of the image segmentation problem, in which we are given a grid graph with weights on the grid cells. We are interested in finding the maximum weight subgraph such that the subgraph can be decomposed into two "star-shaped" images. We show that this problem can be reduced to the problem of finding a maximum-weight closed set in an appropriately defined directed graph which is well known to have efficient algorithms which run very fast in practice. We also show that finding a maximum-weight subgraph that is decomposable into m star-shaped objects is NP-hard for some $m > 2$.

1 Introduction

An area of work that has recently attracted extensive attention in the pattern recognition and computer vision communities is *image segmentation*. In this area of work, we are given an image and are interested in developing algorithms that are able to identify certain objects in the image. One important application of image segmentation is *medical imaging* in which we are interested in developing algorithms that can identify tumors, plan surgeries, measure tissue volumes, and identify other health related issues. There are many other applications of image segmentation including facial recognition and brake light detection.

Image Segmentation as an Optimization Problem. In an attempt to find a "good" segmentation, the problem is often cast as an optimization problem, see for example [19,10,3,4,8,6]. This often involves constructing a weighted grid graph where the grid cells in the graph correspond to the pixels in the input image. The weights are assigned in a way that captures the likelihood of a particular pixel being in the object of interest, and then we attempt find some subset of the grid that optimizes an objective function subject to some constraints. In this context, a subset of the grid cells is often called a *region*. These constraints often times incorporate information about the shape of the region we wish to identify.

* This material is based upon work supported by the National Science Foundation under Grant No. CCF-0830402 and Grant No. CCF-0844765 as well as by the National Institute of Health under Grant No. R01-EB004640.

In many applications, the object that we wish to segment will have some geometric structure, and thus we may be interested in finding an object that satisfies some geometric constraints. There have been several results which show that it is possible to develop algorithms which exploit some geometric structure and can find an optimal region efficiently. Examples of such objects include x -monotone regions, based monotone regions, rectilinear convex regions, and star-shaped regions [3,8,7].

Regions that can be Decomposed into Regions with “Simple” Structure. In a recent paper, Chun et al. [6] consider the maximum-weight region problem with a twist on the constraints of previous work. They are interested in finding a maximum-weight region that may not have simple geometric structure, but can be *decomposed* into objects with simple geometric structure. We say that a subset of the grid cells can be decomposed into m objects of a particular structure if and only if there exists a coloring of the grid cells using m colors such that each of the objects induced by the grid cells of each of the color classes have the desired structure. Chun et al. give an efficient algorithm for computing the maximum-weight region that is decomposable into base monotone regions corresponding to some given k axis parallel base lines, and they give an efficient algorithm for computing the maximum-weight region that can be decomposed into two digital star-shaped regions with respect to two given “center” grid cells.

Digital Geometry Tools. One main challenge when dealing with objects in digital geometry is that many standard geometric objects and definitions from Euclidean geometry do not have “trivial” counterparts in the digital setting. For example, it is a non-trivial task to define line segments between grid cells in the digital setting such that the line segments (1) satisfy some standard axioms of Euclidean line segments and (2) “look comparable” to their corresponding Euclidean line segments. Digital line segments that satisfy (1) are called *consistent* digital line segments. An interesting question that was recently settled is determining whether or not there exist consistent digital line segments which are similar to their Euclidean counterparts. Chun et al. [7] showed that there exists *consistent digital rays* (digital line segments which share a common endpoint) which satisfy all of the given properties and have asymptotically optimal Hausdorff distance with their Euclidean counterparts. They left the following as an open problem: determine if there are consistent digital line segments (line segments with distinct endpoints) with a similar guarantee on the Hausdorff distance. This open problem was recently settled in the affirmative in a result due to Christ et al. [5].

Our Contribution. Given a vertex-weighted grid graph, we consider the problem of finding a region that can be decomposed into two star-shaped regions. Stated more formally, we are given an $n \times n$ grid graph G (let $N := n \cdot n$ denote the total number of grid cells in the grid). For each grid cell g in the graph, we have a corresponding weight $w(g) \in \mathbb{R}$. Given a subset of the grid cells V' , we define the *weight of V'* to be $w(V') := \sum_{v \in V'} w(v)$, and we call V' a *region*.

We are interested in finding a maximum weight region that can be decomposed into two “star-shaped” regions. Recall that a region can be decomposed into two star-shaped regions if we can color each of the grid cells in the region one of two colors such that the grid cells of each color class are a star-shaped region. A region R in the grid is *star-shaped* if there is a grid cell $c \in R$ such that for any grid cell $r \in R$, every grid cell in the digital ray $\text{dig}(c, r)$ is in R (where $\text{dig}(c, r)$ is as defined in [5]). We say that a region R is *star-shaped with respect to a grid cell c'* if for every $r \in R$ we have $\text{dig}(c', r) \subseteq R$. Note that the only difference between the definitions is that in the second definition we are specifying which grid cell must be the “center” and in the first definition we allow any grid cell in the region to be the “center”. See Figure 1 for an illustration of a region that can be decomposed into two star-shaped regions and a region that cannot be decomposed into two star-shaped regions.

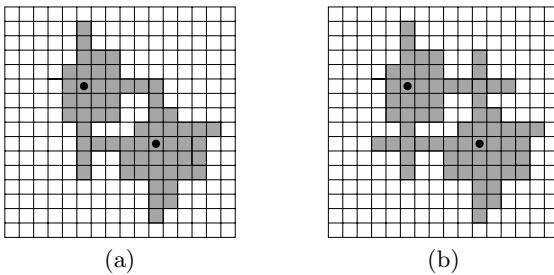


Fig. 1. (a) The shaded region is decomposable into star-shaped regions with respect to the cells with the black dots in them. (b) This shaded region is not decomposable into two star-shaped regions.

A dynamic programming algorithm was given for this problem by Chun et al. [6] with running time $O(N^3)$ where N is the total number of grid cells in the grid graph. Our main contribution is to prove the following theorem.

Theorem 1. *Given a weighted two-dimensional $N := n \times n$ grid graph and two grid cells c_1 and c_2 , the problem of finding the maximum-weight region that can be decomposed into a star-shaped region with respect to c_1 and a star-shaped region with respect to c_2 is equivalent to computing a maximum-weight closed set in an appropriately defined graph with $O(N)$ vertices and $O(N)$ edges.*

Due to its similarities with the maximum-flow problem, a maximum-weight closed set of a directed graph can be computed efficiently [12][11]. The worst case running time of our algorithm is $O(N^2 \log N)$, and in practice the running time is very fast and is easily implemented whereas the Chun et al. algorithm is mainly of theoretical significance. Another strength of our technique is that it could easily be extended to 3D if one were able to compute consistent digital line segments in 3D (which is currently an open problem).

We also show that the problem of finding a region that is decomposable into m star-shaped regions is NP-hard when the location of the m centers is not given. The reduction is from planar vertex cover and involves embedding an instance of planar vertex cover into an appropriately defined grid graph. The analysis takes advantage of the properties of consistent digital line segments.

Organization of the Paper. In Section 2, we show that the problem of finding the maximum weight region in a grid graph that can be decomposed into star-shaped regions with respect to two given centers is equivalent to computing a maximum weight closed set in an appropriately defined directed graph. In Section 3, we give the results of experiments of our algorithm. In Section 4, we show that the problem of finding a region that is decomposable into m star-shaped regions is NP-hard when the location of the m centers is not given.

2 The Algorithm

Our key contribution is to show that finding the maximum-weight region in a grid that can be decomposed into two star-shaped regions is equivalent to finding a maximum-weight closed set in an appropriately defined directed graph. Given a weighted, directed graph $D = (V, E)$, a *closed set* is a subset of the vertices $C \subseteq V$ such that if $u \in C$ and $(u, v) \in E$ then $v \in C$. Intuitively, if C is a closed set then there is no edge from a vertex in C to a vertex in $V \setminus C$. The weight of a closed set C is simply the sum of the weights of the vertices in C . It is well known that a maximum weight closed set can be computed in polynomial time [12][11].

We now will give a high level overview of the directed graph D that we construct, and some intuition as to why finding a maximum weight closed set in this graph is equivalent to a maximum weight region that can be decomposed into two star-shaped regions in G . There are two “sections” of vertices in D , and each each grid cell in G has exactly one vertex in each of these sections. The vertices in a closed set from the first section will determine what grid cells are in the first star in G , and the vertices in a closed set from the second section will determine what grid cells are in the second star in G . There are three sets of edges that we add to D . The first set of edges will have both endpoints in the first section of vertices, and their purpose is to ensure that the vertices chosen in the first section correspond with a star-shaped region with respect to c_1 in G . The second set of edges will have both endpoints in the second section of vertices, and their purpose is to ensure that the vertices in the second section correspond with a star-shaped region with respect to c_2 in G . The final set of edges will have one endpoint in the first section and the other endpoint in the second section, and their purpose is to ensure that the two resulting stars in G can be decomposed into two star-shaped regions. We assign weights to the vertices in D in a way so that the weight of a closed set in D is equal to the weight of the corresponding region (minus a constant) and vice versa.

Definitions. Given a digital ray $\text{dig}(c, g)$ in G , we define the *ray order* of the grid cells in $\text{dig}(c, g)$ to be the natural ordering of the grid cells where c is first in the ordering and g is the last in the ordering.

We will now define two stars over all of the grid cells in G , one with respect to c_1 and the other with respect to c_2 . These stars will be rooted trees over the grid cells in G with c_1 and c_2 being the root of the stars respectively. Intuitively, each path from the root to a leaf represents the grid cells in a digital ray that is in the star. Let B be the set of “boundary” grid cells of G (i.e. a grid cell that has less than four neighbors in G). Fix some $b \in B$, and consider the grid cells in the digital ray $\text{dig}(c_1, b)$ in ray order. We can define a parent/child relationship between these grid cells using this ordering as such: c_1 is the parent of the second grid cell in the ray, the second grid cell is the parent of the third grid cell, etc. We assign this relationship for $\text{dig}(c_1, b)$ for each $b \in B$. We call this star S_1 , and we define the star S_2 with respect to c_2 similarly. It is easy to see that S_1 and S_2 are spanning trees of G by following the properties of consistent digital rays and line segments given in [7,5].

Construction of the Directed Graph. We will now describe how we construct our weighted, directed graph D . Intuitively, there will be a “section” of the graph for S_1 and a “section” of the graph for S_2 . For each grid cell in G , there is a vertex in each such “section”. Let V_1 denote the vertices in the section for S_1 , and let us define V_2 similarly for S_2 . For a grid cell g , let v_g^1 denote its corresponding vertex in V_1 and let v_g^2 denote its corresponding vertex in V_2 .

We will now define three edge sets E_1 , E_2 , and E_3 . E_1 will consist of edges corresponding with the parent/child relationships from S_1 and will have both endpoints in V_1 , E_2 will consist of edges corresponding with the parent/child relationships from S_2 and will have both endpoints in V_2 , and E_3 will consist of edges with their tail in V_1 and their head in V_2 . Let us now define the edge set E_1 . Recall that we can view S_1 as a rooted tree where c_1 is the root and each of the rays define the parent/children relationship in the tree. Suppose g and g' are two grid cells such that g is the parent of g' in S_1 . Then we add the directed edge $(v_{g'}^1, v_g^1)$ to E_1 . Let us define D_1 to be the directed graph with vertex set V_1 and edge set E_1 . Note that D_1 is a rooted tree where the root is $v_{c_1}^1$ and all the edges are pointing “towards” the root. The edge set E_2 is defined similarly on V_2 , except there is one key difference. We think of S_2 as being a tree similarly to how we did with S_1 , but we reverse the direction of the edges. Thus the directed graph $D_2 = (V_2, E_2)$ is a rooted tree, but the edges in the graph orient away from the root. This completes the definition of the edge sets E_1 and E_2 . Now let us define the edge set E_3 . For each grid cell g , we add the directed edge (v_g^1, v_g^2) to E_3 . This completes the construction of the edge sets E_1 , E_2 , and E_3 .

Our directed graph D has vertex set $V := V_1 \cup V_2$ and edge set $E := E_1 \cup E_2 \cup E_3$. We assign weights on the vertices as follows. The weight of each vertex $v_g^1 \in V_1$ is set to be $w(g)$. The weight of each vertex $v_g^2 \in V_2$ is set to be $-w(g)$. This completes the construction of the graph.

We now describe a function T which will take as input a subset of vertices in D and outputs a subset of grid cells in G . Fix any subset $V' \subseteq V$ of D . For any

vertex $v_g^1 \in V' \cap V_1$, the corresponding grid cell g is in $T(V')$. For any vertex $v_g^2 \in V_2 \setminus V'$, the corresponding grid cell g is in $T(V')$. In other words, a grid cell g is in $T(V')$ if v_g^1 is in V' or if v_g^2 is not in V' . If v_g^1 is not in V' and v_g^2 is in V' , then g is not in $T(V')$. We will prove in Lemma 1 that if V' is a closed set of D then $T(V')$ can be decomposed into two star shaped objects whose weight is the same as the weight of V' (minus a constant).

We now define a transformation T' which takes as input a subset of grid cells that be decomposed into two star-shaped regions with respect to c_1 and c_2 and returns a set of vertices in D . The transformation is the inverse of T . Fix R to be any subset of grid cells that can be decomposed into a star-shaped region with respect to c_1 and a star-shaped region with respect to c_2 . Fix such a decomposition, and color the grid cells in the first star red and the cells in the second star blue. Let us call the red grid cells R_1 and the blue grid cells R_2 . For each red cell $r \in R_1$ we have that $v_r^1 \in T'(R)$ and $v_r^2 \in T'(R)$. For each blue cell $b \in R_2$ we have $v_b^1 \notin T'(R)$ and $v_b^2 \notin T'(R)$. For all uncolored cells g we have $v_g^1 \notin T'(R)$ and $v_g^2 \in T'(R)$. This concludes the definition of the transformation $T'(R)$, and in Lemma 2 we will prove that $T'(R)$ is a closed set in D and has weight equal to R (minus a constant).

Note that we have $T'(T(C)) = C$ for every closed set C and $T(T'(R)) = R$ for every decomposable region R . Thus proving Lemma 1 and Lemma 2 will complete the proof that the maximum-weight region in G that is decomposable into two star-shaped regions can be computed by finding a maximum-weight closed set in D .

Lemma 1. *Fix any closed set C of D . Then $T(C) \subseteq S_1 \cup S_2$ can be decomposed into two star-shaped regions and has weight equal to C (minus a constant).*

Proof. We first show that $T(C)$ can be decomposed into two star-shaped regions. Let C_1 be $C \cap V_1$, and abusing notation let $T(C_1) \subseteq T(C)$ be the grid cells g such that $v_g^1 \in C_1$. We will argue that $T(C_1)$ is a star-shaped object with respect to c_1 (the center of star S_1). This will be true as long as for any grid cell $g \in T(C_1)$ we have that the digital ray $\text{dig}(c_1, g) \subseteq T(C_1)$. We can show this is true by considering the construction of D . There is an edge in D from v_g^1 to the vertex corresponding to the grid cell immediately before g in $\text{dig}(c_1, g)$ (recall the definition of the ray ordering of the grid cells in a digital ray). Since C is a closed set, it follows that this vertex must also be in the closed set. It follows from a simple inductive argument that for any grid cell $c \in \text{dig}(c_1, g)$, the vertex v_c^1 must be in C . By the definition of T , it must be that c is in $T(C)$. This then implies that if $g \in T(C_1)$ then $\text{dig}(c_1, g) \subseteq T(C_1)$. We thus have that $T(C_1)$ is star-shaped with respect to c_1 .

Now let C_2 be $C \cap V_2$, and abusing notation let $T(C_2) \subseteq T(C)$ be the grid cells g such that $v_g^2 \notin C_2$. We will now show that $T(C_2)$ is a star shaped object with respect to c_2 . We remind the reader that by the definition of T , vertices in $V_2 \setminus C_2$ correspond with the grid cells in S_2 that are in $T(C_2)$. Again, to show that $T(C_2)$ is star shaped with respect to c_2 , we must show that for any grid cell

$g \in T(C_2)$, we have $\text{dig}(c_2, g) \subseteq T(C_2)$. Suppose for the sake of contradiction that $g \in T(C_2)$ but there is a grid cell $g' \in \text{dig}(c_2, g)$ that is not in $T(C_2)$. Since g' is not in $T(C_2)$, we have $v_{g'}^2 \in C_2$. According to the construction of D , there must be an edge from this vertex to the vertex corresponding to the grid cell immediately after g' in $\text{dig}(c_2, g)$. Since C is a closed set, we must have that this vertex is in C_2 . An inductive argument follows that all of the vertices corresponding to grid cells after g' in $\text{dig}(c_2, g)$ must be in C_2 . This of course implies that $g \notin T(C_2)$, a contradiction. We thus have that $T(C_2)$ is star-shaped with respect to c_2 .

We will now argue that $T(C)$ can be decomposed into two star-shaped regions. To prove this, we will color every grid cell either red or blue so that the red grid cells are a star-shaped region with respect to c_1 and the blue grid cells are a star-shaped region with respect to c_2 . We will prove this by showing that $T(C_1)$ and $T(C_2)$ are disjoint, and thus we can color the grid cells in $T(C_1)$ red and the grid cells in $T(C_2)$ blue to get the desired coloring. This is easy to see from the definition of the edge set E_3 . Let g be some grid cell in $T(C_1)$. By definition, this implies that $v_g^1 \in T(C_1)$. The edge (v_g^1, v_g^2) is in E_3 , and since C is a closed set it must be that $v_g^2 \in C$. This then implies that for any $g \in T(C_1)$, we have $g \notin T(C_2)$. Now let g be some grid cell in $T(C_2)$. By definition we have $v_g^2 \notin C_2$. Since the edge (v_g^1, v_g^2) is in E_3 , it must be that $v_g^1 \notin C_1$ because that would contradict the fact that C is a closed set. Therefore $g \notin C_1$. This completes the proof that $T(C_1)$ and $T(C_2)$ are disjoint and therefore they can be decomposed into two star-shaped regions.

This concludes the proof that $T(C)$ can be decomposed into two star-shaped regions, and we will now prove that C and $T(C)$ have the same weight (minus a constant). First let w_1 be the sum of the weights of the vertices in C_1 , and let w_2 be the sum of the weights of the vertices in C_2 . The weight of the closed set is exactly $w_1 + w_2$. The corresponding grid cell for each vertex in C_1 is also in $T(C)$, and moreover has the exact same weight. So the sum of the weights of the grid cells in $S_1 \cap T(C)$ is w_1 . Recall that the vertices in C_2 correspond to the exact set of grid cells that are not in $T(C)$, and thus the weight of the grid cells in $S_2 \cap T(C)$ is $w(S_2) + w_2$ (we remind the reader that the weight of a vertex in C_2 is the negative of the weight of its corresponding grid cell). Therefore, the weight of the grid cells in $T(C)$ is $w_1 + w_2 + w(S_2)$. Since $w_1 + w_2$ is the weight of C , we conclude that the weight of C is equal to the weight of the grid cells in $T(C)$ minus $w(S_2)$. This concludes the proof of the lemma. \square

We now state a similar lemma for T' . These two lemmas combined complete the proof of correctness of the algorithm. The proof of the lemma is quite similar to the proof of Lemma 1 and has been removed from this version of the paper due to lack of space.

