

# 32nd International Symposium on Algorithms and Computation

ISAAC 2021, December 6–8, 2021, Fukuoka, Japan

Edited by

Hee-Kap Ahn


Kunihiko Sadakane



*Editors*

**Hee-Kap Ahn** 

Pohang University of Science and Technology, Korea  
heekap@postech.ac.kr

**Kunihiko Sadakane** 

The University of Tokyo, Japan  
sada@mist.i.u-tokyo.ac.jp

*ACM Classification 2012*

Theory of computation; Mathematics of computing

**ISBN 978-3-95977-214-3**

*Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-214-3>.

*Publication date*

December, 2021

*Bibliographic information published by the Deutsche Nationalbibliothek*

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

*License*

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):  
<https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.ISAAC.2021.0

ISBN 978-3-95977-214-3

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>



## LIPICs – Leibniz International Proceedings in Informatics

LIPICs is a series of high-quality conference proceedings across all fields in informatics. LIPICs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

### *Editorial Board*

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

**ISSN 1868-8969**

**<https://www.dagstuhl.de/lipics>**



## ■ Contents

Preface	
<i>Hee-Kap Ahn and Kunihiko Sadakane</i> .....	0:xi
Program Committee	
.....	0:xiii
List of External Reviewers	
.....	0:xv

### Invited Talks

Streaming Pattern Matching	
<i>Tatiana Starikovskaya</i> .....	1:1–1:1
Spanning Properties of Variants of the Delaunay Graph	
<i>Prosenjit Bose</i> .....	2:1–2:1

### Regular Papers

Subquadratic Algorithms for Some 3SUM-Hard Geometric Problems in the Algebraic Decision Tree Model	
<i>Boris Aronov, Mark de Berg, Jean Cardinal, Esther Ezra, John Iacono, and Micha Sharir</i> .....	3:1–3:15
Approximate Maximum Halfspace Discrepancy	
<i>Michael Matheny and Jeff M. Phillips</i> .....	4:1–4:15
The VC-Dimension of Limited Visibility Terrains	
<i>Matt Gibson-Lopez and Zhongxiu Yang</i> .....	5:1–5:17
Clustering with Neighborhoods	
<i>Hongyao Huang, Georgiy Klimenko, and Benjamin Raichel</i> .....	6:1–6:17
Approximating Longest Spanning Tree with Neighborhoods	
<i>Ahmad Biniaz</i> .....	7:1–7:11
Self-Improving Voronoi Construction for a Hidden Mixture of Product Distributions	
<i>Siu-Wing Cheng and Man Ting Wong</i> .....	8:1–8:13
Connected Coordinated Motion Planning with Bounded Stretch	
<i>Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer</i> .....	9:1–9:16
Enclosing Depth and Other Depth Measures	
<i>Patrick Schneider</i> .....	10:1–10:15
Illuminating the $x$ -Axis by $\alpha$ -Floodlights	
<i>Bengt J. Nilsson, David Orden, Leonidas Palios, Carlos Seara, and Paweł Żyliński</i> .....	11:1–11:12

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On Geometric Priority Set Cover Problems <i>Aritra Banik, Rajiv Raman, and Saurabh Ray</i> .....	12:1–12:14
The Complexity of Sharing a Pizza <i>Patrick Schnider</i> .....	13:1–13:15
Dynamic Data Structures for $k$ -Nearest Neighbor Queries <i>Sarita de Berg and Frank Staals</i> .....	14:1–14:14
Preference-Based Trajectory Clustering – An Application of Geometric Hitting Sets <i>Florian Barth, Stefan Funke, and Claudius Proissl</i> .....	15:1–15:14
Efficiently Partitioning the Edges of a 1-Planar Graph into a Planar Graph and a Forest <i>Sam Barr and Therese Biedl</i> .....	16:1–16:15
On the Kernel and Related Problems in Interval Digraphs <i>Mathew C. Francis, Pavol Hell, and Dalu Jacob</i> .....	17:1–17:17
Interval Query Problem on Cube-Free Median Graphs <i>Soh Kumabe</i> .....	18:1–18:14
Untangling Circular Drawings: Algorithms and Complexity <i>Sujoy Bhore, Guangping Li, Martin Nöllenburg, Ignaz Rutter, and Hsiang-Yun Wu</i> .....	19:1–19:17
Algorithms and Complexity on Indexing Elastic Founder Graphs <i>Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen</i> .....	20:1–20:18
Partitioning $H$ -Free Graphs of Bounded Diameter <i>Christoph Brause, Petr Golovach, Barnaby Martin, Daniël Paulusma, and Siani Smith</i> .....	21:1–21:14
Clique-Based Separators for Geometric Intersection Graphs <i>Mark de Berg, Sándor Kisfaludi-Bak, Morteza Monemizadeh, and Leonidas Theodorou</i> .....	22:1–22:15
Near-Optimal Distance Oracles for Vertex-Labeled Planar Graphs <i>Jacob Ewald, Viktor Fredslund-Hansen, and Christian Wulff-Nilsen</i> .....	23:1–23:14
A Characterization of Individualization-Refinement Trees <i>Markus Anders, Jendrik Brachter, and Pascal Schweitzer</i> .....	24:1–24:14
Truly Subquadratic Exact Distance Oracles with Constant Query Time for Planar Graphs <i>Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen</i> .....	25:1–25:12
Interval Edge Coloring of Bipartite Graphs with Small Vertex Degrees <i>Anna Małafiejaska, Michał Małafiejski, Krzysztof M. Ociekiewicz, and Krzysztof Pastuszak</i> .....	26:1–26:12
Selected Neighbor Degree Forest Realization <i>Amotz Bar-Noy, David Peleg, Dror Rawitz, and Elad Yehezkel</i> .....	27:1–27:15

MAX CUT in Weighted Random Intersection Graphs and Discrepancy of Sparse Random Set Systems <i>Sotiris Nikolettseas, Christoforos Raptopoulos, and Paul Spirakis</i> .....	28:1–28:16
The Impact of Geometry on Monochrome Regions in the Flip Schelling Process <i>Thomas Bläsius, Tobias Friedrich, Martin S. Krejca, and Louise Molitor</i> .....	29:1–29:17
Piecewise-Linear Farthest-Site Voronoi Diagrams <i>Franz Aurenhammer, Evanthia Papadopoulou, and Martin Suderland</i> .....	30:1–30:11
Effective Resistance and Capacitance in Simplicial Complexes and a Quantum Algorithm <i>Mitchell Black and William Maxwell</i> .....	31:1–31:27
Anonymity-Preserving Space Partitions <i>Úrsula Hébert-Johnson, Chinmay Sonar, Subhash Suri, and Vaishali Surianarayanan</i> .....	32:1–32:16
Impatient PPSZ – A Faster Algorithm for CSP <i>Shibo Li and Dominik Scheder</i> .....	33:1–33:20
Fine-Grained Meta-Theorems for Vertex Integrity <i>Michael Lampis and Valia Mitsou</i> .....	34:1–34:15
Essentially Tight Kernels For (Weakly) Closed Graphs <i>Tomohiro Koana, Christian Komusiewicz, and Frank Sommer</i> .....	35:1–35:15
Filling Crosswords Is Very Hard <i>Laurent Gourvès, Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos</i> .....	36:1–36:16
Grid Recognition: Classical and Parameterized Computational Perspectives <i>Siddharth Gupta, Guy Sa’ar, and Meirav Zehavi</i> .....	37:1–37:15
An Improved Approximation Algorithm for the Matching Augmentation Problem <i>Joseph Cheriyan, Robert Cummings, Jack Dippel, and Jasper Zhu</i> .....	38:1–38:17
Multimodal Transportation with Ridesharing of Personal Vehicles <i>Qian-Ping Gu and JiaJian Liang</i> .....	39:1–39:16
Algorithms for Normalized Multiple Sequence Alignments <i>Eloi Araujo, Luiz C. Rozante, Diego P. Rubert, and Fábio V. Martinez</i> .....	40:1–40:16
Separated Red Blue Center Clustering <i>Marzieh Eskandari, Bhavika Khare, and Nirman Kumar</i> .....	41:1–41:13
On the Extended TSP Problem <i>Julián Mestre, Sergey Pupyrev, and Seeun William Umboh</i> .....	42:1–42:14
Probabilistic Analysis of Euclidean Capacitated Vehicle Routing <i>Claire Mathieu and Hang Zhou</i> .....	43:1–43:16
Exact and Approximation Algorithms for Many-To-Many Point Matching in the Plane <i>Sayan Bandyopadhyay, Anil Maheshwari, and Michiel Smid</i> .....	44:1–44:14

Augmenting Graphs to Minimize the Radius <i>Joachim Gudmundsson, Yuan Sha, and Fan Yao</i> .....	45:1–45:20
Linear-Time Approximation Scheme for $k$ -Means Clustering of Axis-Parallel Affine Subspaces <i>Kyungjin Cho and Eunjin Oh</i> .....	46:1–46:13
Feedback Vertex Set on Geometric Intersection Graphs <i>Shinwoo An and Eunjin Oh</i> .....	47:1–47:12
Streaming Algorithms for Graph $k$ -Matching with Optimal or Near-Optimal Update Time <i>Jianer Chen, Qin Huang, Iyad Kanj, Qian Li, and Ge Xia</i> .....	48:1–48:17
Making Three out of Two: Three-Way Online Correlated Selection <i>Yongho Shin and Hyung-Chan An</i> .....	49:1–49:17
Tight Competitive Analyses of Online Car-Sharing Problems <i>Ya-Chun Liang, Kuan-Yun Lai, Ho-Lin Chen, and Kazuo Iwama</i> .....	50:1–50:14
Skeletons and Minimum Energy Scheduling <i>Antonios Antoniadis, Gunjan Kumar, and Nikhil Kumar</i> .....	51:1–51:16
Machine Covering in the Random-Order Model <i>Susanne Albers, Waldo Gálvez, and Maximilian Janke</i> .....	52:1–52:16
Nearly-Tight Lower Bounds for Set Cover and Network Design with Deadlines/Delay <i>Noam Touitou</i> .....	53:1–53:16
Cryptographic Hardness Under Projections for Time-Bounded Kolmogorov Complexity <i>Eric Allender, John Gouwar, Shuichi Hirahara, and Caleb Robelle</i> .....	54:1–54:17
Identity Testing Under Label Mismatch <i>Clément L. Canonne and Karl Wimmer</i> .....	55:1–55:17
Unique-Neighbor-Like Expansion and Group-Independent Cossystolic Expansion <i>Tali Kaufman and David Mass</i> .....	56:1–56:17
Group Evacuation on a Line by Agents with Different Communication Abilities <i>Jurek Czyzowicz, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Denis Pankratov, and Sunil Shende</i> .....	57:1–57:24
Lower Bounds for Induced Cycle Detection in Distributed Computing <i>François Le Gall and Masayuki Miyamoto</i> .....	58:1–58:19
Distributed Approximations of $f$ -Matchings and $b$ -Matchings in Graphs of Sub-Logarithmic Expansion <i>Andrzej Czygrinow, Michał Hanćkowiak, and Marcin Witkowski</i> .....	59:1–59:13
Inverse Suffix Array Queries for 2-Dimensional Pattern Matching in Near-Compact Space <i>Dhrumil Patel and Rahul Shah</i> .....	60:1–60:14
Dynamic Boolean Formula Evaluation <i>Rathish Das, Andrea Lincoln, Jayson Lynch, and J. Ian Munro</i> .....	61:1–61:19

Shortest Beer Path Queries in Outerplanar Graphs <i>Joyce Bacic, Saeed Mehrabi, and Michiel Smid</i> .....	62:1–62:16
Space-Efficient Algorithms for Reachability in Directed Geometric Graphs <i>Sujoy Bhore and Rahul Jain</i> .....	63:1–63:17
Repetition- and Linearity-Aware Rank/Select Dictionaries <i>Paolo Ferragina, Giovanni Manzini, and Giorgio Vinciguerra</i> .....	64:1–64:16
Pattern Masking for Dictionary Matching <i>Panagiotis Charalampopoulos, Huiping Chen, Peter Christen, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, and Jakub Radoszewski</i> .....	65:1–65:19
Resilient Level Ancestor, Bottleneck, and Lowest Common Ancestor Queries in Dynamic Trees <i>Luciano Gualà, Stefano Leucci, and Isabella Ziccardi</i> .....	66:1–66:17
Computing Shapley Values for Mean Width in 3-D <i>Shuhao Tan</i> .....	67:1–67:17
Simple Envy-Free and Truthful Mechanisms for Cake Cutting with a Small Number of Cuts <i>Takao Asano</i> .....	68:1–68:17
Adaptive Regularized Submodular Maximization <i>Shaojie Tang and Jing Yuan</i> .....	69:1–69:13
$\Gamma$ -Graphic Delta-Matroids and Their Applications <i>Donggyu Kim, Duksang Lee, and Sang-il Oum</i> .....	70:1–70:13
An Approximation Algorithm for Maximum Stable Matching with Ties and Constraints <i>Yu Yokoi</i> .....	71:1–71:16
A Faster Algorithm for Maximum Flow in Directed Planar Graphs with Vertex Capacities <i>Julian Enoch, Kyle Fox, Dor Mesica, and Shay Mozes</i> .....	72:1–72:16
Maximum-Weight Matching in Sliding Windows and Beyond <i>Leyla Biabani, Mark de Berg, and Morteza Monemizadeh</i> .....	73:1–73:16
Quantum Advantage with Shallow Circuits Under Arbitrary Corruption <i>Atsuya Hasegawa and François Le Gall</i> .....	74:1–74:16





## ■ Preface

This volume contains the papers presented at the 32nd International Symposium on Algorithms and Computation (ISAAC 2021). ISAAC 2021 was held in a hybrid manner on December 6–8, 2021 and was organized by Kyushu University, Japan. ISAAC 2021 provided a forum for researchers working in the areas of algorithms, theory of computation, and computational complexity. The technical program of the conference included 72 contributed papers. We received 215 submissions in response to the call for papers. Each submission received at least three reviews. The program committee held electronic meetings using EasyChair. In the end, the Program Committee selected 72 of the submissions for presentation at the symposium.

The conference included two invited presentations, delivered by Prosenjit Bose (Carleton University) and Tatiana Starikovskaya (The École normale supérieure, Paris). Abstracts of their talks are included in this volume. We wish to thank all the authors who submitted extended abstracts for consideration, the program committee members for their scholarly efforts, and all external reviewers who assisted in the evaluation process. We are grateful to Kayamori Foundation of Information Science Advancement, Support Center for Advanced Telecommunications Technology Research (SCAT), KDDI Foundation, Algorithmic Foundations for Social Advancement (AFSA) by Grant-in-Aid for Transformative Research Areas, MEXT, Japan, and Fusion of Computer Science, Engineering and Mathematics Approaches for Expanding Combinatorial Reconfiguration by Grant-in-Aid for Transformative Research Areas, MEXT, Japan, for financial support and the local organizers of ISAAC 2021. Finally, we acknowledge the endorsement from Special Interest Group on Algorithms (SIGAL) of IPSJ.

December 2021

Hee-Kap Ahn and Kunihiko Sadakane





## ■ Program Committee

Hee-Kap Ahn (co-chair, Pohang University of Science and Technology (POSTECH))  
Kevin Buchin (TU Eindhoven)  
Sankardeep Chakraborty (The University of Tokyo)  
T-H. Hubert Chan (University of Hong Kong)  
Johannes Fischer (TU Dortmund)  
Travis Gagie (Dalhousie University)  
Mordecai Golin (Hong Kong University of Science and Technology)  
Hiro Ito (The University of Electro-Communications)  
Akinori Kawachi (Mie University)  
Yusuke Kobayashi (Kyoto University)  
Chung-Shou Liao (National Tsing Hua University, Taiwan)  
Toshimitsu Masuzawa (Osaka University)  
Yota Otachi (Nagoya University)  
Nicola Prezza (Ca' Foscari University of Venice)  
Venkatesh Raman (The Institute of Mathematical Sciences, Chennai)  
Kunihiko Sadakane (co-chair, The University of Tokyo)  
Srinivasa-Rao Satti (Norwegian University of Science and Technology)  
Saket Saurabh (The Institute of Mathematical Sciences)  
Chan-Su Shin (Hankuk University of Foreign Studies)  
Wing-Kin Sung (National University of Singapore)  
Suguru Tamaki (University of Hyogo)  
Sébastien Tixeuil (Sorbonne Université)  
Haitao Wang (Utah State University)  
Prudence Wong (University of Liverpool)  
Jialin Zhang (China Academy of Sciences)





## ■ List of External Reviewers

Ahmed Abdelkader  
Peyman Afshani  
Jarno Alanko  
Sharareh Alipour  
Hyung-Chan An  
Ei Ando  
Haris Angelidakis  
Simon Apers  
Abdullah Arslan  
Yuichi Asahiro  
Tetsuo Asano  
Kyriakos Axiotis  
Sang Won Bae  
Sayan Bandyapadhyay  
Aritra Banik  
Max Bannach  
Jérémy Barbay  
Manu Basavaraju  
Sabyasachi Basu  
Xiaohui Bei  
Rémy Belmonte  
Adam Bene Watts  
Matthias Bentert  
Christoph Berkholz  
Nico Bertram  
Ivona Bezakova  
Anup Bhattacharya  
Sujoy Bhore  
Davide Bilò  
Ahmad Biniaz  
Arijit Bishnu  
Arindam Biswas  
Eric Blais  
Lelia Blin  
Greg Bodwin  
François Bonnet  
Silvia Bonomi  
Quentin Bramas  
Gerth Stølting Brodal  
Janna Burman  
Martin Böhm  
Sergio Cabello  
Pilar Cano  
Yixin Cao  
Paz Carmi  
Carl Johan Casselgren  
Timothy M. Chan  
Ching-Lueh Chang  
Hsien-Chih Chang  
Vincent Chau  
Ho-Lin Chen  
Po-An Chen  
Wei Chen  
Yijia Chen  
Ashish Chiplunkar  
Rajesh Chitnis  
Jongmin Choi  
Jaehoon Chung  
Andre Augusto Cire  
Alessio Conte  
Sabine Cornelsen  
Nicola Cotumaccio  
Mina Dalirrooyfard  
Gautam K Das  
Syamantak Das  
Alexis de Colnet  
Argyrios Deligkas  
Christian Deppe  
Palash Dey  
Patrick Dinklage  
Christoph Dürr  
Eduard Eiben  
Edith Elkind  
Jonas Ellert  
Christian Engels  
Massimo Equi  
Hiroshi Eto  
Yuri Faenza  
Matthew Fahrback  
Chenglin Fan  
Giovanni Farina  
Mohammad Farshi  
Qilong Feng  
Jeremy Fineman  
Till Fluschnik  
Pierre Fraigniaud  
Vincent Froese  
Hiroshi Fujiwara  
Jakub Gajarský  
Vijay Ganesh



Robert Ganian	Pallavi Jain
Jie Gao	Rahul Jain
Xiaofeng Gao	Ragesh Jaiswal
Naveen Garg	Wojciech Janczewski
Daya Gaur	Varunkumar Jayapaul
Pawel Gawrychowski	Shaofeng H.-C. Jiang
Tzvika Geft	Ce Jin
Anirban Ghosh	Kai Jin
Arijit Ghosh	Seungbum Jo
Daniel Gibney	Naonori Kakimura
Nicola Gigante	Hirotsugu Kakugawa
Anastasios Giovanidis	Shahin Kamali
Marc Glisse	Sayaka Kamei
Petr Golovach	Naoyuki Kamiyama
Garance Gourdel	Frank Kammer
Fernando Granha Jeronimo	Panagiotis Kanellopoulos
Qianping Gu	Matthew Katz
Manoj Gupta	Akitoshi Kawamura
Sushmita Gupta	Yasushi Kawase
Gregory Gutin	Shuji Kijima
Adrián Gómez Brandón	Hee-Chul Kim
Daniel Hader	Jae-Hoon Kim
Koki Hamada	Mincheol Kim
Kai Han	Yonghwan Kim
Tesshu Hanaka	Csaba Király
Sariel Har-Peled	Sándor Kisfaludi-Bak
Kun He	Naoki Kitamura
Meng He	Masashi Kiyomi
Qizheng He	Jonathan Klawitter
Klaus Heeger	Boris Klemz
Jacob Hendricks	Peter Kling
Magnus Lie Hetland	Yasuaki Kobayashi
Shuichi Hirahara	Frederic Koehler
Hiroshi Hirai	Sudeshna Kolay
Petr Hliněný	Spyros Kontogiannis
Michael Hoffmann	Matias Korman
Hsiao-Yu Hu	Artur Kraska
Ziyun Huang	Amit Kumar
Ling-Ju Hung	Noboru Kunihiro
Tomohiro I	Florian Kurpicz
David Ilcinkas	Damianos Kypriadis
Fotis Iliopoulos	Arnaud Labourel
Sungjin Im	Manuel Lafond
Tanmay Inamdar	Michael Lampis
Toshimasa Ishii	Alexandra Lassota
Yuni Iwamasa	Francois Le Gall
Taisuke Izumi	Seungjun Lee
Ashwin Jacob	Stefano Leucci

Bo Li	Eunjin Oh
Fei Li	Yoshio Okamoto
Jian Li	Karolina Okrasa
Lvzhou Li	Simon Omlor
Minming Li	Hirotaka Ono
Tongyang Li	Fukuhito Ooshita
Ya-Chun Liang	Katarzyna Paluch
Ching-Chi Lin	Fahad Panolan
Tien-Ching Lin	Charis Papadopoulos
Chih-Hung Liu	Pál András Papp
Fu-Hong Liu	Debasish Pattanayak
Hsiang-Hsuan Liu	Ramamohan Paturi
William Lochet	Will Perkins
Raul Lopes	Geevarghese Philip
Kelin Luo	Giulio Ermanno Pibiri
Maarten Löffler	Rameshwar Pratap
Ramanujan M. Sridharan	Simon Puglisi
Diptapriyo Majumdar	Li Qian
Yury Makarychev	Saladi Rahul
Yoshifumi Manabe	B. V. Raghavendra Rao
Partha Sarathi Mandal	Ivan Rapaport
Jan Marcinkowski	Gaurav Rattan
Andrea Marino	Alexander Ravsky
Fabien Mathieu	Giuseppe Romana
Akihiro Matsuura	Christian Rosenke
Michał Małafiejski	Sanjukta Roy
Andrew McGregor	Davide Rucci
Neeldhara Misra	Abhishek Sahu
Eiji Miyano	Toshiki Saitoh
Atsushi Miyauchi	Thatchaphol Saranurak
Shuichi Miyazaki	Rik Sarkar
Debajyoti Mondal	Jayalal Sarma
Fabrizio Montecchiani	Kevin Schewior
Naoyuki Morimoto	Jens Schlöter
Pat Morin	Christiane Schmidt
Miguel A. Mosteiro	Melanie Schmidt
Mitsuo Motoki	Pascal Schweitzer
Anish Mukherjee	Louisa Seelbach Benkner
Joydeep Mukherjee	Shinnosuke Seki
Veli Mäkinen	Aditi Sethia
Atsuki Nagao	Kazuhisa Seto
Junya Nakamura	Mordechai Shalom
Shin-ichi Nakayama	Xiaohan Shan
N.S. Narayanaswamy	Vikram Sharma
Yakov Nekrich	Masahiro Shibata
Mikhail Nesterenko	Nobutaka Shimizu
Prajakta Nimbhorkar	Sebastian Siebertz
Harumichi Nishimura	Kirill Simonov

**0:xviii External Reviewers**

Sahil Singla  
Friedrich Slivovsky  
Michiel Smid  
Roberto Solis-Oba  
Olivier Spanjaard  
Richard Spence  
Frits Spieksma  
Gawiejnowicz Stanislaw  
Milos Stojakovic  
Martin Suderland  
Yuichi Sudo  
Xiaoming Sun  
Vaishali Surianarayanan  
Akira Suzuki  
Yasuhiro Takahashi  
Asahi Takaoka  
Kenjiro Takazawa  
Eiji Takimoto  
Prafullkumar Tale  
Takeyuki Tamura  
Zihan Tan  
Zhihao Gavin Tang  
Alessandra Tappini  
Erin Taylor  
Jan Arne Telle  
Junichi Teruyama  
Takahisa Toda  
Vera Traub  
Amitabha Tripathi  
Meng-Tsung Tsai  
Konstantinos Tsakalidis  
Kei Uchizawa  
Jorge Urrutia  
Przemysław Uznański  
Tom van der Zanden  
André van Renssen  
Daniel Vaz  
Sebastiano Vigna  
Antoine Vigneron  
Alexandros Voudouris  
Magnus Wahlström  
Bartosz Walczak  
Tomasz Walen  
Matthias Walter  
Changjun Wang  
Kai Wang  
Yipu Wang  
Justin Ward  
Kunihiro Wasa  
Osamu Watanabe  
Hao-Ting Wei  
Andreas Wiese  
Sebastian Wild  
Xiaowei Wu  
Allen Xiao  
Dachuan Xu  
Jie Xue  
Yutaro Yamaguchi  
Masaki Yamamoto  
Katsuhisa Yamanaka  
Koichi Yamazaki  
Kenji Yasunaga  
Yitong Yin  
Yu Yokoi  
Sang Duk Yoon  
Yuichi Yoshida  
Sixie Yu  
Meirav Zehavi  
Chihao Zhang  
Guochuan Zhang  
Jingru Zhang  
Qiankun Zhang  
Yuhao Zhang  
Yiming Zhao  
Samson Zhou  
Michele Zito



# Streaming Pattern Matching

Tatiana Starikovskaya ✉

DIENS, École normale supérieure, PSL Research University, Paris, France

---

## Abstract

Many classical algorithms for string processing assume that the input can be accessed in full via constant-time random access, which poses a serious limitation in the modern era of data deluge. In this talk, we will focus on the streaming model of computation that allows to overcome this issue. In this model of computation, we assume that the input arrives as a stream, one character at a time, which captures a situation when the data are sequential measurements or an output of an algorithm. The space complexity is defined as all the space used, including the space used to store any information about the input, which allows to develop ultra-efficient algorithms.

The first streaming algorithm for pattern matching was presented in the seminal paper of Porat and Porat in FOCS 2009. For a pattern of length  $m$ , the algorithm uses only  $O(\log m)$  space, while any classical algorithm requires  $\Omega(m)$  space. This result served as a foundation of the area of streaming algorithms for pattern matching. After a brief survey of the area, we will discuss two questions in more details: the  $k$ -mismatch problem and the pattern matching with  $k$ -edits problem. In the  $k$ -mismatch problem, one is given a pattern and a text, and the task is to find all substrings of the text that have at most  $k$  mismatches with the pattern. The current best algorithm for this problem was given by Clifford, Kociumaka, and Porat in SODA 2019, and for a pattern of length  $m$  it uses  $O(k \log m)$  space and  $\tilde{O}(\sqrt{k})$  time per character of the text. In the pattern matching with  $k$ -edits problem, the task is similar, but one must find substrings that can be transformed into the pattern by at most  $k$  edits, i.e. substitutions, insertions, and deletions of a character. For this problem, the first streaming algorithm was presented by Kociumaka, Porat, and Starikovskaya in FOCS 2021. The algorithm takes  $\tilde{O}(\text{poly}(k))$  space and  $\tilde{O}(\text{poly}(k))$  time per character of the text.

**2012 ACM Subject Classification** Theory of computation → Pattern matching; Theory of computation → Sketching and sampling

**Keywords and phrases** Streaming algorithms, Pattern matching, Hamming distance, Edit distance

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.1

**Category** Invited Talk



© Tatiana Starikovskaya;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 1; pp. 1:1–1:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Spanning Properties of Variants of the Delaunay Graph

Prosenjit Bose ✉

Carleton University, Ottawa, Canada

---

## Abstract

A weighted geometric graph  $G$  is a graph whose  $n$  vertices are points in the plane and whose  $m$  edges are line segments weighted by the Euclidean distance between their endpoints. A  $t$ -spanner of  $G$  is a connected spanning subgraph  $G'$  with the property that for every pair of vertices  $x, y$ , the shortest path from  $x$  to  $y$  in  $G'$  has weight at most  $t \geq 1$  times the shortest path from  $x$  to  $y$  in  $G$ . The parameter  $t$  is commonly referred to as the spanning ratio or the stretch factor. Typically,  $G$  is a graph with  $\Omega(n^2)$  edges. As such, the goal in this area is to construct a subgraph  $G'$  that possesses several desirable properties such as  $O(n)$  edges and spanning ratio close to 1. In addition, when planarity is one of the desired properties, variants of Delaunay graphs play a vital role in the construction of planar geometric spanners. In this talk, we will provide a comprehensive overview of various results concerning the spanning ratio, among other several other properties, of different types of Delaunay graphs and their subgraphs.

**2012 ACM Subject Classification** Mathematics of computing → Graph theory; Theory of computation → Computational geometry

**Keywords and phrases** Delaunay Graph, Geometric Graph, Graph Spanner

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.2

**Category** Invited Talk



© Prosenjit Bose;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



# Subquadratic Algorithms for Some 3Sum-Hard Geometric Problems in the Algebraic Decision Tree Model

**Boris Aronov** ✉ 

Tandon School of Engineering, New York University, Brooklyn, NY, USA

**Mark de Berg** ✉ 

Eindhoven University of Technology, The Netherlands

**Jean Cardinal** ✉ 

Université libre de Bruxelles (ULB), Brussels, Belgium

**Esther Ezra** ✉ 

School of Computer Science, Bar Ilan University, Ramat Gan, Israel

**John Iacono** ✉ 

Université libre de Bruxelles (ULB), Brussels, Belgium

Tandon School of Engineering, New York University, Brooklyn, NY, USA

**Micha Sharir** ✉ 

School of Computer Science, Tel Aviv University, Israel

---

## Abstract

We present subquadratic algorithms in the algebraic decision-tree model for several 3SUM-hard geometric problems, all of which can be reduced to the following question: Given two sets  $A, B$ , each consisting of  $n$  pairwise disjoint segments in the plane, and a set  $C$  of  $n$  triangles in the plane, we want to count, for each triangle  $\Delta \in C$ , the number of intersection points between the segments of  $A$  and those of  $B$  that lie in  $\Delta$ . The problems considered in this paper have been studied by Chan (2020), who gave algorithms that solve them, in the standard real-RAM model, in  $O((n^2/\log^2 n) \log^{O(1)} \log n)$  time. We present solutions in the algebraic decision-tree model whose cost is  $O(n^{60/31+\varepsilon})$ , for any  $\varepsilon > 0$ .

Our approach is based on a primal-dual range searching mechanism, which exploits the multi-level polynomial partitioning machinery recently developed by Agarwal, Aronov, Ezra, and Zahl (2020).

A key step in the procedure is a variant of point location in arrangements, say of lines in the plane, which is based solely on the *order type* of the lines, a “handicap” that turns out to be beneficial for speeding up our algorithm.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** Computational geometry, Algebraic decision-tree model, Polynomial partitioning, Primal-dual range searching, Order types, Point location, Hierarchical partitions

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.3

**Related Version** *Full Version:* <https://arxiv.org/abs/2109.07587>

**Funding** *Boris Aronov:* Partially supported by NSF Grants CCF-15-40656 and CCF-20-08551, and by Grant 2014/170 from the US-Israel Binational Science Foundation.