Lemma 2. *Fix any subset R of grid cells in G that can be decomposed into two star shaped objects. Then $T'(R)$ is a closed set in D and has weight equal to R (minus a constant).*

3 Experiments

Our algorithm was implemented in ISO C++ on a standard PC with a 2.40GHz Intel R CoreTM2 Duo processor and 2 GB memory, running 32-bit Windows system. The max flow library [2] was utilized as the optimization tool. To simplify our algorithm, the weight of the pixels can be computed using low-level image features.

The running time of our algorithm was evaluated on images of ten different sizes as shown in Figure 2. The sizes of the images range from 100x100 to 700x700. The listed running time for each size of image is the average of 10 runs. We found most of the time was spent on construction of the directed graph. However, the current code is not fully optimized and there is much room for further improvement such as using a pyramid-based strategy to improve the segmentation speed. Figure 2 gives the running time for the execution of our algorithm on 10 different sized images. For each image, the running time is the average of 10 runs. To the best of our knowledge, this is the first time that an algorithm of applied interest has been given for this problem.

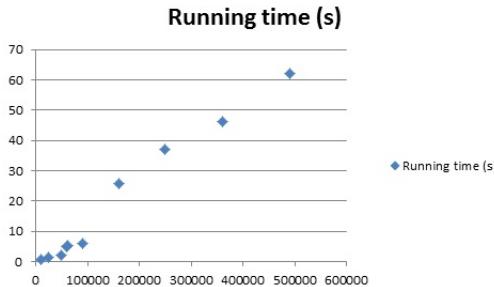


Fig. 2. Chart displaying the average running times of the algorithm for various sized images. The x-axis the number of pixels in the image and the y-axis is the running time of the algorithm in seconds.

We show the results for two images which serve as examples of images containing objects that can be decomposed into two star-shaped regions. In Figure 3, we segment two horses as an illustration of an object that does not have simple structure in itself but can be decomposed into a two star-shaped regions. In Figure 4, we segment a human brain using the black and red dots as the centers of the star shaped objects.

4 NP-Completeness

In this section, we consider the decision version of the problem. In this problem, we would like to know if there is a region in the grid that can be decomposed into m star-shaped regions whose weight is at least K for some $K > 0$. We show

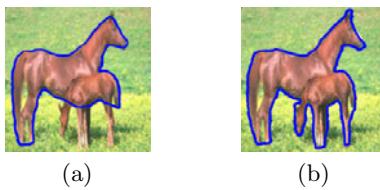


Fig. 3. Horse Segmentation: (a) results using only one center, and (b) result using two centers

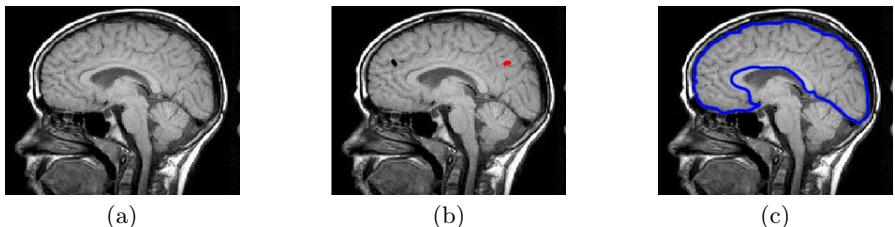


Fig. 4. Brain Segmentation: (a) original image, (b) the centers, and (c) the result of the algorithm

that the problem is NP-complete when we must choose where in the grid to place the m centers. The reduction is from vertex cover in planar graphs. The main idea is to take an instance of planar vertex cover and find a “grid embedding” of the graph of “high enough” resolution. We use the grid embedding to assign weights to the grid cells, and then show that there exists a region that can be decomposed into m star-shaped regions whose weight is at least K if and only if there is a vertex cover in the planar graph of size at most m . The details have been omitted from this version due to lack of space.

Acknowledgments. We thank the anonymous reviewers whose comments have significantly improved the quality of our paper.

References

1. Asano, T., Chen, D.Z., Katoh, N., Tokuyama, T.: Efficient algorithms for optimization-based image segmentation. *Int. J. Comput. Geometry Appl.* 11(2), 145–166 (2001)
2. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.* 23(11), 1222–1239 (2001)
3. Chen, D.Z., Chun, J., Katoh, N., Tokuyama, T.: Efficient Algorithms for Approximating a Multi-dimensional Voxel Terrain by a Unimodal Terrain. In: Chwa, K.-Y., Munro, J.I.J. (eds.) *COCOON 2004. LNCS*, vol. 3106, pp. 238–248. Springer, Heidelberg (2004)

4. Chen, D.Z., Hu, X.S., Luan, S., Wu, X., Yu, C.X.: Optimal Terrain Construction Problems and Applications in Intensity-modulated Radiation Therapy. In: Möhring, R.H., Raman, R. (eds.) *ESA 2002*. LNCS, vol. 2461, pp. 270–283. Springer, Heidelberg (2002)
5. Christ, T., Pálvölgyi, D., Stojakovic, M.: Consistent digital line segments. In: Snoeyink, J., de Berg, M., Mitchell, J.S.B., Rote, G., Teillaud, M. (eds.) *Symposium on Computational Geometry*, pp. 11–18. ACM, New York (2010)
6. Chun, J., Kasai, R., Korman, M., Tokuyama, T.: Algorithms for Computing the Maximum Weight Region Decomposable into Elementary Shapes. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) *ISAAC 2009*. LNCS, vol. 5878, pp. 1166–1174. Springer, Heidelberg (2009)
7. Chun, J., Korman, M., Nöllenburg, M., Tokuyama, T.: Consistent digital rays. *Discrete & Computational Geometry* 42(3), 359–378 (2009)
8. Chun, J., Sadakane, K., Tokuyama, T.: Efficient algorithms for constructing a pyramid from a terrain. *IEICE Transactions* 89-D(2), 783–788 (2006)
9. Fukuda, T., Morimoto, Y., Morishita, S., Tokuyama, T.: Data mining using two-dimensional optimized accociation rules: Scheme, algorithms, and visualization. In: Jagadish, H.V., Mumick, I.S. (eds.) *SIGMOD Conference*, pp. 13–23. ACM Press (1996)
10. Fukuda, T., Morimoto, Y., Morishita, S., Tokuyama, T.: Data mining with optimized two-dimensional association rules. *ACM Trans. Database Syst.* 26(2), 179–213 (2001)
11. Hochbaum, D.S.: A new - old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks* 37(4), 171–193 (2001)
12. Picard, J.-C.: Maximal closure of a graph and applications to combinatorial problems. *Management Science* 22(11), 1268–1272 (1976)

Finding Maximum Sum Segments in Sequences with Uncertainty*

Hung-I Yu¹, Tien-Ching Lin¹, and D.T. Lee^{1,2}

¹ Institute of Information Science, Academia Sinica,
Nankang, Taipei 115, Taiwan

{herbert,kero,dtlee}@iis.sinica.edu.tw

² Department of Computer Science and Engineering,
National Chung Hsing University, Taichung, Taiwan
dtlee@nchu.edu.tw

Abstract. In this paper, we propose to study the famous *maximum sum segment* problem on a sequence consisting of n uncertain numbers, where each number is given as an interval characterizing its possible value. Given two integers L and U , a segment of length between L and U is called a *potential maximum sum segment* if there exists a possible assignment of the uncertain numbers such that, under the assignment, the segment has maximum sum over all segments of length between L and U . We define the *maximum sum segment with uncertainty* problem, which consists of two sub-problems: (1) reporting all potential maximum sum segments; (2) counting the total number of those segments. For $L = 1$ and $U = n$, we propose an $O(n + K)$ -time algorithm and an $O(n)$ -time algorithm, respectively, where K is the number of potential maximum sum segments. For general L and U , we give an $O(n(U - L))$ -time algorithm for either problem.

Keywords: Sequences, Maximum Sum Segments, Uncertainty.

1 Introduction

Molecular biology is to discover the organizing principles that are conserved across all organisms, which is a very fascinating research area. All life on earth depends on three types of molecules: DNA, RNA, and proteins. Traditionally, DNA, RNA and proteins are described by sequences of letters that represent nucleobases or amino acids. Since the organizing principles are hidden in DNA, RNA, and protein sequences, these sequences can be viewed as "algorithms of life". Therefore, analyzing molecular sequences becomes a major topic in molecular biology. The purpose of sequence analysis is to identify biologically significant subsequences in a given biological sequence.

Given a biological sequence $Q = \{q_1, q_2, \dots, q_n\}$, a common approach for identifying significant subsequences in Q is to assign each letter q_i a real number

* Research supported by the National Science Council under the Grants No. NSC-98-2221-E-001-008 and NSC-98-2221-E-001-008.

p_i , based on the optimization criteria of the application and biological experiment results, for constructing a real number sequence $P = \{p_1, p_2, \dots, p_n\}$, and then to search for contiguous subsequences satisfying certain properties in P . This approach was applied to a variety of sequence analysis problems, such as the identification of transmembrane regions, DNA binding domains, regions of high charge, GC-rich regions, and non-coding RNA genes [7], [10], [12].

However, biological experiments are prone to error, and provides us only estimates with numerical errors, instead of precise numbers. They may contain a lower error bound α_i and an upper error bound β_i , for each i , such that p_i is within the interval $[\alpha_i, \beta_i]$. Therefore, the sequence P obtained from experimental data may be different from the exact sequence, and the significant subsequences recognized for P could be insignificant for the exact sequence. In this paper, we propose to study the problem of identifying subsequences that have possibility to be significant in the exact sequence, assuming that the numerical errors of numbers are predictable and can be described by intervals. After identifying these subsequences with potential significance, we are able to concentrate on these smaller subsequences with more precise or more expensive experiments, instead of on the whole sequence, to filter out all insignificant segments.

As a good starting point, we consider the case that significance is measured for each subsequence by the sum of numbers within it, which corresponds to the famous *maximum sum segment* problem when the input sequence consists of exact numbers. Given a sequence P of n real numbers and two positive integers L, U with $L \leq U$, the maximum sum segment problem is to find a segment $P(i, j)$ with $L \leq j - i + 1 \leq U$ that maximizes $\sum_{t=i}^j p_t$. This problem has useful applications in bioinformatics including: (1) finding tandem repeats [13], which is commonly used to map disease genes; (2) locating DNA segments with rich G+C content, which is a key step in gene finding and promoter prediction [2], [4], [5], [9]; (3) designing low complexity filter, which is mostly applied in sequence database search [1]. The best known algorithm for the maximum sum segment problem, by Lin et al. [11], runs in optimal $O(n)$ time, based upon a clever technique called *left-negative decomposition*. Fan et al. [6] give another optimal $O(n)$ time algorithm handling the input sequence in an online manner.

Suppose that we are given an input sequence A with uncertainty, in which each number is not given as an exact value but as an interval characterizing the possible range of its value. A segment of A is called a *potential maximum sum segment* if there exists a possible assignment of the uncertain numbers such that the segment has maximum sum under the assignment. We define the *maximum sum segment with uncertainty (MSSU)* problem, which consists of two sub-problems: (1) reporting all potential maximum sum segments; (2) counting the total number of those segments. When the segment lengths are not constrained, that is, $L = 1$ and $U = n$, we propose an $O(n + K)$ -time algorithm for the MSSU reporting problem and an $O(n)$ -time algorithm for the MSSU counting problem, where K is the number of potential maximum sum segments. For the length-constrained case, we give an $O(n(U - L))$ -time algorithm for either the reporting or counting problem. Due to page limit, the proofs are omitted.

2 Notation and Preliminaries

Let $P = \{p_1, p_2, \dots, p_n\}$ be a sequence of n real numbers. For $1 \leq i \leq j \leq n$, let $P(i, j)$ denote the segment $\{p_i, \dots, p_j\}$ of P . We define the sum and the length of $P(i, j)$ to be $\sum_{t=i}^j p_t$ and $j - i + 1$, respectively. Given two integer constraints L and U with $1 \leq L \leq U \leq n$, a segment $P(i, j)$ is called *feasible* if its length is between L and U . If $P(i, j)$ is feasible and its sum is maximum over all feasible segments, it is called a *maximum sum segment* of P . For a fixed segment $P(i, j)$, we refer to the sums of $P(i, i), P(i, i+1), \dots, P(i, j)$ as the *prefix sums* of $P(i, j)$ and the sums of $P(i, j), P(i+1, j), \dots, P(j, j)$ as the *suffix sums* of $P(i, j)$.

A number a is defined to be *uncertain* if its value has not been recognized exactly, but is known to be some real number within a given interval $[\alpha, \beta]$, where $\alpha, \beta \in \mathbb{R}$. Let $A = \{a_1, a_2, \dots, a_n\}$ be a sequence of n uncertain numbers, where $[\alpha_i, \beta_i]$ is the interval associated with a_i for each i . Let \mathcal{R} be the Cartesian product of the n intervals $[\alpha_i, \beta_i]$. Any element $R \in \mathcal{R}$ is called a *realization*, which describes a possible assignment of real numbers to the uncertain numbers in A . For any realization $R \in \mathcal{R}$, let a_i^R denote the real number assigned to a_i by R , A^R denote the sequence $\{a_1^R, a_2^R, \dots, a_n^R\}$, and $\text{sum}^R(i, j)$ denote the sum of the segment $A^R(i, j)$. A segment $A(i, j)$ is defined to be a *potential maximum sum segment* of A if there exists a realization $R \in \mathcal{R}$ such that $A^R(i, j)$ is a maximum sum segment of A^R .

Our objective is to find all potential maximum sum segments of A . Due to output space consideration, we refer to each segment $A(i, j)$ by a tuple (i, j) , which is called the *location* of $A(i, j)$, and define the term *potential locations* accordingly. The *maximum sum segment with uncertainty* (*MSSU*) problem consists of the following two sub-problems:

Reporting: Report all potential locations in A .

Counting: Count the total number of potential locations in A .

The setting of length constraints L and U further divides the problem into two cases. The MSSU problem is *length-relaxed* when $L = 1$ and $U = n$, and is *length-constrained* otherwise.

For $1 \leq i \leq j \leq n$, let $R_{i,j} \in \mathcal{R}$ be the realization in which $a_t^{R_{i,j}} = \beta_i$ for $i \leq t \leq j$ and $a_t^{R_{i,j}} = \alpha_i$ otherwise. We abbreviate $A^{R_{i,j}}$ as $A^{i,j}$ and call it the i, j -peak sequence. Also, let $a_t^{i,j} = a_t^{R_{i,j}}$ and $\text{sum}^{i,j}() = \text{sum}^{R_{i,j}}()$ for short. These specific sequences are important for identifying potential locations, as shown in the following lemma.

Lemma 1. *A location (i, j) is a potential location if and only if $A^{i,j}(i, j)$ is a maximum sum segment of the i, j -peak sequence.*

Note that this lemma reduces the MSSU problem to $O(n(U - L))$ maximum sum segment problems, which implies a trivial $O(n^2(U - L))$ -time algorithm by using any optimal maximum sum segment algorithm.

For $1 \leq i \leq j \leq n$, let $\alpha(i, j) = \sum_{t=i}^j \alpha_t$ and $\beta(i, j) = \sum_{t=i}^j \beta_t$. For simplicity of description, let $\alpha(i, j) = \beta(i, j) = 0$ if $i > j$, and let $\alpha(i, j) = -\infty$ and $\beta(i, j) = \infty$ if $i < 1$ or $j > n$.

3 The Length-Relaxed MSSU Problem

In this section, we study the length-relaxed MSSU problem and propose linear time algorithms for both the reporting and counting problems. In Subsection 3.1, we provide properties and preprocessing for efficiently determining whether a given location is a potential location. Then, by using the structural similarities between i, j -peak sequences, we propose an $O(n \log n + K)$ -time reporting algorithm and an $O(n \log n)$ -time counting algorithm in Subsection 3.2. Finally, in Subsection 3.3, we improve the time complexities to $O(n + K)$ -time and $O(n)$ -time, respectively.

3.1 Preprocessing

Given a fixed location (i, j) , we first discuss basic properties for determining whether (i, j) is a potential location. A segment $A^{i,j}(x, y)$ is called a *dominating segment* of (i, j) if $\text{sum}^{i,j}(x, y) > \text{sum}^{i,j}(i, j)$. By Lemma 1, (i, j) is a potential location if and only if there is no dominating segment of (i, j) . In the following, we classify the segments in $A^{i,j}$ other than $A^{i,j}(i, j)$ into four types, and then consider the necessary and sufficient conditions that there exists no dominating segment in each type. A segment $A^{i,j}(x, y)$ in $A^{i,j}$ is called a

- *disjoint* segment if $y < i$ or $x > j$,
- *covering* segment if $x < i \leq j \leq y$ or $x \leq i \leq j < y$,
- *inner* segment if $i < x \leq y \leq j$ or $i \leq x \leq y < j$,
- *crossing* segment if $x < i \leq y < j$ or $i < x \leq j < y$.

Obviously, there is no dominating disjoint segment of (i, j) if and only if $\text{sum}^{i,j}(i, j)$ is larger than or equal to the maximum sum appearing to the left of i or to the right of j in $A^{i,j}$. For $1 \leq t \leq n$, we define $\text{Dis}_L(t) = \max\{\alpha(x, y) \mid y < t\}$. Since $\text{sum}^{i,j}(x, y) = \alpha(x, y)$ for $y < i$, $\text{Dis}_L(i)$ represents the maximum value among the sums of disjoint segments to the left of i . Symmetrically, we define $\text{Dis}_R(t) = \max\{\alpha(x, y) \mid x > t\}$. For notational brevity, let $\text{Dis}_L(1) = \text{Dis}_R(n) = -\infty$. Let $\text{Dis}(x, y) = \max\{\text{Dis}_L(x), \text{Dis}_R(y)\}$. Since $\text{sum}^{i,j}(i, j) = \beta(i, j)$, we have the following property.

Property 1. *There is no dominating disjoint segment of (i, j) if and only if $\beta(i, j) \geq \text{Dis}(i, j)$.*

If there exists a dominating covering segment $A^{i,j}(x, y)$, we can see that either $\text{sum}^{i,j}(x, i-1) > 0$ or $\text{sum}^{i,j}(j+1, y) > 0$. Thus, there is no dominating covering segment of (i, j) if and only if i cannot be extended to the left and j cannot be extended to the right by positive-sum segments. We say that i is *left-inextensible* if all suffix sums of $A^{i,j}(1, i-1)$ are non-positive, i.e., $\text{sum}^{i,j}(x, i-1) \leq 0$ for $x \leq i-1$. Similarly, j is *right-inextensible* if all prefix sums of $A^{i,j}(j+1, n)$ are non-positive, i.e., $\text{sum}^{i,j}(j+1, y) \leq 0$ for $j+1 \leq y$.

The extensibility can be checked as follows. For $1 \leq t \leq n$, let $\text{Pos}_L(t)$ be the largest index $x \leq t$ such that $\alpha(x, t) > 0$, and $\text{Pos}_R(t)$ be the smallest index

$y \geq t$ such that $\alpha(t, y) > 0$. Let $Pos_L(t) = 0$ and $Pos_R(t) = n + 1$ in case that such indices do not exist. We can see that, by definition, $Pos_L(i - 1) = 0$ means that $sum^{i,j}(x, i - 1) = \alpha(x, i - 1) \leq 0$ for $x \leq i - 1$ and thus i is left-inextensible. Also, $Pos_R(j + 1) = n + 1$ indicates that j is right-inextensible. (For notational brevity, $Pos_L(0) = 0$ and $Pos_R(n + 1) = n + 1$.) We have the following.