*Mark de Berg:* Partially supported by the Dutch Research Council (NWO) through Gravitation Grant NETWORKS (project no. 024.002.003).

*Jean Cardinal:* Partially supported by the F.R.S.-FNRS (Fonds National de la Recherche Scientifique) under CDR Grant J.0146.18.

*Esther Ezra:* Partially supported by NSF CAREER under Grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation.

*John Iacono:* Partially supported by Fonds National de la Recherche Scientifique (FNRS) under Grant no. MISU F.6001.1.



© Boris Aronov, Mark de Berg, Jean Cardinal, Esther Ezra, John Iacono, and Micha Sharir; licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 3; pp. 3:1–3:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

*Micha Sharir*: Partially supported by ISF Grant 260/18, by Grant 1367/2016 from the German-Israeli Science Foundation (GIF), and by Blavatnik Research Fund in Computer Science at Tel Aviv University.

**Acknowledgements** We thank Zuzana Patáková for helpful discussions on multilevel polynomial partitioning.

## 1 Introduction

Let  $A$  and  $B$  be two sets, each consisting of  $n$  pairwise disjoint line segments in the plane, and let  $C$  be a set of  $n$  triangles in the plane. We study the problem of counting, for each triangle  $\Delta \in C$ , the number of intersection points between the segments of  $A$  and those of  $B$  that lie inside  $\Delta$ . We refer to this problem as *within-triangle intersection-counting*. This is one of four 3SUM-hard problems (among many others) studied by Chan [11], all of which can be reduced to the problem just mentioned.<sup>1</sup> The other three problems are:<sup>2</sup>

- (i) *Intersection of three polygons*. Given three simple  $n$ -gons  $A, B, C$  in the plane, determine whether  $A \cap B \cap C$  is nonempty.
- (ii) *Coverage by three polygons*. Given three simple  $n$ -gons  $A, B, C$  in the plane, determine whether  $A \cup B \cup C$  covers a given triangle  $\Delta_0$ .
- (iii) *Segment concurrency*. Given sets  $A, B, C$ , each consisting of  $n$  pairwise disjoint segments in the plane,<sup>3</sup> determine whether  $A \times B \times C$  contains a concurrent triple.

Chan [11] presents slightly subquadratic algorithms for all four problems, whose running time in the standard real-RAM model (also referred to as the *uniform* model) is  $O((n^2/\log^2 n) \log^{O(1)} n)$ . He has observed that, as already mentioned, all these problems can be reduced in near-linear time to the within-triangle intersection-counting problem, so it suffices to present an efficient subquadratic solution for that problem.

We study the within-triangle intersection-counting problem in the algebraic decision-tree model. In this model only sign tests of polynomial inequalities of constant degree that access explicitly (the endpoint coordinates of) the input segments or vertices of the input triangles count towards the running time. All other operations cost nothing in the model, but are not allowed to access the input segments explicitly. Although originally introduced for establishing lower bounds [7], the algebraic decision-tree model has become a standard model for upper bounds too, used in the study of many problems, including the 3SUM-problem itself [10, 17, 20, 23, 25] and various 3SUM-hard geometric problems [5, 6, 17]. One can interpret the decision-tree model as an attempt to isolate and minimize the cost of the part of the algorithm that explicitly accesses the real representation of the input objects, and ignore the cost of the other purely discrete steps. This has the potential of providing us with an insight about the problem complexity, which might eventually lead to an improved solution also in the uniform real-RAM model.

We show that the within-triangle intersection-counting problem and, hence, also problems (i)–(iii), can be solved in this model with  $O(n^{60/31+\varepsilon})$  sign tests, for any  $\varepsilon > 0$ . Chan [11] also remarks (without providing details) that his algorithm can be implemented in  $O(n^{2-\delta})$  time in the algebraic decision-tree model, for some  $\delta > 0$  that he left unspecified. (With some care, as was communicated to us, one can obtain  $\delta \approx 0.01$ .) Our algorithm is rather

<sup>1</sup> Chan [11] refers to this problem as “triangle intersection-counting.”

<sup>2</sup> The fact that these problems are 3SUM-hard, and the connections between them, are stated in [11].

<sup>3</sup> The segments of one set, say  $C$ , need not be pairwise disjoint. Although not explicitly stated, the technique in [11] for the uniform model can also handle this situation.

different from Chan's, and gives a concrete value for  $\delta$  (any positive  $\delta < 2/31$ ), as mentioned above. Our techniques appear to be of independent interest and to have the potential to apply to other problems, as we demonstrate in the full version [4, Section 4].

If the segments in  $A$  and  $B$  and the triangles in  $C$  were all full lines,<sup>4</sup> then, in general, determining the existence of a concurrent triple of lines in  $A \times B \times C$  (the so-called *concurrency testing* problem) is the dual version of the classical 3SUM-hard *collinearity testing* problem, in which we are given three sets of points in the plane, and wish to determine whether their Cartesian product contains a collinear triple. This problem has recently been studied in the algebraic decision-tree model by Aronov et al. [5], in a restricted version where two of the sets are assumed to lie on two constant-degree algebraic curves, where an algorithm with roughly  $O(n^{28/15})$  comparisons has been presented.

The problems studied here can be regarded as other dual versions of collinearity testing, where restrictions of a different kind are imposed. As noted by Chan [11], the additional disjointness properties that are assumed here make the problem simpler than collinearity testing (albeit by no means simple), and its solution appears to have no bearing on the unconstrained collinearity problem itself. In the full version [4, Section 5] we comment on the substantial differences between this work and the work by Aronov et al. [5].

Our technique is based on hierarchical cuttings of the plane, as well as on tools and properties of segment-intersection range searching. We also use the so-called Fredman's trick in algebraic-geometric settings, in which the problem is mapped into a primal-dual range searching mechanism involving points and surfaces in  $\mathbb{R}^6$ . This reduction exploits the very recent multi-level polynomial partitioning technique of Agarwal et al. [2] (or a similar technique of Matoušek and Patáková [27]). Our range-searching mechanism of points and algebraic surfaces in higher dimensions is a by-product of our analysis, which appears to be broadly applicable in other range-searching contexts, and we thus regard it as a technique of independent interest; see, for example, Proposition 2 and its proof.

**Point location in arrangements.** An additional key ingredient of our approach involves point location in an arrangement of lines in the plane (or an arrangement of curves, or of hyperplanes in higher dimensions). This is of course a well studied problem with several optimal solutions [29], but we adapt and use techniques that are handicapped by the requirement that each operation that examines the real parameters specifying the lines involves at most *three* input lines. In contrast, the persistent data structure of [29], for example, needs to sort the vertices of the arrangement from left to right, thus requiring comparisons of the  $x$ -coordinates of a pair of vertices, which are in general determined by the parameters of *four* input lines. The persistent data structure method has been used in [5, 11] for the study of other 3SUM-hard geometric problems. Here we replace this approach with one that uses solely the relative positions of triples of lines, the so-called *order type* of the arrangement. In this approach each comparison involves only *three* input lines, which, as we show, eventually leads to improved performance of the algorithm.

In standard settings, separating the order-type computation from the rest of the processing makes no sense. This is because obtaining the full order-type information for  $N$  lines already takes  $\Theta(N^2)$  time. This makes the approach based on the order type noncompetitive, as one can just do point location in the line arrangement, in the uniform model, with  $O(N^2)$  preprocessing. Nevertheless, in the applications considered in this paper (see Section 3 and

---

<sup>4</sup> Disjointness then of course cannot be assumed in general, although it might occur when the lines in each set are parallel, as in the dual version of the 3SUM-hard GEOMBASE problem [19].

the full version [4, Section 4]), the input lines have a special representation, which allows us to avoid an explicit construction of their order type and obtain this information implicitly in subquadratic time in the decision-tree model. The rest of the preprocessing, which takes quadratic time and storage in the uniform model, costs nothing in the decision-tree model.

The problem of determining whether and how the order type of an arrangement is sufficient to construct an efficient point-location data structure has, to the best of our knowledge, never been addressed explicitly. As we believe that this kind of “handicapped” point location will be useful for other applications (some of which are mentioned in the full version [4, Section 4]), we present it in some detail in Section 2 and in the full version [4, Section 2.2]. We also present extensions of this technique to arrangements of constant-degree algebraic curves in  $\mathbb{R}^2$ , and to arrangements of planes or hyperplanes in higher dimensions, which will be used in the applications presented in the full version [4, Section 4].

The algorithm for solving the within-triangle intersection-counting problem in the algebraic decision-tree model, and, consequently, also of the other three problems listed at the beginning of this section, is presented in Section 3. Additional applications of our technique are presented in the full version [4, Section 4]; they include: (i) counting intersections between two sets of pairwise disjoint circular arcs inside disks, and (ii) minimum distance problems between lines and two sets of points in the plane.

## 2 Order-type-based point location in arrangements

**Order types.** An arrangement of non-vertical lines in the plane (and, later, curves in the plane, or hyperplanes in higher dimension) can be described in the following combinatorial fashion. We use the notion of an *order type*, defined for a set  $L$  of lines as follows: Given any ordered triple of lines  $(\ell_1, \ell_2, \ell_3)$  from  $L$ , where both  $\ell_2$  and  $\ell_3$  intersect  $\ell_1$ , we record the left-to-right order of the intersections  $\ell_1 \cap \ell_2$  and  $\ell_1 \cap \ell_3$  along  $\ell_1$ ; note that the intersections might coincide. The totality of this information gives, for each line in  $L$ , the left-to-right order of its intersections with every other line it meets. Furthermore, we assume the existence of an “infinitely steep” line  $\ell_\infty$ , placed sufficiently far to the left, the order of whose intersections with the “normal” lines encodes the order of their slopes. This information is dual to the perhaps more familiar notion of an order type for a set of points in the plane (see, e.g., [21]). A higher-dimensional analog of this information involves recording the order in which a line that is the intersection of  $d - 1$  hyperplanes in  $\mathbb{R}^d$  meets the remaining hyperplanes that meet but do not contain it. We also assume a suitable analog of the “infinitely steep line,” recursively defined over the dimension.

Back in the plane, the permutations along each line of the intersection points with the other lines are called *local sequences* [22]. This view allows us to extend the definition of the order type to  $x$ -monotone curves, where each pair of curves is assumed to intersect at most  $s$  points, for some constant  $s$ . In this case the order type gives, for each curve  $\gamma$  in the collection, the labeled left-to-right sequence of intersection points with the other curves, where each intersection point is labeled by the triple  $(i, j, k)$ , where  $i$  and  $j$  are the indices of the two curves that form the intersection, and  $k$  indicates that it is the  $k$ th leftmost intersection point of the two curves. The order type also includes the vertical order of the curves at  $x = -\infty$ . See the full version [4, Section 2.2] for further details.

The significance of the order type is that (a) it only records information for  $(d + 1)$ -tuples of objects, and (b) it contains enough information that lets us construct the arrangement and preprocess it for fast point location, without having to access further the actual parameters that define the objects.



The problem we tackle now is the following: Given the order type of an arrangement, preprocess this information into a point location data structure. The preprocessing stage is not allowed to access the actual geometric description of the objects, such as the coefficients of the equations defining the lines or hyperplanes, but can only exploit the discrete data given by the order type. A query, in contrast, is allowed to examine the coefficients of the few objects that it encounters.

We present two solutions for this problem. First, we show that, for  $d$ -dimensional hyperplane arrangements, for any  $d \geq 2$ , the sampling method of Meiser [28] (see also [16]) can be implemented using only order-type information. Second, we show that for arrangements of  $x$ -monotone curves in the plane, a simple variant of the separating-chain method for point location [15, 26] can be implemented such that only order-type information is used during the preprocessing. We present only the first technique in this version of the paper, and delegate the second one to the full version [4, Section 2.2].

## 2.1 Sampling-based approach for hyperplane arrangements

Let  $H$  be a set of  $N$  non-vertical hyperplanes in  $\mathbb{R}^d$ , where  $d \geq 2$  is a fixed constant. We want to construct a point-location data structure for the arrangement  $\mathcal{A}(H)$  induced by  $H$ , where we are only given the order type of  $H$ . Essentially, we are given, for each intersection line formed by  $d - 1$  hyperplanes, the order of its intersections with the other hyperplanes. (Alternatively, we are given, for each simplex  $\sigma$  formed by  $d + 1$  of the hyperplanes, the  $x_1$ -order of the vertices of  $\sigma$ .) We only require  $H$  not to contain vertical hyperplanes. We do permit more than  $d$  hyperplanes to share a point, as well as other degeneracies. This is indeed a natural scenario for our applications including segment intersection counting and its related problems.

We briefly sketch the randomized method first proposed by Meiser [28] and analyzed in detail by Ezra et al. [16] (see also [10]), and show that the order-type information is sufficient to construct the data structure.

Before considering the point-location structure, we note that the order type suffices to construct a discrete representation of the arrangement  $\mathcal{A}(H)$ , in which each  $j$ -dimensional cell of  $\mathcal{A}(H)$ , for  $j = 1, \dots, d$ , stores the set of all  $(j - 1)$ -dimensional cells that form its boundary (and consequently of all cells, of all dimensions, on its boundary), with respective back pointers from each cell to all higher-dimensional cells that contain it on their boundary. This can be done, e.g., by the Folkman–Lawrence topological representation theorem for oriented matroids [18], which, roughly speaking, implies that, given the order type of  $H$ , one can construct a combinatorial representation for the arrangement  $\mathcal{A}(H)$ , consisting of all *sign conditions*. That is, each face  $f$  of  $\mathcal{A}(H)$  (of any dimension) is encoded by a sign vector  $\{-1, 0, +1\}^{|H|}$  representing the above/below/on relation of  $f$  with respect to each hyperplane in  $H$ ; see [9] for an inductive proof for the planar case, and its generalization to higher dimensions in [8]. Given this property, a naïve actual construction of the combinatorial representation of  $\mathcal{A}(H)$  is easy to derive, and is free of charge in the decision-tree model, once the order type of  $H$  is computed. When we perform a point-location query we report a pointer to the sign vector of the cell of  $\mathcal{A}(H)$  that contains the query point – see below.

**Preprocessing.** Given the arrangement  $\mathcal{A}(H)$  and a fixed  $\varepsilon > 0$ , we first construct a random sample  $S$  of  $O(\frac{d^2}{\varepsilon} \log \frac{d}{\varepsilon})$  hyperplanes of  $H$ ; the size of  $S$  does not depend on  $n$ . We then compute a *canonical* triangulation of the arrangement  $\mathcal{A}(S)$ . For each face of  $\mathcal{A}(S)$ , of any dimension at least 2, we use a fixed rule to designate a *reference vertex*  $p$  of this face. For example, we can take  $p$  to be the lexicographically smallest vertex of the face,

with each vertex represented by the lexicographically smallest  $d$ -tuple of the indices of the hyperplanes that contain it and whose intersection is a single point.<sup>5</sup> We then triangulate each face  $f$  of  $\mathcal{A}(S)$  by the *fan* obtained by adding the vertex  $p$  to each simplex in the triangulations of the lower-dimensional faces composing the boundary of  $f$  and not incident to  $p$ . Next, we construct the *conflict list*  $L(\Delta)$  for each simplex  $\Delta$  of the triangulation, of any dimension, defined as the set of hyperplanes of  $H$  that *cross*  $\Delta$ , i.e., intersect, but not fully contain, it.  $L(\Delta)$  can indeed be constructed using only the order type: Deciding whether a hyperplane  $h \in H$  belongs to  $L(\Delta)$  amounts to testing whether there exist two vertices of  $\Delta$  that lie on different sides of  $h$ , and each such test is an orientation test of the corresponding  $(d + 1)$ -tuple of hyperplanes:  $h$  and the  $d$  hyperplanes forming the vertex.

From standard results on  $\varepsilon$ -nets [24], a suitable choice of the constant of proportionality in the bound on the sample size guarantees that, with high probability, the conflict list size is not larger than  $\varepsilon n$ , for each simplex  $\Delta$ . It remains to recurse, for each simplex  $\Delta$  of the triangulation, with the hyperplanes in  $L(\Delta)$ . (If  $\Delta$  is not full-dimensional, we also record the hyperplanes containing it and the recursive processing involves building a lower-dimensional arrangement within  $\Delta$ .) This leads to a hierarchical data structure in which the number of hyperplanes decreases by a factor  $\varepsilon$  at each level. The construction continues until the number of hyperplanes falls below a suitable constant, at which point we simply store the remaining hyperplanes. Let  $w$  be a leaf in this hierarchy. It will be convenient to further preprocess the set  $H(w)$  of hyperplanes stored at  $w$  into a tree  $\mathcal{T}_w$  that allows us to locate a query point in the arrangement  $\mathcal{A}(H(w))$ . The structure  $\mathcal{T}_w$  is simply a ternary tree of depth  $|H(w)| = O(1)$ , where a node at level  $j$  stores the  $j$ -th hyperplane  $h_j$  of  $H(w)$ , so we can test if a query point is below, on, or above  $h_j$ . Observe that each leaf of  $\mathcal{T}_w$  corresponds to a unique cell in the arrangement  $\mathcal{A}(H(w))$  and, hence, also in  $\mathcal{A}(H)$  – indeed, the sign with respect to every hyperplane in  $H \setminus H(w)$  is determined by the search path to the node  $w$  in the hierarchy.

**Answering queries.** Queries are answered as follows. First, we locate the (open) simplex  $\Delta$  of the canonical triangulation of  $\mathcal{A}(S)$  containing the query point  $q$ . Since  $d$  is assumed to be constant,  $S$  is also of constant size, and so locating  $\Delta$  can be done in  $O(1)$  time. Next, we recurse in the data structure attached to  $\Delta$ . When we reach a leaf  $w$  of the hierarchy, we continue to search in the tree  $\mathcal{T}_w$ . When we reach a leaf in  $\mathcal{T}_w$  we have located  $q$  and can report (a pointer to) the sign vector of the cell containing  $q$ .

The overall number of these recursive steps is  $O(\log n)$ , and thus answering a query costs  $O(\log n)$  arithmetic operations, where the hidden constant<sup>6</sup> is polynomial in  $d$ . As noted, in our applications we only need to determine whether  $q$  lies on a hyperplane of  $H$ .

The following lemma summarizes the result.

► **Lemma 1.** *Let  $H$  be a set of  $n$  hyperplanes in  $\mathbb{R}^d$ , where  $d \geq 2$  is a constant. Using only the order type of  $H$ , we can construct a polynomial-size data structure that guarantees  $O(\log n)$ -time point-location queries in the arrangement  $\mathcal{A}(H)$ ; the implied constant depends polynomially on  $d$ . The (polynomial) preprocessing time and the storage of the data structure cost nothing in the decision-tree model.*

<sup>5</sup> When  $p$  is chosen as the bottommost vertex, the resulting triangulation is referred to as the *bottom-vertex triangulation*, but in general the order type does not provide us with this information.

<sup>6</sup> The value of this constant depends on the storage allocated to the structure. For example, spending  $n^{2d \log d + O(d)}$  on storage guarantees query cost of  $O(d^4 \log n)$  [16].

**Remark.** The query time for  $d = 2$  is better than the time in the second, level-based approach presented in the full version [4, Section 2.2] for curves in the plane (which is  $O(\log^2 n)$ ). However, the sampling-based method does not extend to non-straight curves, since there is no obvious way to extend the notion of a canonical triangulation to the case of curves. The only viable way of doing this seems to use the standard vertical-decomposition technique. Unfortunately (for us), constructing the vertical decomposition requires that we compare the  $x$ -coordinates of vertices defined by different, unrelated pairs of curves. Such a comparison involves *four* input curves and it cannot be resolved from the order-type information alone. For lines in the plane, however, the above technique does yield the improved logarithmic query time.

### 3 The algorithm for within-triangle intersection-counting

Our input consists of two sets  $A, B$ , each of  $n$  pairwise disjoint segments in the plane, and of a set  $C$  of  $n$  triangles in the plane. To simplify the presentation, we assume that the input is in general position, namely that, among the segments of  $A, B$ , and edges of triangles of  $C$ , no two share a supporting line, and no endpoint of one segment lies on another (with the obvious exception of the vertices of a triangle in  $C$ ).<sup>7</sup> These are the only general position assumptions that we need. A triple of segments (one from  $A$ , one from  $B$ , and an edge of a triangle from  $C$ ) are allowed to be concurrent.

**A high-level roadmap of the algorithm.** To avoid various technical issues that complicate the description of our algorithm, we focus in this overview on the simpler segment concurrency problem, where  $C$  is a set of (not necessarily disjoint) segments, and the goal is to determine whether there is a triple  $(a, b, c) \in A \times B \times C$  of concurrent segments. To make the overview even simpler, assume that  $C$  is a set of lines.

We fix a parameter  $g \ll n$  and put  $r := n/g$ . We construct a  $(1/r)$ -cutting  $\Xi(A)$  for the segments of  $A$ , and another such cutting  $\Xi(B)$  for the segments of  $B$ . Since the segments of  $A$  are pairwise disjoint, we can construct  $\Xi(A)$  of size  $O(r)$ , and similarly for  $\Xi(B)$  (see [14]). We overlay the two cuttings and obtain a planar decomposition  $\Xi$ . While the complexity of  $\Xi$  is  $O(r^2)$ , any line of  $C$  crosses only  $O(r)$  of its cells.

For each two-dimensional cell  $\sigma$  of  $\Xi$  (lower-dimensional cells are simpler to handle), we preprocess the sets  $A_\sigma \subseteq A$  and  $B_\sigma \subseteq B$  of those segments that cross  $\sigma$ , each of size at most  $n/r = g$ , into a data structure that supports efficient queries, each specifying a line  $c$  and asking whether  $c$  passes through an intersection point of a segment of  $A_\sigma$  and a segment of  $B_\sigma$ . We pass to the dual plane, obtain sets  $A_\sigma^*$  and  $B_\sigma^*$  of at most  $g$  points (dual to the lines containing the segments) each. (We ignore here “short” segments that have an endpoint inside  $\sigma$ ; see below.) The query is a point  $c^*$  and the task is to determine whether  $c^*$  is collinear with a pair of points  $(a^*, b^*) \in A_\sigma^* \times B_\sigma^*$ . For  $a \in A_\sigma$  and  $b \in B_\sigma$  we define  $\gamma_{a,b}$  to be the line that passes through  $a^*$  and  $b^*$ , and let  $\Gamma_\sigma$  denote the collection of these lines. The query with  $c^*$  then reduces to point location in the arrangement  $\mathcal{A}(\Gamma_\sigma)$ , where we only need to know whether  $c^*$  lies on any of the lines.

We cannot perform this task explicitly in an efficient manner, since the complexity of  $\mathcal{A}(\Gamma_\sigma)$  is  $O(g^2)$  and we have  $O(r^2) = O(n^2/g^2)$  such arrangements, of overall size  $O(n^2)$ . We can do it, though, in the algebraic decision-tree model, in an implicit manner, using the so-called *Fredman’s trick*; see [23] for a simpler yet representative application of Fredman’s

<sup>7</sup> For technical reasons, we also allow a triangle in  $C$  to degenerate to a segment.

trick, as well as [5, 6] for geometric applications of Fredman’s trick. Concretely, we apply the order-type-based machinery of Section 2 to construct  $\mathcal{A}(\Gamma_\sigma)$  and preprocess it for fast point location. More precisely, we first construct the order type of  $\Gamma_\sigma$ : this involves, for each triple of lines  $\gamma_{a_1, b_1}, \gamma_{a_2, b_2}, \gamma_{a_3, b_3}$ , determining the ordering of their intersection points along each of these lines. We express this test, in a straightforward manner, as the sign test of some 12-variate constant-degree polynomial  $G(a_1, a_2, a_3; b_1, b_2, b_3)$ .

We map the triple  $(b_1, b_2, b_3)$  to a point in a six-dimensional parametric space, and  $(a_1, a_2, a_3)$  to an algebraic surface  $\psi_{a_1, a_2, a_3}$  in this space, which is the locus of all triples  $(b_1, b_2, b_3)$  with  $G(a_1, a_2, a_3; b_1, b_2, b_3) = 0$ . We now need to locate the points  $(b_1, b_2, b_3)$  in the arrangement of the surfaces  $\psi_{a_1, a_2, a_3}$ , from which all the sign tests can be resolved, at no extra cost in the algebraic decision-tree model, thereby yielding the desired order type. The subsequent construction of the arrangement  $\mathcal{A}(\Gamma_\sigma)$ , and its preprocessing for fast point location, using the machinery in Section 2, also cost nothing in our model.

To make this process efficient, we group together all the points  $(b_1, b_2, b_3)$ , for  $b_1, b_2, b_3$  in the same cell  $\sigma$ , over all cells  $\sigma$ , into one global set  $P$ , and group the surfaces  $\psi_{a_1, a_2, a_3}$  into another global set  $\Psi$ . We have  $|P|, |\Psi| = O(r) \cdot O(g^3) = O(ng^2)$  (since there are only  $O(r)$  cells of  $\Xi(A)$  (resp., of  $\Xi(B)$ ) from which the triples  $(a_1, a_2, a_3)$  (resp.  $(b_1, b_2, b_3)$ ) are drawn).

Using the recent machinery of Agarwal et al. [2] or of Matoušek and Patáková [27], we can perform this batched point location in 6-space in time

$$O\left(|P|^{6/7+\varepsilon} |\Psi|^{6/7+\varepsilon} + |P|^{1+\varepsilon} + |\Psi|^{1+\varepsilon}\right) = O\left((ng^2)^{12/7+2\varepsilon}\right),$$

for any  $\varepsilon > 0$ . Full details of this step, crucial, albeit rather technical, are given in the full version [4, Section 3.1].

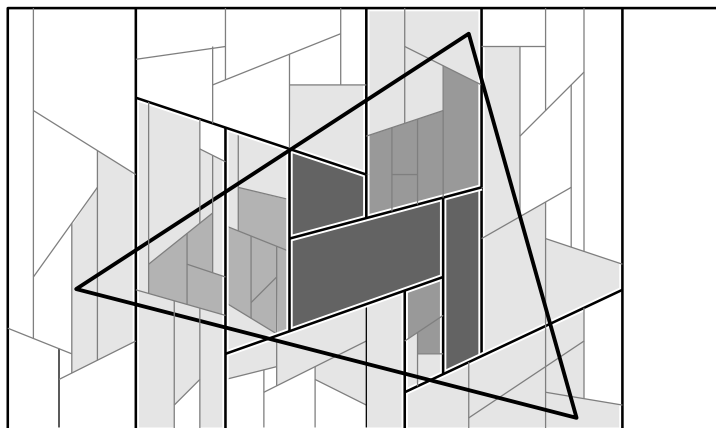
Searching with the dual points  $c^*$  takes  $O\left(\frac{n^2}{g} \log g\right)$  time, because we have  $n$  query lines  $c$ , each line crosses  $O(r) = O(n/g)$  cells  $\sigma$ , and each point location with  $c^*$  in each of the encountered arrangements takes  $O(\log g)$  time. Balancing (roughly) this cost with the preprocessing cost, we choose  $g = n^{2/31}$ , and obtain the total subquadratic running time  $O(n^{2-2/31+\varepsilon}) = O(n^{60/31+\varepsilon})$ .

Quite a few issues were glossed over in this overview. Since the segments of  $A$  and of  $B$  are bounded, a cell  $\sigma$  may contain endpoints of these segments, making the passage to the dual plane more involved. The same applies in the original within-triangle intersection-counting problem, where the triangles of  $C$  may have vertices or more than one bounding edge that lie in or meet  $\sigma$ . We thus need to handle the presence of such “short” segments and/or “short” triangles. Moreover, we need to count intersection points within each triangle, and the number of cells in overlay of the cuttings  $\Xi(A), \Xi(B)$  that a triangle can fully contain is much larger than  $O(r)$ . All these issues require more involved techniques, which are developed below, with some details delegated to the full version [4, Section 3]. Still, the overall runtime of the resulting algorithm remains  $O(n^{60/31+\varepsilon})$ , for any  $\varepsilon > 0$ .

**Hierarchical cuttings.** This ingredient is needed for counting intersection points in cells that are fully contained inside a query triangle. The application of hierarchical cuttings to our problem significantly reduces the query time – see below. Fix a parameter  $g \ll n$  and put  $r := n/g$ . We construct a *hierarchical*  $(1/r)$ -cutting  $\Xi(A)$  for the segments of  $A$ , which is a hierarchy of  $(1/r_0)$ -cuttings, where  $r_0$  is some sufficiently large constant. The top-level cutting  $\Xi_1(A)$  is constructed for  $A$ . Since the segments of  $A$  are pairwise disjoint, we can construct  $\Xi_1(A)$  so that it consists of only  $O(r_0)$  trapezoids (for concreteness, we write this bound as  $br_0$ , for some absolute constant  $b$ ), each of which is crossed by at most

$n/r_0$  segments of  $A$ , which comprise the so-called *conflict list* of the cell  $\sigma$ , denoted as  $A_\sigma$ . The construction time of  $\Xi_1(A)$ , in the real-RAM model, is  $O(n \log r_0) = O(n)$ . See [14, Theorem 1] for details.

For each cell  $\sigma$  of  $\Xi_1(A)$ , we clip the segments in its conflict list  $A_\sigma$  to within  $\sigma$  and apply the cutting-construction step recursively to this set, clipping also the cells of the new cutting to within  $\sigma$  (and ignoring cells, or portions thereof, that lie outside  $\sigma$ , as they are not met by any of the clipped segments of  $A_\sigma$ ). We denote the union of all the resulting  $(1/r_0)$ -cuttings as  $\Xi_2(A)$ . We continue recursively in this manner, until we reach a level  $s$  at which all the cells are crossed by at most  $n/r$  segments. We thus obtain a hierarchy of cuttings  $\Xi_1(A), \Xi_2(A), \dots, \Xi_s(A)$ , for some index  $s = O(\log r)$ . We denote the collective hierarchy as  $\Xi(A)$ . Since we stop the recursion as soon as  $n/r_0^s \leq n/r$ , the overall number of cells of all the levels is  $O((br_0)^s) = O(r^{1+\varepsilon})$ , for any prespecified  $\varepsilon > 0$ , for a suitable choice of  $r_0 = r_0(\varepsilon)$ . Technically, the trapezoids in the cutting are relatively open, and the cutting also includes one- and zero-dimensional cells; as the latter are easier to deal with, we will focus below on the two-dimensional cells of the cutting. At any level  $j$  of the hierarchy, the cells of  $\Xi_j(A)$  are pairwise disjoint. As these cells partition the plane, each intersection point between a segment of  $A$  and a segment of  $B$  lies in precisely one cell of a suitable dimension at each level. See Figure 1 for an illustration.



■ **Figure 1** Interaction of a hierarchical cutting with a triangle. The dark gray cells are the ones inside the triangle at the top level of the hierarchy; the medium gray cells are the ones inside the triangle at the second level (and whose parent cells are not inside the triangle). The light gray cells will be refined and handled at lower levels, since they intersect the triangle boundary.