Property 2. *There is no dominating covering segment of (i, j) if and only if $Pos_L(i - 1) = 0$ (i is left-inextensible) and $Pos_R(j + 1) = n + 1$ (j is right-inextensible).*

Similarly, if there exists a dominating inner segment $A^{i,j}(x, y)$, we can see that either $sum^{i,j}(i, x - 1) < 0$ or $sum^{i,j}(y + 1, j) < 0$. So there is no dominating inner segment of (i, j) if and only if no negative-sum inner segment beginning from i or ending at j can be reduced from $A^{i,j}(i, j)$. We say that i is *irreducible for j* if all prefix sums of $A^{i,j}(i, j - 1)$ are non-negative, i.e. $sum^{i,j}(i, t) \geq 0$ for $i \leq t < j$; and j is *irreducible for i* if all suffix sums of $A^{i,j}(i + 1, j)$ are non-negative, i.e. $sum^{i,j}(t, j) \geq 0$ for $i < t \leq j$.

For $1 \leq t \leq n$, let $Neg_L(t)$ be the largest index $x \leq t$ such that $\beta(x, t) < 0$, which implies that $\beta(x, t) \geq 0$ for $Neg_L(t) + 1 \leq x \leq t$. Obviously, $Neg_L(j) \leq i$ means that j is irreducible for i . Symmetrically, let $Neg_R(t)$ be the smallest index $y \geq t$ such that $\beta(t, y) < 0$, so that $Neg_R(i) \geq j$ gives the fact that i is irreducible for j . (Again, $Neg_L(t) = 0$ and $Neg_R(t) = n + 1$ in case that no proper indices exist.) The following property can be obtained.

Property 3. *There is no dominating inner segment of (i, j) if and only if $Neg_R(i) \geq j$ (i is irreducible for j) and $Neg_L(j) \leq i$ (j is irreducible for i).*

The following property shows that we can ignore the existence of dominating crossing segments while determining potential locations in the length-relaxed MSSU problem.

Property 4. *If there is no dominating covering and inner segment of (i, j) , there is no dominating crossing segment of (i, j) .*

Summarizing the above discussion, we have the following.

Lemma 2. *A location (i, j) is a potential location if and only if the following conditions hold: (a) $\beta(i, j) \geq Dis(i, j)$, (b) $Pos_L(i - 1) = 0$ (i is left-inextensible), (c) $Pos_R(j + 1) = n + 1$ (j is right-inextensible), (d) $Neg_R(i) \geq j$ (i is irreducible for j), and (e) $Neg_L(j) \leq i$ (j is irreducible for i).*

We can pre-compute all values defined above in $O(n)$ time, so that for any location (i, j) each of the five conditions can be verified in constant time. By computing the prefix sums $\alpha(1, t)$ for $1 \leq t \leq n$ in $O(n)$ time, for any x, y , we can compute $\alpha(x, y) = \alpha(1, y) - \alpha(1, x - 1)$ in constant time. Similarly, $\beta(x, y)$ can also be computed in constant time. $Dis_L(t)$ for each t can be obtained in $O(n)$ time by using the algorithm for the maximum sum segment problem in [3]. $Dis_R(t)$ for each t can also be obtained in $O(n)$ time in a symmetric way. Using

a result in Lin *et. al.* [11], $Pos_R(t)$ for each t can be computed in $O(n)$ time, and, with a slight modification, so do $Pos_L(t)$, $Neg_L(t)$ and $Neg_R(t)$ for each t . The above preprocessing takes totally $O(n)$ time. After preprocessing we can obtain a simple $O(n^2)$ -time algorithm for the length-relaxed MSSU problem by checking each of the $O(n^2)$ locations with Lemma 2 in constant time.

3.2 Basic Algorithm

The basic concept of our improvement over the $O(n^2)$ -time solution is as follows. We first construct the set of indices i that may form potential locations (i, j) for each index j . Then, we exploit the property within each set to obtain potential locations for each j efficiently.

For a fixed index j , we say that an index $i \leq j$ is a *candidate* for j , if i is both left-inextensible and irreducible for j (conditions (b) and (d) in Lemma 2). Let C_j denote the set of candidates for j . Note that, by definition, j is irreducible for j itself. Thus, $j \in C_j$ if j is left-inextensible, which also implies that $C_1 = \{1\}$. The next lemma shows the similarity between C_{j-1} and C_j for $j > 1$, so that C_j can be easily obtained from C_{j-1} .

Lemma 3. *For any index i with $1 \leq i < j$, $i \in C_j$ if and only if $i \in C_{j-1}$ and $\beta(i, j-1) \geq 0$.*

The following corollary exhibits the property of indices within each candidate set, which is crucial to efficient identification of potential locations.

Corollary 4. *For any two indices $i_1, i_2 \in C_j$ with $i_1 < i_2$, $\beta(i_1, j) \geq \beta(i_2, j)$.*

Now, we show how to find potential locations ending at j by identifying a "beginning" index and an "end" index within C_j , such that indices of C_j in between them satisfy conditions (a) and (e) in Lemma 2 in addition. Let b_j be the smallest index in C_j if it exists, such that j is irreducible for b_j . We can see that j is also irreducible for any $i > b_j$ by the definition of $Neg_L(j)$. Let e_j be the largest index in C_j if it exists, such that $\beta(e_j, j) \geq Dis(e_j, j)$. Define $b_j = n+1$ and $e_j = 0$ in case they do not exist. Since $Dis(i, j)$ is non-decreasing in i by definition and $\beta(i, j)$ is non-increasing in i for $i \in C_j$ by Corollary 4, the two lemmas follow.

Lemma 5. *For any index $i \in C_j$, $\beta(i, j) \geq Dis(i, j)$ if $i \leq e_j$ and $\beta(i, j) < Dis(i, j)$ if $i > e_j$.*

Lemma 6. *A location (i, j) is a potential location if and only if j is right-inextensible, $i \in C_j$, and $b_j \leq i \leq e_j$.*

Based on the above discussion, we propose the first algorithm for the length-relaxed MSSU problem. The algorithm iterates from $j = 1$ to n for finding all potential locations ending at j . Each iteration j consists of two phases: Phase 1 computes the candidate set C_j from C_{j-1} , and Phase 2 finds potential locations with the help of C_j .

We first explain the details in Phase 1 independently. The algorithm uses a single array C to implement candidate sets, and maintains two invariants:

- **Invariant 1:** At iteration j , the set of indices in C is C_j after Phase 1.
- **Invariant 2:** Indices in C are arranged in increasing order.

At iteration 1, index 1 is obviously left-inextensible and is inserted into C as its single element. Since $C_1 = \{1\}$, Invariants 1 and 2 are both maintained. Consider the computation at iteration j for $j > 1$. In the beginning of Phase 1, Invariant 1 ensures that C represents C_{j-1} . By Lemma 3, each index i in C with $\beta(i, n) < \beta(j, n)$, which implies that $\beta(i, j-1) < 0$, does not belong to C_j and must be removed. By Invariant 2 and Corollary 4, it is easy to see that indices in C are arranged in non-increasing order of $\beta(i, j)$, and in non-increasing order of $\beta(i, n)$, too. Thus, indices to be removed must form a consecutive subarray at the end of C , and can be removed by linearly scanning C from its end. Finally, we insert the index j to the end of C if j is left-inextensible. The insertion does not violate Invariant 2, since j is larger than any index in C . After the computation, Invariant 1 is obviously maintained and Phase 1 of iteration j finishes.

Phase 2 computes potential locations according to Lemma 6. Consider the computation at iteration j for $j \geq 1$. At first, we verify whether j is right-inextensible and terminate this iteration if not. Then, by Lemma 6, any potential location (i, j) satisfies that $i \in C_j$ and $b_j \leq i \leq e_j$. Invariant 1 enables us to verify the conditions inside C . By Invariant 2, if $b_j \leq e_j$, all indices i satisfying $i \in C_j$ and $b_j \leq i \leq e_j$ appear in between b_j and e_j within C , and form a consecutive subarray $C[\pi_{b_j}, \dots, \pi_{e_j}]$, where π_{b_j} and π_{e_j} denote the positions of b_j and e_j in C , respectively. Thus, the main problem is to compute π_{b_j} and π_{e_j} . By definition, π_{b_j} can be located by finding the position of the smallest element i in C with $i \geq Neg_L(j)$. Since C is sorted in increasing order of indices, the finding is easily done by performing binary search in C with the value of $Neg_L(j)$. Similarly, π_{e_j} points to the last element i in C with $\beta(i, j) \geq Dis(i, j)$, and can be determined again by binary search based on Lemma 5. Finally, if $\pi_{b_j} \leq \pi_{e_j}$, we output the location (i, j) for each index i in the subarray $C[\pi_{b_j}, \dots, \pi_{e_j}]$.

The time complexity of the algorithm is analyzed as follows. Pre-computing the required values takes $O(n)$ time as discussed in Subsection 3.1. Phase 1 takes totally $O(n)$ time over n iterations, since each index can be inserted into and removed from C at most once. As to Phase 2, since finding π_{b_j} and π_{e_j} for any j by binary search takes $O(\log n)$ time, overall the finding can be done in total $O(n \log n)$ time. Finally, since there are total K potential locations, the output process over n iterations requires $O(K)$ time. Thus, we have the following.

Theorem 1. *The length-relaxed MSSU reporting problem can be solved in $O(n \log n + K)$ time.*

Since the number of potential locations outputted at iteration j can be computed as $\max\{0, \pi_{e_j} - \pi_{b_j} + 1\}$, the following result can be easily obtained.

Theorem 2. *The length-relaxed MSSU counting problem can be solved in $O(n \log n)$ time.*

3.3 Improved Algorithm

The bottleneck of the previous algorithm is obviously the binary search used for finding π_{b_j} and π_{e_j} . In this subsection, we develop further properties for b_j and e_j and discuss how to modify the algorithm based on the properties, such that the overall time for the finding can be accelerated to $O(n)$.

We first show that b_j can be computed simply by performing a range maximum query [8]. Let F denote the sequence $\{f_1, f_2, \dots, f_n\}$, where $f_i = \beta(i, n)$ if i is left-inextensible and $f_i = -\infty$ otherwise, for $1 \leq i \leq n$. The following lemma can be obtained.

Lemma 7. *If b_j exists, b_j is the smallest index between $\text{Neg}_L(j)$ and j such that $f_{b_j} = \max\{f_i \mid \text{Neg}_L(j) \leq i \leq j\}$.*

This lemma enables us to find b_j directly in the sequence F . The *range maximum query* problem is to preprocess an array R of n numbers so that, given any interval $[i, j]$, the index of the largest element in the subarray $R[i, \dots, j]$ can be answered. Previous results of this problem [8] show that, with an $O(n)$ -time preprocessing, each query can be answered in $O(1)$ time. Using the results, we can preprocess the sequence F in $O(n)$ time for answering range maximum queries, with ties resolved by comparing indices. Thus, if it exists, b_j can be found in $O(1)$ time by querying the interval $[\text{Neg}_L(j), j]$. Determining the existence of b_j can be easily done in the algorithm. Let c be the last element in C , which is also the largest index in C_j by Invariant 2. By definition of b_j we can see that b_j exists if and only if $c \geq \text{Neg}_L(j)$, which can be verified in $O(1)$ time.

From the framework of the algorithm, we still need to know the position π_{b_j} of b_j in C . Thus, we maintain an additional mapping array π such that, for each index i , $\pi[i]$ is the position of i in C if C contains i , and *null* otherwise. The maintenance of π is synchronized with that of C : when an index i is inserted into C , record its position in C into $\pi[i]$; and when it is removed, reset $\pi[i]$ to *null*. Thus, after b_j is found, $\pi_{b_j} = \pi[b_j]$ can also be obtained in $O(1)$ time.

The following lemma characterizes "meaningless" e_j 's, so that we can ignore the computation of such e_j 's, which makes it possible to maintain e_j 's in amortized linear time.

Lemma 8. *For a fixed index j , let (x, j') be an arbitrary location satisfying that $x \in C_j$, $j' < j$, and $\beta(x, j') \geq \text{Dis}(x, j')$. If $e_j < x$, there is no potential location ending at j .*

Based on this lemma, we maintain a pointer ℓ to positions of C in the algorithm that satisfies:

- **Invariant 3:** During iteration j , if $\ell > 0$, there exists some location $(C[\ell], j')$ with $j' \leq j$ such that $\beta(C[\ell], j') \geq \text{Dis}(C[\ell], j')$.

In the following, we show how to maintain ℓ and compute π_{e_j} from ℓ concurrently. Initially, let $\ell = 0$. At the beginning of iteration $j \geq 1$, if $\ell > 0$, by Invariant 3 there is a location $(C[\ell], j')$ with $j' \leq j - 1$ such that $\beta(C[\ell], j') \geq \text{Dis}(C[\ell], j')$,

which also satisfies Invariant 3 for j . During the index removal process in Phase 1, Invariant 3 always holds before $C[\ell]$ is removed. When $C[\ell]$ is going to be removed, by Corollary 4 and Invariant 2, we have $\beta(C[\ell - 1], j') \geq \beta(C[\ell], j') \geq Dis(C[\ell], j') \geq Dis(C[\ell - 1], j')$. Thus, $C[\ell]$ can be removed and Invariant 3 holds again by subtracting 1 from ℓ . Note that ℓ becomes 0 when all elements of C are removed, which also maintains Invariant 3.

Consider Phase 2 of iteration j . We first verify whether $\beta(C[\ell], j) \geq Dis(C[\ell], j)$. If false, $e_j < C[\ell]$ by the definition of e_j . Thus, there is no potential location ending at j by Lemma 8, and the iteration can be terminated directly. Otherwise, we have $e_j \geq C[\ell]$ and $\pi_{e_j} \geq \ell$. Then, π_{e_j} can be found by linearly scanning C from position ℓ toward the end. In order to reduce the computation of later iterations, we increase ℓ synchronously as π_{e_j} increases. By definition, for $\ell \leq \pi_{e_j}$, $\beta(C[\ell], j) \geq Dis(C[\ell], j)$, and thus Invariant 3 holds during the scanning process.

After these modifications, the time complexity is analyzed as follows. At first, an additional sequence F is created and preprocessed in $O(n)$ time. In Phase 1, the extra maintenance of the array π and ℓ obviously does not affect the overall running time. As to Phase 2, finding π_{b_j} by range maximum query for all j takes totally $O(n)$ time. The finding of π_{e_j} is synchronized with the maintenance of ℓ , which takes totally $O(n)$ time by the following lemma.

Lemma 9. ℓ increases at most n times in Phase 2 over n iterations.

Consequently, we obtain the following improved results.

Theorem 3. *The length-relaxed MSSU reporting problem can be solved in $O(n + K)$ time and the length-relaxed MSSU counting problem can be solved in $O(n)$ time.*

4 The Length-Constrained MSSU Problem

Unlike the length-relaxed case, the main difficulty of the length-constrained MSSU problem is the existence of dominating crossing segments. For example, consider the instance $A = \{[0, 1], [0, 1], [-1, 1], [4, 4]\}$, $L = 2$, and $U = 3$. The feasible location $(1, 2)$, with $sum^{1,2}(1, 2) = 2$, has no dominating covering and inner segment in $A^{1,2}$, but has a dominating crossing segment $A^{1,2}(2, 4)$ with $sum^{1,2}(2, 4) = 4$. In fact, a feasible location (i, j) has $O((j - i + 1)(U - L))$ crossing segments, and we do have to compute a segment of maximum sum from them for checking whether (i, j) has a dominating crossing segment. Again by observing similarities, we propose a dynamic programming algorithm of $O(n(U - L))$ time and space to do the computation for all feasible locations (i, j) . Thus, we claim the following result. Due to page limit, the details are omitted.

Theorem 4. *The length-constrained MSSU reporting and counting problem can be solved in $O(n(U - L))$ time and $O(n(U - L))$ space.*

5 Concluding Remarks

In this paper, we proposed a new trend of research problem in sequence analysis. It is to identify some biologically significant subsequences in a given biological sequence with some prior input errors.

As a good starting point, we considered the maximum sum segment problem with uncertainty. We shown that for the length-relaxed case, an optimal $O(n + K)$ -time algorithm for the MSSU reporting problem and an $O(n)$ -time algorithm for the MSSU counting problem can be obtained. For the length-constrained case, we give an $O(n(U - L))$ -time and $O(n(U - L))$ -space algorithm for either the MSSU reporting or counting problem. It would be a great challenge to close the gap between the two cases. Also, it is worthy to consider other sequence problems with some prior input errors.

References

1. Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D.: Basic local alignment search tool. *Journal of Molecular Biology* 215, 403–410 (1990)
2. Barhardi, G.: Isochores and the evolutionary genomics of vertebrates. *Gene* 241, 3–17 (2000)
3. Bentley, J.: Programming pearls: algorithm design techniques. *Communications of the ACM* 27(9), 865–873 (1984)
4. Bernardi, G., Bernardi, G.: Compositional constraints and genome evolution. *Journal of Molecular Evolution* 24, 1–11 (1986)
5. Davuluri, R., Grosse, I., Zhang, M.: Computational identification of promoters and first exons in the human genome. *Nature Genetics* 29, 412–417 (2001)
6. Fan, T.-H., Lee, S., Lu, H.-I., Tsou, T.-S., Wang, T.-C., Yao, A.: An Optimal Algorithm for Maximum-Sum Segment and Its Application in Bioinformatics Extended Abstract. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 251–257. Springer, Heidelberg (2003)
7. Fariselli, P., Finelli, M., Marchignoli, M., Martelli, P.L., Rossi, I., Casadio, R.: Maxsubseq: An algorithm for segment-length optimization. The case study of the transmembrane spanning segments. *Bioinformatics* 19, 500–505 (2003)
8. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: 16th Annual ACM Symposium on Theory of Computing, pp. 135–143. ACM Press, New York (1984)
9. Hannenhalli, S., Levy, S.: Promoter prediction in the human genome. *Bioinformatics* 17, S90–S96 (2001)
10. Huang, X.: An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. *Computer Applications in Biosciences* 10(3), 219–225 (1994)
11. Lin, Y.-L., Jiang, T., Chao, K.-M.: Efficient algorithms for locating the length-constrained heaviest segments, with applications to biomolecular sequence analysis. *Journal of Computer and System Sciences* 65(3), 570–586 (2002)
12. Miklós, C.: Maximum-scoring segment sets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 1(4), 139–150 (2004)
13. Walder, R., Garrett, M., McClain, A., Beck, G., Brennan, T., Kramer, N., Kanis, A., Mark, A., Rapp, A., Sheffield, V.: Short tandem repeat polymorphic markers for the rat genome from marker-selected libraries associated with complex mammalian phenotypes. *Mammalian Genome* 9, 1013–1021 (1998)

Algorithms for Building Consensus MUL-trees

Yun Cui¹, Jesper Jansson^{2,*}, and Wing-Kin Sung^{1,3}

¹ National University of Singapore, 13 Computing Drive, Singapore 117417

{yuncui01@gmail.com, ksung@comp.nus.edu.sg}

² Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan

Jesper.Jansson@ocha.ac.jp

³ Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672

Abstract. A MUL-tree is a generalization of a phylogenetic tree that allows the same leaf label to be used many times. Lott *et al.* [9,10] recently introduced the problem of inferring a so-called *consensus MUL-tree* from a set of conflicting MUL-trees and gave an exponential-time algorithm for a special greedy variant. Here, we study *strict* and *majority rule consensus MUL-trees*, and present the first ever polynomial-time algorithms for building a consensus MUL-tree. We give a simple, fast algorithm for building a strict consensus MUL-tree. We also show that although it is NP-hard to find a majority rule consensus MUL-tree, the variant which we call the *singular majority rule consensus MUL-tree* is unique and can be constructed efficiently.