We apply a similar hierarchical construction for  $B$ , and let  $\Xi(B) = \{\Xi_j(B)\}_{j \leq s}$  denote the resulting hierarchical cutting, which has analogous properties. (We assume for simplicity that the highest index  $s$  is the same in both hierarchies.)

We now overlay  $\Xi(A)$  with  $\Xi(B)$ , that is, at each level  $j$  of the hierarchy, we overlay the cells of  $\Xi_j(A)$  with the cells of  $\Xi_j(B)$ . We denote the  $j$ th level overlay as  $\Xi_j$ , and the entire hierarchical overlay structure as  $\Xi = \{\Xi_j\}_{j \leq s}$ . Since each of  $\Xi_j(A)$  and  $\Xi_j(B)$  consists of at most  $(br_0)^j$  cells, the number of cells of  $\Xi_j$  is at most  $O((br_0)^{2j})$ . Since we have  $r_0^s \approx r$  (up to a factor of  $r_0$ ), it follows that the overall complexity of all the overlays is  $O(r^{2+2\varepsilon})$ , provided that we choose  $r_0$ , as above, to be sufficiently large, as a function of  $\varepsilon$ .

For simplicity of exposition, we ignore lower-dimensional faces of the cuttings, and regard each of the overlays  $\Xi_j$  as a decomposition of the plane into pairwise openly disjoint convex polygons, each of complexity linear in  $j \leq s = O(\log r)$ . Each cell  $\sigma$  of the overlay is

identified by the pair  $(\tau, \tau')$ , where  $\tau$  and  $\tau'$  are the respective cells of  $\Xi_j(A)$  and  $\Xi_j(B)$  whose intersection is  $\sigma$ ; we simply write  $\sigma = (\tau, \tau')$ . Each bottom-level cell  $\sigma$  of the final overlay  $\Xi_s$  is crossed by at most  $n/r = g$  segments of  $A$  and by at most  $g$  segments of  $B$ .

**Classifying the segments and triangles.** Let  $\sigma = (\tau, \tau')$  be a cell of  $\Xi_j$ , for any level  $j$  of the hierarchy. Call a segment  $e$  of  $A$  *long* (resp., *short*) *within*  $\sigma$  if  $e$  crosses  $\sigma$  and neither of its endpoints lies in  $\sigma$  (resp., at least one endpoint lies in  $\sigma$ ). Let  $A_\sigma^l$  (resp.,  $A_\sigma^s$ ) denote the set of long (resp., short) segments of  $A$  within  $\sigma$ . Apply analogous definitions and notations to the segments of  $B$ . Denote by  $C_\sigma$  (resp.,  $C_\sigma^{(0)}$ ) the set of triangles with at least one edge that crosses  $\sigma$  (resp., that fully contain  $\sigma$ ). Call a triangle  $\Delta \in C_\sigma$  *long* (resp., *short*) in  $\sigma$  if  $\sigma$  does not (resp., does) contain a vertex of  $\Delta$ , and denote by  $C_\sigma^l$  (resp.,  $C_\sigma^s$ ) the set of long (resp., short) triangles in  $C_\sigma$ .

For each triangle  $\Delta \in C$ , each of its edges crosses only  $O((br_0)^j)$  cells of  $\Xi_j$ . Indeed, as such an edge crosses from one cell of  $\Xi_j$  to an adjacent cell, it does so by crossing the boundary of either a cell of  $\Xi_j(A)$  or a cell of  $\Xi_j(B)$ , and the total number of such crossings is  $O((br_0)^j)$ . In particular, the edge crosses at most  $O(r^{1+\varepsilon})$  cells of the final overlay  $\Xi_s$ . It follows that  $\sum_{\sigma \in \Xi} |C_\sigma^l| \leq \sum_{\sigma \in \Xi} |C_\sigma| = O(nr^{1+\varepsilon})$ , but clearly  $\sum_{\sigma \in \Xi} |C_\sigma^s|$  is only  $O(n \log r)$ . In contrast,  $\Delta$  can fully contain many more cells of  $\Xi_s$ , perhaps almost all of them, but the hierarchical nature of the construction allows us to deal with a much smaller number of such interior cells, by collecting them at higher levels of the hierarchy; see below for details.

**The algorithm: A quick review.** The high-level structure of the algorithm is as follows (see also the “roadmap” overview given earlier). We construct the hierarchies  $\Xi(A) = \{\Xi_j(A)\}_{j \geq 1}$  and  $\Xi(B) = \{\Xi_j(B)\}_{j \geq 1}$ . For each cell  $\tau$  of  $\Xi_j(A)$  (resp.,  $\tau'$  of  $\Xi_j(B)$ ), we compute its conflict list  $A_\tau$  (resp.,  $B_{\tau'}$ ), which, as we recall, is the set of all segments of  $A$  that cross  $\tau$  (resp., segments of  $B$  that cross  $\tau'$ ). We then form the hierarchical overlay  $\Xi = \{\Xi_j\}_{j \geq 1}$ , and for each cell  $\sigma = (\tau, \tau')$  of any overlay  $\Xi_j$ , we compute the subset  $A_\sigma$  of the segments of  $A_\tau$  that cross  $\sigma$ , and the subset  $B_\sigma$  of the segments of  $B_{\tau'}$  that cross  $\sigma$ . We partition  $A_\sigma$  into the subsets  $A_\sigma^l$  and  $A_\sigma^s$  of long and short segments (within  $\sigma$ ), respectively, and apply an analogous partition to  $B_\sigma$ . The additional overall cost for constructing these sets, over all hierarchical levels, is  $O(r^{2+\varepsilon} \cdot n/r) = O(nr^{1+\varepsilon}) = O(n^{2+\varepsilon}/g)$ . (The cost at the bottom level dominates the entire cost over all levels.)

We also trace each triangle  $c \in C$  through the cells of  $\Xi$  that are crossed by its edges, and form, for each cell  $\sigma$  of the overlay, the list  $C_\sigma$  of triangles of  $C$  with at least one edge that crosses  $\sigma$ . We partition  $C_\sigma$  into the subsets  $C_\sigma^l$  and  $C_\sigma^s$ , as defined earlier. As we show below, we can handle, in a much more efficient way, the short triangles of  $C_\sigma^s$ , as well as the triangles of  $C_\sigma^l$  all three of whose edges cross  $\sigma$ , simply because the overall number of such triangle-cell interactions is small. We therefore focus on the triangles of  $C_\sigma^l$  that have only one or two edges crossing  $\sigma$ . For triangles with two crossing edges we use a standard two-level data structure (where in each level we consider only one crossing edge). This lets us assume, without loss of generality, that each triangle in  $C_\sigma^l$  is a halfplane. Each of these halfplanes can be represented by its bounding line, that is the line supporting the appropriate crossing edge of the triangle. We flesh out the details below.

We also assume, for now, that all the segments of  $A_\sigma$  and of  $B_\sigma$  are long in  $\sigma$  (and so we drop the superscript  $l$ ). This is the hard part of the analysis, requiring the involved machinery presented below. After handling this case, we will address the much simpler situations that involve short segments and/or short triangles (or triangles with three edges crossing  $\sigma$ ). The cost of handling short segments or short triangles within cells is lower, even in the uniform model, since the overall number of short objects within cells is smaller.

**Handling the long segments.** We preprocess each level  $j$  of the overlay, to compute, for each of its cells  $\sigma = (\tau, \tau')$ , the number of intersection points between the (long) segments of  $A_\sigma$  and those of  $B_\sigma$  (which, due to the clipping, lie in  $\sigma$ ). This is a standard procedure that involves computing the number of pairs of segments from  $A_\sigma \times B_\sigma$  whose intersection points with the boundary of  $\sigma$  interleave (these are precisely the pairs of intersecting segments), and can be implemented to take  $O((|A_\sigma| + |B_\sigma|) \log(|A_\sigma| + |B_\sigma|))$  time; see, e.g., [1]. We store the resulting count at  $\sigma$ .

Consider a two-dimensional cell  $\sigma$ , a segment  $a \in A_\sigma$ , a segment  $b \in B_\sigma$ , and a triangle  $\Delta \in C_\sigma$ . By assumption,  $\Delta$  has only one edge  $c$  or two edges  $c_1, c_2$  crossing  $\sigma$ . When  $a$  and  $b$  intersect inside  $\sigma$ , the intersection lies in  $\Delta$  if and only if the triple  $(a, b, c)$ , or each of the triples  $(a, b, c_1)$ ,  $(a, b, c_2)$ , has a prescribed orientation, reflecting the condition that the point  $a \cap b$  lies on the side of  $c$  (or the sides of  $c_1, c_2$ ) that contain  $\Delta$ . This orientation (or orientations) can be positive, negative, or zero, depending on the relative order of the slopes of  $a, b$ , and  $c$  (or of  $c_1$  and  $c_2$ ), and on whether  $\Delta$  lies to the left or to the right of  $c$  (or of  $c_1, c_2$ ).

For each halfplane  $c^+$  that represents a triangle  $\Delta \in C_\sigma$  (the halfplane contains  $\Delta$  and is bounded by the line supporting the single (relevant) edge  $c$  of  $\Delta$  that crosses  $\sigma$ ), we want either (i) to represent the set of pairs  $(a, b) \in A_\sigma \times B_\sigma$  that have a prescribed orientation of the triple  $(a, b, c)$ , as the disjoint union of complete bipartite graphs, or (ii) to count the number of such pairs. The subtask (i) arises in cases where  $\Delta$  has two edges crossing  $\sigma$  and is needed for the first level of the data structure, which we query with the first crossing edge of  $\Delta$ . The subtask (ii) arises in the second level of the structure, which we query with the second crossing edge of  $\Delta$ , and in cases where only one edge of  $\Delta$  crosses  $\sigma$ .

We also count the number of intersections within  $\sigma$ , in  $O((|A_\sigma| + |B_\sigma|) \log(|A_\sigma| + |B_\sigma|))$  time. As a matter of fact, with a simple modification of the procedure, we can, within the same time bound, represent the set of all pairs of segments  $(a, b) \in A_\sigma \times B_\sigma$  that intersect each other (inside  $\sigma$ ) as the disjoint union of complete bipartite graphs, so that the overall size of their vertex sets is  $O((|A_\sigma| + |B_\sigma|) \log(|A_\sigma| + |B_\sigma|))$ . This follows from standard planar segment-intersection range searching machinery; see, e.g., [1]. In what follows we focus on just one such graph, and to simplify the presentation we denote it as  $A_\sigma \times B_\sigma$ , with a slight abuse of notation.

**Preparing for Fredman's trick.** We use the infrastructure developed by Aronov et al. [5], with suitable modifications, but adapt it to the order-type context. We preprocess  $A$  and  $B$  into a data structure that we will then search with the points dual to the lines supporting the edges of the triangles of  $C$ . For each  $a \in A, b \in B$ , we define  $\gamma_{a,b}$  to be the line that passes through  $a^*$  and  $b^*$ , where  $a^*$  (resp.,  $b^*$ ) is the point dual to  $a$  (resp.,  $b$ ). By our general position assumption,  $a^* \neq b^*$ , so  $\gamma_{a,b}$  is well defined. Let  $\Gamma_0$  denote the set of these  $n^2$  lines. Our goal in task (ii) is to count, for each cell  $\sigma$  of any of the overlays, for each point  $c^*$  dual to an edge of a triangle  $\Delta \in C_\sigma$ , the number of lines of  $\Gamma_0$  that lie above  $c^*$ , the number of lines that are incident to  $c^*$ , and the number of lines that lie below  $c^*$ . In task (i), we want to represent each of these sets of lines as the disjoint union of a small number of precomputed canonical sets. This calls for preprocessing the arrangement  $\mathcal{A}(\Gamma_0)$  into a suitable point location data structure, which we will then search with each  $c^* \in C^*$ , and retrieve the desired data from the outcome of each query.

As in, e.g., [5], a naïve implementation of this approach will be too expensive. Instead, we return to the hierarchical partitions  $\Xi(A), \Xi(B)$ , and  $\Xi$ , and iterate, over all cells  $\sigma = (\tau, \tau')$  of the bottom level  $\Xi_s$ , defining  $\Gamma_\sigma := \{\gamma_{a,b} \mid (a, b) \in A_\sigma \times B_\sigma\}$ . In principle, we want to



construct the separate arrangements  $\mathcal{A}(\Gamma_\sigma)$ , over the cells  $\sigma$ , preprocess each of them into a point location data structure, and search, for each triangle  $\Delta \in C$ , in the structures that correspond to the cells of  $\Xi$  that are either crossed by (at most) one or two edges of  $\Delta$ , or fully contained in  $\Delta$ . This is also too expensive if implemented naïvely, so we use instead Fredman’s trick, combined with the machinery developed in Section 2.

We first observe that, for each triangle  $\Delta \in C$ , finding the cells  $\sigma$  (at any level of the hierarchy) that  $\Delta$  fully contains is easy and inexpensive. We go over the hierarchy of the overlays  $\Xi_j$ . At the root we find, by brute force, all the (constantly many) cells of  $\Xi_1$  that  $\Delta$  fully contains, and add their intersection counts to our output counter. We then recurse, in the same manner, in the at most  $br_0$  cells of  $\Xi_1$  that  $\Delta$  crosses. Thus the number of cells we visit is at most  $O(r_0^2) \cdot (1 + br_0 + (br_0)^2 + \dots + (br_0)^s) = O(r^{1+\varepsilon})$ , so the overall cost of this step<sup>8</sup> is  $O(nr^{1+\varepsilon}) = O(n^{2+\varepsilon}/g)$ .

We therefore focus, for each triangle  $\Delta$  of  $C$ , only on the cells that it crosses (at every level of the hierarchy), and restrict the analysis for now to cells at which  $\Delta$  is long, with at most two of its edges crossing the cell. Repeating most of the analysis just given, the number of these cells is  $O(r^{1+\varepsilon})$  (with a smaller constant of proportionality, since we now do not have the factor  $O(r_0^2)$ , as above).

**Constructing  $\mathcal{A}(\Gamma_\sigma)$  in the decision-tree model.** Consider the step of constructing  $\mathcal{A}(\Gamma_\sigma)$  for some fixed bottom-level cell  $\sigma$ . Following the technique in Section 2, we perform this step using only the order type of  $\Gamma_\sigma$ , and we begin by considering the task of obtaining the order-type information. That is, we want to determine, for each ordered triple  $(\gamma_{a_1,b_1}, \gamma_{a_2,b_2}, \gamma_{a_3,b_3})$  of lines of  $\Gamma_\sigma$ , whether the point  $\gamma_{a_1,b_1} \cap \gamma_{a_2,b_2}$  lies to the left or to the right of the point  $\gamma_{a_1,b_1} \cap \gamma_{a_3,b_3}$ . Let  $G(a_1, a_2, a_3; b_1, b_2, b_3)$  denote the 12-variate polynomial (of constant degree) whose sign determines the outcome of the above comparison. (The immediate expression for  $G$  is a rational function, which we turn into a polynomial by multiplying it by the square of its denominator, without affecting its sign; our general position assumption ensures that none of the denominators vanishes.)

Once the signs of all expressions  $G(a_1, a_2, a_3; b_1, b_2, b_3)$  are determined, we can apply Lemma 1. The rest of the preprocessing, which constructs a discrete representation of the arrangement, say, in the DCEL format [13], and turns this representation into an efficient point location data structure, can be carried out at no cost in the algebraic decision-tree model.

We search the structure with each triangle  $\Delta \in C_\sigma$ . We may assume that  $\Delta$  is long in  $\sigma$  and that only one or two edges of  $\Delta$  cross  $\sigma$ , as the other cases are easy to handle. Assuming further that there is only one such edge  $c$ , locating the dual point  $c^*$  in  $\mathcal{A}(\Gamma_\sigma)$  takes  $O(\log g)$  time, as shown in Section 2 (noting that  $\Gamma_\sigma$  consists of only  $g^2$  lines). With suitable preprocessing, locating  $c^*$  gives us, for free in our model, the three sets of the lines that pass above  $c^*$ , are incident to  $c^*$ , or pass below  $c^*$ . The case where two edges of  $\Delta$  cross  $\sigma$  is handled using a two-level version of the structure; see below for details. The point location cost now goes up to  $O(\log^2 g)$ .

Consider then the step of computing the order type of the lines of  $\Gamma_\sigma$ , that is, of computing the sign of  $G(a_1, a_2, a_3; b_1, b_2, b_3)$ , for every triple of segments  $a_1, a_2, a_3 \in A_\sigma$  and every triple of segments  $b_1, b_2, b_3 \in B_\sigma$ . To this end, we play Fredman’s trick. We fix a bottom-level cell  $\tau$  of  $\Xi(A)$ . For each triple  $(a_1, a_2, a_3) \in A_\tau^3$ , we define the surface

$$\psi_{a_1, a_2, a_3} = \{(b_1, b_2, b_3) \in \mathbb{R}^6 \mid G(a_1, a_2, a_3; b_1, b_2, b_3) = 0\},$$

<sup>8</sup> It is for making this step efficient that we use hierarchical partitions. A single-shot partition would have forced the query to visit up to  $\Theta(r^2)$  such cells, which would make it too expensive.



and denote by  $\Psi$  the collection of these surfaces, over all cells  $\tau$ . We have  $N := |\Psi| = O((n/g)^{1+\varepsilon} \cdot g^3) = O(n^{1+\varepsilon}g^2)$ . Similarly, we let  $P$  denote the set of all triples  $(b_1, b_2, b_3)$ , for  $b_1, b_2, b_3 \in B_{\tau'}^3$ , over all cells  $\tau'$  of  $\Xi(B)$ . We have  $M := |P| = O(n^{1+\varepsilon}g^2)$ . These bounds pertain to the bottommost level of the hierarchy; they are smaller at levels of smaller indices.

We apply a batched point-location procedure to the points of  $P$  and the surfaces of  $\Psi$ . The output of this procedure is a collection of complete bipartite subgraphs of  $P \times \Psi$ , so that, for each such subgraph  $P_\alpha \times \Psi_\alpha$ ,  $G(a_1, a_2, a_3; b_1, b_2, b_3)$  has a fixed sign for all  $(b_1, b_2, b_3) \in P_\alpha$  and all  $(a_1, a_2, a_3) \in \Psi_\alpha$ , see, e.g., [3, 12] for the use of such structures in similar contexts. This tells us the desired signs of  $G(a_1, a_2, a_3; b_1, b_2, b_3)$ , for every pair of triples  $(a_1, a_2, a_3) \in A_\tau^3$ ,  $(b_1, b_2, b_3) \in B_{\tau'}^3$ , over all pairs of cells  $(\tau, \tau') \in \Xi(A) \times \Xi(B)$ , and these signs give us the orientation (i.e., the order of the intersection points) of every triple of lines  $\gamma_{a,b}$ . That is, we obtain the order type of the lines. As remarked in Section 2, we may assume that this also includes the sorting of the lines at  $x = -\infty$ , but, for the sake of concreteness, we address this simpler task in the full version [4, Section 3].

The batched point location step proceeds by using the recent multilevel polynomial partitioning technique of Agarwal et al. [2, Corollary 4.8]. We delegate the full, and rather technical, details of the analysis to the full version [4, Section 3.1], and just summarize the result here.

► **Proposition 2.** *Let  $T(M, N)$  denote the maximum possible sum of the sizes of the vertex sets of the complete bipartite graphs produced by the recursive process described above, over all input sets of at most  $M$  points and at most  $N$  surfaces. Then we have*

$$T(M, N) = O\left(M^{6/7+\varepsilon}N^{6/7+\varepsilon} + M^{1+\varepsilon} + N^{1+\varepsilon}\right),$$

for any  $\varepsilon > 0$ , where the constant of proportionality depends on  $\varepsilon$ . The same asymptotic bound also holds for the cost (in the uniform model) of constructing these graphs.

In summary, the information collected so far allows us to obtain the combinatorial structure of each of the arrangements  $\mathcal{A}(\Gamma_\sigma)$ , over all cells  $\sigma$  of  $\Xi$ , and subsequently construct an order-type-based point-location data structure for each of them, at no extra cost in the algebraic decision-tree model. The overall cost of this phase, in this model, is thus  $O((n^{1+\varepsilon}g^2)^{12/7+\varepsilon})$ , for any  $\varepsilon > 0$ . By replacing  $\varepsilon$  by some small multiple thereof, we can write this bound as  $O((ng^2)^{12/7+\varepsilon})$ , for any  $\varepsilon > 0$ .

Fredman’s trick, as applied above, separates the handling of the conflict lists  $A_\tau$ , over the trapezoids  $\tau$  of  $\Xi(A)$ , and the conflict lists  $B_{\tau'}$ , over the trapezoids  $\tau'$  of  $\Xi(B)$ . For a cell  $\sigma = (\tau, \tau')$  of  $\Xi$ , not all the segments in  $A_\tau$  necessarily cross  $\sigma$ , so we have to retain (for  $\sigma$ ) only those that do cross it, and apply a similar pruning to  $B_{\tau'}$ . As we show in the full version [4, Section 3], the cost of this filtering step is  $O(g)$  for each  $\sigma$ , for an overall cost of  $O((n/g)^2 \cdot g) = O(n^2/g)$ .

**Searching with the elements of  $C$ .** We now need to search the structures computed in the preceding phase with the dual features of the triangles of  $C$ . Due to lack of space, we delegate the description to the full version [4, Section 3], where we show that the total searching time, for all the elements of  $C$ , is  $O\left(\frac{n^{2+\varepsilon} \log^2 g}{g}\right)$ , concluding (see once again our high-level roadmap):

► **Theorem 3.** *Let  $A$  and  $B$  be two sets each consisting of  $n$  pairwise disjoint segments in the plane, and let  $C$  be a set of  $n$  triangles in the plane. We can count, for each triangle  $\Delta \in C$ , the number of intersection points of segments of  $A$  with segments of  $B$  that lie inside  $\Delta$ , in the algebraic decision-tree model, at the subquadratic cost  $O(n^{60/31+\varepsilon})$ , for any  $\varepsilon > 0$ .*

► **Corollary 4.** *We can solve, in the algebraic decision-tree model, at the cost of  $O(n^{60/31+\varepsilon})$ , for any  $\varepsilon > 0$ , each of the problems (i) intersection of three polygons, (ii) coverage by three polygons, and (iii) segment concurrency, as listed in the introduction.*

---

## References

- 1 Pankaj K. Agarwal. Parititioning arrangements of lines II: Applications. *Discret. Comput. Geom.*, 5:533–573, 1990. doi:10.1007/BF02187809.
- 2 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, and Joshua Zahl. Efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50(2):760–787, 2021. doi:10.1137/19M1268550.
- 3 Pankaj K. Agarwal, Marco Pellegrini, and Micha Sharir. Counting circular arc intersections. *SIAM J. Comput.*, 22(4):778–793, 1993. doi:10.1137/0222050.
- 4 Boris Aronov, Mark de Berg, Jean Cardinal, Esther Ezra, John Iacono, and Micha Sharir. Subquadratic algorithms for some 3SUM-hard geometric problems in the algebraic decision tree model. *CoRR*, abs/2109.07587, 2021. arXiv:2109.07587.
- 5 Boris Aronov, Esther Ezra, and Micha Sharir. Testing polynomials for vanishing on cartesian products of planar point sets. In *36th International Symposium on Computational Geometry, SoCG 2020*, volume 164 of *LIPIcs*, pages 8:1–8:14, 2020. doi:10.4230/LIPIcs.SocG.2020.8.
- 6 Luis Barba, Jean Cardinal, John Iacono, Stefan Langerman, Aurélien Ooms, and Noam Solomon. Subquadratic algorithms for algebraic 3SUM. *Discret. Comput. Geom.*, 61(4):698–734, 2019. doi:10.1007/s00454-018-0040-y.
- 7 Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 80–86. ACM, 1983. doi:10.1145/800061.808735.
- 8 Jürgen Bokowski, Simon King, Susanne Mock, and Ileana Streinu. The topological representation of oriented matroids. *Discret. Comput. Geom.*, 33(4):645–668, 2005. doi:10.1007/s00454-005-1164-4.
- 9 Jürgen Bokowski, Susanne Mock, and Ileana Streinu. On the Folkman-Lawrence topological representation theorem for oriented matroids of rank 3. *Eur. J. Comb.*, 22(5):601–615, 2001. doi:10.1006/eujc.2000.0482.
- 10 Jean Cardinal, John Iacono, and Aurélien Ooms. Solving  $k$ -SUM using few linear queries. In *24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 25:1–25:17, 2016. doi:10.4230/LIPIcs.ESA.2016.25.
- 11 Timothy M. Chan. More logarithmic-factor speedups for 3SUM, (median, +)-convolution, and some geometric 3SUM-hard problems. *ACM Trans. Algorithms*, 16(1):7:1–7:23, 2020. doi:10.1145/3363541.
- 12 Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica*, 11(2):116–132, 1994. doi:10.1007/BF01182771.
- 13 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. URL: <https://www.worldcat.org/oclc/227584184>.
- 14 Mark de Berg and Otfried Schwarzkopf. Cuttings and applications. *Int. J. Comput. Geom. Appl.*, 5(4):343–355, 1995. doi:10.1142/S0218195995000210.
- 15 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15(2):317–340, 1986. doi:10.1137/0215023.
- 16 Esther Ezra, Sarel Har-Peled, Haim Kaplan, and Micha Sharir. Decomposing arrangements of hyperplanes: VC-dimension, combinatorial dimension, and point location. *Discret. Comput. Geom.*, 64(1):109–173, 2020. doi:10.1007/s00454-019-00141-7.



- 17 Esther Ezra and Micha Sharir. A nearly quadratic bound for point-location in hyperplane arrangements, in the linear decision tree model. *Discret. Comput. Geom.*, 61(4):735–755, 2019. doi:10.1007/s00454-018-0043-8.
- 18 Jon Folkman and Jim Lawrence. Oriented matroids. *J. Comb. Theory, Ser. B*, 25(2):199–236, 1978. doi:10.1016/0095-8956(78)90039-4.
- 19 Anka Gajentaan and Mark H. Overmars. On a class of  $O(n^2)$  problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.
- 20 Omer Gold and Micha Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. In *25th Annual European Symposium on Algorithms, ESA 2017*, volume 87 of *LIPICs*, pages 42:1–42:13, 2017. doi:10.4230/LIPICs.ESA.2017.42.
- 21 Jacob E. Goodman and Richard Pollack. Multidimensional sorting. *SIAM J. Comput.*, 12(3):484–507, 1983. doi:10.1137/0212032.
- 22 Jacob E. Goodman and Richard Pollack. Semispaces of configurations, cell complexes of arrangements. *J. Comb. Theory, Ser. A*, 37(3):257–293, 1984. doi:10.1016/0097-3165(84)90050-5.
- 23 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018. doi:10.1145/3185378.
- 24 David Haussler and Emo Welzl.  $\varepsilon$ -nets and simplex range queries. *Discret. Comput. Geom.*, 2:127–151, 1987. doi:10.1007/BF02187876.
- 25 Daniel M. Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-SUM and related problems. *J. ACM*, 66(3):16:1–16:18, 2019. doi:10.1145/3285953.
- 26 D. T. Lee and Franco P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6(3):594–606, 1977. doi:10.1137/0206043.
- 27 Jirí Matousek and Zuzana Patáková. Multilevel polynomial partitions and simplified range searching. *Discret. Comput. Geom.*, 54(1):22–41, 2015. doi:10.1007/s00454-015-9701-2.
- 28 Stefan Meiser. Point location in arrangements of hyperplanes. *Inf. Comput.*, 106(2):286–303, 1993. doi:10.1006/inco.1993.1057.
- 29 Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986. doi:10.1145/6138.6151.



# Approximate Maximum Halfspace Discrepancy

Michael Matheny 

Amazon, Seattle, WA, USA

Jeff M. Phillips  

University of Utah, Salt Lake City, UT, USA

---

## Abstract

Consider the geometric range space  $(X, \mathcal{H}_d)$  where  $X \subset \mathbb{R}^d$  and  $\mathcal{H}_d$  is the set of ranges defined by  $d$ -dimensional halfspaces. In this setting we consider that  $X$  is the disjoint union of a red and blue set. For each halfspace  $h \in \mathcal{H}_d$  define a function  $\Phi(h)$  that measures the “difference” between the fraction of red and fraction of blue points which fall in the range  $h$ . In this context the maximum discrepancy problem is to find the  $h^* = \arg \max_{h \in \mathcal{H}_d} \Phi(h)$ . We aim to instead find an  $\hat{h}$  such that  $\Phi(h^*) - \Phi(\hat{h}) \leq \varepsilon$ . This is the central problem in linear classification for machine learning, in spatial scan statistics for spatial anomaly detection, and shows up in many other areas. We provide a solution for this problem in  $O(|X| + (1/\varepsilon^d) \log^4(1/\varepsilon))$  time, for constant  $d$ , which improves polynomially over the previous best solutions. For  $d = 2$  we show that this is nearly tight through conditional lower bounds. For different classes of  $\Phi$  we can either provide a  $\Omega(|X|^{3/2 - o(1)})$  time lower bound for the exact solution with a reduction to APSP, or an  $\Omega(|X| + 1/\varepsilon^{2 - o(1)})$  lower bound for the approximate solution with a reduction to 3SUM.

A key technical result is a  $\varepsilon$ -approximate halfspace range counting data structure of size  $O(1/\varepsilon^d)$  with  $O(\log(1/\varepsilon))$  query time, which we can build in  $O(|X| + (1/\varepsilon^d) \log^4(1/\varepsilon))$  time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases** range spaces, halfspaces, scan statistics, fine-grained complexity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.4

**Related Version** *Full Version:* <https://arxiv.org/abs/2106.13851>

**Funding** *Jeff M. Phillips:* Thanks to supported by NSF CCF-1350888, IIS-1251019, ACI-1443046, CNS-1514520, and CNS-1564287.

## 1 Introduction

Let  $X$  be a set of  $m$  points in  $\mathbb{R}^d$  for constant  $d$  where  $X$  can either be the union of a red,  $R$ , and blue set,  $B$ , of points  $X = R \cup B$  (possibly not disjoint) or a set of weighted points where each point has weight  $w(x)$  for  $x \in X$ . Now let  $(X, \mathcal{H}_d)$  be the associated range space of all subsets of  $X$  defined by intersection with a halfspace.

We are interested in finding the halfspace  $h^*$  and value  $\Phi^*$  that maximizes a function  $\Phi_X(h) : \mathcal{H}_d \rightarrow \mathbb{R}$  for some class of functions  $\Phi$ . We characterize them by reframing it as a function of  $\mu_R$  and  $\mu_B$  so  $\Phi_X(h) = \phi(\mu_R(h), \mu_B(h))$ , where  $\mu_R(h) = |R \cap h|/|R|$  and  $\mu_B(h) = |B \cap h|/|B|$  are the fraction of red or blue points, respectively, in the range  $h$ . In particular, we only consider functions  $\Phi_X(h)$  which can be calculated in  $O(1)$  time from  $\mu_R(h)$  and  $\mu_B(h)$  as  $\phi(\mu_R(h), \mu_B(h))$  (e.g.,  $\phi(\mu_R, \mu_B) = |\mu_R - \mu_B|$ ). Given such a fixed  $\phi$ , or one from a class, we state the two main problems: exact and  $\varepsilon$ -additive error.