1 Introduction

To describe tree-like evolutionary history, scientists often use a data structure known as the *phylogenetic tree* [3,17]. In traditional applications, phylogenetic trees were always distinctly leaf-labeled, and in fact, the computational efficiency of most existing methods for constructing and comparing phylogenetic trees implicitly depends on this uniqueness property. The *multi-labeled phylogenetic tree*, or *MUL-tree* for short, is a generalization of the standard phylogenetic tree model that allows the same leaf label to be used more than once in a single tree structure; for some examples, see Fig. 2 and 3. MUL-trees have applications in different research fields such as Molecular Systematics [10,14,15], Biogeography [4,12], the study of host-parasite cospeciation [13], and Computer Science [8].

Ideally, one would like to generalize tools and concepts that have been demonstrated to be useful for single-labeled phylogenetic trees to MUL-trees. Unfortunately, certain basic problems become NP-hard when extended to MUL-trees. For example, given a multiset \mathcal{S} of *splits* (bipartitions of a fixed multiset L of leaf labels), it is NP-hard to determine whether there exists an unrooted MUL-tree leaf labeled by L such that the multiset of all its splits is equal to \mathcal{S} ,

* Funded by the Special Coordination Funds for Promoting Science and Technology and KAKENHI grant number 23700011.

whereas the corresponding problem for single-labeled trees is solvable in polynomial time [7]. As another example, given a set R of *rooted triplets* (single-labeled phylogenetic trees with exactly three leaves each), a classical algorithm by Aho *et al.* [1] can check if there exists a single-labeled phylogenetic tree that is consistent with all of the rooted triplets in R in polynomial time; on the other hand, it is NP-hard to decide if there exists a MUL-tree consistent with R having at most d leaf duplications, even if restricted to $d = 1$ [6]. In short, MUL-trees pose new and sometimes unexpected algorithmic challenges.

A *consensus tree* is a phylogenetic tree that summarizes the branching of a given set of (conflicting) phylogenetic trees. Different types of consensus trees for single-labeled trees, along with fast algorithms for constructing them, have been developed since the 1970's and are widely used today; see, e.g., [3][7]. The problem of constructing a *consensus MUL-tree* was introduced in [9][10], where an exponential-time algorithm was provided for a specific, greedy type of consensus MUL-tree.

1.1 Definitions

A rooted *multi-labeled phylogenetic tree*, or *MUL-tree* for short, is a rooted, unordered leaf-labeled tree in which every internal node has at least two children. Importantly, in a MUL-tree, the same label may be used for more than one leaf. Fig. 2 and 3 show some examples. The multiset of all leaf labels that occur in a MUL-tree T is denoted by $\Lambda(T)$. For any multiset X and $x \in X$, the *multiplicity* of x in X is the number of occurrences of x in X and is denoted by $\text{mult}_X(x)$. Below, the multiset union operation is expressed by the symbol \uplus .

Let L be a multiset and let T be a MUL-tree with $\Lambda(T) = L$. If $\text{mult}_L(\ell) = 1$ for all $\ell \in L$ then T is a *single-labeled phylogenetic tree*. Next, any submultiset C of L is called a *cluster* of L , and if $|C| = 1$ then C is called *trivial*. Let $V(T)$ be the set of all nodes in T . For any $u \in V(T)$, the subtree of T rooted at u is written as $T[u]$, and $\Lambda(T[u])$ is referred to as the *cluster associated with u* . The *cluster collection* of T is the multiset $\mathcal{C}(T) = \biguplus_{u \in V(T)} \{\Lambda(T[u])\}$. When a cluster C belongs to $\mathcal{C}(T)$, we say that T *contains* C or that C *occurs in* T . Thus, when a cluster C does not occur in a MUL-tree T , we have $\text{mult}_{\mathcal{C}(T)}(C) = 0$.

Let $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ be a given set of MUL-trees satisfying $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$. Two popular types of consensus trees for single-labeled trees are the *strict consensus tree* [16] and the *majority rule consensus tree* [11]. We extend their definitions as follows.

- A *strict consensus MUL-tree of \mathcal{T}* is a MUL-tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T) = \bigcap_{i=1}^k \mathcal{C}(T_i)$, where \bigcap is the intersection of multisets. Formally, for every $C \in \mathcal{C}(T)$, $\text{mult}_{\mathcal{C}(T)}(C) = \min_{1 \leq i \leq k} \text{mult}_{\mathcal{C}(T_i)}(C)$.
- A cluster that occurs in more than $k/2$ of the MUL-trees in \mathcal{T} is a *majority cluster*. A *majority rule consensus MUL-tree of \mathcal{T}* is a MUL-tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of all majority clusters, and for any $C \in \mathcal{C}(T)$, $\text{mult}_{\mathcal{C}(T)}(C)$ equals the largest integer j such that the following condition holds: $|\{T_i : \text{mult}_{\mathcal{C}(T_i)}(C) \geq j\}| > k/2$.

Next, we introduce a new kind of consensus tree. For any MUL-tree T , a cluster C in $\mathcal{C}(T)$ is called *singular* if $C \sqcup C \not\subseteq \Lambda(T)$. Note that if $C \in \mathcal{C}(T)$ is singular then $mult_{\mathcal{C}(T)}(C) = 1$ (but not the other way around).

- A *singular majority rule consensus MUL-tree* of \mathcal{T} is a MUL-tree T such that $\Lambda(T) = L$ and $\mathcal{C}(T)$ consists of: (1) all trivial clusters in T_1, T_2, \dots, T_k ; and (2) all singular clusters that occur in more than $k/2$ of the MUL-trees in \mathcal{T} .

1.2 Our Results and Organization of the Paper

From here on, let \mathcal{T} be a given set of MUL-trees and L a fixed multiset of leaf labels with $\Lambda(T_i) = L$ for every $T_i \in \mathcal{T}$. Define $k = |\mathcal{T}|$ and $n = |L|$. Also, let q equal the number of distinct elements in L . In other words, $q \leq n$. We define $m = \max_{\ell \in L} mult_L(\ell)$ and call m the *multiplicity* of L .

The paper is organized as follows. Section 2 highlights some key properties of consensus MUL-trees. Next, we explain how to construct a strict consensus MUL-tree in $O(nqk)$ time in Section 3. Section 4 shows that constructing a majority rule consensus MUL-tree is NP-hard, even if restricted to instances where $k = 3$ and $m = 3$. However, the singular majority rule consensus MUL-tree admits an efficient algorithm running in $O(n^3k)$ time, described in Section 5. To our knowledge, these are the first ever polynomial-time algorithms for building a consensus MUL-tree.

Our new results for strict and majority rule consensus MUL-trees, along with previously known results for single-labeled phylogenetic trees (corresponding to the case $m = 1$), are summarized in Fig. 1. Our results also hold for the analogous *unrooted* MUL-tree versions of the problems, with the same computational complexities. Due to space constraints, most proofs have been omitted from this conference version of the paper.

Strict consensus		Majority rule consensus	
	$k \geq 2$		$k \geq 3$
$m = 1$	Always exists; always unique; $O(nk)$ time. (Day 2)	$m = 1$	Always exists; always unique; $O(n)$ time. (Day 2)
$m \geq 2$	Always exists; may not be unique; $O(nqk)$ time. Sections 2 and 3	$m = 2$	Always exists; may not be unique; $O(nq)$ time. Sections 2 and 3
		$m \geq 3$	Always exists; may not be unique; $O(nq)$ time. Sections 2 and 3

Fig. 1. The complexity of building strict and majority rule consensus MUL-trees. For $k = 2$, a strict consensus and a majority rule consensus MUL-tree are equivalent.

2 Preliminaries

It is possible for two non-isomorphic MUL-trees to have identical cluster collections. See T_1 and T_2 in Fig. 2 for an example. This property was first observed by Ganapathy *et al.* [4] for unrooted MUL-trees, and their example was simplified by Holm *et al.* [7]. (The example given here is the same as Fig. 1 (b)–(c) in [7], adapted to rooted MUL-trees.)

Define the *delete* operation on any non-root, internal node u in a MUL-tree T as letting all children of u become children of the parent of u , and then removing u and the edge between u and its parent. Note that any delete operation on a node u in T effectively removes one occurrence of a cluster from $\mathcal{C}(T)$, namely $\Lambda(T[u])$, without affecting the other clusters.

Lemma 1. *Let $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ be a set of MUL-trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$. A strict consensus MUL-tree of \mathcal{T} always exists but might not be unique.*

Proof. To prove the existence, let $Z = \bigcap_{i=1}^k \mathcal{C}(T_i)$ (using the intersection of multisets), and construct a MUL-tree T with $\Lambda(T) = L$ and $\mathcal{C}(T) = Z$ as follows. Set T equal to T_1 . Since $Z \subseteq \mathcal{C}(T)$, we have $\text{mult}_Z(C) \leq \text{mult}_{\mathcal{C}(T)}(C)$ for every $C \in \mathcal{C}(T)$. For each $C \in \mathcal{C}(T)$, arbitrarily select $(\text{mult}_{\mathcal{C}(T)}(C) - \text{mult}_Z(C))$ nodes u in T with $\Lambda(T[u]) = C$ and delete them. This yields a MUL-tree T with $\text{mult}_Z(C) = \text{mult}_{\mathcal{C}(T)}(C)$ for every $C \subseteq L$ and $\Lambda(T) = L$, so T is a strict consensus MUL-tree of \mathcal{T} .

To prove the non-uniqueness, consider $\mathcal{T} = \{T_1, T_2\}$ in Fig. 2. Each of T_1 and T_2 is a strict consensus MUL-tree of the set $\mathcal{T} = \{T_1, T_2\}$. \square

Next, we consider majority rule consensus MUL-trees. For $k = 2$, a majority rule consensus MUL-tree of \mathcal{T} is equivalent to a strict consensus MUL-tree of \mathcal{T} . If $k \geq 3$, the non-uniqueness and non-existence follow from the examples in Fig. 2 and 3. Hence:

Lemma 2. *Let $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ be a set of MUL-trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$. If $k = 2$, a majority rule consensus MUL-tree of \mathcal{T} always exists but might not be unique. If $k \geq 3$, a majority rule consensus MUL-tree might not exist and might not be unique.*

Finally, we consider singular majority rule consensus MUL-trees. Let S be the set of all singular, non-trivial clusters that occur in at least $k/2$ of the MUL-trees in \mathcal{T} . For any cluster $C \in S$ and any singular majority rule consensus MUL-tree T of \mathcal{T} , we have $\text{mult}_{\mathcal{C}(T)}(C) = 1$. Thus, for every $C \in S$, there is a unique node t_C in T such that $C = \Lambda(T[t_C])$. For any two clusters $C, C' \in S$, we say that C is an ancestor (the parent) cluster of C' in T if the node t_C is an ancestor (the parent) of the node $t_{C'}$.

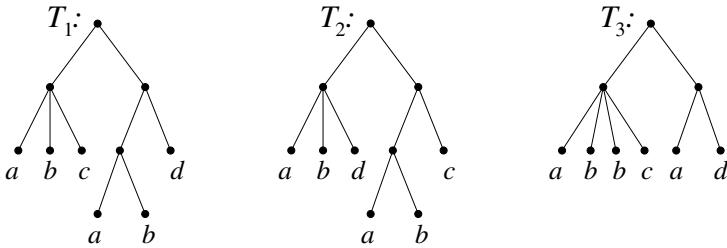


Fig. 2. Let T_1, T_2, T_3 be the three MUL-trees shown above with $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, a, b, b, c, d\} = L$. Then $T_1 \neq T_2$ although $\mathcal{C}(T_1) = \mathcal{C}(T_2) = \{\{a\}, \{a\}, \{b\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, b, c\}, \{a, b, d\}, L\}$. Each of T_1 and T_2 is a strict consensus MUL-tree of $\{T_1, T_2\}$, and also a majority rule consensus MUL-tree of $\{T_1, T_2, T_3\}$.

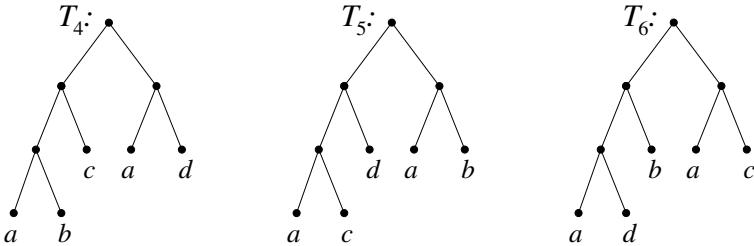


Fig. 3. Here, $\mathcal{T} = \{T_4, T_5, T_6\}$, $\Lambda(T_4) = \Lambda(T_5) = \Lambda(T_6) = \{a, a, b, c, d\} = L$. The non-trivial majority clusters are $\{\{a, b\}, \{a, c\}, \{a, d\}, \{a, a, b, c, d\}\}$. For any MUL-tree T that contains all these clusters, $\text{mult}_{\Lambda(T)}(a) \geq 3$ while $\text{mult}_L(a) = 2$, i.e., $\Lambda(T) \neq L$. Thus, a majority rule consensus MUL-tree of \mathcal{T} does not exist. Also, all the non-trivial majority clusters above are singular, so no singular majority rule consensus MUL-tree exists.

Lemma 3. Let $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ be a set of MUL-trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$. If $k \geq 3$ then a singular majority rule consensus MUL-tree of \mathcal{T} might not exist, but if it does, it is unique.

Proof. The non-existence follows from the example in Fig. 3. ■

Next, we prove the uniqueness. For the sake of obtaining a contradiction, suppose there exist two different singular majority rule consensus MUL-trees A, B of \mathcal{T} . Since $A \neq B$, there are two clusters $C, C' \in S$ such that C' is the parent cluster of C in A while C' is not the parent cluster of C in B . It follows from the definition of a singular cluster that C' must be an ancestor cluster of C in B . Thus, there exists another cluster C'' such that C' is an ancestor cluster of C'' and C'' is the parent cluster of C in B . This means that $C \subsetneq C'' \subsetneq C'$, so C'' cannot be an ancestor cluster of C' in A . Hence, C'' is not an ancestor cluster of C in A , and so A must contain at least two copies of all elements in C . But then $C \uplus C \subseteq L$, contradicting the definition of a singular cluster. □

Observe that the results in Lemmas 1, 2, and 3 hold even when restricted to instances with $m = 2$, i.e., when $\text{mult}_L(x) \leq 2$ for all $x \in L$.

3 Building a Strict Consensus MUL-tree

This section describes a simple algorithm for constructing a strict consensus MUL-tree. Our algorithm, named **Strict_consensus**, is essentially an implementation of the existence proof in Lemma 1. The basic strategy is to remove clusters from the cluster collection $\mathcal{C}(T_1)$ by delete operations on suitable internal nodes from T_1 until a strict consensus MUL-tree is obtained. To identify which clusters to remove, the algorithm uses vectors of integers to represent clusters in \mathcal{T} , as explained next. A *leaf label numbering function* is a bijection from the set of distinct leaf labels in L to the set $\{1, 2, \dots, q\}$. We fix an arbitrary leaf label numbering function f . For every $T_i \in \mathcal{T}$ and node $u \in V(T_i)$, define a vector D_i^u of length q such that, for every $j \in \{1, 2, \dots, q\}$, the j th element equals $\text{mult}_{A(T_i[u])}(f^{-1}(j))$. In other words, each element of the vector D_i^u counts how many times the corresponding leaf label occurs in the subtree rooted at node u in T_i . Clearly, D_i^ℓ contains exactly one 1 and $q - 1$ 0's for any leaf ℓ of T_i , and D_i^u for any internal node u equals the sum of its children's D_i -vectors.

For each MUL-tree T_i in \mathcal{T} , **Strict_consensus** first computes all D_i^u -vectors by one bottom-up traversal of T_i and initializes a trie A_i augmented with leaf counters to store the cluster collection $\mathcal{C}(T_i)$. More precisely, the q elements of each D_i^u -vector are concatenated into a string of length q which is inserted into A_i . Next, for each cluster in T_1 (i.e., for each leaf in the trie A_1), the algorithm calculates how many of its occurrences to remove from T_1 to obtain a strict consensus MUL-tree by subtracting its minimum number of occurrences among T_2, \dots, T_k from the number of occurrences in T_1 ; the tries A_1, \dots, A_k are used to retrieve these numbers efficiently. Finally, the necessary delete operations are performed on T_1 .

Theorem 1. Let $\mathcal{T} = \{T_1, \dots, T_k\}$ be a set of MUL-trees with $A(T_1) = \dots = A(T_k)$. Algorithm **Strict_consensus** constructs a strict consensus MUL-tree of \mathcal{T} in $O(nqk)$ time.

4 Building a Majority Rule Consensus MUL-tree

This section demonstrates that constructing a majority rule consensus MUL-tree is computationally hard. Define the following decision problem:

Majority rule consensus MUL-tree (MCMT):

Input: A set $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ of MUL-trees and a multiset L of leaf labels such that $A(T_i) = L$ for every $T_i \in \mathcal{T}$.

Question: Is there a majority rule consensus MUL-tree of \mathcal{T} ?

To prove the result, we will reduce the 1-IN-3 SAT problem to MCMT. 1-IN-3 SAT is known to be NP-hard [5] and is defined as:

1-in-3 Satisfiability (1-IN-3 SAT):

Input: A Boolean formula F in conjunctive normal form where every clause contains at most 3 literals (3-CNF).

Question: Does there exist a truth assignment to F such that each clause contains exactly one true literal?

First, define *non-mono-replace* on any Boolean formula F in 3-CNF as:

- For every clause C_u in F consisting of three positive literals, arbitrarily select one of its literals x_k and replace $C_u = (x_i \vee x_j \vee x_k)$ by two clauses $(x_i \vee x_j \vee \bar{y}_u) \wedge (y_u \vee x_k)$, where y_u is a newly added Boolean variable. Similarly, for every clause C_u in F consisting of three negative literals, arbitrarily select one of its literals \bar{x}_k and replace $C_u = (\bar{x}_i \vee \bar{x}_j \vee \bar{x}_k)$ by $(\bar{x}_i \vee \bar{x}_j \vee y_u) \wedge (\bar{y}_u \vee \bar{x}_k)$, where y_u is a newly added Boolean variable.