- **Problem MAX-HALFSPACE:** *From a given set  $X = R \cup B \subset \mathbb{R}^d$  points where  $|X| = m$  and a Lipschitz constant function  $\Phi_X(h) : 2^X \rightarrow \mathbb{R}$ , find  $h^* = \arg \max_{h \in \mathcal{H}_d} \Phi_X(h)$ .*
- **Problem  $\varepsilon$ -MAX-HALFSPACE:** *From a given set  $X = R \cup B \subset \mathbb{R}^d$  points where  $|X| = m$  and a Lipschitz constant function  $\Phi_X(h) : 2^X \rightarrow \mathbb{R}$  where  $h^* = \arg \max_{h \in \mathcal{H}_d} \Phi_X(h)$ , find  $\hat{h} \in \mathcal{H}_d$  such that  $\Phi_X(h^*) - \Phi_X(\hat{h}) \leq \varepsilon$ .*



© Michael Matheny and Jeff M. Phillips;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

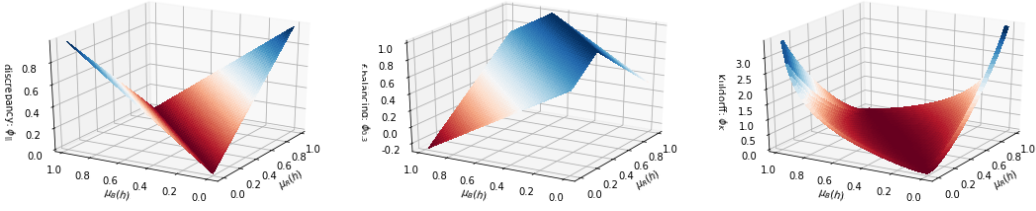
Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 4; pp. 4:1–4:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 4:2 Approximate Maximum Halfspace Discrepancy



■ **Figure 1** Plots of common  $\phi$  functions. From left to right:  $\phi_{||}$ ,  $\phi_f$  (for  $f = 0.3$ ), and  $\phi_K$ .

We typically write  $\Phi_X$  as  $\Phi$  when the set  $X$  is clear. Our algorithms will explicitly compute  $\mu_R(h)$  (and  $\mu_B(h)$  separately) for each  $h$  evaluated, and so work with all functions  $\phi$ . Notable useful examples of  $\phi$ , shown in Figure 1, include:

- **discrepancy:**  $\phi_{||}(h) = |\mu_R(h) - \mu_B(h)|$ .  
This measures the maximum disagreement between the proportions of the two sets  $B$  and  $R$ : the central task of building a linear classifier (on training data) in machine learning is often formulated to maximize precisely this function [26, 34, 12, 15]. Given a coloring of  $X$  into  $R$  and  $B$ , this also measures the discrepancy of that coloring [32, 10].
- **$f$ -balancing:**  $\phi_f(h) = 1 - |(\mu_R(h) - \mu_B(h)) - f|$ , for a fraction  $f \in (0, 1)$ .  
This scores how well a halfspaces strikes a balance of  $f$  between the two sets. By minimizing this function (or maximizing  $1 - \phi_f(h)$ ), the goal is to find a range  $h$  that exhibits an imbalance between the sets of  $f$ . Maximizing this function (say with  $f = 1/2$ ) is the problem of finding good ham sandwich cuts [27].
- **Kulldorff:**  $\phi_K(h) = \mu_R(h) \log \frac{\mu_R(h)}{\mu_B(h)} + (1 - \mu_R(h)) \log \frac{1 - \mu_R(h)}{1 - \mu_B(h)}$ .  
This is the Kulldorff discrepancy function [23] which arises in spatial scan statistics [29, 30, 23, 24, 21, 33, 3, 2], for detecting spatial anomalies. This function specifically is derived as the log-likelihood ratio test under a Poisson model, but other similar convex, non-linear functions arise naturally from other models [24, 2]. In this setting the most common range shape model is a disk, and the best algorithms [16, 17, 29] operate by lifting to one-dimension higher where the ranges correspond to halfspaces.

All of these  $\phi$  functions are Lipschitz continuous over  $\mu_B$  and  $\mu_R$ , and thus this extends to a combinatorial notion of Lipschitz over the associated  $\Phi$  with respect to the combinatorial range  $h$ : that is,  $|\Phi_X(h) - \Phi_X(h')| \leq c(|h \cap R| - |h' \cap R|/|R| + |h \cap B| - |h' \cap B|/|B|)$  for constant  $c$ . For  $\phi_{||}$  and  $\phi_f$ ,  $c = 1$ , and for  $\phi_K$  it is bounded in a reasonable range of  $\mu_B, \mu_R$  [2, 28]. This means that approximating each of  $\mu_B(h)$  and  $\mu_R(h)$  up to additive error translates into at most additive error in  $\Phi_X$ .

We also consider the hardness of these problems, and for this we will restrict the classes of functions  $\phi$  considered in these problems. We will show the largest lower bound on *concave* functions  $\phi$  (like  $\phi_f$ ), and this construction will apply to a Lipschitz functions (again like  $\phi_f$ ). However, the class of convex functions (like  $\phi_{||}$  and  $\phi_K$ ) are more prevalent, and we present a smaller lower bound, but which applies to convex Lipschitz functions (including  $\phi_{||}$  and  $\phi_K$ ).

We also note that for all of the above functions and linked challenges, the  $\varepsilon$ -approximate versions are just as relevant and common as the exact ones. For instance in machine learning, typically  $\varepsilon$ -additive error is the baseline, on assumptions that the input  $X$  is drawn from a fixed but unknown distribution [38, 25]. Spatial scan statistics are applicable to data sets containing thousands to 100s of millions of spatial data points, such as census data, disease incidents, geo-located social media posts. The exact algorithms are polynomial in  $m$  and therefore can have massive runtimes, which become infeasible in the full data sets. Usually

the exact algorithms are not even attempted and the set of ranges considered are defined using some heuristic such as disks with centerpoints of a grid or centered at data points. On the other hand the  $\varepsilon$ -additive error versions are scalable with runtimes of  $O(m + \text{poly}(\frac{1}{\varepsilon}))$ , depending only on the accuracy of the solution not the scale of the data or assumptions about the data distribution.

**Our Results.** We connect this problem to the approximate range counting problem for constant  $d \geq 2$  in the additive error model by designing a data structure of size  $O(\frac{1}{\varepsilon^d})$  that can be constructed in time  $O(m + \frac{1}{\varepsilon^d} \log^4 \frac{1}{\varepsilon})$  with constant probability and supports range counting queries in time  $O(\log \frac{1}{\varepsilon})$ . This structure implies a halfspace scanning algorithm that runs in time  $O(m + \frac{1}{\varepsilon^d} \log^4 \frac{1}{\varepsilon})$ . The data structure is closely related to cuttings, but we do not need the set of crossing lines, but only an approximation of their total count.

In the other direction, we show instances where  $\phi$  is linear that are as hard as  $\Omega(m^{3/2-o(1)})$  on the exact problem, reduced from APSP, and instances where  $\phi$  is concave that are as hard as  $\Omega(m + \frac{1}{\varepsilon^{2-o(1)}})$ , reduced from 3SUM. This implies (conditionally) that this class of scanning algorithms requires  $\Omega(m + \frac{1}{\varepsilon^{2-o(1)}})$  time, and any further algorithmic improvements (beyond  $\text{polylog}(1/\varepsilon)$  factors) would either require specific and new assumptions on  $\phi$  or improvement on a classic hard problem.

**Relation to Prior Work.** The best prior algorithms for  $\varepsilon$ -MAX-HALFSPACE required  $O(m + (1/\varepsilon)^{d+1/3} \log^{2/3}(1/\varepsilon))$  time [28] and the best exact algorithm requires  $O(m^d)$  time [16, 17]. Conditioned on 3SUM, without allowing restrictions to  $\phi$  beyond linearity, our lower bound shows the prior exact algorithms are shown tight for  $d = 2$  by setting  $m = 1/\varepsilon$ .

For the related problem of the range space defined by axis-aligned rectangles [39], the story for instance is more complicated with respect to  $\phi$ . For linear  $\phi$  the exact problem can be solved in  $O(m^2)$  time [7] which is tight [6] assuming no subcubic algorithm for MAX-WEIGHT-3-CLIQUE (and hence APSP). The  $\varepsilon$ -approximate version can be solved in  $O(m + \frac{1}{\varepsilon^{2d-2}} + \frac{1}{\varepsilon^2} \log \log \frac{1}{\varepsilon})$  time with constant probability [28], and by the result of Backurs *et al.* [6] this cannot be improved beyond  $\Omega(m + 1/\varepsilon^2)$  in  $\mathbb{R}^2$  under the same assumptions [28]. However, for general functions  $\phi$  the best known runtimes increase to  $O(m^4)$  and  $O(m + 1/\varepsilon^4)$  for the exact and approximate versions, respectively [28]. Although for a big class of convex functions (like  $\phi_K$ ) can be reduced back to  $O(m + 1/\varepsilon^{2.5})$  in the approximate case [28]. Thus, this paper shows the situation appears significantly simpler for the halfspaces case, and we provide a new connection to a (usually) separate [41, 6, 40] class of conditional hardness problems through 3SUM.

Our approximate range counting structure is also new and may be of independent interest, allowing very fast halfspace queries (in  $O(\log 1/\varepsilon)$  time) in moderate dimensions  $d$  where  $1/\varepsilon^d$  may not be too large. For example,  $3d$  disk queries maps to the  $d = 4$  setting, and with  $\varepsilon = 0.01$  (1% error) then  $1/\varepsilon^d = 100$  million – which can fit in the memory of most modern systems, and allow for very fast queries. Similar structures are possible for the exact range counting paradigm [8, 10], and these could be adapted to the approximate setting after constructing an appropriate  $\varepsilon$ -sample of  $X$  [38, 25, 29]. But these would instead use higher  $\tilde{O}(1/\varepsilon^{d+1})$  preprocessing time (where  $\tilde{O}(z)$  hides  $\text{polylog}(z)$  terms). Most effort in approximate range counting has come in the low-space regime. For instance with *relative*  $(1 + \varepsilon)$  error, one can build a data structure of expected size  $O(\text{poly}(1/\varepsilon) \cdot m)$  that answers approximate range counting queries in  $O(\text{poly}(1/\varepsilon) \cdot \log m)$  time in  $\mathbb{R}^3$  [1]. However for  $d > 3$  with linear space, the query time becomes polynomial in  $m$  at  $\tilde{O}(m^{1-1/\lfloor d/2 \rfloor})$  [5, 4, 35].



A related line of work on robust halfspace learning under specific noise models in high dimensions has witnessed significant progress in the last few years [12, 15, 13, 11]. These models and algorithms also start with a sufficiently large iid sample (of size roughly  $1/\varepsilon^2$ ), and ultimately achieve a result with additive  $\varepsilon$  error from the opt. However, they assume a specific noise model, but using this achieve runtimes polynomial in  $d$ , whereas our results grow exponentially with  $d$  but do not require any assumptions on the noise model. A recent focus is on Massart noise, where given a perfect classifier, each point has its sign flipped independently with an assigned probability at most  $\eta < 1/2$ . Only recently [11], in this Massart noise model it was achieved a proper learned function (a halfspace) in time polynomial in  $1/\varepsilon$  and in  $d$ . Recent previous work was not proper (the learned function may not be a halfspace) [12, 15], or takes time exponential in  $1/\varepsilon$  [13]. Indeed, under Gaussian distributed data, the proper halfspace learning requires  $d^{\Omega(1/\varepsilon)}$  time [14], under the statistical query model [36].

## 2 Background and Notation

**Useful Sampling Properties.** An  $\varepsilon$ -sample [19, 38] is a subset  $S \subset X$  so for a range space  $(X, \mathcal{R})$  it preserves the density for all ranges as  $\max_{A \in \mathcal{R}} \left| \frac{|X \cap A|}{|X|} - \frac{|S \cap A|}{|S|} \right| \leq \varepsilon$ . An  $\varepsilon$ -net is a subset  $N \subset X$  so for a range space  $(X, \mathcal{R})$  it hits large ranges, specifically for all ranges  $A \in \mathcal{R}$  such that  $|X \cap A| \geq \varepsilon|X|$  we guarantee that  $N \cap A \neq \emptyset$ . For halfspaces, a random sample  $S \subset X$  of size  $O(\frac{1}{\varepsilon^2}(d + \log \frac{1}{\delta}))$  is an  $\varepsilon$ -sample with probability at least  $1 - \delta$  [38, 25], and a random sample  $N \subset X$  of size  $O(\frac{d}{\varepsilon} \log \frac{1}{\delta})$  is an  $\varepsilon$ -net [20] with probability at least  $1 - \delta$ .

**Enumeration.** Given a set of points  $X \subset \mathbb{R}^d$ , Sauer's Lemma [37] shows that there are at most  $O(|X|^d)$  combinatorially distinct ranges, where each range defined by a halfspace contains the same subset of points. We can always take a halfspace and rotate it until it intersects at most  $d$  boundary points without changing the set of points contained inside. This observation immediately implies a simple algorithm for scanning the point set; we can just enumerate all subsets of at most  $d$  points, compute a halfspace that goes through these points, and count the red and blue points lying underneath to evaluate  $\Phi$ .

Our work builds upon this simple algorithm by dividing it into two steps and optimizing both. We first define a set of prospective ranges  $\hat{\mathcal{H}}_d$  to scan and then secondly compute the function  $\Phi$  on each region. Matheny *et al.* [28] showed that a simple random sample  $X_0 \subset X$  of size  $O(\frac{1}{\varepsilon})$  (for constant  $d$ ) induces a small range space  $(X_0, \mathcal{H}_d)$ . Each range (a subset of  $X_0$ ) maps to a canonical geometric halfspace  $h_0$ , and this geometric halfspace in turn induces a range  $h_0 \cap X$ , an element of  $(X, \mathcal{H}_d)$ . We refer to this subset of  $(X, \mathcal{H}_d)$  as  $(X, \hat{\mathcal{H}}_d)$ , it is of size  $O(1/\varepsilon^d)$ . Now for each range  $h \cap X$  in  $(X, \mathcal{H}_d)$  there is a range  $\hat{h} \cap X \in (X, \hat{\mathcal{H}}_d)$  such that the symmetric difference between them is at most  $\varepsilon|X|$ . So if every range in  $(X, \hat{\mathcal{H}}_d)$  needs to be explicitly checked, and  $f(1/\varepsilon)$  is the time to compute  $\Phi$ , this would imply a  $(1/\varepsilon^d)f(1/\varepsilon)$  lower bound. Note that  $f(1/\varepsilon)$  may take super-constant time because we may need to construct the (approximate)  $\mu_R(h)$  and  $\mu_B(h)$  values.

**Cuttings.** Cuttings are a useful tool to formalize divide and conquer steps in geometric algorithm design. Given  $\mathbb{R}^d$ , a set of halfspaces  $\mathcal{H}_d$  of size  $m$ , and some parameter  $r$  a  $\frac{1}{r}$ -cutting is a partition of  $\mathbb{R}^d$  into a disjoint set of constant complexity cells where each cell is crossed by at most  $m/r$  halfspaces. The set of halfspaces crossing a cell in the cutting is referred to as the conflict list of the cell. It is well established that the number of partitions



is of size  $O(r^d)$  and the partitioning can be constructed in  $O(r^{d-1}m)$  time if the conflict lists are needed [31]. If the crossing information is not needed then faster algorithms can be used. For instance, the arrangement of a  $\frac{1}{r}$ -net over  $\mathcal{H}_d$  defines a partitioning of  $\mathbb{R}^d$  into disjoint cells where each cell is crossed by at most  $m/r$  halfspaces, and since a simple random sample can be used to generate the net, a cutting of size  $O(r^d \log^d r)$  can be computed in  $O(m + r^d \log^d r)$  time with constant probability. For  $r \ll m$ , this can be a substantial runtime improvement with a small increase in size.

When a cutting is restricted to a single cell there are better bounds on the size of the partitioning. We will need this better bound for our proof and we restate it here.

► **Theorem 1** ([9, 8]). *Denote the vertices corresponding to  $d$ -way intersections of  $\mathcal{H}_d$  as  $\mathcal{A}(\mathcal{H}_d)$ . A  $\frac{1}{r}$ -cutting of a cell  $\Delta$  containing  $|\mathcal{A}(\mathcal{H}_d) \cap \Delta| = \eta$  vertices can be constructed with  $O(\eta \left(\frac{r}{m}\right)^d + r^{d-1})$  cells.*

### 3 Approximate Halfspace Range Counting and the Upper Bound

In this section, instead of operating on  $R \cup B = X \in \mathbb{R}^d$ , as is common for halfspace range searching, we work on the set  $H$  of dual halfspaces in  $\mathbb{R}^d$ . In this setting a query halfspace  $h \in \mathcal{H}_d$  in the dual representation is  $q_h \in \mathbb{R}^d$ , and the desired quantity is the number of halfspaces in  $H$  below  $q_h$ . We apply the construction separately for  $R$  and  $B$ , so the halfspaces  $H$  may be weighted, with positive weights.

We will construct  $L$  decompositions of  $\mathbb{R}^d$  into disjoint trapezoidal cells:  $\Delta_0, \Delta_1, \dots, \Delta_L$ . For each level of cells  $\Delta_i$  any cell  $\Delta \in \Delta_i$  has a set of children cells  $S_\Delta$ . The next level of cells  $\Delta_{i+1}$  is the disjoint union of all the  $S_\Delta$  children cells of  $\Delta \in \Delta_i$ ; that is  $\Delta_{i+1} = \bigcup_{\Delta \in \Delta_i} S_\Delta$ . Initially  $\Delta_0 = \mathbb{R}^d$  and therefore contains the entire domain and a corresponding sample of this initial cell will be denoted as  $\hat{H}$ ; we will bound its required size in Lemma 6. Define  $\Delta \cap H$  to be the set of halfspaces in  $H$  that lie completely underneath  $\Delta$  (that is, if  $q \in \Delta$  then  $q \notin h$  for any  $h \in \Delta \cap H$ ), and  $\Delta \cap H$  is the set of halfspaces in  $H$  that cross  $\Delta$ .

Importantly, we maintain an estimate of the weight of each cell  $m_i(\Delta)$ , as we recursively build the decomposition. It depends on a sufficiently large constant  $r$ . For a cell  $\Delta$  we will consider a sample  $H_\Delta \subset H \cap \Delta$  of halfspaces. For each cell  $\Delta' \in \Delta_{i+1}$  where  $\Delta' \in S_\Delta$  we take a sample  $H_{\Delta'} \subset (H_\Delta \cap \Delta')$  of size  $\frac{|H_\Delta \cap \Delta'|}{r}$  with replacement. The value  $\hat{m}_{i+1}(\Delta')$  estimates the number of halfplanes lying below it, and is defined recursively as  $\hat{m}_{i+1}(\Delta') = r^{i+1}|H_\Delta \cap \Delta'| + \hat{m}_i(\Delta)$ ; with  $\hat{m}_0(\Delta) = 0$ . If a cell  $\Delta$  has a small number of lines crossing it, specifically if  $|H_\Delta| \leq \log \frac{1}{\varepsilon}$  then the recursion terminates. Otherwise, we *split* the cell, and create a  $1/t_\Delta$ -cutting of each  $(\Delta, H_\Delta)$ , and let  $S_\Delta$  be its cells; the value  $t_\Delta = \max\left(\frac{|H_\Delta| r^{2i+1}}{|\hat{H}|}, 1\right)$  is 1 if  $|H_\Delta|$  is small, otherwise it is at most  $r$ . We recurse on each cell  $\Delta'$  in each  $S_\Delta$  until each is sufficiently small, which will require  $L = O(\log \frac{|\hat{H}|}{\varepsilon}) = O(\log \frac{1}{\varepsilon})$  levels. See Algorithm 1 for details.

**Complexity analysis.** As cells are subsampled and split the number of lines and vertices lying inside of a cell drops off quickly with the level.

► **Lemma 2.** *A cell  $\Delta \in \Delta_i$  with sample  $H_\Delta$  is of size  $|H_\Delta| \leq |\hat{H}|/r^{2i}$ .*

**Proof.** Consider a cell  $\Delta^* \in \Delta_{i-1}$  where  $\Delta \in S_{\Delta^*}$  then  $|H_\Delta| = \frac{1}{r}|H_{\Delta^*} \cap \Delta|$  by construction, and since  $\Delta$  is a cell in a  $\frac{1}{t_{\Delta^*}}$ -cutting of  $H_{\Delta^*}$  then  $|H_{\Delta^*} \cap \Delta| \leq |H_{\Delta^*}|/t_{\Delta^*} = |\hat{H}|/r^{2i-1}$ ; hence  $|H_\Delta| = |\Delta \cap H_{\Delta^*}|/r \leq |\hat{H}|/r^{2i}$ . ◀

## 4:6 Approximate Maximum Halfspace Discrepancy

■ **Algorithm 1**  $\text{SampleCut}(\Delta_0, \hat{H})$ .

---

**for** levels  $i = [0, L]$  **do**  
  **for** each cell in that level  $\Delta \in \Delta_i$  with  $|H_\Delta| > \log \frac{1}{\varepsilon}$  **do**  
    Set  $t_\Delta = \max(\frac{|H_\Delta| r^{2i+1}}{|\hat{H}|}, 1)$   
    Build  $S_\Delta$ , the cells of a  $\frac{1}{t_\Delta}$ -cutting on  $(\Delta, H_\Delta)$ .  
    **for** each child cell  $\Delta' \in S_\Delta$  **do**  
       $\hat{m}_{i+1}(\Delta') = r^{i+1}|H_\Delta \cap \Delta'| + \hat{m}_i(\Delta)$   
      Sample  $H_{\Delta'} \subset H_\Delta$  where  $|H_{\Delta'}| = |\Delta' \cap H_\Delta|/r$

---

The number of vertices in the cells drops off quickly as well.

► **Lemma 3.** *A cell  $\Delta \in \Delta_i$  with sample  $H_\Delta$  has  $\mathbb{E}[|\mathcal{A}(H_\Delta)|] \leq \frac{|\mathcal{A}(\hat{H}) \cap \Delta|}{r^{di}}$  expected vertices.*

**Proof.** Consider a vertex lying inside of  $\Delta$  induced by the intersection of  $d$  halfspaces  $h_1, \dots, h_d \in H_{\Delta^*}$  in  $\Delta$  (where  $\Delta \subset \Delta^* \in \Delta_{i-1}$ ). The probability that this vertex is in  $H_\Delta$  is  $\Pr(h_1 \in H_\Delta \wedge \dots \wedge h_d \in H_\Delta) \leq \frac{1}{r^d}$ . The probability that a vertex survives through  $i$  samples is then upper bounded by  $\frac{1}{r^{di}}$  and by linearity of expectation  $\mathbb{E}[|\mathcal{A}(H_\Delta)|] \leq \frac{|\mathcal{A}(\hat{H}) \cap \Delta|}{r^{di}}$ . ◀

Combining these results we show the expected number of cells increases as  $O(r^{di})$ . This leverages Theorem 1 using the number of vertex dependent bound for size of the cutting.

► **Lemma 4.** *At a level  $i$  the expected number of cells is  $\mathbb{E}[|\Delta_i|] = O(r^{di})$ .*

**Proof.** The number of cells at a level  $i$  is in expectation

$$\begin{aligned} \mathbb{E}[|\Delta_i|] &= \mathbb{E} \left[ \sum_{\Delta \in \Delta_{i-1}} O(|\mathcal{A}(H_\Delta) \cap \Delta| \frac{t_\Delta^d}{|H_\Delta|^d} + t_\Delta^{d-1}) \right] \\ &= C \cdot \mathbb{E} \left[ \sum_{\Delta \in \Delta_{i-1}} |\mathcal{A}(H_\Delta) \cap \Delta| \frac{t_\Delta^d}{|H_\Delta|^d} + t_\Delta^{d-1} \right] \end{aligned}$$

from Theorem 1, for some sufficiently large constant  $C < r/4$ . Since  $t_\Delta = \max(\frac{|H_\Delta| r^{2i+1}}{|\hat{H}|}, 1)$  we can divide cells in  $\Delta_{i-1}$  into a set  $\Delta_{i-1}^+$  where  $t_\Delta > 1$ , and is therefore split, and a set of cells  $\Delta_{i-1} \setminus \Delta_{i-1}^+$  where  $t_\Delta = 1$ , and is therefore not split. The set of non split cells  $\Delta_{i-1} \setminus \Delta_{i-1}^+$  cannot be larger than  $|\Delta_{i-1}|$ .

$$\begin{aligned} \mathbb{E}[|\Delta_i|] &= C \cdot \mathbb{E} \left[ \sum_{\Delta \in \Delta_{i-1}} |\mathcal{A}(H_\Delta) \cap \Delta| \frac{t_\Delta^d}{|H_\Delta|^d} + t_\Delta^{d-1} \right] \\ &\leq C \cdot \mathbb{E} \left[ |\Delta_{i-1}| + \sum_{\Delta \in \Delta_{i-1}^+} |\mathcal{A}(H_\Delta) \cap \Delta| \frac{r^{2di-d}}{|\hat{H}|^d} + t_\Delta^{d-1} \right] \end{aligned}$$

By Lemma 2 we can bound  $t_\Delta \leq \frac{|H_\Delta| r^{2i+1}}{|\hat{H}|} \leq r$ , to replace the last term.

$$\begin{aligned} \mathbb{E}[|\Delta_i|] &\leq C \cdot \mathbb{E} \left[ |\Delta_{i-1}| + \sum_{\Delta \in \Delta_{i-1}^+} |\mathcal{A}(H_\Delta) \cap \Delta| \frac{r^{2di-d}}{|\hat{H}|^d} + r^{d-1} \right] \\ &\leq C \cdot \frac{r^{2di-d}}{|\hat{H}|^d} \mathbb{E} \left[ \sum_{\Delta \in \Delta_{i-1}^+} |\mathcal{A}(H_\Delta) \cap \Delta| \right] + C \cdot (r^{d-1} + 1) \mathbb{E}[|\Delta_{i-1}|] \end{aligned}$$

By Lemma 3  $\mathbb{E}[|\mathcal{A}(H_\Delta) \cap \Delta|] \leq \frac{|\mathcal{A}(\hat{H}) \cap \Delta|}{r^{di-d}}$ , and this yields

$$\begin{aligned} \mathbb{E}[|\Delta_i|] &\leq C \cdot \frac{r^{2di-d}}{|\hat{H}|^d} \mathbb{E} \left[ \sum_{\Delta \in \Delta_{i-1}^+} \frac{|\mathcal{A}(\hat{H}) \cap \Delta|}{r^{di-d}} \right] + C \cdot (r^{d-1} + 1) \mathbb{E}[|\Delta_{i-1}|] \\ &= C \cdot \frac{r^{di}}{|\hat{H}|^d} \mathbb{E} \left[ \sum_{\Delta \in \Delta_{i-1}^+} |\mathcal{A}(\hat{H}) \cap \Delta| \right] + C \cdot (r^{d-1} + 1) \mathbb{E}[|\Delta_{i-1}|]. \end{aligned}$$

Since a vertex in  $\mathcal{A}(\hat{H}) \cap \Delta$  can only be in one cell  $\sum_{\Delta \in \Delta_{i-1}^+} |\mathcal{A}(\hat{H}) \cap \Delta| \leq |\hat{H}|^d$ , since this quantity upper bounds the number of vertices in  $\mathcal{A}(\hat{H})$ . Hence

$$\mathbb{E}[|\Delta_i|] \leq \dots \text{chain of inequalities} \dots \leq C \cdot (r^{di} + (r^{d-1} + 1) \mathbb{E}[|\Delta_{i-1}|])$$

Now finally we show  $\mathbb{E}[|\Delta_i|] \leq 2C \cdot r^{di}$  by inductively assuming  $\mathbb{E}[|\Delta_{i-1}|] \leq 2C \cdot r^{d(i-1)}$  for  $C < r/4$  and  $r$  sufficiently large

$$\begin{aligned} \mathbb{E}[|\Delta_i|] &\leq C \cdot (r^{di} + (r^{d-1} + 1)(2C r^{d(i-1)})) \\ &\leq C \cdot (r^{di} + r^{di}/2 + r^{di-d+1}/2) \\ &\leq 2C \cdot r^{di} = O(r^{di}). \end{aligned} \quad \blacktriangleleft$$

**Sampling Error.** Consider now that we wish to estimate  $|\hat{H} \cap \Delta|$ , the number of planes crossing under some  $\Delta$ . We will use that  $\Delta$  lies within a nested sequence of cutting cells  $\Delta = \Delta_i \subset \Delta_{i-1} \subset \dots \subset \Delta_0$  with  $\Delta_i \in \mathbf{\Delta}_i, \Delta_{i-1} \in \mathbf{\Delta}_{i-1}, \dots, \Delta_0 \in \mathbf{\Delta}_0$ , where  $i \leq L$ , and with corresponding samples  $\hat{H} = H_{\Delta_0}$  and  $H_{\Delta_1}, \dots, H_{\Delta_\ell}$ . We can also define  $\hat{m}_i(\Delta)$  more generally for a cell  $\Delta$  that is the subset of a cell  $\Delta_i \in \mathbf{\Delta}_i$  (and its ancestors), but not necessary one of those cells. It is defined  $\hat{m}_i(\Delta) = r^i |H_{\Delta_{i-1}} \cap \Delta| + \sum_{j=0}^{i-1} r^j |H_{\Delta_{j-1}} \cap \Delta_j|$ . By this definition  $\hat{m}_0(\Delta) = |\hat{H} \cap \Delta|$ , and we are left to bound  $|\hat{m}_i(\Delta_i) - |\hat{H} \cap \Delta_i||$ .

► **Lemma 5.**  $|\hat{m}_i(\Delta) - |\hat{H} \cap \Delta|| = O(i \sqrt{|\hat{H}| \log \frac{2\ell}{\delta'}})$  with probability  $1 - \delta'$ .

**Proof.** By the triangle inequality we expand

$$\begin{aligned}
 |\hat{m}_i(\Delta) - \hat{m}_0(\Delta)| &\leq \sum_{j=0}^{i-1} |\hat{m}_{j+1}(\Delta) - \hat{m}_j(\Delta)| \\
 &= \sum_{j=0}^{i-1} \left| r^{j+1} |H_{\Delta_j} \cap \Delta| + \sum_{\ell=0}^j r^\ell |H_{\Delta_{\ell-1}} \cap \Delta_\ell| \right. \\
 &\quad \left. - \left( r^j |H_{\Delta_{j-1}} \cap \Delta| + \sum_{\ell=0}^{j-1} r^{\ell-1} |H_{\Delta_{\ell-2}} \cap \Delta_{\ell-1}| \right) \right| \\
 &= \sum_{j=0}^{i-1} |r^{j+1} |H_{\Delta_j} \cap \Delta| + r^j |H_{\Delta_{j-1}} \cap \Delta_j| - r^j |H_{\Delta_{j-1}} \cap \Delta|| \\
 &= \sum_{j=0}^{i-1} r^j |r |H_{\Delta_j} \cap \Delta| - |(H_{\Delta_{j-1}} \cap \Delta_j) \cap \Delta|| \\
 &\leq \sum_{j=0}^{i-1} r^j \cdot r \cdot C_d \sqrt{|H_{\Delta_j}| \log \frac{1}{\delta^\dagger}}.
 \end{aligned}$$

The last inequality follows since  $H_{\Delta_j}$  is a random sample from  $H_{\Delta_{j-1}} \cap \Delta_j$ , and the  $\cap \Delta$  restriction is a constant VC-dimension range [38]; the constant  $C_d$  depends only on  $d$ , and  $\delta^\dagger$  is the probability of failure for each term in the sum. In particular we use  $\left| \frac{|H_{\Delta_j} \cap \Delta|}{|H_{\Delta_j}|} - \frac{|(H_{\Delta_{j-1}} \cap \Delta_j) \cap \Delta|}{|H_{\Delta_{j-1}} \cap \Delta_j|} \right| \leq C_d \sqrt{\frac{1}{|H_{\Delta_j}|} \log \frac{1}{\delta^\dagger}}$  and multiply by  $|H_{\Delta_{j-1}} \cap \Delta_j| = r |H_{\Delta_j}|$ .

Applying Lemma 2 then  $|H_{\Delta_j}| \leq \frac{|\hat{H}|}{r^{2j}}$ . Hence

$$\sum_{j=0}^{i-1} r^j \sqrt{|H_{\Delta_j}| \log \frac{1}{\delta^\dagger}} \leq \sum_{j=0}^{i-1} r^j \sqrt{\frac{|\hat{H}|}{r^{2j}} \log \frac{1}{\delta^\dagger}} \leq \sum_{j=0}^{i-1} \sqrt{|\hat{H}| \log \frac{1}{\delta^\dagger}}.$$

And to ensure a failure probability of  $1 - \delta'$  for the sequence of samples, set  $\delta^\dagger = \delta'/2\ell$ .

$$|\hat{m}_i(\Delta) - |\hat{H} \cap \Delta|| \leq i \cdot C_d \sqrt{|\hat{H}| \log \frac{2\ell}{\delta'}}. \quad \blacktriangleleft$$

Now how large does  $\hat{H}$  need to be to ensure a correct estimate of the cells at the leaves of the arrangement. This largely depends on the number of  $\varepsilon$ -samples taken in total.

► **Lemma 6.** *We can ensure that  $|\hat{m}_i(\Delta) - |\hat{H} \cap \Delta|| \leq \varepsilon |\hat{H}|$  for all  $\Delta \in \mathbf{\Delta}_i$  with probability  $\delta$  by setting  $|\hat{H}| = O(\frac{i^3}{\varepsilon^2} \log \frac{i}{\delta})$ .*

**Proof.** We can set the probability of  $\delta' = \delta/(2|\mathbf{\Delta}_i|)$  in Lemma 5 to ensure a failure probability of  $\delta$  for each cell  $\Delta \in \mathbf{\Delta}_i$  and therefore  $|\hat{m}_i(\Delta) - |\hat{H} \cap \Delta|| = O(i \sqrt{|\hat{H}| \log \frac{2i|\mathbf{\Delta}_i|}{\delta}})$ . So if  $\varepsilon |\hat{H}| = |\hat{m}_i(\Delta) - |\hat{H} \cap \Delta|| = O(i \sqrt{|\hat{H}| \log \frac{2i|\mathbf{\Delta}_i|}{\delta}})$  then  $|\hat{H}| = O(\frac{i^2}{\varepsilon^2} \log \frac{2i|\mathbf{\Delta}_i|}{\delta})$ .

From Lemma 4 we know that  $|\mathbf{\Delta}_i| = O(r^{di})$ , and so  $|\hat{H}| = O(\frac{i^3}{\varepsilon^2} \log \frac{i}{\delta})$ . ◀

**Runtime.** The time to compute an estimate for  $\Delta \in \mathbf{\Delta}_i$  is linear in the number of lines in  $H_\Delta \leq \frac{|\hat{H}|}{r^{2i}}$ . We can then devise the expected runtime for computing estimates for all cells in a level  $i$ , of which there are  $\mathbb{E}[|\mathbf{\Delta}_i|] = O(r^{di})$ , so the total for level  $i$  is  $\sum_{\Delta \in \mathbf{\Delta}_i} O(\frac{|\hat{H}|}{r^{2i}}) = O(\frac{|\mathbf{\Delta}_i| |\hat{H}|}{r^{2i}}) = O(r^{i(d-2)} |\hat{H}|)$ . The total expected runtime over all levels is then  $\sum_{i=0}^L O(r^{i(d-2)} |\hat{H}|)$ . Furthermore, if  $d = 2$  then this will be  $O(\frac{L^4}{\varepsilon^2} \log \frac{L}{\delta})$ . If  $d > 2$

then each layer of the cutting will dominate the previous layers in runtime and so the total time will be bounded by the time to compute the last layer as  $\sum_{i=0}^L O(r^{i(d-2)}|\hat{H}|) = O(r^{(L+1)(d-2)}\frac{L^3}{\varepsilon^2} \log \frac{L}{\delta})$ .

Next we want to achieve that all cells  $\Delta \in \mathbf{\Delta}_L$  are of size  $|H_\Delta| < \log_r \frac{1}{\varepsilon}$ . This can be achieved via  $|H_\Delta| < |\hat{H}|/(r^{2L}) \leq \log_r \frac{1}{\varepsilon}$  by setting the maximum level  $L > \frac{1}{2} \log_r \frac{|\hat{H}|}{\log_r \frac{1}{\varepsilon}} = O(\log \frac{1}{\varepsilon})$ , via Lemma 6. Then a query can at first recursively descend into the structure for  $O(L) = O(\log \frac{1}{\varepsilon})$  steps, and upon reaching a leaf node, can enumerate the leaf node's sample which will take at most  $O(\log \frac{1}{\varepsilon})$  time again. With  $L = O(\log \frac{1}{\varepsilon})$ , the total time to compute the cell division for  $d = 2$  is  $O(\frac{L^4}{\varepsilon^2} \log \frac{L}{\delta}) = O(\frac{1}{\varepsilon^2} \log^4 \frac{1}{\varepsilon} \log \frac{\log \frac{1}{\varepsilon}}{\delta})$ . For  $d > 2$ , then  $O(r^{(L+1)(d-2)}\frac{L^3}{\varepsilon^2} \log \frac{L}{\delta}) = O(\frac{1}{\varepsilon^d} \log^3 \frac{1}{\varepsilon} \log \frac{\log \frac{1}{\varepsilon}}{\delta})$  since  $r^{(L+1)(d-2)} = r r^{(d-2) \log_r \frac{1}{\varepsilon}} = r r^{\log_r \frac{1}{\varepsilon^{d-2}}} = \frac{r}{\varepsilon^{d-2}}$ .

► **Theorem 7.** *We can build an  $\varepsilon$ -approximate halfspace range counting data structure of size  $O(1/\varepsilon^d)$  that for any halfspace  $h \in \mathcal{H}_d$  returns in  $O(\log(1/\varepsilon))$  time returns a count  $\hat{m}(h)$  so that  $|\hat{m}(h) - |h \cap X|| \leq \varepsilon|X|$ . The total expected construction time, with probability  $1 - \delta$ , for  $d = 2$  is  $O(|X| + \frac{1}{\varepsilon^2} \log^4 \frac{1}{\varepsilon} \log \frac{\log \frac{1}{\varepsilon}}{\delta})$ , and for constant  $d > 2$  is  $O(|X| + \frac{1}{\varepsilon^d} \log^3 \frac{1}{\varepsilon} \log \frac{\log \frac{1}{\varepsilon}}{\delta})$ .*

**Finding the Maximum Range.** To query the structure we need a set of viable halfspaces that cover the space well enough to approximately hit the maximum region. We can use a random sample of points from the primal space of size  $O(\frac{1}{\varepsilon})$  to induce a set of  $O(1/\varepsilon^d)$  halfspaces  $\hat{\mathcal{H}}_d$ , as was done in [28], to get a constant probability that at least one halfspace is  $O(\varepsilon)$ -close to the maximum region. Then we repeat the procedure  $\log \frac{1}{\delta}$  times and take the maximum found region to magnify the success probability to  $1 - \delta$  (see [28] for details). For each query hyperplane  $h$  we can query a structure constructed over  $R$  and over  $B$  and then compute the function value from this; we return the  $h$  which maximizes  $\Phi(h)$ .

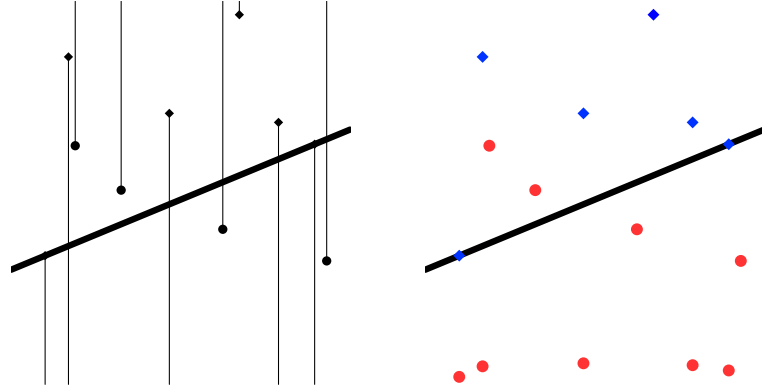
The set of query hyperplanes from this tactic is of size  $O(\frac{1}{\varepsilon^d})$  for constant  $d$  and at each level we can determine which cell the dual point of the halfspace falls into by testing a constant  $O(r^d)$  number of constant sized cells. At a leaf of the structure we check if the remaining, at most  $\log \frac{1}{\varepsilon}$ , dual halfspaces are below the query dual point, to determine the total count. The query structure has  $O(\log \frac{1}{\varepsilon})$  levels and at each level a constant amount of work is done; it is repeated for each of  $O(\frac{1}{\varepsilon^d})$  halfspaces, and the entire endeavor is repeated  $\log \frac{1}{\delta}$  times to reduce the probability of failure. The full runtime is  $O(\frac{1}{\varepsilon^d} \log \frac{1}{\varepsilon} \log \frac{1}{\delta})$ , plus the construction time of the data structure (from Theorem 7) which dominates the cost.

► **Theorem 8.** *We can solve  $\varepsilon$ -MAX-HALFSPACE with probability  $1 - \delta$ , in expected time  $O(|X| + \frac{1}{\varepsilon^d} \log^4 \frac{1}{\varepsilon} \log \frac{\log \frac{1}{\varepsilon}}{\delta})$  for  $d = 2$  and  $O(|X| + \frac{1}{\varepsilon^d} \log^3 \frac{1}{\varepsilon} \log \frac{\log \frac{1}{\varepsilon}}{\delta})$  for constant  $d > 2$ .*

## 4 Conditional Lower Bounds

Our upper bounds only restrict  $\Phi_X(h) = \phi(\mu_R(h), \mu_B(h))$  to be a Lipschitz function. Next we show that an algorithm that can operate on this entire class of functions for  $d = 2$  has a conditional lower bound of  $O(m + \frac{1}{\varepsilon^{2-o(1)}})$ , depending on 3SUM [18].

However, the first bound requires  $\phi$  is also concave (unlike  $\phi_{||}$  or  $\phi_K$  which are convex). So we consider a different convex function  $\phi(\mu_R, \mu_B) = \mu_R - \mu_B$  on weighted points; solving for it, and again after flipping all signs of weights, corresponds with  $\phi_{||}$  (which can be used to approximate  $\phi_K$ ). We show MAX-HALFSPACE for this  $\phi$  is lower bounded by  $O(m^{3/2-o(1)})$  conditional on APSP [41] requiring  $\Omega(n^{3-o(1)})$  time.



■ **Figure 2** On left converting the POINT-COVERING problem into the dual makes it such that a line contains every ray that it intersects. On right the equivalent bichromatic discrepancy problem where red points have been placed on the lower envelope.

### 4.1 Lower Bounds by 3SUM

Gajentaan and Overmars identified a large class of problems in computational geometry called 3SUM hard [18]. 3SUM can be reduced to each of these problems, so an improvement in any one of them would imply an improvement in 3SUM. While there are some loose lower bounds for this problem, 3SUM is conjectured to **not** be solvable in  $O(m^{2-o(1)})$  time [22, 40]. 3SUM reduces to the following problem in  $O(m \log m)$  time [18].

- **Problem POINT-COVERING:** *From a given set of  $m$  halfspaces determine if there is a point at depth  $k$  ( $k$  halfspaces lie above this point) where  $k \leq m/2$ .*

Through standard point-line duality, we transform this to its equivalent dual problem.

- **Problem LINE-COVERING:** *From a given set of  $m$  rays, oriented upwards or downwards, determine if there is a line cutting through  $k$  rays where  $k \leq m/2$ .*

Define the piecewise linear function (for  $k \leq m/2$ ) on points  $R, B \in \mathbb{R}^2$ :

$$\Phi(h) = \phi_{LC}(\mu_R, \mu_B) = |R| - |\mu_R|R| - \mu_B|B| - k = |R| - ||h \cap R| - |h \cap B| - k|.$$

► **Lemma 9.** *LINE-COVERING is reducible to MAX-HALFSPACE in  $\mathbb{R}^2$  with  $\phi_{LC}$  in  $O(m \log m)$  time.*

**Proof.** Construct the lower envelope of all endpoints of the rays; this takes  $O(m \log m)$  time. Each upwards oriented ray is replaced with a point at its end point, and placed in  $R$ . Each downward oriented ray is replaced with two points: its endpoint generates a point in  $B$ , and where it intersects the lower envelope generates a point in  $R$ . See Figure 2.

Lines now correspond to halfplanes below those lines. Upward ray intersections require lines above them. Downward rays require lines between the corresponding  $B$  and  $R$  points: if a line is above both, they cancel in  $\phi_{LC}$ ; if it is below both, it includes neither; if a line is between them, it only identifies the  $R$  point. But the lines below both are below the lower envelope, and cannot be the optimal halfspace. Thus scanning the generated point set  $R \cup B$ , if it identifies a halfspace  $h$  where  $\Phi(h) > |R|$ , only then is LINE-COVERING satisfied. ◀

We can also reduce the exact version to the approximate version. If we run an  $\varepsilon$ -approximate MAX-HALFSPACE algorithm, and set  $\varepsilon = \frac{1}{2|R|}$  then the approximate range  $h'$  found will be off by at most a count of  $1/2$  from the optimal range, and hence must be the optimal solution.

► **Theorem 10.** *In  $\mathbb{R}^2$ ,  $\varepsilon$ -MAXHALFSPACE for  $\phi_{LC}$  takes  $\Omega(m + 1/\varepsilon^{2-o(1)})$  assuming the full input of size  $m$  needs to be read, and 3SUM requires  $\Omega(n^{2-o(1)})$  time.*

Since  $\phi_{LC}$  is concave and 1-Lipschitz, this implies any algorithm that works for all concave  $\phi$  or all 1-Lipschitz  $\phi$  must also take at least this long.

## 4.2 Lower Bound by All Pairs Shortest Path

We next provide a new construction that directly applies to the  $\phi_{||}$  function on weighted points in  $\mathbb{R}^2$  via a reduction from All Pairs Shortest Path (APSP). We first show the MAX-WEIGHT-3-CLIQUE problem reduces to APSP via another problem NEGATIVE-TRIANGLE.

- **Problem APSP:** *Given an edge-weighted undirected graph with  $n$  vertices, find the shortest path between every pair of vertices.*
- **Problem NEGATIVE-TRIANGLE:** *Given an edge-weighted undirected graph with  $n$  vertices, determine if there is a triangle with negative total edge weight.*
- **Problem MAX-WEIGHT- $K$ -CLIQUE:** *Given an edge-weighted undirected graph with  $n$  vertices, find the  $K$ -clique with the maximum total edge weight.*

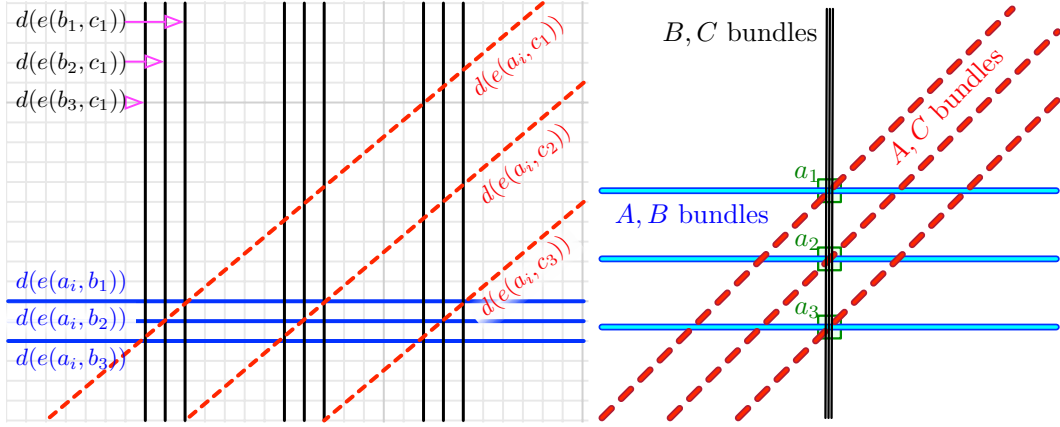
While the APSP is more well-known, Williams and Williams [41] showed it is equivalent to the NEGATIVE-TRIANGLE problem, which will be more useful. Both have well-known  $O(n^3)$  algorithms and are conjectured to not be solvable in less than  $O(n^{3-o(1)})$  time, and improving that bound on one would improve on the other. Moreover, Backurs *et al.* [6] used MAX-WEIGHT- $K$ -CLIQUE as a hard problem, believed to require  $O(n^{K-o(1)})$  time, for which to reduce to several other problems involving rectangles. They note that for  $K = 3$ , MAX-WEIGHT-3-CLIQUE is a special case of NEGATIVE-TRIANGLE. That is, for a guessed max-weight  $\omega$ , one can subtract  $\omega/3$  from each edge weight, then multiply all weights by  $-1$ . If there exists a negative triangle with the new weights, there exists a triangle with weight  $\omega$  in the original. One can resolve the max weight with logarithmically number of steps of binary search. Hence, NEGATIVE-TRIANGLE (and hence also APSP) reduces to MAX-WEIGHT-3-CLIQUE.

For MAX-WEIGHT-3-CLIQUE it is equivalent to assume a 3-partite graph  $G = (V, E)$  [6]; that is, the vertices  $V$  are the disjoint union of three independent sets  $A = \{a_1, a_2, \dots, a_n\}$ ,  $B = \{b_1, b_2, \dots, b_n\}$ , and  $C = \{c_1, c_2, \dots, c_n\}$ . Each independent set will have exactly  $n$  vertices and each vertex, for instance  $a_i$ , will have an edge with every vertex in  $B$  and  $C$ . Denote the edge between  $a_i$  and  $b_j$  as  $e(a_i, b_j)$  and the weight as  $w(e(a_i, b_j))$ . We reduce to a dual of halfspace scanning in  $\mathbb{R}^2$ :

- **Problem MAX-WEIGHT-POINT:** *Given  $m$  weighted lines find a point which maximizes the sum of weights of all lines passing below that point (or intersecting that point).*

Our reduction will rely on a planar geometric realization of any such graph  $G$  with  $m = O(n^2)$  lines, where the lines correspond to edges, and triple configurations of lines correspond with cliques. Given such an instance, if we can solve the MAX-HALFSPACE algorithm (in the dual as MAX-WEIGHT-POINT) in better than  $O(m^{\frac{3}{2}-o(1)})$  time we can recover the solution to the MAX-WEIGHT-3-CLIQUE problem in better than  $O(n^{3-o(1)})$  time.

**The double weighted line gadget.** Our full construction will use a special gadget which will ensure the max weighted point will correspond with a vertex at a triple intersection of a planar arrangement of lines, with each line corresponding with an edge in  $G$ . For this to hold we instantiate each edge as a *double line*  $d(e)$ . This consists of two parallel lines  $\ell_u(e)$  and



■ **Figure 3** MAX-WEIGHT-POINT construction with  $n = 3$ . Left: intersection of  $a_i$  blue and red bundles with all black double lines. Right: relation of the bundles, with 3  $a_i$  intersections marked.

$\ell_l(e)$ , separated vertically by a gap of a small positive value  $\alpha \ll 1$ , with  $\ell_u(e)$  above  $\ell_l(e)$ . In what follows we will refer to a double line as a single line; to be precise this will refer to a line at the midpoint between  $\ell_l$  and  $\ell_u$ . Let  $\bar{w} = 2 \max_{e \in E} |w(e)| + 1$  be a large enough value so for any  $e \in E$  that  $\bar{w} + w(e)$  is strictly positive, and any  $|w(e)| < \bar{w}/2$ . Then we denote the weight of the double line as  $w(d(e)) = w(e) + \bar{w}$ . This is transferred to the lines as  $w(\ell_u(e)) = -w(d(e))$  and  $w(\ell_l(e)) = w(d(e))$ .

Now given a query point  $x \in \mathbb{R}^2$  we say it is *on*  $d(e)$  if it lies between lines  $\ell_l(e)$  and  $\ell_u(e)$ , and *not on* otherwise. This will allow us to control the effect of  $e$  on query  $x$  in a precise way.

► **Lemma 11.** *Any point  $x$  that lies on  $d(e)$  will have weight contributed to it by  $e$  of exactly  $w(e) + \bar{w}$ ; otherwise that contribution will be 0.*

**Reduction to triple intersections.** Our construction will place double lines for each edge in the graph. The edges from  $A$  to  $B$  will be *blue* lines; from  $A$  to  $C$  will be *red* lines, and from  $B$  to  $C$  will be *black* lines. All lines of the same color will be parallel; this will ensure that any query point  $x \in \mathbb{R}^2$  can only be on one double line of each color. For easy of exposition, in our construction description (and illustration; see Figure 3) the black lines will be vertical, which makes ambiguous the “above” relation; so the final step will be to rotate the entire construction clockwise by a small angle (less than  $\pi/4$  radians).

The construction will now lay out the double lines so that every clique  $\{a_i, b_j, c_k\}$  will be realized as a triple intersection of double lines  $d(e(a_i, b_j))$ ,  $d(e(a_i, c_k))$ , and  $d(e(b_j, c_k))$  and so there are no other types of triple intersections. Such a triple intersection will have weight precisely  $w(e(a_i, b_j)) + w(e(a_i, c_k)) + w(e(b_j, c_k)) + 3\bar{w} > (3/2)\bar{w}$ , and any other point (e.g., a double intersection) must have weight strictly less than  $(3/2)\bar{w}$ .

► **Lemma 12.** *The maximum weight point must occur at a triple intersection of three double lines of different colors, and thus must correspond the max weight 3-clique.*

**The full construction.** The blue lines (using  $A$  to  $B$  edges) will all be horizontal. Edge  $e(a_i, b_j)$  will have  $y$ -coordinate  $y_{i,j} = -j - i(5n^2)$  so that  $y_{i,j+1} = y_{i,j} - 1$  and so  $y_{i+1,j} = y_{i,j} - 3n^2$ . Set  $y_{1,1} = -5n^2 - 1$ . This bundles the blue lines associated with the same  $a_i$  point. Figure 3 shows a single bundle (left) and structure of all bundles (right).



The red lines (using  $A$  to  $C$  edges) will be at a 45 degree angle (a slope of 1), similarly clustered by their  $c_k$  values. Double line  $d(e(a_i, c_k))$  will have equation  $\mathbf{y} = \mathbf{x} + o_{i,k}$ . We define the offsets  $o_{i,k} = -k(3n) - i(5n^2)$  so that  $o_{1,1} = -3n - 5n^2$ ;  $o_{i,k+1} = o_{i,k} - 3n$  and  $o_{i+1,k} = o_{i,k} - 5n^2$ . And in particular, double lines  $d(e(a_i, b_j))$  (at horizontal  $y_{i,j}$ ) and  $d(e(a_i, c_k))$  (at offset  $o_{i,k}$ ) will intersect at  $x$ -value  $x_{j,k} = y_{i,j} - o_{i,k}$ .

The black lines (using  $B$  and  $C$ ) will be vertical. Edge  $d(e(b_j, c_k))$  will have  $x$  coordinate  $x_{j,k}$ , defined  $x_{j,k} = y_{i,j} - o_{i,k} = -j + (3n)k$ . Also, these  $x_{j,k}$  values will be distinct for different values of  $j, k$ , but independent of the choice of  $a_i$ . Moreover, these are the same  $x$ -coordinates where the corresponding red and blue lines intersect. Thus each black line  $d(e(b_j, c_k))$  intersects the intersection of blue line  $d(e(a_i, b_j))$  and  $d(e(a_i, c_k))$  for each  $a_i$ .

Finally, note that all black lines are in the  $x$  range  $[-3n^2, 0]$  we can argue that they do not cause any other triple intersections. Because the each red bundle has offsets separated by more than  $3n^2$  then a red bundle associated with  $a_i$  cannot intersect a blue bundle associated  $a_{i'}$  for  $i \neq i'$  since the red lines have linear slope and the blue bundles are also separated by more than  $3n^2$ . Thus the intended triple intersections (of  $d(e(a_i, b_j))$ ,  $d(e(a_i, c_k))$ , and  $d(e(b_j, c_k))$ ) are the only ones in this construction.

In total there are  $n^2$  blue,  $n^2$  blue, and  $n^2$  black double lines, thus  $m = O(n^2)$ . Hence, MAX-WEIGHT-3-CLIQUE on  $n$  vertices reduces to MAX-HALFSPACE in time  $O(n^2)$ . Then reversing the dual mapping, we consider each dual line as two points, one in  $R$  and one in  $B$ , and this corresponds with the MAX-HALFSPACE problem in  $\mathbb{R}^2$  with  $\phi_{||}$ . Then since APSP reduces to MAX-WEIGHT-3-CLIQUE, we obtain the following theorem.

► **Theorem 13.** *In  $\mathbb{R}^2$ , MAX-HALFSPACE for  $\phi_{||}$  on  $m$  points requires  $\Omega(m^{3/2-o(1)})$  time assuming that APSP on  $n$  vertices requires  $\Omega(n^{3-o(1)})$  time.*

## 5 Conclusions and Discussion

We have mostly closed the planar  $\varepsilon$ -MAX-HALFSPACE problem with an  $\tilde{O}(m + 1/\varepsilon^d)$  algorithm in  $\mathbb{R}^d$  and conditional (to 3SUM)  $\Omega(m + 1/\varepsilon^{2-o(1)})$  lower bound in  $\mathbb{R}^2$ . However, the lower bound uses a piecewise-linear function  $\phi_{LC}$ , and while all known algorithmic improvements that depend on  $\phi$  take advantage of this linear structure (e.g., for rectangles [3, 28]), the function  $\phi_{LC}$ , perhaps strangely, is concave. Also surprisingly we can prove another conditional lower bound for MAX-HALFSPACE using a convex (in fact, again linear) function  $\phi$ , but this one is smaller at  $\Omega(m^{3/2-o(1)})$  in  $\mathbb{R}^2$ , and because some point may have very small weight in the construction, does not directly apply to  $\varepsilon$ -MAX-HALFSPACE, which allows  $\varepsilon$  additive error. Moreover, this reduces to APSP, which does not appear to be in the same class as 3SUM [40].

We leave several curious aspects to future work. Is there a real fine-grained complexity difference between the  $\phi$  variants in the problems? That is, can we improve upper bounds for convex  $\phi$ , or improve conditional lower bounds in this case? And can we condition the results on 3SUM in the convex  $\phi$  case? The convex  $\phi$  lower bound construction relies on weighted points, can we obtain improved algorithmic runtime by only allowing  $\{-1, +1\}$  weights? Moreover, are the polynomial terms  $(1/\varepsilon)^d$  in the algorithmic runtime correct in constant dimensions larger than  $d = 2$ ? And can these results help resolve polynomial terms in the high-dimensional robust statistics settings? An anonymous reviewer has suggested to use data structure for relative-error approximate counting on an  $\varepsilon$ -sample of halfspaces, and has argued that this can reduce the space requirements to  $o(1/\varepsilon^d)$ ; given our focus on the runtime of the maximum halfspace problem, we did not pursue this space improvement. If the runtime can be reduced to  $o(1/\varepsilon^d)$  clearly such a reduction in the space complexity would also be required.




## References

- 1 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *SODA*, 2009.
- 2 Deepak Agarwal, Andrew McGregor, Jeff M. Phillips, Suresh Venkatasubramanian, and Zhengyuan Zhu. Spatial scan statistics: Approximations and performance study. In *KDD*, 2006.
- 3 Deepak Agarwal, Jeff M. Phillips, and Suresh Venkatasubramanian. The hunting of the bump: On maximizing statistical discrepancy. In *SODA*, 2006.
- 4 Pankaj K. Agarwal. Simplex range searching. *Journey Through Discrete Mathematics*, pages 1–30, 2017.
- 5 Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SICOMP*, 38:899–921, 2008.
- 6 Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In *ICALP*, 2016.
- 7 Jérémy Barbay, Timothy M. Chan, Gonzalo Navarro, and Pablo Pérez-Lantero. Maximum-weight planar boxes in time (and better). *Information Processing Letters*, 114(8):437–445, 2014.
- 8 Mark De Berg and Otfried Schwarzkopf. Cuttings and applications. *International Journal of Computational Geometry and Applications*, 5:343–355, 1995.
- 9 Bernard Chazelle. Geometric discrepancy revisited. In *FOCS*, 1993.
- 10 Bernard Chazelle. *The Discrepancy Method*. Cambridge University Press, 2001.
- 11 Sitan Chen, Frederic Koehler, Ankur Moitra, and Morris Yau. Classification under misspecification: Halfspaces, generalized linear models, and connections to evolvability. *NeurIPS*, 2020. [arXiv:2006.04787](https://arxiv.org/abs/2006.04787).
- 12 Ilias Diakonikolas, Themis Gouleakis, and Christos Tzamos. Distribution-independent pac learning of halfspaces with massart noise. *arXiv preprint*, 2019. [arXiv:1906.10075](https://arxiv.org/abs/1906.10075).
- 13 Ilias Diakonikolas, Daniel M Kane, Vasilis Kontonis, Christos Tzamos, and Nikos Zarifis. Agnostic proper learning of halfspaces under gaussian marginals. *arXiv preprint*, 2021. [arXiv:2102.05629](https://arxiv.org/abs/2102.05629).
- 14 Ilias Diakonikolas, Daniel M Kane, and Nikos Zarifis. Near-optimal sq lower bounds for agnostically learning halfspaces and relus under gaussian marginals. *arXiv preprint*, 2020. [arXiv:2006.16200](https://arxiv.org/abs/2006.16200).
- 15 Ilias Diakonikolas, Vasilis Kontonis, Christos Tzamos, and Nikos Zarifis. Learning halfspaces with tsybakov noise. *arXiv preprint*, 2020. [arXiv:2006.06467](https://arxiv.org/abs/2006.06467).
- 16 David Dobkin and David Eppstein. Computing the discrepancy. In *Proceedings 9th Annual Symposium on Computational Geometry*, 1993.
- 17 David P. Dobkin, David Eppstein, and Don P. Mitchell. Computing the discrepancy with applications to supersampling patterns. *ACM Transactions on Graphics*, 15:354–376, 1996.
- 18 Anka Gajentaan and Mark H. Overmars. On a class of  $o(n^2)$  problems in computational geometry. *Computational Geometry*, 5:165–185, 1995.
- 19 Sariel Har-Peled. *Geometric Approximation Algorithms*. AMS, 2011.
- 20 David Haussler and Emo Welzl. epsilon-nets and simplex range queries. *Discrete and Computational Geometry*, 2:127–151, 1987.
- 21 Lan Huang, Martin Kulldorff, and David Gregorio. A spatial scan statistic for survival data. *BioMetrics*, 63:109–118, 2007.
- 22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 3SUM hardness in (dynamic) data structures. Technical report, arXiv, 2014. [arXiv:1407.6756](https://arxiv.org/abs/1407.6756).
- 23 Martin Kulldorff. A spatial scan statistic. *Communications in Statistics: Theory and Methods*, 26:1481–1496, 1997.
- 24 Martin Kulldorff. *SatScan User Guide*, 7.0 edition, 2006. URL: <http://www.satscan.org/>.
- 25 Yi Li, Philip M. Long, and Aravind Srinivasan. Improved bounds on the samples complexity of learning. *J. Comp. and Sys. Sci.*, 62:516–527, 2001.