Below, we use the non-mono-replace operation to ensure that the Boolean formula we reduce from has a special structure. The relationship between F and the result of applying non-mono-replace on F is given by:

Lemma 4. *Let F be a Boolean formula in 3-CNF and let F' be the 3-CNF Boolean formula obtained by applying the non-mono-replace operation on F . There exists a truth assignment for F such that every clause contains exactly one true literal if and only if there exists a truth assignment for F' such that every clause contains exactly one true literal.*

We now describe the reduction. Let F be any given Boolean formula in 3-CNF. As in the proof of Theorem 3.1 in [7], assume w.l.o.g. that: (i) No single clause in F contains a variable x_i as well as its negation \bar{x}_i as literals; and (ii) for every variable x_i in F , both x_i and its negation \bar{x}_i appear somewhere in F as literals. Then, apply non-mono-replace on F to obtain a Boolean formula F' with s variables and t clauses, for some positive integers s, t (this does not affect properties (i) and (ii) above). Lastly, construct three MUL-trees T_1, T_2, T_3 based on F' as follows. Let $X = \{x_1, \dots, x_s\}$ and $Z = \{z_1, \dots, z_t\}$ be two sets in one-to-one correspondence with the variables and clauses of F' . Say that x_i is positive (negative) in z_j if x_i corresponds to a variable in F' that occurs positively (negatively) in j th clause. Define the leaf label multiset L for T_1, T_2, T_3 as $L = \{x, x : x \in X\} \cup \{z, z, z : z \in Z\}$. (In other words, L contains two copies of every element in X and three copies of every element in Z .) Next, for each $x \in X$, define two subsets Z_x, \tilde{Z}_x of Z by $Z_x = \{z \in Z : x \text{ is positive in } z\}$ and $\tilde{Z}_x = \{z \in Z : x \text{ is negative in } z\}$. Let $\mathcal{W} = \{Z_x \cup \{x\} : x \in X\}$ and $\tilde{\mathcal{W}} = \{\tilde{Z}_x \cup \{x\} : x \in X\}$. From $\mathcal{W}, \tilde{\mathcal{W}}$, construct three MUL-trees T_1, T_2, T_3 with $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = L$ whose sets of non-trivial clusters are: $\mathcal{W} \cup \tilde{\mathcal{W}}$, $\mathcal{W} \cup \{X \cup Z\}$, and $\tilde{\mathcal{W}} \cup \{X \cup Z\}$, respectively. Then, the set of non-trivial majority clusters for $\{T_1, T_2, T_3\}$ is: $\mathcal{W} \cup \tilde{\mathcal{W}} \cup \{X \cup Z\}$. It is straightforward to show that T_1, T_2, T_3 are valid MUL-trees. Because of the non-mono-replace operation, for every $z_j \in Z$, there is exactly one or two subtrees attached to the root of T_2 (T_3) that contains an occurrence of z_j . The reduction's correctness follows from:

Lemma 5. A majority rule consensus MUL-tree for T_1, T_2, T_3 exists if and only if there exists a truth assignment for F' such that every clause contains exactly one true literal.

Theorem 2. The MCMT problem is NP-hard, even if restricted to inputs where $k = 3$ and each leaf label occurs at most 3 times.

5 Building a Singular Majority Rule Consensus MUL-tree

Here, we present a polynomial-time algorithm for building a singular majority rule consensus MUL-tree. By Lemma 3 in Section 2, when a singular majority rule consensus MUL-tree of \mathcal{T} exists, it is unique.

Our algorithm consists of two phases. The first phase constructs the set S of all singular, non-trivial clusters that occur in at least $k/2$ of the MUL-trees in \mathcal{T} . To implement Phase 1, we enumerate all non-trivial clusters that occur in \mathcal{T} and count their occurrences using the technique described in Section 3. The second phase builds the singular majority rule consensus tree of \mathcal{T} by calling a top-down, recursive procedure `Build_MUL-tree(L, S)`, listed in Fig. 4. The cluster associated with the root of T is L , and the clusters associated with the children of the root of T belong to a set $\mathcal{F} \subseteq S$ of maximal elements in S . More precisely, we let $\mathcal{F} = \{C \in S : C \text{ is not a submultiset of any cluster } C' \in S\}$. Then:

```

Algorithm Build_MUL-tree
Input: A multiset  $L$ , and a set  $S$  of singular, non-trivial clusters of  $L$ .
Output: A MUL-tree leaf-labeled by  $L$  that contains all clusters in  $S$ , if one exists;
          otherwise, FAIL.

1 Let  $\mathcal{F}$  be the empty set.
2 for every  $X \in S$  do
3.1   if  $X$  is not a submultiset of any cluster currently in  $\mathcal{F}$  then
        Delete every cluster from  $\mathcal{F}$  that is a submultiset of  $X$ . Insert  $X$  into  $\mathcal{F}$ .
        If  $L \subsetneq \bigcup_{C \in \mathcal{F}} C$  then return FAIL.
      endif
    endfor
3 for every  $C \in \mathcal{F}$  do
  Compute  $S|C = \{X \in S : X \subseteq C\}$ .
  If  $S|C \neq \emptyset$ , let  $T_C = \text{Build\_MUL-tree}(C, S|C)$ ; otherwise, let  $T_C = \text{null}$ .
  endfor
4 Let  $T$  be a MUL-tree whose root is attached to: (1) the root of  $T_C$  for each
    $C \in \mathcal{F}$  with  $T_C \neq \text{null}$ ; and (2) all leaves labeled by  $L \setminus (\bigcup_{C \in \mathcal{F}} C)$ .
5 return  $T$ 
End Build_MUL-tree

```

Fig. 4. Algorithm `Build_MUL-tree`

Lemma 6. $\mathcal{F} = \{C \in S : C \text{ is not a submultiset of any cluster } C' \in S\}$ equals the set of all clusters associated with children of the root of the unique singular majority consensus MUL-tree of \mathcal{T} .

Steps 1 and 2 of `Build_MUL-tree` compute \mathcal{F} in a greedy fashion. After each update to \mathcal{F} in Step 2, if L is a proper submultiset of $\biguplus_{C \in \mathcal{F}} C$ then no MUL-tree leaf-labeled by L containing all clusters in S exists, and the algorithm reports FAIL. Step 3 builds a sub-MUL-tree T_C for each C in \mathcal{F} by recursively calling `Build_MUL-tree`($C, S|C$), where $S|C = \{X \in S : X \subseteq C\}$; the base case of the recursion is when $S|C = \emptyset$. Then, in Step 4, the T_C -trees and all “leftover leaves” not in $\biguplus_{C \in \mathcal{F}} C$ are assembled into the final consensus MUL-tree T , which is returned in Step 5.

`Build_MUL-tree` constructs a MUL-tree with $O(|L|)$ internal nodes. For each such node, it may need to execute all the steps of the procedure, which takes $O(|L||S|)$ time because $|\biguplus_{C \in \mathcal{F}} C| \leq |L|$. The total running time of Phase 2 is $O(|L|^2|S|) = O(n^3k)$ since $|L| = n$ and $|S| = O(nk)$.

Theorem 3. Let $\mathcal{T} = \{T_1, \dots, T_k\}$ be a set of MUL-trees with $\Lambda(T_1) = \dots = \Lambda(T_k)$. Our algorithm constructs the singular majority consensus MUL-tree of \mathcal{T} (if it exists) in $O(n^3k)$ time.

References

1. Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D.: Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing* 10, 405–421 (1981)
2. Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification* 2(1), 7–28 (1985)
3. Felsenstein, J.: *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland (2004)
4. Ganapathy, G., Goodson, B., Jansen, R., Le, H.-S., Ramachandran, V., Warnow, T.: Pattern identification in biogeography. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3(4), 334–346 (2006)
5. Garey, M., Johnson, D.: *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York (1979)
6. Guillermot, S., Jansson, J., Sung, W.-K.: Computing a smallest multilabeled phylogenetic tree from rooted triplets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8(4), 1141–1147 (2011)
7. Huber, K.T., Lott, M., Moulton, V., Spillner, A.: The complexity of deriving multilabeled trees from bipartitions. *J. of Comp. Biology* 15(6), 639–651 (2008)
8. Huber, K.T., Spillner, A., Suchen, R., Moulton, V.: Metrics on multilabeled trees: Interrelationships and diameter bounds. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 8(4), 1029–1040 (2011)
9. Lott, M., Spillner, A., Huber, K.T., Moulton, V.: PADRE: a package for analyzing and displaying reticulate evolution. *Bioinformatics* 25(9), 1199–1200 (2009)
10. Lott, M., Spillner, A., Huber, K.T., Petri, A., Oxelman, B., Moulton, V.: Inferring polyploid phylogenies from multiply-labeled gene trees. *BMC Evolutionary Biology* 9, 216 (2009)

11. Margush, T., McMorris, F.R.: Consensus n -Trees. *Bulletin of Mathematical Biology* 43(2), 239–244 (1981)
12. Nelson, G., Platnick, N.: Systematics and Biogeography: Cladistics and Vicariance. Columbia University Press (1981)
13. Page, R.D.M.: Parasites, phylogeny and cospeciation. *International Journal for Parasitology* 23, 499–506 (1993)
14. Page, R.D.M.: Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas. *Systematic Biology* 43(1), 58–77 (1994)
15. Scornavacca, C., Berry, V., Ranwez, V.: Building species trees from larger parts of phylogenomic databases. *Information and Computation* 209(3), 590–605 (2011)
16. Sokal, R.R., Rohlf, F.J.: Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology* 30(3), 309–325 (1981)
17. Sung, W.-K.: Algorithms in Bioinformatics: A Practical Introduction. Chapman & Hall/CRC (2010)
18. Wareham, H.T.: An efficient algorithm for computing Mi consensus trees. B.Sc. Honours thesis, Memorial University of Newfoundland, Canada (1985)

Adaptive Phenotype Testing for AND/OR Items

Francis Y.L. Chin, Henry C.M. Leung, and S.M. Yiu

Department of Computer Science,
The University of Hong Kong,
Pokfulam, Hong Kong
`{chin, cmleung2, smyiu}@cs.hku.hk`

Abstract. The combinatorial group testing problem is concerned with the design of experiments so as to minimize the number of tests needed to find the sets of items responsible for a particular phenotype (an observable property). The traditional group testing problem only considers the OR problem, i.e. the phenotype appears as long as one of the responsible items exists. In this paper, we introduce the phenotype testing problem which is a generalization of the well-studied combinatorial group testing problem. In practice, there are more than one phenotype and the responsible items and their mechanism (AND or OR) for each phenotype are unknown where an AND mechanism means that the phenotype only appears if all of the responsible items exist.

This phenotype testing problem has an important application in biological research, known as phenotype knockout study. New algorithms for designing adaptive experiments for solving the phenotype testing problem with n items using $O(\log n)$ tests in the worst cases are introduced. When the number of phenotypes is small, say at most 2, and the number of responsible items for each phenotype is at most 2, algorithms with near-optimal number of tests are presented.

Keywords: adaptive phenotype testing, combinatorial group testing, knockout study.

1 Introduction

Phenotype knockout study [10,13,14] is a new and critical application in the study of biology. Phenotype refers to an observed physical characteristic of an organism (e.g. blue eyes, black hair, tumor). Many important phenotypes are induced by genes. Given a set of phenotypes and a set of genes, the problem is to determine which subset of genes induces which phenotype and the mechanism (AND or OR). Knowing which subset of genes and its mechanism for the phenotypes is useful in drug design such as tumor therapy. There are two simple mechanisms for a subset of genes to induce a phenotype, the *OR-mechanism* and the *AND-mechanism*. In the OR (AND)-mechanism, a subset of genes can induce a phenotype as long as one (all) of these genes is (are) active, in other words, the phenotype disappears if and only if all (any one of) these genes are (is)

inactive. Besides AND and OR mechanisms, there are more complicated mechanisms between the responsible genes and phenotypes, which are not discussed in this paper [3].

To discover which subset of genes and its mechanism for each phenotype, biologists can knockout a gene [7,8] and observe if some particular phenotype still exists. However, such a test is expensive and more importantly, it is very time consuming. Since the number of genes responsible for a phenotype is usually small, in practice, instead of checking one gene against each phenotype, we can check a subset of genes and several phenotypes together in one test. For example, for the OR-mechanism, if some phenotype disappears, we can conclude that none of the genes in the subset is responsible for their phenotype. Given n items and k phenotypes, the *phenotype testing problem* is to design how to group the items (genes) into subsets such that, based on the test results on these subsets, one can identify the *responsible items* (genes) for each phenotype and find out the OR- or AND-mechanism of these items (genes) to induce the phenotype. From now on, we shall use “gene” and “item” interchangeably as long as no confusion arises. The items that are responsible for the phenotype with the OR-(AND-) mechanism, called OR-(AND-) phenotype, are referred to as *OR-items (AND-items)*. The objective is to have a design with as few subsets (or tests) as possible.

It turns out that this problem is related to the well-studied combinatorial group testing problem in computer science. In the *combinatorial group testing (CGT) problem* [12,5,9,11], we are given a set of items in which some of them are contaminated (or defective). We assume that a test can determine if a subset of items contains any contaminated ones. If the result is negative, all the items in the subset are not contaminated. An important objective of this problem is to design the grouping of the items into subsets in order to minimize the number of tests. The phenotype testing problem is a generalization of the CGT problem by considering more than one phenotype, each phenotype with different mechanisms and responsible items. One can easily see that the CGT problem is equivalent to the phenotype testing problem with only one OR-phenotype.

The CGT problem has been studied under two different scenarios, adaptive and non-adaptive. In the *adaptive (or sequential) scenario*, group tests are divided into stages, conducted one by one after the test results in previous stages are known. In the *non-adaptive scenario*, there is only one stage and all the group tests are performed together at the same time. It is assumed that after one simple stage of group tests, we should be able to discover all the responsible items in the non-adaptive scenario. Usually fewer tests are required under the adaptive scenario at the expense of more test stages. Non-adaptive tests will be conducted if time is more critical. In the phenotype testing problem, we can also consider these two scenarios. In fact, the non-adaptive phenotype testing problem with OR-mechanism has been studied in another application called *DNA library screening* [4,6,12] in bioinformatics, which is referred as the *pooling design problem*. In this paper, we will focus on the adaptive phenotype testing (APT) problem which minimizes the number of tests.

Table 1. Summary of our results

Num. of phenotypes (k)	Items per phenotype (d)	Num. of tests required
General solution		
$k = 1$	$d \geq 1$	$d\lceil\log_2 n\rceil + (d + 2)$
$k \geq 2$	$d \geq 1$	$4d\lceil\sqrt{k/2}\rceil \cdot \lceil\log_2 n\rceil$
Special solution		
$k = 1$	$d = 2$	$2\lceil\log_2 n\rceil$
$k \geq 2$	$d = 1$	$\lceil\log_2 n\rceil$
$k = 2$	$d = 2$	$2\lceil\log_2 n\rceil + 2\lceil\sqrt{\log_2 n - 1}\rceil$

1.1 Our Contributions

For the adaptive case, existing solutions assume that there is only one phenotype. However, one test can produce test results for several phenotypes simultaneously and we shall show how to make use of this property to solve the phenotype testing problem. The difficulty of the phenotype testing problem arises from the fact that the subsets of items for testing might be different for each phenotype. For example, assume S_1 and S_2 are two disjoint sets of items, phenotypes P_1 is positive for S_1 and phenotypes P_2 is positive for S_2 respectively. For detecting the responsible items for P_1 , splitting S_1 would be recommended, whereas splitting S_2 would be recommended for detecting the responsible items for P_2 . Also, the mechanism of each phenotype is usually unknown. So the design of the tests for phenotype testing might be different from that for CGT when there are more than one phenotype. In this paper, we first provide several algorithms to solve the APT problem for general k and d , maximum size of S_1 and S_2 . Then, for some special cases, algorithms with near-optimal number of tests are presented. The number of tests required by our algorithms is given in the Table II.

2 Preliminaries

Given a set of n items (represented by a set of integers $U = \{1, \dots, n\}$) and a phenotype P_r with a set $U_r \subseteq U$ of responsible items, we define P_r as a function $2^U \rightarrow \{0, 1\}$ such that for any subset S of U , $P_r(S) = 1$ if and only if $U_r \subseteq S$, where U_r is a set of AND-items for P_r (AND-mechanism) or $U_r \cap S \neq \emptyset$ where U_r is a set of OR-items for P_r (OR-mechanism). In practice, the size of U_r is small and bounded by a constant d , i.e. $|U_r| \leq d$.

Given a set of n integers $U = \{1, \dots, n\}$ and a set of phenotype P_r , $r = 1, \dots, k$, each with a hidden set U_r of responsible items where $|U_r| \neq d$, the Adaptive Phenotype Testing (APT) problem is to design an algorithm for constructing the minimum number of subsets S_1, \dots, S_q of U for testing sequentially such that we can deduce U_r and its mechanism from $P_r(S_1), \dots, P_r(S_q)$, for all $r = 1, \dots, k$, where the k test results of $\{P_r(S_i) | r = 1, \dots, k\}$ can be provided in a single test on subset S_i . Note that the construction of S_i might depend on the test results on S_1, \dots, S_{i-1} . Note that the APT problem with only

AND-phenotypes is equivalent to the APT problem with only OR-phenotypes by considering testing the complement set of items.

3 APT Algorithms for General k and d

In this section, we provide several algorithms to solve the APT problem for general $k \geq 1$ and $d \geq 1$. For only one phenotype with any number of responsible items ($k = 1$ and $d \geq 1$), we give an algorithm using at most $d\lceil\log_2 n\rceil + (d+2)$ tests. Then, for more than one phenotype, with at most d responsible items, we provide an algorithm to solve the APT problem with $4d\lceil\sqrt{k/2}\rceil \cdot \lceil\log_2 n\rceil$ tests.

3.1 One Phenotype ($k = 1$, $d \geq 1$)

Huang's Generalized Binary Splitting Algorithm [9] can solve the APT problem with one OR-phenotype ($k = 1$) with at most d OR-items in $t(n, d) = \log_2 \binom{n}{d} + d \leq d \log_2 n + d$ tests. We shall solve the APT problem with unknown mechanism by first using two tests to determine the mechanism of the phenotype and at the same time reduce the size of the problem.

Theorem 1. *The APT problem with $k = 1$ unknown phenotype and at most d ($d \geq 1$) responsible items can be solved in $\max_{s=0, \dots, \lceil\log_2 n\rceil} \{t(n/2^s, d) + 2s\}$ test.*

Proof. Partition U into two equal size subsets S_1 and S_2 . Apply a test on S_1 and S_2 respectively. There are 4 outcomes.

- $P(S_1) = 1$ and $P(S_2) = 1$: It is an OR-phenotype and the APT problem of size n with OR-mechanism can be solved using $t(n, d)$ tests.
- $P(S_1) = 0$ and $P(S_2) = 0$: It is an AND-phenotype and the APT problem of size n with AND-mechanism can be solved using $t(n, d)$ tests (by converting AND-items to OR-items).
- $P(S_1) = 1$ and $P(S_2) = 0$, or $P(S_1) = 0$ and $P(S_2) = 1$: Reduces to the original problem of half of the size, which can be solved recursively.

The number of required tests would be the maximum of these cases as given in the statement of the theorem. \square

Corollary 1. *The APT problem with $k = 1$ and $d \geq 1$ can be solved with $d\lceil\log_2 n\rceil + (d+2)$ tests.*

3.2 Multiple Phenotype ($k \geq 2$ and $d \geq 1$)

When there are more than one phenotype, the algorithm mentioned in Section 3.1 can be repeated k times and the APT problem can be solved with $kd\lceil\log_2 n\rceil + k(d+2)$ tests. However, test on a phenotype may also provide information of another phenotype as long as both phenotypes need to be tested on the same or disjoint subset of items. In order to reduce the number of tests, we should

design the subsets to be tested by different phenotypes such that each test can provide information to determine the responsible items for several phenotypes.