- 26 Ming C. Lin and Dinesh Manocha. Applied computational geometry. towards geometric engineering: Selected papers. *Springer Science & Business Media*, 114, 1996.
- 27 Chi-Yuan Lo, Jirka Matousek, and William Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11:433–452, 1994.
- 28 Michael Matheny and Jeff M. Phillips. Computing approximate statistical discrepancy. In *International Symposium on Algorithm and Computation*, 2018.
- 29 Michael Matheny and Jeff M. Phillips. Practical low-dimensional halfspace range space sampling. In *European Symposium on Algorithms*, 2018.
- 30 Michael Matheny, Raghvendra Singh, Liang Zhang, Kaiqiang Wang, and Jeff M. Phillips. Scalable spatial scan statistics through sampling. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2016.
- 31 Jiri Matousek. Approximations and optimal geometric divide-and-conquer. In *Proceedings 23rd Symposium on Theory of Computing*, pages 505–511, 1991.
- 32 Jiri Matousek. *Geometric Discrepancy*. Springer, 1999.
- 33 Daniel B. Neill and Andrew W. Moore. Rapid detection of significant spatial clusters. In *KDD*, 2004.
- 34 Tan Nguyen and Scott Sanner. Algorithms for direct 0–1 loss optimization in binary classification. In *International Conference on Machine Learning*, 2013.
- 35 Saladi Rahul. Approximate range counting revisited. In *SoCG*, 2017.
- 36 Lev Reyzin. Statistical queries and statistical algorithms: Foundations and applications. *arXiv preprint*, 2020. [arXiv:2004.00557](https://arxiv.org/abs/2004.00557).
- 37 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13:145–147, 1972.
- 38 Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theo. of Prob and App*, 16:264–280, 1971.
- 39 Zhewei Wei and Ke Yi. Tight space bounds for two-dimensional approximate range counting. *ACM Transactions on Algorithms (TALG)*, 14(2):1–17, 2018.
- 40 Virginia Vassilevska Williams. Some open problems in fine-grained complexity. *ACM SIGACTT News*, 49:29–35, 2018.
- 41 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of ACM*, 2018.



# The VC-Dimension of Limited Visibility Terrains

Matt Gibson-Lopez   

The University of Texas at San Antonio, TX, USA

Zhongxiu Yang  

The University of Texas at San Antonio, TX, USA

---

## Abstract

Visibility problems are fundamental to computational geometry, and many versions of geometric set cover where coverage is based on visibility have been considered. In most settings, points can see “infinitely far” so long as visibility is not “blocked” by some obstacle. In many applications, this may be an unreasonable assumption. In this paper, we consider a new model of visibility where no point can see any other point beyond a sight radius  $\rho$ . In particular, we consider this visibility model in the context of terrains. We show that the VC-dimension of limited visibility terrains is exactly 7. We give lower bound construction that shatters a set of 7 points, and we prove that shattering 8 points is not possible.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases** VC-dimension, Terrain Guarding, Limited Visibility

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.5

## Supplementary Material

*Software (Source Code for Lower Bound):* <https://github.com/utsa-saga/vc7proof>  
archived at `swh:1:dir:33dfe57e9e3cb7a1fad1a49f6bd65a0fa1e32a67`

*Software (Source Code for Upper Bound):* <https://github.com/utsa-saga/vc8proof>  
archived at `swh:1:dir:bffd67e203db12523de2cc9f76baa84d3b6046d0`

**Funding** Supported by the National Science Foundation under Grant No. 1733874.

## 1 Introduction

Visibility is fundamental to computational geometry, and in particular, variants of geometric set cover where coverage is determined by what a point “sees” have been widely considered in the literature. One of the most famous such applications is the *art gallery problem* where we are given a simple polygon  $P$  and we wish to place the minimum number of “guard” points  $G$  such that each point inside  $P$  is seen by some guard in  $G$ . Usually, the way that visibility is defined, is that two points  $p$  and  $q$  see each other if the line segment  $\overline{pq}$  does not contain any point outside of  $P$ . The intuition is that  $P$  is modelling some room (or art gallery), and if the line segment connecting the two points exits  $P$ , then there is a wall of the polygon that “blocks” their vision. The geometric set cover problem with respect to this definition of visibility has been considered in many different contexts, for example simple polygons [9, 15, 8, 1, 13], monotone polygons [20, 22, 12, 21, 14], rectilinear polygons [21, 22], staircase polygons [4, 23, 25], and terrains [18, 17, 2, 16, 6, 11, 19].

A *terrain*  $T$  is an  $x$ -monotone polygonal chain defined by a sequence of  $n$  vertices  $p_1, \dots, p_n$ . We let  $x(p_i)$  and  $y(p_i)$  denote the coordinates of  $p_i$ . The  $x$ -monotone property implies that  $x(p_i) < x(p_{i+1})$  for each  $i \in \{1, \dots, n-1\}$ . Note that given this definition, a vertical line intersects  $T$  in at most one point. Consider three vertices of  $T$ ,  $p_i, p_j$ , and  $p_k$  such that  $i < j < k$ . We say  $p_j$  *pierces*  $\overline{p_i p_k}$  if  $p_j$  is strictly above  $\overline{p_i p_k}$ . Similarly, if there is some  $p_j$  that pierces  $\overline{p_i p_k}$  then we say that  $\overline{p_i p_k}$  is *pierced*. Generally visibility in terrains is defined in the following way:  $p_i$  and  $p_k$  see each other if and only if  $\overline{p_i p_k}$  is not pierced.



© Matt Gibson-Lopez and Zhongxiu Yang;

licensed under Creative Commons License CC-BY 4.0

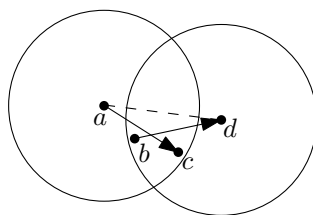
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 5; pp. 5:1–5:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1**  $a$  sees  $c$ .  $b$  sees  $d$ . In an LVT,  $a$  may be too far from  $d$  to see it.

## 1.1 A Restricted Model of Visibility

An issue with the usual visibility model for various applications is that points are assumed to be able to see “infinitely far”. That is, a guard that is very far from  $p$  is assumed to see  $p$  just as clearly as another guard that is very close to  $p$ . Distance does not matter so long as the line segment connecting the points is not “blocked”. In many applications, this assumption is not true, and a model that considers the distance between a guard and the points it sees may be desirable. To address this undesirability, there has been some research that has considered restricted visibility models for exploring/guarding polygons or polygonal environments. Much of this past work is motivated by robotics, often times in an online setting where a robot is trying to learn about its surroundings but has restrictions on how far it can see, e.g. [3, 10, 24]. Other past work has considered restricted visibility models when computing an optimal “watchman tour” where a single guard patrols a known polygon, e.g. [26, 7].

In this paper we consider a *limited visibility* model of the terrain guarding problem, where each point has a sight radius  $\rho$  and cannot see any points that are outside of its sight radius. So to contrast with most of the past work with limited visibility models, we are assuming that we have full knowledge of the terrain we wish to guard, we can use multiple guards, and the guards are static. We call an instance of this problem a *limited visibility terrain* (LVT). An LVT can be defined by a set of  $n$ ,  $x$ -monotone vertices and a sight radius  $\rho$ . We say that two points  $p$  and  $q$  see each other if and only if (1)  $\overline{pq}$  is not pierced and (2)  $d(p, q) \leq \rho$ , where  $d(p, q)$  is the Euclidean distance between  $p$  and  $q$ . In this paper, we assume each point has the same sight radius, although it would be interesting to also consider the scenario where each point has its own radius. For any point  $p$ , we denote  $disc(p)$  to be a disc of radius  $\rho$  centered at  $p$ . Then clearly  $p$  cannot see any points that are not inside  $disc(p)$  in this model.

## 1.2 Order Claim Generalization

One of the key properties of (regular) terrains is the order claim.

▷ **Claim 1.** Let  $a, b, c, d$  be four points on a regular terrain such that  $x(a) < x(b) < x(c) < x(d)$ . If (1)  $a$  sees  $c$  and (2)  $b$  sees  $d$ , then  $a$  sees  $d$ .

The intuition here is pretty simple. Since  $a$  sees  $c$ , it must be that  $\overline{ac}$  is not pierced (so  $b$  is “below”  $\overline{ac}$ ). Similarly  $\overline{bd}$  is not pierced (so  $c$  is below  $\overline{bd}$ ). This implies that  $\overline{ad}$  is “above”  $\overline{ac}$  and  $\overline{bd}$ , and therefore  $\overline{ad}$  is not pierced (implying that  $a$  and  $d$  see each other).

For LVTs the order claim does not hold if  $a \notin disc(d)$ . See Figure 1 for an illustration. We can however apply a generalization of the order claim.

▷ **Claim 2.** Let  $a, b, c, d$  be four points on an LVT such that  $x(a) < x(b) < x(c) < x(d)$ . If (1)  $\overline{ac}$  is not pierced and (2)  $\overline{bd}$  is not pierced, then  $\overline{ad}$  is not pierced.

### 1.3 VC-Dimension

An interesting measure of the complexity of a set system is the notion of VC-dimension. To define this, we say that a finite set of points  $G$  in  $T$  is *shattered* if for every subset of  $G' \subseteq G$  there exists some point  $v \in T$  such that  $v$  sees every point in  $G'$  and does not see any point in  $G \setminus G'$ . In this context, we call  $v$  a *viewpoint*. The VC-dimension is the largest  $d$  such that there exists some LVT  $T$  and point set  $G$  of size  $d$  that can be shattered.

Brönnimann and Goodrich give a polynomial-time  $O(\log OPT)$ -approximation algorithm for any set system with constant VC-dimension [5] where  $OPT$  is the size of an optimal cover. For regular terrains, King showed that the VC-dimension is 4 [17]. This result relies heavily on the order claim, which as we mentioned above does not hold in the limited visibility model. One would certainly expect the VC-dimension to increase in this model.

### 1.4 Our Results

We prove the following theorem.

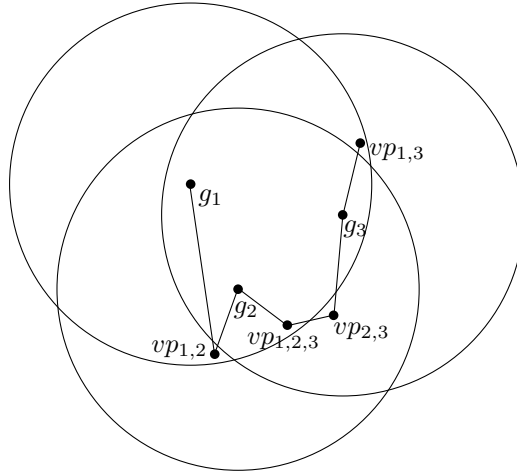
► **Theorem 3.** *The VC-Dimension of limited visibility terrains is 7.*

We show that for any set of 8 points on a LVT, it is not possible to shatter the set, and we show that it is possible to shatter a set of 7 points (thereby proving that our upper bound analysis is tight). We show several key structural lemmas about visibility in a LVT, and then use a computer program to show that there is no permutation of the 8 points and the  $2^8$  needed viewpoints such that there is a LVT with the points in left-to-right order according to the permutation that satisfies all of the visibility requirements. As is often the case with VC-dimension proofs, a direct proof often involves a tedious case analysis, so we use a computer program that is much easier to verify is correct rather than having to analyze each case “manually”. Our proof of Theorem 3 then implies that the algorithm of Brönnimann and Goodrich [5] is a  $O(\log OPT)$ -approximation algorithm for guarding the vertices of a LVT.

## 2 Upper Bound

In this section, we show that the VC-dimension of LVTs is at most 7. We will do this by showing that any set of 8 points cannot be shattered. Suppose that it is possible to shatter 8 points, that is there exists a LVT that shatters a set  $G = \{g_1, \dots, g_8\}$  which are indexed according to increasing x-coordinate without loss of generality. Motivated by the connection to geometric set cover, we sometimes call the points of  $G$  *guards*. For any subset  $S \subseteq \{1, \dots, 8\}$ , let  $vp_S$  denote a point that should see  $g_i$  for each  $i \in S$  and does not see  $g_j$  for any  $j \notin S$ . For example,  $vp_{1,6,8}$  should see  $g_1, g_6$ , and  $g_8$  and should not see any other point of  $G$ . Let  $V$  be the set of all such viewpoints. Clearly  $|V| = 2^8 = 256$ .

We begin with a high-level overview of our proof strategy for showing that  $G$  cannot be shattered. Suppose the contrary, and let  $T$  denote a LVT that does shatter  $G$ . We say the *ordering* of  $T$  is the permutation of  $G \cup V$  when sorting  $G \cup V$  by their x-coordinates (note  $T$  may have vertices that do not correspond with  $G$  or  $V$ ). We say any subsequence of the ordering is a *partial ordering* of  $T$ . The intuition is that an ordering describes the left-to-right order of the points of  $V$  and  $G$  with respect to  $T$ , and a partial ordering describes the left-to-right ordering of *some* of the points of  $V$  and  $G$ . See Figure 2 for an illustration.



■ **Figure 2** The ordering of  $T$  is  $(g_1, vp_{1,2}, g_2, vp_{1,2,3}, vp_{2,3}, g_3, vp_{1,3})$ . Partial orderings of  $T$  include  $(g_1, g_2, vp_{2,3}, g_3)$ ,  $(vp_{1,2}, vp_{1,2,3}, g_3, vp_{1,3})$ , etc.

Now we view the problem in the “opposite direction”; instead of starting with a LVT and considering its order, we start with an order and consider whether there is a LVT that has this order. We call any permutation  $O$  of  $V \cup G$  a *candidate ordering* (CO) of  $V \cup G$ . We say  $O$  is *realizable* if there is a LVT  $T$  that shatters  $G$  such that  $O$  is the ordering of  $T$ . For every subset  $V' \subseteq V$ , we call a permutation of  $V' \cup G$  a *candidate partial ordering* (CPO), and we say it is realizable if it is a partial ordering of any LVT  $T$  that shatters  $G$ .

We obtain our result by showing that there is no realizable CPO. We do this by first proving a set of structural lemmas. Given a CPO  $O$ , the lemmas will imply some properties that must be satisfied by *any* realization of  $O$ . For example, we might be able to argue that any realization of  $O$  must satisfy  $y(g_i) < y(g_j)$ , or we might be able to argue that a point  $v \in V$  that is not supposed to see  $g_i$  must be outside the disc of  $g_i$  or  $\overline{vg_i}$  must be pierced in any realization. If the lemmas contradict each other then we can determine that  $O$  is not realizable.

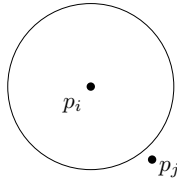
The proofs of VC-dimension upper bounds are often tedious case analyses. We instead use a computer program that helps to automate this case analysis. Our program, given a CPO  $O$ , determines if our lemmas can be used to show that  $O$  is not realizable. The CPOs we give as inputs to the program are all of the CPOs based off  $G$  and four points of  $V$ . Each of the CPOs can be analyzed in parallel, and therefore picking four points of  $V$  provides a nice tradeoff between the number of CPOs that must be considered and the amount of time it takes to evaluate a single CPO (the time to evaluate all cases on a 3.7 GHz CPU with 6 cores is less than 24 hours).

## 2.1 Structural Lemmas

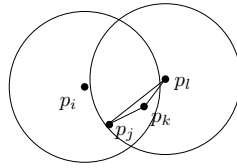
We now give a set of structural lemmas that will help us argue when a CPO is not realizable. We use the following definitions. Suppose that  $p_i$  and  $p_j$  are two points on an LVT such that  $d(p_i, p_j) \geq \rho$  and  $|x(p_i) - x(p_j)| < \rho$  (note that this implies that  $y(p_i) \neq y(p_j)$ ). Then if  $y(p_i) > y(p_j)$  we say that  $p_i$  is *high outside disc*( $p_j$ ), and similarly we say that  $p_j$  is *low outside disc*( $p_i$ ). See Figure 3.

► **Lemma 4.** *Let  $p_i, p_j, p_k$ , and  $p_l$  be four points on a LVT such that  $x(p_i) < x(p_j) < x(p_k) < x(p_l)$ . If (1)  $p_i$  sees  $p_k$  and (2)  $p_j$  sees  $p_l$ , then  $d(p_j, p_k) \leq \rho$ .*

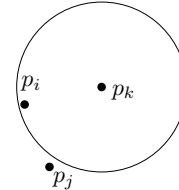




■ **Figure 3**  $p_j$  is low outside of  $disc(p_i)$ .



■ **Figure 4**  $d(p_j, p_k) \leq \rho$ .



■ **Figure 5**  $y(p_j) < y(p_i)$  and  $y(p_j) < y(p_k)$ .

**Proof.** Without loss of generality, assume that  $y(p_j) \leq y(p_k)$ . Then it must be that  $y(p_k) \leq y(p_l)$ , or else  $p_k$  would pierce  $\overline{p_j p_l}$ . Consider the triangle  $\triangle p_j p_k p_l$ . We have  $\angle p_j p_k p_l > \frac{\pi}{2}$ , thus  $\overline{p_j p_l}$  is the longest side of the triangle. Then we have  $d(p_j, p_k) < d(p_j, p_l) \leq \rho$ . See Figure 4 for illustration. ◀

► **Lemma 5.** Let  $p_i, p_j$ , and  $p_k$  be three points on a LVT such that  $x(p_i) < x(p_j) < x(p_k)$ . If (1)  $p_k$  sees  $p_i$  but does not see  $p_j$  and (2)  $\overline{p_j p_k}$  is not pierced, then  $p_j$  is low outside  $disc(p_k)$ . Moreover, we also have  $y(p_j) < y(p_i)$ .

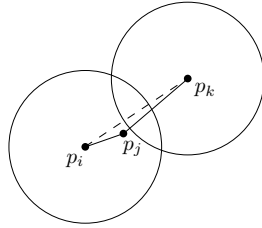
**Proof.** See Figure 5 for illustration. Since  $\overline{p_j p_k}$  is not pierced and they do not see each other, then clearly  $d(p_j, p_k) > \rho$ . Since  $p_i \in disc(p_k)$ , we must have  $x(p_k) - x(p_i) \leq \rho$ , and since  $x(p_j)$  is “between”  $x(p_i)$  and  $x(p_k)$  it therefore must be that  $x(p_k) - x(p_j) < \rho$ . So it remains to show that  $y(p_j) < y(p_k)$  and  $y(p_j) < y(p_i)$ . For the sake of contradiction, suppose that  $y(p_j) \geq y(p_i)$ , clearly  $y(p_k) > y(p_j)$ , otherwise  $p_j$  will pierce  $\overline{p_i p_k}$ . Consider the  $\triangle p_i p_j p_k$ . We have  $\angle p_i p_j p_k$  is greater than  $\frac{\pi}{2}$ , see Figure 6 for illustration. Therefore  $d(p_i, p_k) > d(p_j, p_k) > \rho$ , contradicting that  $p_i$  sees  $p_k$ . Thus the initial assumption  $y(p_j) \geq y(p_i)$  is wrong. We can derive a contradiction for  $y(p_j) \geq y(p_k)$  similarly. ◀

► **Lemma 6.** Let  $p_i, p_j$ , and  $p_k$  be three points on a LVT such that  $x(p_i) < x(p_j) < x(p_k)$ . If (1)  $y(p_i) < y(p_j)$  and  $d(p_i, p_j) > \rho$ , and (2)  $\overline{p_i p_k}$  is not pierced, then  $y(p_j) < y(p_k)$  and  $d(p_i, p_k) > \rho$ .

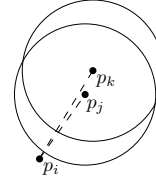
**Proof.** Clearly we must have  $y(p_j) < y(p_k)$  or else  $p_j$  would pierce  $\overline{p_i p_k}$ . See Figure 7 for illustration. In  $\triangle p_i p_j p_k$ , clearly the longest edge is  $\overline{p_i p_k}$ , thus  $d(p_i, p_k) > d(p_i, p_j) > \rho$ . ◀

► **Lemma 7.** Let  $p_i$  and  $p_k$  be two points on a LVT that that do not see each other, but  $d(p_i, p_k) \leq \rho$ . Then there must be some point  $p_j \in (p_i, p_k)$  that pierces  $\overline{p_i p_k}$ . If there is some point  $p_l$  such that  $d(p_i, p_l) \leq \rho$  and  $d(p_k, p_l) \leq \rho$ , then either  $d(p_j, p_l) \leq \rho$  or  $p_j$  is high outside  $disc(p_l)$ .

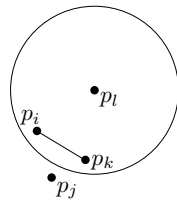
**Proof.** It must be that  $x(p_l) - x(p_i) \leq \rho$ , and since  $p_j$  is “between”  $p_i$  and  $p_k$  then we have  $x(p_l) - x(p_j) < \rho$ . We then complete the lemma by showing that if  $p_j$  is such that  $d(p_j, p_l) > \rho$  and  $y(p_j) < y(p_l)$  then it cannot pierce  $\overline{p_i p_k}$ . In this case,  $p_j$  must be low outside  $disc(p_l)$ . By convexity, we have that  $\overline{p_i p_k}$  is contained inside of  $disc(p_l)$ , and therefore it cannot be that  $p_j$  pierces  $\overline{p_i p_k}$ , see Figure 8 for illustration. ◀



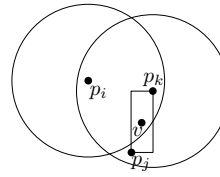
■ **Figure 6** Contradiction with  $p_i$  seeing  $p_k$ .



■ **Figure 7**  $y(p_i) < y(p_k)$ .



■ **Figure 8** Contradiction with  $d(p_j, p_l) > \rho$  and  $y(p_j) < y(p_l)$ .



■ **Figure 9** Contradiction with  $d(v, p_k) \leq \rho$ .

► **Lemma 8.** *Let  $p_i, p_j$ , and  $p_k$  be three points on an LVT such that  $x(p_i) < x(p_j) < x(p_k)$  such that  $p_j$  is low outside  $\text{disc}(p_i)$  and  $d(p_j, p_k) \leq \rho$ . Then there is no point  $v : x(p_j) < x(v) < x(p_k)$  such that  $v$  is low outside  $\text{disc}(p_k)$  and  $d(p_i, v) \leq \rho$ .*

**Proof.** If the lemma is incorrect, then there would be a  $v$  that satisfies the following three properties: (1)  $y(v) < y(p_k)$ , (2)  $d(v, p_k) > \rho$ , and (3)  $d(p_i, v) \leq \rho$ . We will prove the lemma by showing that any point that satisfies conditions (1) and (3) cannot also satisfy condition (2). To that end, let  $v$  be any point satisfying  $y(v) < y(p_k)$  and  $d(p_i, v) \leq \rho$ . We will show that it must be such that  $d(v, p_k) \leq \rho$ . See Figure 9 for illustration. Given  $x(p_i) < x(p_j) < x(v)$ ,  $d(p_i, p_j) > \rho$ ,  $d(p_i, v) \leq \rho$ , from Lemma 5 clearly  $y(v) > y(p_j)$ . Since  $y(v) < y(p_k)$ , then we have  $x(p_j) < x(v) < x(p_k)$  and  $y(p_j) < y(v) < y(p_k)$ . This implies that  $v$  is inside of the axis-parallel rectangle with  $p_j$  as lower-left corner and  $p_k$  as upper-right corner. If  $a$  and  $b$  are any two points in this rectangle, then we have  $d(a, b) \leq d(p_j, p_k) \leq \rho$ , and therefore it must be that  $d(v, p_k) \leq \rho$ . ◀

## 2.2 Procedure

We use a computer program to show that there does not exist a realizable CO of  $G \cup V$ . Our strategy is to consider a starting set of viewpoints  $V_{start} := \{vp_{1,3,5,7,8}, vp_{1,2,4,6,8}, vp_{1,3,6,8}, vp_{1,2,4,5,7,8}\}$ , and to show that every CPO of  $G \cup V_{start}$  is not realizable. This implies there is no realizable CO of  $G \cup V$ . If not, then consider the subsequence of a realizable order when considering the vertices in  $G \cup V_{start}$ . This would be a realizable CPO of  $G \cup V_{start}$ .

Suppose  $O$  is some CPO of  $G \cup V_{start}$  that we are considering. It may be that our lemmas can “directly” show that  $O$  is not realizable, that is we can apply the lemmas to the points in  $O$  to derive a contradiction. In fact, this is true for roughly half of the CPOs of  $G \cup V_{start}$  (which is why we pick this set as a starting point). If we cannot directly show  $O$  is not

realizable, we consider *extensions* of  $O$ . That is, we consider a viewpoint  $v$  that is not in  $O$ , and we consider adding  $v$  to  $O$  in one of the  $|O| + 1$  “gaps” in  $O$ . Intuitively, we add one more viewpoint to  $O$  to obtain a new CPO  $O'$  while ensuring that the order of the points in  $O$  is the same in  $O'$ . Then we say that  $O'$  is an extension of  $O$ . Using this terminology, note that  $O$  is realizable if and only if there is at least one realizable extension of  $O$ . Similarly, if we are considering a CPO  $O$  of  $G \cup V'$  and there is a viewpoint  $v \in V \setminus V'$  such that our lemmas imply a contradiction for *every* extension of  $O$  that includes  $v$ , then we can in turn say that  $O$  is not realizable. When considering viewpoints for extensions, we restrict ourselves to the set  $V_{test}$  which is the set of all 96 viewpoints such satisfies either (1) they see both  $g_1$  and  $g_8$ , or (2) see  $g_2$  and  $g_7$  and exactly one of  $g_1$  and  $g_8$ . The intuition is that  $V_{test}$  consist of the more “difficult” viewpoints to find a spot in an order, so by only considering extensions from  $V_{test}$  we can focus the search to “difficult” viewpoints only, speeding up our program. And indeed it is sufficient to consider only these viewpoints given that our program verifies that no CPO of  $G \cup V_{start}$  is realizable, even if we only needed to extend to  $G \cup V_{test}$ .

Our program consists of two main functions: *isNotRealizable()* (Algorithm 1 and 2 in the paper due to the length of the algorithm) and *candidateOrderingIsExtendable()* (Algorithm 3 in the paper). *isNotRealizable()* takes a CPO as input and returns true if our lemmas can directly show that the order is not realizable, and returns false otherwise. *candidateOrderingIsExtendable()* takes a CPO as input and returns true if there is an extension such that the lemmas cannot directly show it is not realizable, and otherwise returns false. In this paper, we give color-coded pseudocode that explains what our program does. The actual C++ source code that we use as well as a color-coded rich text version of the source code is provided at <https://github.com/utsa-saga/vc8proof> (the colors are used to help with the readability of the code, pairing the “sections” of the source code with the “sections” of the pseudocode from the paper).

Given 4 fixed starting viewpoints  $V_{start}$  and guard set  $G = \{g_1, g_2, \dots, g_8\}$ , we consider all 11,880 CPOs of  $G \cup V_{start}$ . Our program handles each of these 11,880 orderings independently (cases can be tested simultaneously in parallel). Let  $O_1$  denote one such ordering. We pass  $O_1$  to *candidateOrderingIsExtendable()*. This function calls *isNotRealizable()* on  $O_1$ . If we determine that it is not realizable, then *candidateOrderingIsExtendable()* returns false immediately. If we cannot determine that  $O_1$  is not realizable, then we add a new viewpoint from  $V_{test}$  to  $O_1$  to obtain a new candidate ordering and repeat until we find a candidate ordering of  $G \cup V_{test}$  for which *isNotRealizable()* returns false (indicating that it might be realizable) or determine that every candidate ordering of  $G \cup V_{test}$  extending from  $O_1$  is not realizable. In our actual program, we use some heuristics for computing a “good” ordering of the points in  $V_{test} \setminus O_i$  that are not shown in *candidateOrderingIsExtendable()*.

It is not too difficult to see that *candidateOrderingIsExtendable()* is correct given that *isNotRealizable()* is correct. We will formally show that *isNotRealizable()* is correct. That is, we show that if *isNotRealizable()* returns true for some candidate ordering  $O_i$ , then a contradiction about any realization can be derived from our lemmas. To help show the correctness of this function, we define some notation. We use three main variables to detect lemma contradictions: `cannotBlockWithTerrain[{u, v}]`, `tooFarAway[{u, v}]`, and `higherPoint[{u, v}]`.

The first variable is set to true if it must be that every realization must satisfy that  $\overline{uv}$  is not pierced (following from the order claim generalization that we presented in Section 1.2. We consider this variable for all pairs  $u, v \in O$ . The second variable is set to be true when every realization (according to our lemmas) must satisfy that  $d(u, v) > \rho$ , it is set to false if  $d(u, v) \leq \rho$  in every realization, and it is set to unknown if our lemmas do not imply this one

way or the other. We consider this variable for  $u, v$  pairs such that at least one of  $u$  and  $v$  is in  $G$ . As for the third variable, it will be set to either  $u$ ,  $v$ , or unknown depending on if we can determine which point must have the larger y-coordinate in any realization according to our lemmas. This again is defined for all pairs  $u, v$  pairs such that at least one of  $u$  and  $v$  is in  $G$ .

In order to determine the values of variables 2 and 3, we define some extra notation and variables that are used as “helpers”. Let  $O$  be any candidate ordering of  $G \cup V'$ , where  $V'$  is any subset of  $V$ . For ease of description, we add to  $O$  a “dummy” point  $vp_{-\infty}$  to the left of every point of  $O$  and a “dummy” point  $vp_{\infty}$  to the right of every point of  $O$ . Since we have a fixed ordering of the points, we can determine for any two points of  $u, v \in O$  whether or not  $u$  and  $v$  are can be pierced by applying the order claim generalization we presented in Section 1.2. For any guard  $g \in G$ , let  $L(g)$  denote all viewpoints to the left of  $g$  in  $O$ , and similarly let  $R(g)$  denote all viewpoints to the right of  $g$  in  $O$ . Let  $leftMostPointGuardSees(g)$  denote the leftmost point of  $L(g)$  that sees  $g$ . If there is no such point, then we define it to be  $g$ . Similarly we let  $rightMostPointGuardSees(g)$  denote the rightmost point in  $R(g)$  that sees  $g$ , and if no such point exists we set it to be  $g$ . Let  $closestHighOutsideLeft(g)$  denote the rightmost point  $v \in L(G)$  such that  $g$  does not see  $v$ ,  $cannotBlockWithTerrain[\{g, v\}]$  is true, and  $v$  sees some  $g' \in G$  right of  $g$  (our lemmas imply this view point must be high outside  $disc(g)$ ). If no such point exists, we let  $closestHighOutsideLeft(g)$  be  $vp_{-\infty}$ . We similarly define  $closestHighOutsideRight(g)$  to be the leftmost point  $v \in R(g)$  such that  $g$  does not see  $v$ ,  $cannotBlockWithTerrain[\{g, v\}]$  is true, and  $v$  sees some  $g' \in G$  left of  $g$ . If no such point exists, we define  $closestHighOutsideRight(g)$  to be  $vp_{\infty}$ . For any two points  $u, v \in O$ , we say  $u < v$  if  $u$  is to the left of  $v$  in  $O$ .

► **Lemma 9.** *If our function  $isNotRealizable(O)$  returns true, then  $O$  is not realizable.*

**Proof.** We will show a contradiction to realization for each line of  $isNotRealizable()$  that returns true. To this end, we will also show that the assignments for our variables  $cannotBlockWithTerrain$ ,  $tooFarAway$ , and  $highestPoint$  are in fact correct.

**Gold Block.** In the gold block (Line 1), we set the values for the variable  $cannotBlockWithTerrain$ . If for points  $a$  and  $d$  we have  $cannotBlockWithTerrain[\{a, d\}]$  set to true, then it is true that in any realization of  $O$  that  $\overline{ad}$  is not pierced as either they see each other or there are points  $b$  and  $c$  such that  $\overline{ac}$  is not pierced and  $\overline{ad}$  is not pierced (second Claim from Section 1.2). If  $cannotBlockWithTerrain[\{a, d\}]$  is false then we make no assumptions about whether  $\overline{ad}$  is pierced.

**Green Block.** Now consider the green block (Lines 2–3), and in particular suppose we return true in line 3. Without loss of generality assume that it returned true because  $leftMostPointGuardSees(g) < closestHighOutsideLeft(g)$ . Let  $p$  denote  $leftMostPointGuardSees(g)$ , and let  $q$  denote  $closestHighOutsideLeft(g)$ . Then  $g$  does not see  $q$  and  $\overline{qg}$  cannot be pierced which implies that in any realization we must have  $d(q, g) > \rho$ . But we can apply Lemma 4 with  $p_i, p_j, p_k,$  and  $p_l$  as  $p, q, g, g'$  respectively, where  $g'$  is a guard that  $q$  sees to the right of  $g$  (which exists by the definition of  $closestHighOutsideLeft(g)$ ). Then in any realization,  $d(q, g) \leq \rho$  by Lemma 4, a contradiction.

We also remark that  $closestHighOutsideRight(g)$  must indeed be high outside  $disc(g)$  in any realization. This follows from Lemma 5 by letting  $p_k$  be  $closestHighOutsideRight(g)$ ,  $p_j$  be  $g$ , and  $p_i$  be any guard that  $closestHighOutsideRight(g)$  sees to the left of  $g$ . Similarly,  $closestHighOutsideLeft(g)$  must be high outside of  $disc(g)$ .

**Blue Block.** Now consider the blue block of code (Lines 4–23) where we set the `tooFarAway` and `higherPoint` variables for each pair of guards. We claim that if `tooFarAway[{gi, gj}]` is set to true, then the lemmas imply that any realization must have  $d(g_i, g_j) > \rho$ , and if it is set to false then every realization must have  $d(g_i, g_j) \leq \rho$ . Similarly we claim that if we set `higherPoint[{gi, gj}]` to be  $g_i$ , then any realization must have  $y(g_i) > y(g_j)$  and if it is set to be  $g_j$  then any realization must have  $y(g_i) < y(g_j)$ . We initially set both variables to unknown, and then we change them if we detect a reason to change them.

Suppose we set `tooFarAway[{gi, gj}]` to be false in Line 7. Then let  $p$  be `leftMostPointGuardSees(gj)` and let  $q$  be `rightMostPointGuardSees(gi)`. Then from the if-statement, we have that  $p < g_i < g_j < q$  and applying Lemma 4 we have that  $d(g_i, g_j) \leq \rho$  in any realization, therefore our variable setting is correct.

Now suppose that we set `higherPoint[{gi, gj}]` to be  $g_i$  in Line 9. If `closestHighOutsideRight(gj)` and `rightMostPointGuardSees(gi)` are the same point, then we have that  $y(g_i) > y(g_j)$  in any realization by Lemma 5. If they are not the same point, then we have  $g_i < g_j < p < q$  where  $p$  is `closestHighOutsideRight(gj)` and  $q$  is `rightMostPointGuardSees(gi)`. If we had  $y(g_i) \leq y(g_j)$  then it would have to be that  $d(g_i, p) > \rho$  and  $y(g_i) < y(p)$  and therefore we can apply Lemma 6 to  $g_i$ ,  $p$ , and  $q$  to get that  $d(g_i, q) > \rho$ , a contradiction. Therefore it must be that  $y(g_i) > y(g_j)$  in any realization. A symmetric argument holds for Line 13.

Now suppose that we set `higherPoint[{gi, gj}]` to be  $g_j$  in Line 17 (and therefore also set `tooFarAway[{gi, gj}]` to be true in the next line). Then from the if-statement, we can apply Lemma 6 to  $g_i$ , `closestHighOutsideRight(gi)`, and  $g_j$  to see that both assignments must be true in any realization. The argument is symmetric in Line 22.

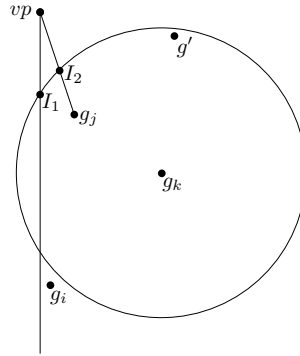
Note that each of the “return true” statements in this block occur if we get contradictory statements about which of  $g_i$  and  $g_j$  has the larger y-coordinate, and therefore if we return true in this block then  $O$  is indeed not realizable.

**Light Green Block.** Now consider the light green block (Lines 24–26). If we return true in line 26, then we have that  $\overline{g_i g_j}$  cannot be pierced in any realization. But at the same time we would need a  $g_k$  between  $g_i$  and  $g_j$  such that  $y(g_k) > y(g_i)$  and  $y(g_k) > y(g_j)$ . Clearly the latter implies that  $g_k$  pierces  $\overline{g_i g_j}$ , and therefore  $O$  cannot be realized.

**Red Block.** Now consider the red block of code (Lines 27–41). In this block we set the `tooFarAway` and `higherPoint` variables for a guard  $g$  and a viewpoint  $p$ . We will argue that if we set these variables to something other than unknown then any realization must satisfy that assignment. If we set `tooFarAway[{g, p}]` to be false in Line 30 then its either because  $g$  sees  $p$  (in which case they obviously cannot be too far away) or it’s because `leftMostPointGuardSees(g) < p < g < g'` where  $g'$  is a guard that  $p$  sees. In this case we can apply Lemma 4 to see that it must be that  $d(p, g) \leq \rho$  in any realization.

Now suppose we enter the else statement starting at Line 34. In this case we have that  $g$  does not see  $p$  and we have `cannotBlockWithTerrain[{g, p}]` is true. Therefore clearly the reason why  $g$  does not see  $p$  is because  $d(g, p) > \rho$ , and therefore our assignment of `tooFarAway[{g, p}]` to be true is correct. Then further suppose we set `higherPoint[{g, p}]` to be  $g$  in Line 36. Then we can apply to Lemma 5 to `leftMostPointGuardSees(g)`,  $p$ , and  $g$  to see that indeed it must be  $y(p) < y(g)$ . The argument for Line 38 where we set `higherPoint[{g, p}]` to be  $p$  follows symmetrically.

**Yellow Block.** Now consider the yellow block (Lines 42–51). Consider any viewpoint  $vp$  that satisfies the for-loop condition in Line 43. Then we have that  $vp$  does not see  $g_j$ , but  $d(vp, g_j) \leq \rho$ . Therefore any realization must have some point that pierces  $\overline{vp g_j}$ . Moreover



■ **Figure 10**  $x(g') > x(g_k)$  and  $vp$  sees  $g'$ .

we have `cannotBlockWithTerrain`  $\{g_i, g_j\}$  is true from Line 42. Then maybe  $g_i$  is a point that pierces  $\overline{vp g_j}$ , but if it does not pierce it then certainly no point between  $g_i$  and  $g_j$  can pierce  $\overline{vp g_j}$  (as it would also pierce  $\overline{g_i g_j}$ ). This means that if  $g_i$  does not pierce  $\overline{vp g_j}$ , then we would need to find a point between  $vp$  and  $g_i$  (possibly a point not in  $O$ ) to pierce  $\overline{vp g_j}$ . The yellow block first considers if it is possible for  $g_i$  to pierce, and if it cannot, then we consider the possibility of a “blocker” between  $vp$  and  $g_i$ .

First consider the if-statement in Line 44. If this condition is true, then we have that there is some guard  $g_k$  such that  $vp$  and  $g_j$  are inside of  $disc(g_k)$  but  $g_i$  is low outside  $disc(g_k)$ . Lemma 7 implies that any point that pierces  $\overline{vp g_j}$  cannot be low outside  $disc(g_k)$ , and therefore  $g_i$  cannot pierce  $\overline{vp g_j}$ .

Next consider the if-statement in Line 46. If this condition is true, then we have that there is some guard  $g_k$  to the right of  $g_j$  such that  $vp$  is high outside  $disc(g_k)$ ,  $g_i$  is low outside  $disc(g_k)$ , and  $g_j$  is inside  $disc(g_k)$ . Since  $vp$  is high outside  $disc(g_k)$ , we have that a ray shot straight down from  $vp$  will first intersect  $disc(g_k)$  at a point  $I_1$  such that  $y(I_1) > y(g_k)$  (recall that this is true since  $vp$  must see a guard to the right of  $g_k$  in order for `higherPoint`  $\{vp, g_k\}$  to be  $vp$ ). Now consider walking along  $\overline{vp g_j}$  starting at  $g_j$  and walking towards  $vp$ . Let  $I_2$  denote the point that this walk exits  $disc(g_k)$ . Then we have that  $y(I_2) > y(I_1)$ . See Figure 10. Therefore any point that is low outside  $disc(g_k)$  cannot pierce  $\overline{vp g_j}$ , and therefore  $g_i$  cannot pierce  $\overline{vp g_j}$ .

So now suppose that `giCanBeBlocker` was set to false by one of the two cases above. Then indeed  $g_i$  cannot pierce  $\overline{vp g_j}$  and we now need to consider a blocker strictly between  $vp$  and  $g_i$ . There are two potential issues this might cause. The first is that if  $g_i$  does not pierce  $\overline{vp g_j}$  but some point  $b$  between  $vp$  and  $g_i$  does pierce  $\overline{vp g_j}$ , then it must be that  $b$  also pierces  $\overline{vp, g_i}$ . So if we have `cannotBlockWithTerrain`  $\{vp, g_i\}$  set to true, then we cannot use such a blocker  $b$ . But moreover, using a blocker  $b$  in this range may cause us to have to flip `cannotBlockWithTerrain`  $\{vp_2 g_j\}$  from false to true if we use such a blocker  $b$ . Indeed, for any point  $vp_2$  to the left of  $vp$ , if we have `cannotBlockWithTerrain`  $\{vp_2, g_i\}$  is true, then the use of this blocker  $b$  will force us to have `cannotBlockWithTerrain`  $\{vp_2, g_j\}$  to be true by the second claim in Section 1.2. If we previously determined that  $vp_2$  cannot be too far away from  $g_j$ , then this would imply that  $vp_2$  sees  $g_j$ , a contradiction. Therefore in this case we would not be able to use a new blocker  $b$ .

So if we determine that  $g_i$  cannot be a blocker and a new point between  $vp$  and  $g_i$  cannot be the blocker, then it is not possible to pierce  $\overline{vp g_j}$  without changing other visibilities and therefore  $O$  cannot be realized, and the algorithm returns true accordingly.



**Purple Block.** Now consider the purple block (Lines 52–54). If there are guards  $g_i$  and  $g_j$  and viewpoints  $vp_1$  and  $vp_2$  that satisfy the if-statement condition then we have  $vp_2$  sees  $g_j$  and is low outside  $disc(g_i)$ , and we have  $vp_1$  sees  $g_i$  and is low outside  $disc(g_j)$ . The left-to-right order of  $g_i < vp_2 < vp_1 < g_j$  then contradicts Lemma 8, and therefore  $O$  is not realizable and we return true accordingly. ◀

### 3 Lower Bound

In this section, we show that the VC-Dimension of LVTs is at least 7. See Figure 12 for the coordinates of the vertices of our LVT where a set of 7 vertices is shattered. We remark that the main value of this construction is that it shows that our upper bound proof we give in Section 2 is tight. We do not claim that this LVT is particularly interesting outside of the fact that it proves that the upper bound cannot be improved.

Here, we assume that the radius  $\rho$  that each vertex can see is 1. The vertices that are shattered are the vertices  $G = \{g_1, \dots, g_7\}$ . The viewpoints are labeled  $vp$  where the subscript denotes which vertices of  $G$  the viewpoint sees. Some of the vertices are neither a viewpoint nor a vertex in  $G$  and are labeled “-” (their role is simply to block a viewpoint from seeing a vertex in  $G$ ). The table does not contain a viewpoint for the empty set or for the subsets of size 1. It is trivial to add these points: the empty set point can be placed to the right of all points a distance greater than 1 to each vertex in  $G$ , and a viewpoint that should see only one vertex  $g_i$  can be placed in a steep “canyon” just to the left of  $g_i$  so that every other  $g_j$  is blocked from seeing it.

To formally verify that our coordinates are correct, we wrote a computer program that ensures each viewpoint sees exactly which guards they are supposed to see. Our program to verify the problem is available at <https://github.com/utsa-saga/vc7proof>.

The aspect ratio of our construction is very large which makes it difficult to produce a static figure that is helpful in visualizing the construction. Nonetheless, see Figure 11 for a static figure of the LVT. Note that in this figure, we use letters  $A$  through  $G$  to represent the guards instead of  $g_1 - g_7$  (e.g.,  $B$  represents  $g_2$  and  $BEF$  represents  $vp_{2,5,6}$ ). See Figure 11 for the overall the boundary of the LVT, where the viewpoints are listed in blue (many of the labels are not visible because the points are so close together) and the guards in  $G$  are red. There is a dynamic version of the figure that allows zooming in-and-out here: <https://www.geogebra.org/calculator/ybvdrjag>. Finally we have dynamic figures for each guard. They show the disc of the guard so it is possible to see which portions of the LVT are too far away from certain guards. In these figures, green line segments indicate that the guard and viewpoint see each other (their line segment is not pierced, and they are close enough to see each other); red line segments indicate their line segment is pierced; purple viewpoints indicate the viewpoints that are outside of the disc of the guard.

- $g_1$ : <https://www.geogebra.org/calculator/vcjuryvhmm>
- $g_2$ : <https://www.geogebra.org/calculator/hjnqeuyqy>
- $g_3$ : <https://www.geogebra.org/calculator/nktasnvc>
- $g_4$ : <https://www.geogebra.org/calculator/pgb5tzj2>
- $g_5$ : <https://www.geogebra.org/calculator/bptpyugq>
- $g_6$ : <https://www.geogebra.org/calculator/cx8khxkc>
- $g_7$ : <https://www.geogebra.org/calculator/t83pypcpq>

## 5:12 The VC-Dimension of Limited Visibility Terrains

■ **Algorithm 1** boolean `isNotRealizable( $O$ )` – Part 1 of 2.

---

```

1: For each pair of points  $u, v$ , set cannotBlockWithTerrain[ $\{u, v\}$ ] to true if  $u$  sees  $v$  or the
   order claim generalization implies that  $\overline{uv}$  cannot be pierced, otherwise set it to be false.
2: if there is a  $g \in G$  such that
    $leftMostPointGuardSees(g) < closestHighOutsideLeft(g)$  or
    $closestHighOutsideRight(g) < rightMostPointGuardSees(g)$  then
3:   return true
4: for every pair  $g_i, g_j \in G$  such that  $i < j$  do
5:   tooFarAway[ $\{g_i, g_j\}$ ] = unknown, higherPoint[ $\{g_i, g_j\}$ ] = unknown
6:   if  $leftMostPointGuardSees(g_j) < g_i$  and
    $g_j < rightMostPointGuardSees(g_i)$  then
7:     tooFarAway[ $\{g_i, g_j\}$ ] = false
8:     if  $closestHighOutsideRight(g_j) \leq rightMostPointGuardSees(g_i)$  then
9:       higherPoint[ $\{g_i, g_j\}$ ] =  $g_i$ 
10:    if  $leftMostPointGuardSees(g_j) \leq closestHighOutsideLeft(g_i)$  then
11:      if higherPoint[ $\{g_i, g_j\}$ ] is  $g_i$  then
12:        return true
13:      higherPoint[ $\{g_i, g_j\}$ ] =  $g_j$ 
14:      if  $closestHighOutsideRight(g_i) < g_j$  and
      cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true then
15:        if higherPoint[ $\{g_i, g_j\}$ ] is  $g_i$  then
16:          return true
17:        higherPoint[ $\{g_i, g_j\}$ ] =  $g_j$ 
18:        tooFarAway[ $\{g_i, g_j\}$ ] = true
19:      if  $g_i < closestHighOutsideLeft(g_j)$  and
      cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true then
20:        if higherPoint[ $\{g_i, g_j\}$ ] is  $g_j$  then
21:          return true
22:        higherPoint[ $\{g_i, g_j\}$ ] =  $g_i$ 
23:        tooFarAway[ $\{g_i, g_j\}$ ] = true
24: for every pair  $g_i, g_j \in G$  such that  $i < j$  do
25:   if cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true and there is a  $k \in (i, j)$  such that
   higherPoint[ $\{g_i, g_k\}$ ] and higherPoint[ $\{g_k, g_j\}$ ] are both  $g_k$  then
26:     return true
27: for all  $g \in G$  do
28:   for all viewpoints  $p \in L(g)$  do
29:     if  $g$  sees  $p$  or ( $leftMostPointGuardSees(g) < p$  and
      $p$  sees a guard right of  $g$ ) then
30:       tooFarAway[ $\{g, p\}$ ] = false, higherPoint[ $\{g, p\}$ ] = unknown
31:     else if cannotBlockWithTerrain[ $\{g, p\}$ ] is false then
32:       tooFarAway[ $\{g, p\}$ ] = unknown, higherPoint[ $\{g, p\}$ ] = unknown
33:     else
34:       tooFarAway[ $\{g, p\}$ ] = true
35:       if  $leftMostPointGuardSees(g) < p$  then
36:         higherPoint[ $\{g, p\}$ ] =  $g$ 
37:       else if  $p$  sees a guard to the right of  $g$  then
38:         higherPoint[ $\{g, p\}$ ] =  $p$ 
39:       else
40:         higherPoint[ $\{g, p\}$ ] = unknown
41:   Symmetrically repeat steps 28 - 40 for viewpoints  $p \in R(g)$ .

```

---



■ **Algorithm 2** boolean `isNotRealizable( $O$ )` – Part 2 of 2.

---

```

42: for every pair  $g_i, g_j \in G$  such that  $i < j$  and cannotBlockWithTerrain[ $\{g_i, g_j\}$ ] is true do
43:   for each viewpoint  $vp$  such that leftMostPointGuardSees( $g_j$ ) < vp < g_i and  $vp$  does
      not see  $g_j$  and tooFarAway[ $\{vp, g_j\}$ ] is false do
44:     if there is a  $g_k \in G$  such that tooFarAway[ $\{vp, g_k\}$ ] is false and tooFarAway[ $\{g_i, g_k\}$ ]
      is true and higherPoint[ $\{g_k, g_i\}$ ] is  $g_k$  and tooFarAway[ $\{g_j, g_k\}$ ] is false then
45:        $g_i$ CanBeBlocker = false
46:     else if there is a  $g_k \in G$  such that  $g_j < g_k$  and tooFarAway[ $\{vp, g_k\}$ ] is true and
      higherPoint[ $\{vp, g_k\}$ ] is  $vp$  and tooFarAway[ $\{g_i, g_k\}$ ] is true and higherPoint[ $\{g_k, g_i\}$ ]
      is  $g_k$  and tooFarAway[ $\{g_j, g_k\}$ ] is false then
47:        $g_i$ CanBeBlocker = false
48:     if  $g_i$ CanBeBlocker is false then
49:       if cannotBlockWithTerrain[ $\{vp, g_i\}$ ] is true or (there is a  $vp_2$  such that
      leftMostPointGuardSees( $g_j$ ) < vp_2 < vp and  $vp_2$  sees  $g_i$  and  $vp_2$  does not see  $g_j$ 
      and tooFarAway[ $\{vp_2, g_j\}$ ] is false) then
50:         return true
51:   Repeat Lines 43 - 50 symmetrically, looking for a  $vp$  to the right of  $g_j$  such that  $g_i$  needs
      a blocker to prevent it from seeing  $vp$ .
52: for every pair  $g_i, g_j \in G$  such that  $i < j$  do
53:   if there are viewpoints  $vp_1$  and  $vp_2$  such that  $g_i < vp_2 < vp_1 < g_j$  and  $vp_1$  sees  $g_i$ 
      and tooFarAway[ $\{vp_1, g_j\}$ ] is true and higherPoint[ $\{vp_1, g_j\}$ ] is  $g_j$  and  $vp_2$  sees  $g_j$  and
      tooFarAway[ $\{g_i, vp_2\}$ ] is true and higherPoint[ $\{g_i, vp_2\}$ ] is  $g_i$  then
54:     return true
55: return false

```

---

■ **Algorithm 3** boolean `candidateOrderingIsExtendable( $O_i$ )`.

---

```

1: if isNotRealizable( $O_i$ ) then
2:   return false
3: else if  $V_{test} \setminus O_i = \emptyset$  then
4:   return true
5: for all  $v \in V_{test} \setminus O_i$  do
6:   for  $j = 0$  to  $|O_i|$  do
7:     Let  $O_{i+1}$  be a candidate ordering by inserting  $v$  into position  $j$  in  $O_i$ .
8:     if candidateOrderingIsExtendable( $O_{i+1}$ ) then
9:       return true
10: return false

```

---

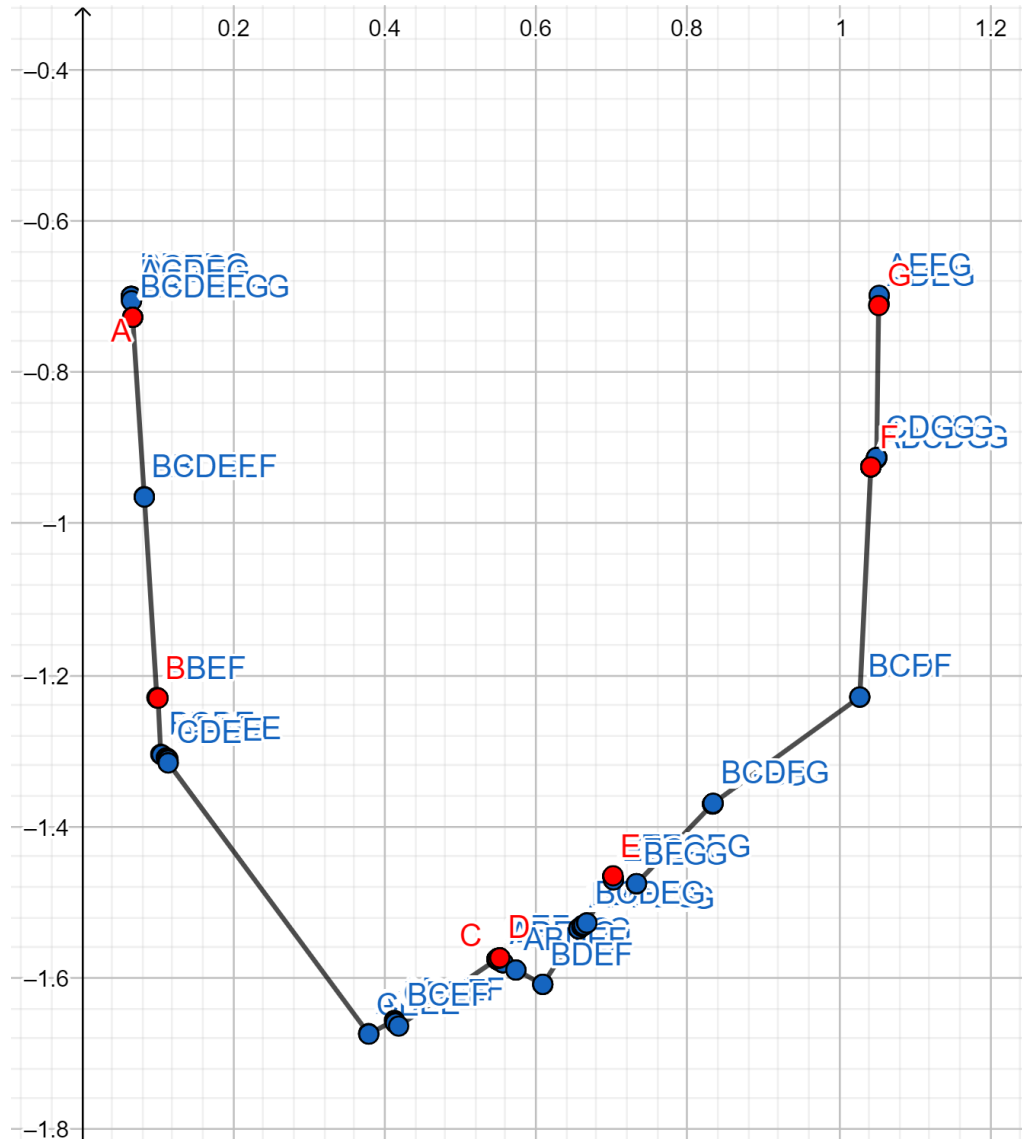


Figure 11 Terrain Boundary.