Consider solving the APT problem ($k = 1$ and $d = 2$) for phenotype P_p with at most two responsible items x_1 and y_1 using at most $3\lceil \log_2 n \rceil - 1$ tests as follows. First, we represent the n items by n distinct length- $\log_2 n$ binary numbers, e.g. item $b = b[1] \dots b[\lceil \log_2 n \rceil]$. The idea is to deduce the binary representation of the responsible items. If the tested subset is formed according to binary numbers of the items, e.g. subset containing all items with 1 at their i -th digit, then a positive phenotype test on this subset would indicate that some responsible items have 1 in the i -th digit. Assume we perform 2 tests $P_p(S_i, 0)$ and $P_p(S_i, 1)$ such that $b \in S_{i,0}$ iff $b[i] = 0$ and $b \in S_{i,1}$ iff $[i] = 1$ for every digit i of the length- $\log_2 n$ binary numbers. Note that the same two tests would also give the test results of other phenotypes in $S_{i,0}$ and $S_{i,1}$ at the same time. If $P_p(S_i, 0) = P_p(S_i, 1) = 0$ or 1, we can conclude that some responsible items are in $S_{i,0}$ and some in $S_{i,1}$, i.e. these two sets of responsible items have different i -th digit, and P_p is AND- or OR-phenotype. Otherwise, $P_p(S_i, 0) \neq P_p(S_i, 1)$, the set $S_{i,\alpha}$ with $P_p(S_{i,\alpha}) = 1$ contains all responsible items for P_p , i.e. the i -th digit of all responsible items is α which is either 0 or 1. If there is only one responsible item for P_p , the binary string $b = b[1] \dots b[\lceil \log_2 n \rceil]$ such that $P_p(S_{j,b[j]}) = 1$ represents the responsible item of P_p .

The algorithm starts with $i = 1$, we perform a test on $P_p(S_1, 0)$ and $P_p(S_1, 1)$. If $P_p(S_1, 0) \neq P_p(S_1, 1)$, i.e. the first digit of all responsible items for P_p are the same (equals 0 if $P_p(S_1, 0) = 1$, otherwise equals 1). After having determined the first digit, we can continue the test on $P_p(S_2, 0)$ and $P_p(S_2, 1)$ and so on. Assume the j -th digit is the first digit with $P_p(S_j, 0) = P_p(S_j, 1)$, i.e. the first $j - 1$ digits of all responsible items are the same but different at the j -th digit. There are two possible length- j binary numbers $b[1]b[2] \dots b[j-1]0$ and $b[1]b[2] \dots b[j-1]1$ representing the prefixes of the binary representations of all responsible items of P_p . For each $i > 0$, there are at most d length- i binary numbers $b[1]b[2] \dots b[i]$ representing the prefixes of the binary representations of all responsible items of P_p . In order to determine the $(i+1)$ -th digits of the responsible items of P_p , two tests on items with the prefixes of their binary representations $b[1]b[2] \dots b[i]0$ and $b[1]b[2] \dots b[i]1$ have to be performed. A total of $2d$ tests might be needed for each digit and $2d\lceil \log_2 n \rceil$ in total for a particular phenotype. Thus, $2dk\lceil \log_2 n \rceil$ tests are needed for k phenotypes.

Many of these tests can be shared. For example, when the responsible items of two phenotypes with the same mechanism have the same prefix $b[1]b[2] \dots b[i]$, the test on the $(i+1)$ -th positions of these two items can be performed at the same time. Similarly, when there are two disjoint sets of phenotypes S_α and S_β with the same mechanism each has a responsible item with prefix $b[1] \dots b[i]$ and $b'[1] \dots b'[i]$ respectively, the test on the $(i+1)$ -th positions of these items can be performed at the same time. However, two disjoint set with different prefixes of the same phenotype cannot be tested together as we cannot associate the test results to which set or both sets of items. Thus, we can construct a graph G with each vertex represents a subset of prefixes of some responsible items of different phenotypes, on which a single test can be performed without mixing up their

results. There is an edge between two vertices u and v if and only if there is a phenotype P_p having some responsible items with prefixes in u and v at the same time, i.e. the tests on these two subset of prefixes cannot be performed together. Given two vertices u and v with no edge in between, the test results of the phenotypes having responsible item with prefix in u will not be affected by those items with prefix in v of different phenotypes, and vice versa. The test on the responsible items with prefixes in these two vertices u and v can be performed at the same time. Vertices without edges connecting them can be merged together until a clique is formed. The maximum number of prefixes that cannot be performed at the same time is the same as the size of the largest clique in the graph G .

Lemma 1. *Given x phenotypes each with at most d responsible items with the same mechanism. Let each vertex in G represents the prefixes of some responsible items and there is an edge between two vertices u and v if and only if a phenotype P_p has two items with prefixes in u and v at the same time, the size of the largest clique formed by G is at most d .*

Proof. Given a phenotype P_p with at most d responsible items, there are at most $\binom{d}{2}$ edges corresponding to P_p . Since there are x phenotypes, there are at most $x\binom{d}{2}$ edges in graph G . Since a clique of size c has $\binom{c}{2}$ edges and $\binom{c}{2} \leq x\binom{d}{2}$, which implies $c \leq d\sqrt{x}$. \square

Theorem 2. *The number of tests needed for solving the APT problem with k phenotypes each with at most d responsible items is at most $4d\lceil\sqrt{k/2}\rceil \cdot \lceil\log_2 n\rceil$.*

Proof. Assume there is x phenotypes with OR-mechanism and $k-x$ phenotypes with AND-mechanism. By Lemma 1, at most $2d\sqrt{x} + 2d\sqrt{k-x}$ tests are needed for determining the responsible items. The maximum number of tests happens when $x = k/2$. Thus at most $4d\lceil\sqrt{k/2}\rceil$ tests are needed for each digit and at most $4d\lceil\sqrt{k/2}\rceil \cdot \lceil\log_2 n\rceil$ tests are needed for solving the APT problem. \square

4 APT Algorithms for Special k and d

For some particular k and d , we are able to derive better algorithms to solve the APT problem. For $k = 1$, $d = 2$, we can use $2\log_2 n$ tests to solve the problem with 4 times fewer tests than the general solution given in Section 3.1. For $k = 2$ and $d = 1$, only $\log_2 n$ tests are needed. And for $k = 2$ and $d = 2$, we can have a solution which uses at most $2\lceil\log_2 n\rceil + 2\lceil\sqrt{\log_2 n - 1}\rceil$ tests compared to the solution in Section 3.2 which uses at most $4(\lceil\log_2 n\rceil - 1) + 2$ tests.

4.1 $k = 1$ and $d = 2$

Assume that we only have one phenotype ($k = 1$) with unknown mechanism and at most 2 responsible items, we will show that the APT problem can be solved by $2\log_2 n$ tests which matches the lower bound $\log_2((\binom{n}{1}) + (\binom{n}{2}) + (\binom{n}{2})) = 2\log_2 n$

($\binom{n}{1}$ cases for exactly one responsible items and $\binom{n}{2}$ cases for two AND-items and OR-items) when n is power of 2. In Lemma 2, we show that using a binary search technique, we can locate the responsible items if the mechanism of the phenotype is known and the responsible items are in two disjoint known subsets.

Lemma 2. *Given a phenotype with two responsible items of known mechanism, let $U = \{1, \dots, n\}$ be divided into two disjoint subsets S_1 and S_2 with each subset containing one responsible item. Locating the responsible item in S_1 or S_2 takes $\log_2 |S_1|$ and $\log_2 |S_2|$ tests respectively.*

Proof. Without loss of generality, we show how to locate the responsible item x in S_1 . If the phenotype is an AND-phenotype, we know that the other responsible item y is in S_2 . So, we divide S_1 into two subsets S_{11} and S_{12} of equal size, and test $(S_{11} \cup S_2)$, if the result is positive, then x is in S_{11} , otherwise, x is in S_{12} . Each round, we can remove half of the items of S_1 from consideration. So, $\log_2 |S_1|$ tests are sufficient. If the phenotype is an OR-phenotype, again we divide S_1 into two equal subsets S_{11} and S_{12} , we only need to test S_{11} to see if it contains x . So, $\log_2 |S_1|$ tests are sufficient. Similarly $\log_2 |S_2|$ tests are sufficient for determine the responsible item in S_2 . \square

Based on Lemma 2, we can solve the APT problem with one phenotype of unknown mechanism and at most two responsible items x and y ($x = y$ when there is only one responsible item) using a recursive algorithm as follows. We divide U into two disjoint sets S_1 and S_2 , and test S_1 and S_2 with four outcomes each required $2\lceil \log_2 n \rceil$ tests:

- (a) $P_1(S_1) = 1$ and $P_1(S_2) = 0$: Both responsible items x and y are in S_1 , so we can recursively work on S_1 only.
- (b) $P_1(S_1) = 0$ and $P_1(S_2) = 1$: Similar to Case (a), both responsible items x and y are in S_2 .
- (c) Both $P_1(S_1)$ and $P_1(S_2)$ equal 0: It is an AND-phenotype with exactly two responsible items, w.l.o.g $x \in S_1$ and $y \in S_2$. By Lemma 2, we can locate x in S_1 and y in S_2 using $\log_2 \lceil (n/2) \rceil = \lceil \log_2 n \rceil - 1$ additional tests.
- (d) Both $P_1(S_1)$ and $P_1(S_2)$ equal 1: Similar to Case (c), it is an OR-phenotype with exactly two responsible items. By Lemma 2, we can locate x in S_1 and y in S_2 using $\log_2 \lceil (n/2) \rceil = \lceil \log_2 n \rceil - 1$ additional tests.

4.2 $k \geq 1$ and $d = 1$

Since there is only one responsible item for each phenotype, the mechanisms of the phenotype can be ignored as both OR- and AND-mechanisms are the same. The responsible items can be determined by performing a binary search on all phenotypes at the same time until the some responsible items are found in different subsets, i.e. the test results are different for the two phenotypes. Then a binary search on these these subsets can be performed simultaneously because the test result of a phenotype does not affect the test result of other phenotypes. The total number of tests needed is $\lceil \log_2 n \rceil$.

4.3 $k = 2$ and $d = 2$

The technique described in the Section 4.1 cannot be applied when $d = 2$ and the two phenotypes have different mechanisms. We divide U into two disjoint subsets of equal size S_1 and S_2 and test S_1 and S_2 . There are two possible outcomes for each phenotype. (1) $P(S_1) \neq P(S_2)$: One subset does not contain any responsible item related to the phenotype, or (2) $P(S_1) = P(S_2) = 1$ (or 0), the phenotype is OR-mechanism (AND-mechanism) with exactly two responsible items, i.e. each subset contains one responsible item for the phenotype. There are three cases for the outcomes of the two phenotypes. Case (a): the outcomes for both phenotypes are (1). Case (b): the outcomes for both phenotypes are (1) and (2) respectively. Case (c): the outcomes for both phenotypes are (2). Cases (a) and (b) are relatively easier and the phenotypes can be determined using at most $2\lceil\log_2 n\rceil + 1$ tests for each case by binary searching the responsible items for both phenotypes at the same time. Here, we mainly focus on Case (c).

Let x_1, y_1 be the responsible items for P_1 and x_2, y_2 be the responsible items for P_2 . Note that when P_1 (P_2) has only one responsible item, $x_1 = y_1$ ($x_2 = y_2$). Without loss of generality, let $\{x_1, x_2\} \subseteq S_1$, $\{y_1, y_2\} \subseteq S_2$ and the mechanisms of phenotype P_1 and P_2 are known. If the two phenotypes are of the same mechanism, similar to the case for $k = 2$ and $d = 1$, we can perform a binary search on S_1 for x_1 and x_2 together using $(\lceil\log_2 n\rceil - 1)$ tests and then on S_2 for y_1 and y_2 using another $(\lceil\log_2 n\rceil - 1)$ tests. Thus, $2\lceil\log_2 n\rceil$ tests in total will be needed.

In the situation where the two phenotypes are of different mechanisms, assume P_1 is OR-mechanism and P_2 is AND-mechanism. Initially, $\{x_1, x_2\} \subseteq S_1$ and $\{y_1, y_2\} \subseteq S_2$. Partition S_1 into two subsets of equal size S_{11} and S_{12} and S_2 into S_{21} and S_{22} . Two tests are performed on $S_{11} \cup S_{21}$ and $S_{11} \cup S_{22}$ respectively. Depending on the test outcomes, the sizes of sets containing the responsible items will be reduced. If the test outcome of the OR-mechanism phenotype P_1 on $S_{11} \cup S_{21}$ and $S_{11} \cup S_{22}$ is:

- (0,1): then $x_1 \in S_{12}$ and $y_1 \in S_{22}$
- (1,0): then $x_1 \in S_{12}$ and $y_1 \in S_{21}$
- (1,1): then $x_1 \in S_{11}$ and $y_1 \in S_2$
- (0,0): this case is not possible

If the test outcome of the AND-mechanism phenotype P_2 on $S_{11} \cup S_{21}$ and $S_{11} \cup S_{22}$ is:

- (0,1): then $x_2 \in S_{11}$ and $y_2 \in S_{22}$
- (1,0): then $x_2 \in S_{11}$ and $y_2 \in S_{21}$
- (1,1): this case is not possible
- (0,0): then $x_2 \in S_{12}$ and $y_2 \in S_2$

Cases that halve the sizes of the sets containing the responsible items or produce disjoint sets containing x_1 and x_2 (y_1 and y_2) are not problematic (Lemma B) and the responsible items for the phenotypes can be determined with $2\lceil\log_2 n\rceil$ tests. In fact, the outcome (1,1) for the OR-mechanism phenotype P_1 with outcome (0,1) or (1,0) for the AND-mechanism phenotype P_2 will result in $\{x_1, x_2\} \subseteq S_{11}$, $y_2 \in S' = S_{21}$ or S_{22} and $y_1 \in S_2$ where $S' \in S_2$. While the outcome (0,0) for

the AND-mechanism phenotype P_2 with outcome (0,1) or (1,0) for the OR-mechanism phenotype P_1 will result in $\{x_1, x_2\} \subseteq S_{12}$, $y_1 \in S' = S_{21}$ or S_{22} and $y_2 \in S_2$ where $S' \subseteq S_2$. A sub-problem P that solving the APT problem with three subsets: $\{x_1, x_2\} \subseteq T$, $y_1 \in T_O$ and $y_2 \in T_A$ where $T_A \subseteq T_O$ or $T_O \subseteq T_A$. Assume $T_A \subseteq T_O$, we can partition subset $T(T_A)$ into two equal-size disjoint subsets T_1 and T_2 (T_{A1} and T_{A2}) and perform 2 tests ($T_1 \cup T_{A1}$ and $T_1 \cup T_{A2}$) with the different cases ($P_1(T_1 \cup T_{A1})$, $P_2(T_1 \cup T_{A1})$), ($P_1(T_1 \cup T_{A2})$, $P_2(T_1 \cup T_{A2})$):

Case i: (0,0), (1,0): $\{x_1, x_2\} \subseteq T_2$, $y_2 \in T_A$, $y_1 \in T_{A2}$ to be solved by recursion.
Case ii: (1,0), (1,1): $\{x_1, x_2\} \subseteq T_1$, $y_2 \in T_{A2}$, $y_1 \in T_O$ (1,1), (1,0): $\{x_1, x_2\} \subseteq T_1$, $y_2 \in T_{A1}$, $y_1 \in T_O$ We can recursively divided T_1 and T_{A1} to determine x_1 , x_2 and y_2 using $2\lceil \log_2 n \rceil$ tests. However, it takes $\lceil \log_2 n \rceil - 1$ extra test to determine y_1 , so a total of $3\lceil \log_2 n \rceil - 1$ might be needed. In order to determine y_1 more efficiently, we perform a test on $T_O - T_{A1}$ every s steps (halving of T_1). If $P_1(T_O - T_{A1}) = 1$, $y_2 \in T_{A1}$, $y_1 \in T_O - T_{A1}$ the problem can be solved by Lemma 3 using $2\lceil \log_2 n \rceil + \lceil \sqrt{\log_2 n} \rceil + 1$ tests in total. If $P_1(T_O - T_{A1}) = 0$, the problem can be solved by recursion with the size of T_O at most $\lceil n/2^s \rceil$. In the worst case, $\lceil \log_2 n - 1 \rceil/s$ tests on $T_O - T_{A1}$ are needed and an extra s tests are needed for determining y_1 in $T_O - T_{A1}$ of size $2^s - 1$ using binary searching. Thus, $\lceil \log_2 n - 1 \rceil/s + s$ extra tests are needed for determining y_1 with the minimum value when $s = \lceil \sqrt{\log_2 n - 1} \rceil$. The total number of tests required is $2\lceil \log_2 n \rceil + 2\lceil \sqrt{\log_2 n - 1} \rceil$. **Case iii:** Other cases: $\{x_1, x_2\}$, y_1 and y_2 are in disjoint subsets and the problem can be solved easily by Lemma 3 in the Appendix and the phenotypes can be determined with $2\lceil \log_2 n \rceil$ tests. Thus, the APT problem can be solved using at most $2\lceil \log_2 n \rceil + 2\lceil \sqrt{\log_2 n - 1} \rceil$ tests.

Lemma 3. Let U be divided into three disjoint subsets S_1 , S_2 and S_3 . If $y_1 \in S_1$, $y_2 \in S_2$, $\{x_1, x_2\} \subseteq S_3$ and the mechanisms of P_1 and P_2 are known, determining all responsible items takes at most $\lceil \log_2(\max\{|S_1|, |S_2|\}) \rceil + \lceil \log_2 |S_3| \rceil$ tests.

Proof. When both phenotypes are OR-mechanism, we determine x_1 and x_2 by binary search for both phenotypes on subset S_3 which requires $\lceil \log_2 |S_3| \rceil$ tests. We can then determine y_1 and y_2 by binary search for both phenotypes on subsets S_1 and S_2 simultaneously using $\lceil \log_2(\max\{|S_1|, |S_2|\}) \rceil$ tests. Similarly, for both phenotypes are AND-mechanism.

If P_1 is OR-mechanism and P_2 is AND-mechanism, we can divide S_3 into two equal subsets S_{31} and S_{32} and test $(S_{31} \cup S_2)$, if the results are (i) positive (or negative) for both phenotypes, both items $\{x_1, x_2\}$ are in S_{31} (or S_{32}), then we can recursively solve the problem with half of the size of S_3 , otherwise, (ii) x_1 and x_2 are in different set S_{31} and S_{32} , and we can determine x_1 and x_2 at the same time by binary search and including S_2 in every test. $y_1 \in S_1$, $y_2 \in S_2$ can then be determined by performing binary search simultaneously. \square

5 Conclusions

In this paper, we introduced the phenotype testing problem which is an important generalization of the combinatorial group testing problem. We have obtained

several interesting results for the adaptive version of the problem for handling any number of phenotypes (k) and any number of responsible items (d) in each phenotype. For some special cases with k and d smaller than 2, algorithms using near-optimal number of tests are also presented. In this paper, we only consider two common mechanisms, namely AND-version and OR-version, on how the subset of items relates to a phenotype. More complicated mechanisms, such as mixing AND and OR in the same subset of inducing items, should be modeled and considered. Also, even for the OR-version and AND-version of the problems, the lower bound and upper bound are still not closed yet. Finding a better algorithm which uses fewer tests or finding a better lower bound would be desirable.

References

1. Bishop, M.A., Macula, A.J., Renz, T.E., et al.: Hypothesis group testing for disjoint pairs. *Jour. Comb. Optim.* 15, 7–16 (2008)
2. Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. *SIAM Jour. on Comput.* 34(5), 1253–1270 (2005)
3. Chin, F., Leung, H., Yiu, S.M.: Non-Adaptive Complex Group Testing with Multiple Positive Sets. *Theo. Applic. Models Comput. (TAMC)*, 172–183 (2011)
4. Deng, P., Hwang, F.K., Wu, W., et al.: Improved construction for pooling design. *Jour. Comb. Optim.* 15, 123–126 (2008)
5. Du, D.Z., Hwang, F.: Combinatorial group testing and its applications, 2nd edn. World Scientific, Singapore (2000)
6. Du, D.Z., Hwang, F.K., Wu, W., et al.: New construction for transversal design. *Jour. Comput. Biol.* 13(4), 990–995 (2006)
7. Elbashir, S.M., Harborth, J., Lendeckel, W., et al.: Duplexes of 21-Nucleotide RNAs Mediate RNA Interference in Cultured Mammalian Cells. *Nature* 411, 494–498 (2001)
8. Fire, A., Xu, S., Montgomery, M.K., et al.: Potent and Specific Genetic Interference by Double-Stranded RNA in *Caenorhabditis Elegans*. *Nature* 391, 806–811 (1998)
9. Hwang, F.K.: A method for detecting all defective members in a population by group testing. *Jour. Amer. Statist. Assoc.* 67, 605–608 (1972)
10. Jendreyko, N., Popkov, M., Rader, C., et al.: Phenotypic Knockout of VEGF-R2 and Tie-2 with an Intrabody Reduces Tumor Growth and Angiogenesis in vivo. *PNAS* 102(23), 8293–8298 (2005)
11. Li, C.H.: A Sequential Method for Screening Experimental Variables. *Jour. Amer. Stat. Assoc.* 57(298), 455–477 (1962)
12. Ngo, H.Q., Du, D.Z.: A Survey on Combinatorial Group Testing Algorithms with Applications to DNA Library Screening. In: Du, D.-Z., Pardalos, P.M., Wang, J. (eds.) *Discrete Mathematical Problems with Medical Applications. DIMACS Series*, vol. 55. American Mathematical Society, Providence (2000)
13. Ruberti, F., Capsoni, S., Comparini, A., et al.: Phenotypic Knockout of Nerve Growth Factor in Adult Transgenic Mice Reveals Severe Deficits in Basal Forebrain Cholinergic Neurons, Cell Death in the Spleen, and Skeletal Muscle Dystrophy. *Jour. Neurosci.* 20(7), 2589–2601 (2000)
14. Yang, A.G., Bai, X., Huang, X.F., et al.: Phenotypic Knockout of HIV Type 1 Chemokine Coreceptor CCR-5 by Intrakines as Potential Therapeutic Approach for HIV-1 Infection. *Proc. Natl. Acad. Sci.* 94, 11567–11572 (1997)

An Index Structure for Spaced Seed Search

Taku Onodera and Tetsuo Shibuya

Human Genome Center, Institute of Medical Science, University of Tokyo
4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan
{tk-ono,tshibuya}@hgc.jp

Abstract. In this paper, we introduce an index structure of texts which supports fast search of patterns with “don’t care”s in predetermined positions. This data structure is a generalization of the suffix array and has many applications especially for computational biology. We propose three algorithms to construct the index. Two of them are based on a variant of radix sort but each utilizes different types of referential information to sort suffixes by multiple characters at a time. The other is for the case when “don’t care”s appear periodically in patterns and can be combined with the others.

1 Introduction

String searching algorithms can be classified into online search and index search. Though construction of indices is computationally expensive, it makes successive searches much faster. Hence, it is a good idea to prepare an index for the text when the text is large and static and many searches are likely to be performed later.

In this paper, we consider an index structure which supports search of patterns with “don’t care”s. A “don’t care” is a special character that can match any single character and it is also called a ”wild card”. Such queries occur, for example, in spaced seed search. In the research of computational biology, search of homologous regions between two sequences is a common task and to do that the following three steps are widely used: a)extract a short segment called a seed from one of the sequences; b)find the occurrences of the seed in the other; c)examine the surrounding region of each hit by alignment. A spaced seed is a seed including “don’t care”s. Ma et. al.^[10] found that it is possible to optimize the arrangement of “don’t care”s in a spaced seed to make homology search faster and more sensitive. Since then, many researches about spaced seeds, both theoretical and practical, have been made ^{[2], [9], [14]}.

The gapped suffix array ^[3] is a generalization of the suffix array. A (g_0, g_1) -gapped suffix array of a text is the indices of the heads of suffixes, sorted ignoring i -th characters for $i \in [g_0, g_0 + g_1]$. If the (g_0, g_1) -gapped suffix array is given, patterns whose i -th characters are “don’t care”s for $i \in [g_0, g_0 + g_1]$ can be searched in $O((m - g_1) \log n)$ -time where n and m are the length of the text and the pattern respectively. The gapped suffix array can be constructed in $O(n)$ -time.

Because the gapped suffix array is not applicable to the search of spaced seeds as it is, we generalize it as follows. First, fix a binary string $b = b_1 b_2 \dots b_w \in \{0, 1\}^w$. Then, we consider an array containing the indices of the heads of suffixes sorted ignoring i -th characters for $i \in \{1 \leq j \leq w | b_j = 0\}$. For example, if $b = 101$ suffixes are sorted according to their first and third characters while the second and those following the third are ignored. If this array is given, patterns whose i -th characters are “don’t care”’s for $i \in \{1 \leq j \leq w | b_j = 0\}$ can be searched in $O(k \log n)$ -time where $k := \#\{1 \leq i \leq w | b_i = 1\}$.

The problems considered in the rest of the paper and our results for them are summarized as follows.

Problem 1. Given a text $T = T[1, 2, \dots, n]$ and a binary string $b = b_1 b_2 \dots b_w$, sort the suffixes of T ignoring i -th characters for $i \in \{1 \leq j \leq w | b_j = 0\}$ and construct an array $b\text{-}SA[1, 2, \dots, n]$ s.t. $b\text{-}SA[i]$ is the index of the head of the i -th suffix.

Problem 2. Given a text $T = T[1, 2, \dots, n]$, $b = b_1 b_2 \dots b_w$, $b\text{-}SA$ and the rank corresponding to $b\text{-}SA$ where $b\text{-}SA$ is the same as that in Problem 1, sort the suffixes of T ignoring i -th characters for $i \in \{1 \leq j \leq n | b_{j \% w} = 0\}$ where $j \% w$ is the remainder of j divided by w , and construct an array $b^*\text{-}SA[1, 2, \dots, n]$ s.t. $b^*\text{-}SA[i]$ is the index of the head of the i -th suffix.

Result for Problem 1. Given T and b , $b\text{-}SA$ (and the corresponding rank) can be obtained either in $O(gn)$ -time and $O(n)$ -space where g is the number of runs of 1 in b or in $O(\frac{wn}{\epsilon \log w})$ -time and $O(w^\epsilon n)$ -space where ϵ can be any constant s.t. $0 < \epsilon < 1$.

Result for Problem 2. Given T , b , $b\text{-}SA$ and corresponding rank, $b^*\text{-}SA$ can be obtained in $O(n)$ -time and $O(n)$ -space.

Though it requires much memory, the second algorithm for Problem 1 can be faster than the first when g is large (For example, when there is $\exists c > 0$ s.t. $g > cn$). The algorithm for Problem 2 requires not only $b\text{-}SA$ but also the rank. When the rank is not given, one can compute it by checking i -th characters of suffixes for $i \in \{1 \leq j \leq w | b_j = 1\}$, but this naive method takes $O(kn)$ -time. The result for Problem 1 means one can provide both $b\text{-}SA$ and the rank in $O(gn)$ -time to the solver of Problem 2.

There are other types of indices which aid the search of patterns with “don’t care”’s [13], [7], [15]. These are based on ordinary indices like the suffix tree. These indices support more flexible search than ours because they do not assume positions in patterns where “don’t care”’s occur. On the other hand, these indices work as a filter in general and, after referring to it, one needs to determine whether each candidate is actually a match. Thus, in terms of search time, our index is better for inputs to which these indices generate many false positive candidates.

In section 2, we set up basic definitions and notations. The usage of the index is also explained here. In sections 3 and 4, we describe the results for the problems 1 and 2 respectively. We conclude in section 5.

2 Preliminaries

2.1 Definitions and Notations

Let Σ be a finite totally ordered set. A string T is either an element of $\cup_{n=1}^{\infty} \Sigma^n$ or an empty string. That is, the set of strings is equal to $\Sigma^* := \cup_{n=0}^{\infty} \Sigma^n$ where Σ^0 is the set consists only of the empty string. The length of T is $n \geq 0$ s.t. $T \in \Sigma^n$, and is denoted by $|T|$. We denote the i -th character of a string T by $T[i]$. For string T of length n and $i, j \in \mathcal{N}$ s.t. $1 \leq i \leq j$, let $T[i, j]$ be $T[i]T[i+1]\dots T[j]$ if $j \leq n$, $T[i]T[i+1]\dots T[n]$ if $i \leq n < j$ and the empty string if $n < i$. Let T_i be $T[i, n]$ for $i \leq n$. The lexicographic order \leq between $T_1, T_2 \in \Sigma^*$ is defined as follows: a) $\phi \leq T$ for $\forall T \in \Sigma^*$; b) $T_1 \leq T_2$ if $|T_1| > 0, |T_2| > 0$ and $T_1[1] < T_2[1]$; c) $T_1 \leq T_2$ if $|T_1| > 0, |T_2| > 0, T_1[1] = T_2[1]$ and $T_1[2] \leq T_2[2]$. Throughout the paper we assume integer alphabets, i.e. the size of Σ is $O(|T|)$.

A “don’t care” is a special character not included in Σ and we denote it by ? . A pattern P is an element of $\cup_{m=1}^{\infty} (\Sigma \cup \{\text{?}\})^m$. The length of P is $m \geq 1$ s.t. $P \in (\Sigma \cup \{\text{?}\})^m$ and denoted by $|P|$. The i -th character of a pattern P is denoted by $P[i]$. If a pattern P and a string T satisfy $P[j] = T[i+j-1]$ for $\forall j \in \{i | 1 \leq i \leq m, P[i] \neq \text{?}\}$, we say P matches T at position i .

For $w \in \mathcal{N}$, a “don’t care” position of length w , $b = b_1b_2\dots b_w$, is an element of $\{0, 1\}^w$. Let $\mathbf{1}_b$, $\mathbf{0}_b$ and $|b|$ denote $\{1 \leq i \leq w | b_i = 1\}$, $\{1 \leq i \leq w | b_i = 0\}$ and $\#\mathbf{1}_b$ respectively. Let $b[i, j]$ denote $b_ib_{i+1}\dots b_j$ for $1 \leq i \leq j \leq w$. Let $b(T)$ denote $T[i_1]T[i_2]\dots T[i_k]$ (if $i_k \leq n$), $T[i_1]T[i_2]\dots T[i_j]$ (if $i_j \leq n < i_{j+1}, 1 \leq j < k$), ϕ (if $n < i_1$) where $\{i_j\}_{j=1}^k$ is the subsequence of $\{i\}_{i=1}^w$ s.t. $\{i_j\}_j$ equals to $\mathbf{1}_b$ as sets. The b -lexicographic order \leq_b between $T_1, T_2 \in \Sigma^*$ is defined as $T_1 \leq_b T_2$ iff $b(T_1) \leq b(T_2)$. \leq_b is a preorder. We write $T_1 =_b T_2$ if $T_1 \leq_b T_2$ and $T_2 \leq_b T_1$.

The leftmost rank of an element e in a finite totally ordered set S is $\#\{e' \in S | e' \leq e, e \neq e'\} + 1$.

The suffix array SA of a text T is the unique permutation of $\{1, \dots, n\}$ s.t. $T_{SA[i]}$ is lexicographically increasing. SA_h is a permutation s.t. $T_{SA_h[i]}[1, h]$ is lexicographically non-decreasing. The h -rank of T_i is the leftmost rank of $T_i[1, h]$ in $\{T_i[1, h]\}_{i=1}^n$ ordered lexicographically. Let $RSA_h[1, \dots, n]$ be an array s.t. $RSA_h[i]$ is the h -rank of T_i . The height array Hgt of a string T is an array s.t. $Hgt[i]$ is the length of the longest common prefix of $T_{SA[i-1]}$ and $T_{SA[i]}$ for $1 < i \leq n$.

The b -gapped suffix array $b-SA$ of a string T of length n is the unique permutation of $\{1, \dots, n\}$ s.t. a) $T_{b-SA[i]} \leq_b T_{b-SA[i+1]}$ ($1 \leq i < n$); b) $b-SA[i] < b-SA[j]$ if $T_{b-SA[i]} =_b T_{b-SA[j]}$ and $i > j$.

Let $b-SA_h := b[1, h]-SA$. The b -rank of T_i is the leftmost rank of T_i in $b-SA$. Let (b, h) -rank be $b[1, h]$ -rank. Let $b-RSA_h[1, \dots, n]$ be an array s.t. $b-RSA_h[i]$ is the (b, h) -rank of T_i .

2.2 Search Method

The b -gapped suffix array can be applied to pattern matching in almost the same way as the suffix array. If b -gapped suffix array $b-SA$ of a string T and

$b \in \{0, 1\}^w$ is given, a pattern P of length at most w s.t. $P[i] = ?$ for $i \in \mathbf{0}_b$ and $P[i] \in \Sigma$ for $i \in \mathbf{1}_b$ can be located by binary search in $O(|b| \log n)$ -time. All the occurrences of the pattern can be enumerated in $O(|b| \log n + Occ(P))$ -time where $Occ(P)$ is the number of occurrences of P .

3 Constructing the b -Suffix Array for Given T and b

In this section, we consider constructing b -SA for given T and b . An obvious way to do this is to use ordinary radix sort, that is, to repeat sorting the suffixes by i -th character while i runs through all $i \in \mathbf{1}_b$ downwards from the largest to the smallest. It takes $O(|b|n)$ -time and $O(n)$ -space. We propose two more elaborate algorithms to construct b -suffix arrays. We first explain the underlying ideas shared in common. Then we describe each algorithm.

3.1 Underlying Ideas

In SA_h suffixes of the same h -rank neighbor each other. These groups are called h -groups. When sorted by first h' characters where $h < h'$, an h -group can split into several h' -groups but each suffix never move beyond the boundaries of the h -group it belongs to. Thus the members of an h -group are still located together in $SA_{h'}$. In particular, this is true for h -groups of $\forall h \leq n$ in $SA = SA_n$. In some suffix array construction algorithms, $SA = SA_n$ is obtained by repeatedly deriving SA_{2h} from SA_h [11], [8].

Similarly, in b - SA_h , suffixes of the same (b, h) -rank align consecutively, which we call a (b, h) -group. We consider constructing b -SA in a similar way as that for SA above. But unfortunately, the method to double h at a time no longer works for b - SA_h . We try other methods to increase h . In both of the following subsections we use a modification of radix sort to derive b - SA_h of larger h from that of smaller h . This method is based on the sort algorithms presented in [11] and [12].

3.2 $O(gn)$ -Time, $O(n)$ -Space Algorithm

For a “don’t care” position b , we call the subregion from b_{i_1} to b_{i_2} a run of 1 iff a) $b_i = 1 (i_1 \leq i \leq i_2)$; b) $i_1 = 1$ or $b_{i_1-1} = 0$; c) $i_2 = w$ or $b_{i_2+1} = 0$. A run of 1 of length r is denoted by 1^r . A run of 0 is defined similarly.

In this subsection, we prove the following theorem.

Theorem 1. *Given a text T of length n and a “don’t care” position b of length w , the b -suffix array b -SA can be constructed in $O(gn)$ -time and $O(n)$ -space where g is the number of runs of 1 in b where ϵ can be any constant s.t. $0 < \epsilon < 1$.*

b can be divided into runs as $0^{r'_1}1^{r_1}\dots0^{r'_g}1^{r_g}0^{r'_{g+1}}$. Let i_j be the index of the head of the j -th run of 1. Suppose b - $SA_{i_{j-1}+r_{j-1}-1}$ and b - $RSA_{i_{j-1}+r_{j-1}-1}$ are given and we compute b - $SA_{i_j+r_j-1}$ and b - $RSA_{i_j+r_j-1}$. Because $b_i = 0$ for $i \in$

$[i_{j-1} + r_{j-1}, i_j - 1]$, $b-SA_{i_{j-1}+r_{j-1}-1} = b-SA_{i_j-1}$ and $b-RSA_{i_{j-1}+r_{j-1}-1} = b-RSA_{i_j-1}$. Hence, the $(b, i_j - 1)$ -groups are already collected and the next thing to do is to sort each elements of a $(b, i_j - 1)$ -group according to characters between the i_j -th and the $i_j + r_j - 1$ -th. $T_i[i_j, i_j + r_j - 1] = T_{i+i_j-1}[1, r_j]$ and the right hand sides for different i s can be compared in constant time if we have RSA_{r_j} , the array of r_j -ranks.

RSA_{r_j} is easily computed when we have SA and Hgt . The boundaries of h -groups are those places where suffixes with different $h + 1$ -th characters lie next to each other. They can be located by scanning Hgt once marking is s.t. $Hgt[i] < h$. Thus we calculate SA and Hgt in advance.

We prepare queues for each r_j -rank. We enumerate $b-SA_{i_j-1}$ in ascending order and put suffixes into the queues corresponding to $RSA_{r_j}[i + i_j - 1]$. Then we enumerate queues from the one corresponding to the lowest r_j -rank to the one corresponding to the highest. From each queue, we pop all the suffixes out aligning them by the following rule: a) Suffixes of lower $(b, i_j - 1)$ -rank are aligned before suffixes of higher $(b, i_j - 1)$ -rank; b) suffixes of the same $(b, i_j - 1)$ -rank are aligned according to the order they are popped out. We maintain $b-RSA_{i_j-1}$ to find the $(b, i_j - 1)$ -rank of each suffix in constant time. After that, suffixes in a $(b, i_j - 1)$ -group are arranged according to the i -th characters for $i \in [i_j, i_j + r_j - 1] \cap 1_b$, which means we got $b-SA_{i_j+r_j-1}$. $b-SA$ is obtained by repeating this procedure for $j = 1$ to g ,

Next, we explain the detail. The following data are used¹: $S1: b-SA_{i_j-1}$, $R1: b-RSA_{i_j-1}$, $C1$: counter for $S1$, B : marker for the heads of $(b, i_j + r_j - 1)$ -groups, $S2$: buckets, $R2: RSA_{r_j}$, $C2$: counter for $S2$, SA : suffix array, Hgt : height array.

We show how to update $S1$ and $S2$ from $b-SA_{i_j-1}$ and $b-RSA_{i_j-1}$ to $b-SA_{i_j+r_j-1}$ and $b-RSA_{i_j+r_j-1}$ respectively. First, we assign RSA_{r_j} to $R2$ using SA and Hgt . Next, we put suffixes in $S1$, say $T_{S1[i]}$ into buckets in $S2$ corresponding to the r_j -rank of $T_{S1[i]+r_j-1}$, that is $R2[i1]$ where $i1 := S1[i] + i_j - 1$. Because there can be several suffixes of the same rank, we use $C2$ to record offsets as $S2[R2[i1] + C2[R2[i1]]] \leftarrow S1[i]$. Suffixes of $(b, i_j - 1)$ -groups of only 1 member and T_i s for $i > n - i_j + 1$ are already in their final position and can be skipped. Then we return the suffixes in $S2$ to the tail of the $(b, i_j - 1)$ -groups they belong. When i is the index of the head of an r_j -group in $S2$, the index of the head of the $(b, i_j - 1)$ -group suffix $S2[i + j]$ belongs to is $k := R1[S2[i + j]]$. Again, there can be several elements to return to a $(b, i_j - 1)$ -group and we use $C1$ to record offsets as $S1[k + C1[k]] \leftarrow S2[i + j]$. We set $B[i]$ to 1 every time we push back a suffix to $S1[i]$. At this time, the suffixes returned to the same $(b, i_j - 1)$ -group from the same r_j -group make up a $(b, i_j + r_j - 1)$ -group. Each time we finish returning suffixes from one bucket, we traverse the same bucket again setting $B[i]$ to 0 except the heads of $(b, i_j + r_j - 1)$ -groups. After returning every suffixes to $S1$, we update $S2$ to $b-RSA_{i_j+r_j-1}$ referring to B . The whole process in this paragraph is repeated while j runs from 1 to g .

¹ It is possible to implement without $C1$ and $C2$ but in that case, we need to scan $S1$ and $S2$ to find the appropriate positions to put suffixes increasing the time complexity.

SA and Hgt can be made in $O(n)$ -time [4], [6], [12], [5]. In each update from $b-SA_{i_j-1}$ to $b-SA_{i_j + r_j - 1}$, we initialize $R2, C1, C2, B$, move suffixes from $S1$ to $S2$, return suffixes from $S2$ to $S1$ and update $R2$. All of these are done in $O(n)$ -time. Therefore the total time complexity is $O(gn)$. Because we only use constant number of data of $O(n)$ -size, the total space complexity is $O(n)$. Therefore theorem [1] follows.

The pseudocode of the algorithm is in appendix [A].

3.3 $O(\frac{wn}{\epsilon \log w})$ -Time and $O(w^\epsilon n)$ -Space Algorithm

Below we prove the following theorem.

Theorem 2. *Given a text T of length n and a “don’t care” position b of length w , the b -suffix array $b-SA$ can be constructed in $O(\frac{wn}{\epsilon \log w})$ -time and $O(w^\epsilon n)$ -space. where ϵ can be any constant s.t. $0 < \epsilon < 1$.*

Fix v s.t. $1 \leq v \leq w$. For brevity, we assume v divides w evenly. Suppose $b-SA_{(t-1)v}$ and $b-RSA_{(t-1)v}$ are given and we compute $b-SA_{tv}$ and $b-RSA_{tv}$. $(b, (t-1)v)$ -groups are already collected and we need to sort elements in a $(b, (t-1)v)$ -group according to i -th characters for $i \in [(t-1)v+1, tv] \cap \mathbf{1}_b$. By the same method as that of subsection [3.2], this sorting is done in $O(n)$ -time if $b[(t-1)v+1, tv]$ -RSA is given. Thus, we calculate all b' -RSAs for all “don’t care” positions b' of length v in advance. By computing b' -SAs and b' -RSAs from those with smaller $|b'|$ s to larger ones, this preprocessing is done in $O(2^v n)$ -time.

The total time complexity is $O(2^v n + wn/v)$ and the total space complexity is $O(2^v n)$. In particular, if we choose v to be $\epsilon \log w$ where ϵ is a constant s.t. $0 < \epsilon < 1$, the time complexity is $O(\frac{wn}{\epsilon \log w})$ and the space complexity is $O(w^\epsilon n)$. Therefore theorem [2] follows. This value of v is chosen so that the time needed for preprocessing and those for main process are close to each other. ²

This algorithm is suitable for the case of multiple “don’t care” positions and a single text. Since b' -SAs and b' -RSAs depend only on T , it is possible to share these data among the calculations of different b -STs for the same text and different “don’t care” positions. ³ In particular, when you have a text and are going to be given multiple “don’t care” positions of the same length w , each b -suffix array can be constructed in $O(wn/\epsilon \log w)$ -time for $1 \leq \forall \epsilon$. ⁴

4 Constructing b -Suffix Arrays for Periodic b

In this section we prove the following theorem.

² The main process takes longer. The v that balances them is expressed by using the Lambert W function.

³ SA and Hgt in the algorithm of subsection [3.2] can also be shared. But in terms of time complexity, the fact makes no difference.

⁴ $1 \leq \epsilon$ because it should be taken so that the preprocessing takes longer than the construction of each b -suffix array.

Theorem 3. Given a text T of length n , a “don’t care” position b of length p , the b -suffix array $b\text{-}SA$ and corresponding rank $b\text{-}RSA = b\text{-}RSA_p$, $b^*\text{-}SA[i]$ is obtained in $O(n)$ -time and $O(n)$ -space where b^* is a “don’t care” position obtained by repeating b enough times to be longer than T .

$T_i \leq_{b^*} T_j$ iff a) $T_i[1, p] \leq_b T_j[1, p]$ and $T_j[1, p] \not\leq_b T_i[1, p]$ ($\Leftrightarrow b\text{-}RSA[i] < b\text{-}RSA[j]$) or b) $T_i[1, p] =_b T_j[1, p]$ ($\Leftrightarrow b\text{-}RSA[i] = b\text{-}RSA[j]$) and $T_{i+p} \leq_{b^*} T_{j+p}$. Thus, \leq_{b^*} on $\{T_i\}_i$ is equivalent to the lexicographic order on $\{b\text{-}RSA[i]b\text{-}RSA[i+p]\dots b\text{-}RSA[i+\lfloor(n-i)/p\rfloor p]\}_i$ seen as strings. To sort $\{T_i\}_i$ by \leq_{b^*} , it suffices to sort $\{rs(i)\}_i$ by lexicographic order where $rs(i) := b\text{-}RSA[i]b\text{-}RSA[i+p]\dots b\text{-}RSA[i+\lfloor(n-i)/p\rfloor p]$. For i s.t. $1 \leq i \leq p$ and $0 \leq k \leq \lfloor(n-i)/p\rfloor$, $rs(i+kp)$ is a suffix of $rs(i)$. We consider the string $rs := rs(1)0rs(2)0\dots 0rs(p)$. $\{rs(i)\}_i$ can be sorted by sorting the suffixes of rs . It is easy to ignore the suffixes starting from 0 because they gather to the head when sorted. Because the comparison of two suffixes ends before or at the time when either of them reaches the end, comparison of corresponding $rs(i)$ s is not affected by inserting 0s.

Almost all of the work is reduced to suffix sorting of rs , which can be done in $O(n)$ -time. Therefore, the total time complexity is $O(n)$. Space complexity is also $O(n)$. Therefore theorem 3 follows.

An argument similar to this is made in [4].

Below we describe the pseudocode of the algorithm.

Input: b -suffix array $b\text{-}SA$, b -rank $b\text{-}RSA$, assume p evenly divides n

Output: b^* -suffix array $b^*\text{-}SA$

```

construct  $rs = rs(1)0rs(2)0\dots 0rs(p)$ 
construct the suffix array of  $rs$   $SA1$ 
 $n' \leftarrow n/p$ 
for  $i = p$  to  $n + p - 1$  do
     $j \leftarrow SA1[i]$ 
     $q \leftarrow \lfloor j/(n'+1) \rfloor$ 
     $r \leftarrow j \% (n'+1)$ 
     $b^*\text{-}SA[i-p+1] \leftarrow q + 1 + pr$ 

```

5 Conclusion

We introduced an index structure based on the suffix array, which supports efficient search of patterns with “don’t care”s in predetermined positions. We designed three algorithms to construct the index. Two of them are for general inputs and the other is for the case of patterns with periodic “don’t care”s.

References

1. Andersson, A., Nilsson, S.: A new efficient radix sort. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 714–721. IEEE Computer Society, Washington, DC, USA (1994)

2. Brown, D.G.: A Survey of Seeding for Sequence Alignment, pp. 117–142. John Wiley & Sons, Inc. (2007)
3. Crochemore, M., Tischler, G.: The Gapped Suffix Array: A New Index Structure for Fast Approximate Matching. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 359–364. Springer, Heidelberg (2010)
4. Kärkkäinen, J., Sanders, P.: Simple Linear Work Suffix Array Construction. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 943–955. Springer, Heidelberg (2003)
5. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time Longest-Common-prefix Computation in Suffix Arrays and its Applications. In: Amir, A., Landau, G.M. (eds.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001)
6. Ko, P., Aluru, S.: Space Efficient Linear Time Construction of Suffix Arrays. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 200–210. Springer, Heidelberg (2003)
7. Lam, T.W., Sung, W.-K., Tam, S.-L., Yiu, S.-M.: Space Efficient Indexes for String Matching with Don't Cares. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 846–857. Springer, Heidelberg (2007)
8. Larsson, N.J., Sadakane, K.: Faster suffix sorting. Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1-20/(1999), Department of Computer Science, Lund University, Sweden (May 1999)
9. Lin, H., Zhang, Z., Zhang, M.Q., Ma, B., Li, M.: Zoom! zillions of oligos mapped. Bioinformatics 24(21), 2431–2437 (2008)
10. Ma, B., Tromp, J., Li, M.: Patternhunter: faster and more sensitive homology search. Bioinformatics 18(3), 440–445 (2002)
11. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. In: Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1990, pp. 319–327. Society for Industrial and Applied Mathematics, Philadelphia (1990)
12. Nong, G., Zhang, S., Chan, W.H.: Linear suffix array construction by almost pure induced-sorting. In: Storer, J.A., Marcellin, M.W. (eds.) DCC, pp. 193–202. IEEE Computer Society (2009)
13. Sohel Rahman, M., Iliopoulos, C.S.: Pattern matching algorithms with don't cares. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plasil, F., Bieliková, M. (eds.) SOFSEM (2), pp. 116–126. Institute of Computer Science AS CR, Prague (2007)
14. Rumble, S.M., Lacroute, P., Dalca, A.V., Fiume, M., Sidow, A., Brudno, M.: Shrimp: Accurate mapping of short color-space reads. PLoS Comput. Biol. 5(5), e1000386 (2009)
15. Tam, A., Wu, E., Lam, T.-W., Yiu, S.-M.: Succinct Text Indexing with Wildcards. In: Karlgren, J., Tarhio, J., Hyryrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 39–50. Springer, Heidelberg (2009)

A Pseudocode of the Algorithm in 3.2

Input: text T of length n , “don’t care” position $b \in \{0, 1\}^w$

Output: b -gapped suffix array $S1$

```

/* R1:heads of  $(b, i_j - 1)$ -groups, C1:counter for  $S1$ , B:boundary marker */
/* S2:buckets, R2: $r_j$ -ranks, C2:counter for  $S2$  */
for  $i = 1$  to  $n$  do
     $S1[i] \leftarrow i$ 
construct  $SA$  and  $Hgt$ 
for  $j = 1$  to  $k$  do
     $R2 \leftarrow RSA_{r_j}$ 
    initialize  $C1, C2, B$  by 0
    for  $i = 1$  to  $n$  do
        if  $R1[S1[i]] = i$  then
             $B[i] \leftarrow 1$ 
        if  $i = n$  or  $R1[S1[i + 1]] = i + 1$  then
            continue
         $i1 \leftarrow S1[i] + i_j - 1$ 
        if  $i_1 > n$  then
            continue
         $S2[R2[i1] + C2[R2[i]]] \leftarrow S1[i]$ 
         $C2[R2[i1]] \leftarrow C2[R2[i1]] + 1$ 
     $i \leftarrow 1$ 
while true do
    while  $i \leq n$  and  $C2[i] = 0$  do
         $i \leftarrow i + 1$ 
    if  $i > n$  then
        break
    for  $j = 0$  to  $C2[i] - 1$  do
         $k \leftarrow R1[S2[i + j]]$ 
         $S1[k + C1[k]] \leftarrow S2[i + j]$ 
         $B[k + C1[k]] \leftarrow 1$ 
         $C1[k] \leftarrow C1[k] + 1$ 
    for  $j = C2[i] - 1$  to  $0$  do
         $k \leftarrow R1[S2[i + j]]$ 
         $l \leftarrow k + C1[k] - 1$ 
        while  $l > 1$  and  $B[l] = 1$  and  $S1[l - 1] + i_j - 1 \leq n$  and  $R2[S1[l] + i_j - 1] =$ 
             $R2[S1[l - 1] + i_j - 1]$  do
             $B[l] \leftarrow 0$ 
             $l \leftarrow l - 1$ 
    for  $i = 1$  to  $n$  do
        if  $B[i] = 1$  then
            left =  $i$ 
         $R1[S1[i]] \leftarrow left$ 

```

Author Index

- Abel, Zachary 574
Ahn, Hee-Kap 50, 60
Alam, Muhammad Jawaherul 281
Amir, Amihood 683, 714
Ando, Ryuta 474
Angelini, Patrizio 271
Angelino, Elaine 384
Apostolico, Alberto 683
Arora, Sanjeev 6
Atsonios, Ioannis 504
Avis, David 415
- Bae, Sang Won 60
Barba, Luis 70
Bazgan, Cristina 643
Beaumont, Olivier 504
Belmonte, Rémy 110
Biedl, Therese 281
Bock, Adrian 10
Brandstädt, Andreas 100
Buchin, Kevin 250
- Campos, Victor 634
Castelli Aleardi, Luca 312
Chan, Ho-Leung 564
Chen, Danny Z. 584, 594, 604
Chen, Gen-Huey 200
Chen, Ho-Lin 445
Chimani, Markus 40
Chin, Francis Y.L. 754
Chopin, Morgan 643
Cook IV, Atlas F. 240
Cui, Yun 744
- de Berg, Mark 240, 260
Demaine, Erik D. 574
Demaine, Martin L. 574
de Montgolfier, Fabien 435
Devillers, Olivier 312
Di Battista, Giuseppe 271
Dörnfelder, Martin 624
Doty, David 445
Dumitrescu, Adrian 220, 230
- Eisenberg, Estrella 714
Eisenstat, Sarah 574
Elmasry, Amr 150
- Farzan, Arash 302
Fellows, Michael R. 643
Felsner, Stefan 281
Fischer, Johannes 302
Frati, Fabrizio 271
Friedrich, Tobias 190
Fujito, Toshihiro 544
Fujiwara, Hiroshi 544
Fürer, Martin 484
- Gagie, Travis 653
Gawrychowski, Paweł 653
Gerasch, Andreas 281
Gibson, Matt 724
Goetzmann, Kai-Simon 210
Golin, Mordecai 180
Golovach, Petr A. 110
Goodrich, Michael T. 292, 374, 384
Grant, Elyot 10
Gu, Qian-Ping 364
Gudmundsson, Joachim 240
Guillemot, Sylvain 354
Guo, Jiong 614, 624
- Han, Dongfeng 724
Hanusse, Nicolas 504
Harks, Tobias 210
Hartung, Sepp 614
Hasan, Masud 230
Hatano, Kohei 534
He, Meng 140, 150, 160
Heggernes, Pinar 110
Hellerstein, Lisa 703
Hilscher, Evan 220
Hon, Wing-Kai 673
Hsiao, Sheng-Ying 344
Hurwitz, Jeremy 524
- Iacono, John 180
Itsykson, Dmitry 464
Iwama, Kazuo 415

- Jansson, Jesper 744
 Kakimura, Naonori 693
 Kamiński, Marcin 110
 Kamiyama, Naoyuki 130
 Kao, Mong-Jen 494
 Kaufmann, Michael 281
 Khani, M. Reza 20
 Kijima, Shuji 514, 534
 Kim, Sang-Sub 50
 Kim, Yusik 504
 Kirkpatrick, David 554
 Kitano, Takuma 544
 Klein, Sulamita 634
 Klimm, Max 210
 Knauer, Christian 50, 60
 Kobourov, Stephen G. 281
 Komusiewicz, Christian 624
 Könemann, Jochen 10
 Korman, Matias 70, 80
 Krizanc, Danny 180
 Lam, Tak-Wah 564
 Landau, Gad M. 683
 Langerman, Stefan 70
 Lässig, Jörg 405
 Le, Van Bang 120
 Lee, D.T. 494, 734
 Lee, Lap-Kei 564
 Lee, Mira 60
 Leung, Henry C.M. 754
 Levy, Avivit 683, 714
 Lewenstein, Moshe 683
 Lewenstein, Noa 714
 Lin, Ching-Chi 200
 Lin, Tien-Ching 734
 Lu, Chen-Hua 673
 Lynch, Jayson 574
 Makino, Kazuhisa 693
 Matsui, Tomomi 474
 Miller, Konstantin 210
 Misra, Pranabendu 333
 Mitzenmacher, Michael 384
 Mosca, Raffaele 100
 Munro, J. Ian 140, 150, 160
 Navarro, Gonzalo 323
 Nekrich, Yakov 170, 395
 Nevries, Ragnar 120
 Nicholson, Patrick K. 150, 160
 Niedermeier, Rolf 614
 Nisgav, Aviv 425
 Ogata, Masatora 514
 Onodera, Taku 764
 Özkan, Özgür 703
 Paku, Daichi 415
 Pan, Jiangwei 564
 Patt-Shamir, Boaz 425
 Paulusma, Daniël 110
 Porat, Ely 683
 Puglisi, Simon J. 653
 Radoszewski, Jakub 90
 Raman, Rajeev 180
 Raman, Venkatesh 333
 Ramanujan, M.S. 333
 Rao, S. Srinivasa 180
 Russo, Luís M.S. 323
 Rytter, Wojciech 90
 Sakai, Yoshifumi 663
 Salavatipour, Mohammad R. 20
 Sampaio, Rudini 634
 Sanità, Laura 10
 Sauerwald, Thomas 190
 Saurabh, Saket 333
 Schardl, Tao B. 574
 Schlipf, Lena 50
 Seimi, Kento 693
 Seki, Shinnosuke 445
 Sellie, Linda 703
 Shah, Rahul 673
 Shapiro-Ellowitz, Isaac 574
 Shibuya, Tetsuo 764
 Shin, Chan-Su 50, 60
 Silva, Ana 634
 Silveira, Rodrigo I. 70
 Simons, Joseph A. 292
 Sitchinava, Nodari 374
 Sokolov, Dmitry 464
 Sonka, Milan 724
 Soto, Mauricio 435
 Speckmann, Bettina 250, 260
 Stauffer, Alexandre 190
 Suchý, Ondřej 614
 Sudholt, Dirk 405
 Sung, Wing-Kin 744

- Takeda, Masayuki 534
Takimoto, Eiji 534
Thaler, Justin 384
Thankachan, Sharma V. 673
Ting, Hing-Fung 564
Tseng, Kuan-Chieh Robert 554
Tsou, Cheng-Hsiao 200

van der Weele, Vincent 260
van 't Hof, Pim 110
Verbeek, Kevin 250
Viennot, Laurent 435
Vigneron, Antoine 50, 60

Wagner, Dorothea 1
Wang, Chunhao 364
Wang, Haitao 584, 594, 604

Weller, Mathias 624
Woste, Matthias 40
Wu, Xiaodong 724

Yamakami, Tomoyuki 454
Yamashita, Masafumi 514
Yamauchi, Yukiko 514
Yasutake, Shota 534
Yiu, S.M. 754
Yu, Huiwen 484
Yu, Hung-I 734
Yu, Wei 30

Zhang, Guochuan 30
Zhang, Qin 374, 564
Zhou, Gelin 140