Coordinate	Label	Coordinate	Label
(0.065221878837284,-0.699711771625512)	VP3,5,7	(0.573309531425741,-1.58973181578975)	VP1,2,4,5,6
(0.065221883996657,-0.699711787103631)	VP2,3,5,7	(0.608804212112144,-1.60869870293362)	VP2,4
(0.065224793883047,-0.699755445718236)	VP1,2,3,5,7	(0.608804366893335,-1.6086950913725)	VP2,4,6
(0.065235112629113,-0.700075326846282)	VP1,3,5,7	(0.608806017892706,-1.60869251168598)	VP2,4,5,6
(0.065609528330117,-0.705692491048488)	VP4,5,7	(0.656334162272601,-1.53539071931979)	VP1,2,4,6,7
(0.065611746860522,-0.705696463765723)	VP1,3,4,5,7	(0.656375437256865,-1.53532622715688)	VP1,2,4,5,6,7
(0.067070817554251,-0.727468502027634)	VP1,3,4,5,6,7	(0.656376469131471,-1.53532674309418)	VP1,2,5,6,7
(0.067071333491554,-0.727471546057723)	VP1,7	(0.656534954752298,-1.53526755476674)	VP2,4,6,7
(0.067071849428857,-0.727471597651453)	g1	(0.656535377820887,-1.535266894367)	VP2,4,5,6,7
(0.067074429115374,-0.727520611695267)	VP1,2,3,4,5,6,7	(0.660353313865299,-1.5324679344966)	VP2,4,7
(0.067074945052677,-0.727534542002456)	VP2,3,4,5,6,7	(0.660902529124661,-1.53132874493092)	VP1,2,4
(0.082446781067165,-0.96428681986565)	VP1,3,4,5,6	(0.660902787093312,-1.53132513336979)	VP1,2,4,7
(0.082549968527825,-0.964802757168949)	VP1,2,3,4,5,6	(0.660915685525895,-1.53130191619114)	VP1,2,4,5,7
(0.082555127900858,-0.965009132090268)	VP2,3,4,5,6	(0.663061984707618,-1.5304660977598)	VP2,4,5,7
(0.099059962233389,-1.22973656241294)	VP2,5,6	(0.666998070394486,-1.52760625728761)	VP2,3,4
(0.099885461918668,-1.2302009059859)	VP1,2,5,6	(0.666998586331789,-1.52759748635346)	VP2,3,4,7
(0.099916418156866,-1.2307168432892)	VP2,6	(0.667003745704822,-1.52758716760739)	VP2,3,4,5,7
(0.100607258205983,-1.23053110586002)	VP1,2	(0.701416763834859,-1.46577787867218)	VP1,2,3,4,5,7
(0.100607774143286,-1.23051046836788)	g2	(0.701421923207892,-1.46578303804522)	VP1,5,7
(0.10417290909082,-1.3046919338362)	VP2,4,5	(0.701427082580925,-1.465780974296)	VP1,2,5,7
(0.105560772254956,-1.30552775226754)	VP2,3,4,5	(0.701435337577777,-1.46578200617061)	VP1,2,5
(0.111473413750761,-1.30925281959736)	VP1,2,4,5	(0.701436369452384,-1.46578303804522)	VP1,5
(0.113506206725759,-1.31048075037921)	VP1,2,3,4,5	(0.701674732486508,-1.46526194136888)	g5
(0.113764175377408,-1.31254449959241)	VP1,3,4,5	(0.702087482329147,-1.47062768932319)	VP5,7
(0.113769334750441,-1.31600643889755)	VP4,5	(0.702190669789807,-1.47042131440187)	VP5,6,7
(0.114022144029058,-1.3161560607155)	VP3,4,5	(0.732501986358618,-1.47586445295168)	VP2,5
(0.378697980621395,-1.67370061190164)	VP1,2,3,5	(0.732527783223783,-1.47578706235618)	VP2,5,7
(0.378955949273044,-1.67421654920494)	VP1,3,5	(0.732630970684442,-1.47558068743486)	VP2,5,6,7
(0.379007543003374,-1.67442292412626)	VP3,5	(0.732638709743992,-1.47558352509003)	VP2,6,7
(0.412749842639122,-1.65615874358948)	VP1,2,3,5,6	(0.833197469843465,-1.37048425875288)	VP2,3,5
(0.41378171724572,-1.65822249280267)	VP1,3,5,6	(0.833238744827728,-1.37032947756189)	VP2,3,5,6,7
(0.414204785834425,-1.65983221718897)	VP5,6	(0.833239539113207,-1.3703289054906)	VP2,3,7
(0.414297654549019,-1.65977030471257)	VP3,5,6	(0.833239539164801,-1.37032890546481)	VP2,7
(0.41842515297541,-1.6637946156783)	VP2,3,5,6	(0.833239539371176,-1.37032890487148)	VP2,3,6,7
(0.548214340993282,-1.57551774308386)	VP1,2,3,5,6,7	(0.833806275861357,-1.36950397787661)	-
(0.548216920679799,-1.57552032277038)	VP1,3,5,6,7	(0.833820231965412,-1.36949386550546)	VP2,3,4,6,7
(0.548219500366316,-1.57552548214341)	VP3,5,6,7	(1.02748913951977,-1.22916903137931)	VP2,3,4,6
(0.548441353406734,-1.57567252427485)	VP1,2,3,6	(1.02756188662794,-1.22911641093374)	VP2,3
(0.5484516721528,-1.57568284302092)	VP1,3,6	(1.02756188667953,-1.22911640577437)	VP2,3,6
(0.548456831525833,-1.57569316176698)	VP3,6	(1.04219283672648,-0.925087451372909)	-
(0.549466004891086,-1.57464864669645)	VP1,2,3,6,7	(1.04219289863896,-0.925087358504194)	VP1,2,3,4,6
(0.549466262859737,-1.57464890466511)	VP1,3,6,7	(1.04219335266378,-0.925075584814933)	g6
(0.549466520828389,-1.57464942060241)	VP3,6,7	(1.04219336298253,-0.925075770552362)	VP6,7
(0.549468068640299,-1.5746483887278)	VP1,2,3,7	(1.04219338356843,-0.925075589974306)	VP1,6
(0.54946832660895,-1.57464890466511)	VP3,7	(1.04219338362002,-0.925075585330871)	VP1,6,7
(0.549472196138725,-1.57465096841432)	VP1,2,3	(1.04219341354439,-0.925075562113692)	VP1,2,6
(0.549472712076029,-1.57465148435162)	VP1,3	(1.04219341457626,-0.925075533221203)	VP1,2,6,7
(0.549473228013332,-1.57464064966825)	g3	(1.04219355903871,-0.925075275252551)	VP1,2,3,4,6,7
(0.549576415473991,-1.57639483649947)	VP3,4,6	(1.04219361063244,-0.925075223658821)	VP1,2,7
(0.549989165316631,-1.57618846157815)	VP3,4,6,7	(1.04219367254491,-0.92507512047136)	VP1,2,3,4,7
(0.551640164687187,-1.57660121142079)	VP1,3,4	(1.049421634283,-0.914440053244729)	VP1,3,4,7
(0.55204775156793,-1.57366036879198)	VP1,2,3,4	(1.04942627771873,-0.91443391359082)	VP1,3,7
(0.552052914529826,-1.57360877506165)	g4	(1.04951940440198,-0.913640092455965)	VP1,3,4,6
(0.552070456398138,-1.5738187615441)	VP4,7	(1.04951966237063,-0.913621776681697)	VP1,3,4,6,7
(0.552071488272745,-1.57381721373219)	VP4,6,7	(1.0498287088153,-0.912847870726749)	VP3,4,5,6
(0.552073552021958,-1.57381514998297)	VP4,5,6,7	(1.04982891519023,-0.912847561164367)	VP3,4
(0.552506939356729,-1.57805615461609)	VP4,6	(1.04982922475261,-0.912796276996419)	VP3,4,5,6,7
(0.552517258102795,-1.57804583587003)	VP4,5,6	(1.04984470287171,-0.912771512005861)	VP3,4,5,7
(0.556087544241624,-1.58007862884502)	VP1,4,5,6	(1.04984728255822,-0.912767900444738)	VP3,4,7
(0.556110761420272,-1.58009410696412)	VP1,4,6	(1.05251209872976,-0.711993478552486)	g7
(0.556156679840266,-1.58004664073222)	VP1,4	(1.05302803603306,-0.71044566664259)	VP1,4,5,7
(0.556157195777569,-1.58003477417424)	VP1,4,6,7	(1.053131232349372,-0.71018769799094)	VP1,4,5
(0.556167514523635,-1.58002703511469)	VP1,4,7	(1.05313638286675,-0.699027974120585)	VP1,5,6,7
(0.556180412956217,-1.58000639762256)	VP1,4,5,6,7	(1.05316372754383,-0.698461732930214)	VP1,5,6
(0.573305403927315,-1.58973697516278)	VP1,2,4,6		

Figure 12 The coordinates of our construction.

## References

- 1 J. Abello, H. Lin, and S. Pisupati. On visibility graphs of simple polygons. *Congressus Numerantium*, 90:119–128, 1992.
- 2 Boaz Ben-Moshe, Matthew J. Katz, and Joseph S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '05*, pages 515–524, USA, 2005. Society for Industrial and Applied Mathematics.
- 3 Amitava Bhattacharya, Subir Kumar Ghosh, and Sudeep Sarkar. Exploring an unknown polygonal environment with bounded visibility. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, René S. Renner, and C. J. Kenneth Tan, editors, *Computational Science — ICCS 2001*, pages 640–648, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- 4 Therese Biedl and Saeed Mehrabi. Grid-obstacle representations with connections to staircase guarding. In Fabrizio Frati and Kwan-Liu Ma, editors, *Graph Drawing and Network Visualization*, pages 81–87, Cham, 2018. Springer International Publishing.
- 5 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995. doi:10.1007/BF02570718.
- 6 Alon Efrat and Sarel Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006. doi:10.1016/j.ipl.2006.05.014.
- 7 Sándor P Fekete, Joseph SB Mitchell, and Christiane Schmidt. Minimum covering with travel cost. *Journal of combinatorial optimization*, 24(1):32–51, 2012.
- 8 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. In *SWAT*, pages 96–104, 1988.
- 9 Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997. doi:10.1007/BF02770871.
- 10 Subir Kumar Ghosh, Joel Wakeman Burdick, Amitava Bhattacharya, and Sudeep Sarkar. Online algorithms with discrete visibility - exploring unknown polygonal environments. *IEEE Robotics Automation Magazine*, 15(2):67–76, 2008. doi:10.1109/MRA.2008.921542.
- 11 Matt Gibson, Gaurav Kanade, Erik Krohn, and Kasturi Varadarajan. An approximation scheme for terrain guarding. In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 140–148, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 12 Matt Gibson, Erik Krohn, and Matthew Rayford. Guarding monotone polygons with half-guards. In Joachim Gudmundsson and Michiel H. M. Smid, editors, *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada*, pages 168–173, 2017.
- 13 Matt Gibson, Erik Krohn, and Qing Wang. The vc-dimension of visibility on the boundary of a simple polygon. In Khaled Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation*, pages 541–551, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- 14 Matt Gibson, Erik Krohn, and Qing Wang. The vc-dimension of visibility on the boundary of monotone polygons. *Computational Geometry*, 77:62–72, 2019. Canadian Conference on Computational Geometry. doi:10.1016/j.comgeo.2018.10.006.
- 15 Alexander Gilbers and Rolf Klein. New results on visibility in simple polygons. In Frank Dehne, Marina Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures*, pages 327–338, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- 16 James King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 629–640. Springer, 2006. doi:10.1007/11682462\_58.
- 17 James King. Vc-dimension of visibility on terrains. In *In Proc. 20th Canadian Conference on Comput. Geom.*, pages 27–30, 2008.
- 18 James King and Erik Krohn. Terrain guarding is np-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011. doi:10.1137/100791506.

- 19 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi R. Varadarajan. Guarding terrains via local search. *JoCG*, 5:168–178, 2014.
- 20 Erik Krohn and Bengt J. Nilsson. The complexity of guarding monotone polygons. In *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG 2012, Charlottetown, Prince Edward Island, Canada, August 8-10, 2012*, pages 167–172, 2012. URL: <http://2012.cccg.ca/papers/paper26.pdf>.
- 21 Erik Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013. doi:10.1007/s00453-012-9653-3.
- 22 Erik A. Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, pages 1–31, 2012.
- 23 T. Prellberg. Uniform q-series asymptotics for staircase polygons. *Journal of Physics A: Mathematical and General*, 28(5):1289–1304, March 1995. doi:10.1088/0305-4470/28/5/016.
- 24 Abdulmuttalib Turkey Rashid, Abduladhem Abdulkareem Ali, Mattia Frasca, and Luigi Fortuna. Path planning with obstacle avoidance based on visibility binary tree algorithm. *Robotics and Autonomous Systems*, 61(12):1440–1449, 2013. doi:10.1016/j.robot.2013.07.010.
- 25 Sven Schuierer, Gregory J. E. Rawlins, and Derick Wood. A generalization of staircase visibility. In H. Bieri and H. Noltemeier, editors, *Computational Geometry-Methods, Algorithms and Applications*, pages 277–287, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- 26 Israel A. Wagner, Michael Lindenbaum, and Alfred M. Bruckstein. Mac versus pc: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *The International Journal of Robotics Research*, 19(1):12–31, 2000. doi:10.1177/02783640022066716.



# Clustering with Neighborhoods

Hongyao Huang ✉

Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Georgiy Klimenko ✉

Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Benjamin Raichel ✉

Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

---

## Abstract

In the standard planar  $k$ -center clustering problem, one is given a set  $P$  of  $n$  points in the plane, and the goal is to select  $k$  center points, so as to minimize the maximum distance over points in  $P$  to their nearest center. Here we initiate the systematic study of the clustering with neighborhoods problem, which generalizes the  $k$ -center problem to allow the covered objects to be a set of general disjoint convex objects  $\mathcal{C}$  rather than just a point set  $P$ . For this problem we first show that there is a PTAS for approximating the number of centers. Specifically, if  $r_{opt}$  is the optimal radius for  $k$  centers, then in  $n^{O(1/\varepsilon^2)}$  time we can produce a set of  $(1 + \varepsilon)k$  centers with radius  $\leq r_{opt}$ . If instead one considers the standard goal of approximating the optimal clustering radius, while keeping  $k$  as a hard constraint, we show that the radius cannot be approximated within any factor in polynomial time unless  $P = NP$ , even when  $\mathcal{C}$  is a set of line segments. When  $\mathcal{C}$  is a set of unit disks we show the problem is hard to approximate within a factor of  $\frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}} \approx 6.99$ . This hardness result complements our main result, where we show that when the objects are disks, of possibly differing radii, there is a  $(5 + 2\sqrt{3}) \approx 8.46$  approximation algorithm. Additionally, for unit disks we give an  $O(n \log k) + (k/\varepsilon)^{O(k)}$  time  $(1 + \varepsilon)$ -approximation to the optimal radius, that is, an FPTAS for constant  $k$  whose running time depends only linearly on  $n$ . Finally, we show that the one dimensional version of the problem, even when intersections are allowed, can be solved exactly in  $O(n \log n)$  time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases** Clustering, Approximation, Hardness

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.6

**Related Version** *Full Version:* <https://arxiv.org/abs/2109.13302> [22]

**Funding** Partially supported by NSF CAREER Award 1750780.

## 1 Introduction

In the standard  $k$ -center clustering problem, one is given a set  $P$  of  $n$  points in a metric space and an integer parameter  $k \geq 0$ , and the goal is to select  $k$  points from the metric space (or from  $P$  in the discrete  $k$ -center problem), called centers, so as to minimize the maximum distance over points in  $P$  to their nearest center. Equivalently, the problem can be viewed as covering  $P$  with  $k$  balls with the same radius  $r$ , where the goal is to minimize  $r$ . It is well known that it is NP-hard to approximate the optimal  $k$ -center radius  $r_{opt}$  within any factor less than 2 in general metric spaces [21], and that the problem remains hard to approximate within a factor of roughly 1.82 in the plane [12]. For general metric spaces, the standard greedy algorithm of Gonzalez [19], which repeatedly selects the next center to be the point from  $P$  which is furthest from the current set of centers, achieves an optimal 2-approximation to  $r_{opt}$ . An alternative algorithm due to Hochbaum and Shmoys [20] also achieves an optimal approximation ratio of 2 by approximately searching for the optimal radius, observing that if  $r \geq r_{opt}$  then all points will be covered after  $k$  rounds of repeatedly removing points in  $2r$  radius balls centered at any remaining point of  $P$ .



© Hongyao Huang, Georgiy Klimenko, and Benjamin Raichel;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiro Sadakane; Article No. 6; pp. 6:1–6:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider a natural generalization of  $k$ -center clustering in the plane, where the objects which we must cover are general disjoint convex objects rather than points. Specifically, in the *clustering with neighborhoods* problem the goal is to select  $k$  center points so that balls centered at these points with minimum possible radius intersect all the convex objects. This generalization is natural as real world objects may not be well modeled as individual points. This generalized setting has previously been considered for other classical point based problems in the plane, such as the Traveling Salesperson Problem [10], where the authors referred to these objects as neighborhoods. (We instead typically refer to them as *objects*.) To the best of our knowledge we are the first to consider the general problem of clustering convex objects in this context, though as we discuss below many closely related problems have been considered, some of which equate to special or extreme cases of our problem. We remark that since a point is a convex set, the hardness results for  $k$ -center clustering immediately apply to clustering with neighborhoods.

### Related Work

As clustering is a fundamental data analysis task, countless variants have been considered. Here we focus on variants which share our  $k$ -center objective of minimizing the maximum radius of the balls at the chosen centers. Bandyapadhyay et al. [7] considered the colorful  $k$ -center problem, where the points are partitioned into color classes  $P_1, \dots, P_c$  and the goal is to find  $k$  balls with minimum radius which cover at least  $t_i$  points from each color class  $P_i$ . When our convex objects have bounded diameter our problem can be approximately cast as an instance of colorful  $k$ -center by replacing each object with the set  $P_i$  of grid points it intersects and setting  $t_i = 1$ . General colorful clustering, however, is more challenging as the color classes can be interspersed, which is why [7] assumes the number of color classes is a constant, allowing for a constant factor approximation, which subsequently was improved [5, 23]. Note that colorful  $k$ -center itself generalizes the  $k$ -center with outliers problem [8], corresponding to the case with a single color class  $P$  with  $n - t$  outliers allowed.

Xu and Xu [31] considered  $k$ -center clustering on point sets (KCS) where given points sets  $S_1, \dots, S_n$  the goal is to find  $k$  balls of minimum radius such that each  $S_i$  is entirely contained in one of the balls. Again when our objects have bounded diameter we can relate our problem to KCS by discretizing the objects. Their requirement that all of  $S_i$  be covered by a single ball implies that the optimal radius is at least the radius of the largest object, whereas in our case as only a single point of  $S_i$  needs to be covered the radius can be arbitrarily smaller. In particular, while [31] achieves a  $(1 + \sqrt{3})$ -approximation, we show our problem cannot in general be approximated within any factor in polynomial time unless  $P = NP$ .

For the special case when  $k = 1$  or  $k = 2$ , there are several prior results which closely relate to our problem. When  $k = 1$ , i.e. the one-center problem, the solution can be derived from the farthest object Voronoi diagram, for which Cheong et al. [9] gave a near linear time algorithm for polygon objects. For disk objects, Ahn et al. [4] gave a near quadratic time algorithm for the two-center problem. Several papers have also considered generalizing to higher dimensions, but restricting the convex objects to affine subspaces of dimension  $\Delta$ . Gao et al. [16] introduced the 1-center problem for  $n$  lines, achieving a linear time  $(1 + \varepsilon)$ -approximation, as well as a  $(1 + \varepsilon)$ -approximation for higher dimensional flats or convex sets whose running time depends exponentially on  $\Delta$ . Later in [17] the same authors considered the more challenging  $k = 2$  and  $k = 3$  cases for lines, providing a  $(2 + \varepsilon)$ -approximation in quasi-linear time. Subsequently, [26] considered the problem for axis-parallel flats, where they provide an improved approximation for  $k = 1$ , hardness results for  $k = 2$ , and an



approximation for larger  $k$  where the time depends exponentially on both  $k$  and  $\Delta$ . While our focus is on the  $k$ -center objective, we remark that  $k$ -means clustering for lines was considered by Marom and Feldman [27], who gave a PTAS for constant  $k$ .

The  $k$ -center problem for points in a metric space can also be viewed as clustering the vertices according to the shortest path metric of a positively weighted graph. This allows one to consider specific graph classes, for example, Eisenstat et al. [11] gave a polynomial time bi-criteria approximation scheme for  $k$ -center in planar graphs (i.e. they allow both the number of centers and radius to be violated). We remark, however, that for our problem, and the various others described above where the objects are not points, the complete graph with all pairwise distances between the objects, is not necessarily metric (i.e. it may not be its own metric completion). For example, the triangle inequality would be violated if you had two small convex objects (e.g. points) which are far from one another but both are close to some other large convex object. Note that this non-metric behavior is what allows us to prove a stronger hardness of approximation result than that for points in the plane [12].

Finally, we note that there is a polynomial time algorithm for  $k$ -center when  $k$  is a constant and the objects are points in  $d$ -dimensional Euclidean space, for constant  $d$ . Specifically, Agarwal and Procopiuc [3] gave an  $n^{O(k^{1-1/d})}$  time exact algorithm, as well as a  $O(n \log k) + (k/\varepsilon)^{O(k^{1-1/d})}$  time  $(1 + \varepsilon)$ -approximation. Later Bădoiu et al. [6] removed the bounded dimension assumption, achieving a  $2^{O((k \log k)/\varepsilon^2)} \cdot dn$  time  $(1 + \varepsilon)$ -approximation.

## Our Contribution

In this paper we initiate the systematic study of the NP-hard clustering with neighborhoods problem. While this problem allows centers to be placed anywhere in the plane, in Section 3 we first argue that one can compute a cubic sized set of points  $P$  and a cubic sized set of radii  $R$ , such that for any integer  $k \geq 0$  there is an optimal set  $S \subseteq P$  of  $k$  centers with optimal radius  $r_{opt} \in R$ . This naturally leads to a PTAS for approximating the optimal number of centers by using Minkowski sums to reduce the problem to instances of geometric hitting set, for which there is a well known PTAS [29]. Specifically, if  $r_{opt}$  is the optimal radius for  $k$  centers, then in  $n^{O(1/\varepsilon^2)}$  time we can produce a set of  $(1 + \varepsilon)k$  centers with radius  $\leq r_{opt}$ .

In clustering problems, however, often the emphasis is on approximating the radius, while keeping  $k$  as a hard constraint. In Section 4 we prove this problem is significantly harder, by adapting the hardness proof of [12] for planar  $k$ -center. Specifically, we show that the radius cannot be approximated within any factor in polynomial time unless  $P = NP$ , even when the convex objects are restricted to disjoint line segments. On the other hand, for disjoint unit disks, a more in depth proof shows the problem is APX-hard, and in particular cannot be approximated within  $\frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}} \approx 6.99$  in polynomial time unless  $P = NP$ . Complementing this result, in Section 5 we present our main result, showing that when the objects are disjoint disks (of possibly varying radii) there is a  $(5 + 2\sqrt{3})$ -approximation for the optimal radius. Significantly, for the case of disks, our approximation factor of  $5 + 2\sqrt{3} \approx 8.46$  is close to our hardness bound of  $\frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}} \approx 6.99$ . Moreover, while our approximation holds for disks of varying radii, interestingly our hardness bound applies even for disks of uniform radii.

Further probing the complexity of clustering with neighborhoods, in Section 6 we show there is an FPTAS for unit disks when  $k$  is bounded by a constant. Specifically, we give an  $O(n \log k) + (k/\varepsilon)^{O(k)}$  time  $(1 + \varepsilon)$ -approximation to the optimal radius, by carefully reducing to the algorithm of [3] for  $k$ -center. Finally in Section 7, by utilizing the searching procedure of [15], we show that in one dimension the problem can be solved exactly in  $O(n \log n)$  time even when intersections are allowed, contrasting our hardness results in the plane.

## 2 Preliminaries

Given points  $x, y \in \mathbb{R}^d$ ,  $\|x - y\|$  denotes their Euclidean distance. Given two closed sets  $X, Y \subset \mathbb{R}^d$ ,  $\|X - Y\| = \min_{x \in X, y \in Y} \|x - y\|$  denotes their distance. For a single point  $x$  we write  $\|x - Y\| = \|\{x\} - Y\|$ . For a point  $x$  and a value  $r \geq 0$ , let  $B(x, r)$  denote the closed ball centered at  $x$  and with radius  $r$ .

Let  $\mathcal{C}$  be a set of  $n$  pairwise disjoint convex objects in the plane. For simplicity, we assume  $\mathcal{C}$  is in general position. We work under the standard assumption that the objects in  $\mathcal{C}$  are semi-algebraic sets of constant descriptive complexity. Namely, the boundary of each object is composed of a set of algebraic arcs where the sum of the degrees of these arcs is bounded by a constant, and any natural standard operation on such objects, such as computing the distance between any pair of objects, can be carried out in constant time. See Agarwal et al. [1] for a more detailed discussion of this model. Our analysis generalizes to the case where  $n$  is the total complexity of  $\mathcal{C}$  and individual objects in  $\mathcal{C}$  are not required to have constant complexity, however, assuming constant complexity simplifies certain structural statements and the polynomial degree of  $n$  in our running time statements.

► **Problem 1** (Clustering with Neighborhoods). *Given a set  $\mathcal{C}$  of  $n$  disjoint convex objects in the plane, and an integer parameter  $k \geq 0$ , find a set of  $k$  points  $S$  (called centers) which minimize the maximum distance to a convex object in  $\mathcal{C}$ . That is,*

$$S = \arg \min_{S' \subset \mathbb{R}^2, |S'|=k} \max_{C \in \mathcal{C}} \|C - S'\|.$$

Let  $S$  be any set of  $k$  points, and let  $r = \max_{C \in \mathcal{C}} \|C - S\|$ . We refer to  $r$  as the *radius* of the solution  $S$ , since  $r$  is the minimum radius such that the set of all balls  $B(s, r)$  for  $s \in S$ , intersect all  $C \in \mathcal{C}$ . If  $S$  is an optimal solution then we refer to its radius  $r_{opt}$  as the optimal radius.

In this paper we will consider two types of approximations.<sup>1</sup> Let  $\mathcal{C}, k$  be an instance of Problem 1 with optimal radius  $r_{opt}$ . For a value  $\alpha \geq 1$ , we refer to a polynomial time algorithm as an  $\alpha$ -size-approximation if it returns a solution  $S$  of radius  $\leq r_{opt}$  where  $|S| \leq \alpha k$ . Alternatively, we refer to a polynomial time algorithm as an  $\alpha$ -radius-approximation if it returns a solution  $S$  of radius  $\leq \alpha r_{opt}$  where  $|S| = k$ . Often we refer to the latter radius case simply as an  $\alpha$ -approximation.

## 3 Canonical Sets and a PTAS for Approximating the Size

In this section we show that while Problem 1 allows centers to be placed anywhere in the plane, we can compute a canonical cubic sized set of points  $P$  and a set of corresponding radii  $R$ , such that for any integer  $k \geq 0$  there is an optimal set  $S \subseteq P$  of  $k$  centers with optimal radius  $r_{opt} \in R$ . We then use this property to give a PTAS for Problem 1 when approximating the size of an optimal solution. Specifically, for any fixed  $\varepsilon > 0$ , we give a  $(1 + \varepsilon)$ -size-approximation with running time  $n^{O(1/\varepsilon^2)}$ . In Section 5, we will again use this canonical set when designing our constant factor radius-approximation for disks.

The *bisector* of two convex objects  $C, C'$  is the set of all points  $x$  in the plane such that  $\|x - C\| = \|x - C'\|$ . Let  $\beta(C, C')$  denote the bisector of  $C$  and  $C'$ . As discussed in [24], any set  $\mathcal{C}$  of  $n$  disjoint constant-complexity convex objects in general position satisfies the conditions of an abstract Voronoi diagram [25]. In particular we can assume the following:

<sup>1</sup> We refrain from using the standard bi-criteria approximation terminology to emphasize that in each case only the size or only the radius is being approximated, not both.

- 1) For any  $C, C' \in \mathcal{C}$  we have that  $\beta(C, C')$  is an unbounded simple curve.
  - 2) The intersection of any two bisectors is a discrete set with a constant number of points.
- We point out that in the following lemma there is a single pair of sets  $P, R$  which works simultaneously for all values of  $k$ .

► **Lemma 2.** *Let  $\mathcal{C}$  be a set of  $n$  disjoint convex objects. In  $O(n^3 \log n)$  time one can compute a set of  $O(n^3)$  points  $P$ , and a corresponding set of  $O(n^3)$  radii  $R$ , such that for any value  $k \geq 0$  for the instance  $\mathcal{C}, k$  of Problem 1 there is an optimal set of  $k$  centers  $S$  with optimal radius  $r_{opt}$  such that  $S \subseteq P$  and  $r_{opt} \in R$ .*

**Proof.** Let  $I(\mathcal{C})$  be a set containing exactly one (arbitrary) point from each convex object in  $\mathcal{C}$ . For any number of centers  $k$ , let  $S$  be any optimal solution, and let  $r_{opt}$  be the optimal radius. Consider an arbitrary center  $s \in S$ . Let  $\mathcal{C}'$  be the subset of objects in  $\mathcal{C}$  which intersect the ball  $B(s, r_{opt})$ . We can assume  $\mathcal{C}'$  is non-empty, as otherwise the center  $s$  does not cover any convex object within radius  $r_{opt}$  and so can be thrown out. Moreover, if  $|\mathcal{C}'| = 1$  then we can assume  $s$  is the point from  $I(\mathcal{C})$  which intersects this one convex object. So assume  $|\mathcal{C}'| > 1$ , and let  $C$  be the convex object in  $\mathcal{C}'$  which lies furthest from  $s$ . Now consider moving  $s$  continuously toward the convex object  $C$ . As we do so the distance from  $s$  to  $C$  monotonically decreases. Thus so long as  $C$  remains the furthest convex object from  $s$  in  $\mathcal{C}'$ , the ball  $B(s, r_{opt})$  still intersects all of  $\mathcal{C}'$  (i.e. we did not increase the solution radius). Now if  $C$  always remains the furthest, when  $s$  eventually reaches and intersects  $C$  then this will imply its distance to all objects in  $\mathcal{C}'$  is zero, which is a contradiction as we assumed the convex objects do not intersect. Otherwise, at some point  $C$  is no longer the furthest, which implies we must have crossed a bisector  $\beta(C, C')$  for some other convex object  $C' \in \mathcal{C}'$ .

So far we have shown one can assume each center  $s$  either is in  $I(\mathcal{C})$ , or lies on the bisector  $\beta = \beta(C, C')$  of the two objects,  $C, C'$ , which lie furthest away from  $s$  among the set of objects  $\mathcal{C}'$  which intersect the ball  $B(s, r_{opt})$ . In the latter case, let  $T_\beta$  denote the set of all points  $p$  on  $\beta$  such that there exists a third object  $X \in \mathcal{C}$  such that  $\|p - X\| = \|p - C\|$  (or equivalently  $\|p - X\| = \|p - C'\|$ ). Note that such points lie at intersections of bisectors and thus from the above discussion before the lemma, we know  $T_\beta$  is a discrete set. As  $\beta$  is a simple curve, we can view points in  $T_\beta$  as being ordered along  $\beta$ . Suppose that  $s \notin T_\beta$ , and let  $p$  and  $q$  be the points of  $T_\beta$  which come immediately before and after  $s$  along  $\beta$ , and let  $[p, q]$  denote the portion of  $\beta$  lying between these points. (This interval may be unbounded to one side if  $s$  comes after or before all points in  $T_\beta$ .) Recall that  $\mathcal{C}'$  is the subset of objects intersecting  $B(s, r_{opt})$ , and  $C$  and  $C'$  are the furthest from  $s$  among those in  $\mathcal{C}'$ . Observe that for any other point  $z$  in  $[p, q]$ ,  $C$  and  $C'$  must also be the furthest objects from  $z$  among those in  $\mathcal{C}'$ , as otherwise as we move continuously along  $\beta$  from  $s$  to  $z$  we must cross another point from  $T_\beta$  before reaching  $z$  and there are no such points in  $(p, q)$ . Thus if we replace  $s$  with the point in  $[p, q]$  minimizing the distance to  $C$  (or equivalently  $C'$ ) then all objects previously intersected by  $B(s, r_{opt})$  will remain intersected by  $B(s, r_{opt})$ .

Let  $M(\mathcal{C})$  be a set containing, for each bisector  $\beta$ , the set  $T_\beta$  and one minimum distance point from each such interval  $[p, q]$ . We thus have argued that the points of  $S$  can be assumed to lie in  $P = I(\mathcal{C}) \cup M(\mathcal{C})$ . As for the running time and size of these sets, first observe that  $I(\mathcal{C})$  has size  $n$  and can be trivially computed in  $O(n)$  time. For the set  $M(\mathcal{C})$ , first observe that there are  $O(n^2)$  bisectors. For any bisector  $\beta$ , the set  $T_\beta$  of intersection points of  $\beta$  with other bisectors that are equidistant at the intersection point, has size  $O(n)$ , since by general position every point is equidistant to at most 3 objects and as mentioned above any pair of bisectors intersect in a constant number of points. (In other words, we ultimately consider all  $O(n^3)$  points equidistant to three objects, as opposed to all  $O(n^4)$  bisector intersections.) Thus the set  $M(\mathcal{C})$ , and correspondingly  $P$ , has size  $O(n^3)$  as claimed. For the running

## 6:6 Clustering with Neighborhoods

time, as the objects in  $\mathcal{C}$  all have constant complexity, so do their bisectors, and thus  $T_\beta$  can be computed in  $O(n)$  time. The minimum points of  $M(\mathcal{C})$  on  $\beta$ , can thus be computed by sorting  $T_\beta$  along  $\beta$ , in  $O(n \log n)$  time, and then computing the minimum point in constant time for each constant complexity interval between consecutive pairs of points from  $T_\beta$  along  $\beta$ . Thus over all  $O(n^2)$  bisectors it takes  $O(n^3 \log n)$  time to compute  $M(\mathcal{C})$ . ◀

We now argue the canonical sets  $P$  and  $R$  from the above lemma naturally lead to a PTAS for size-approximation by using Minkowski sums. For sets  $A, B \subset \mathbb{R}^2$ , let  $A \oplus B = \{a + b \mid a \in A, b \in B\}$  denote their Minkowski sum. Let  $B(r)$  denote the ball of radius  $r$  centered at the origin. Then we write  $\mathcal{C} \oplus B(r) = \{C \oplus B(r) \mid C \in \mathcal{C}\}$ . A set of points  $S$  is called a *hitting set* for a set of objects if every object has non-empty intersection with  $S$ .

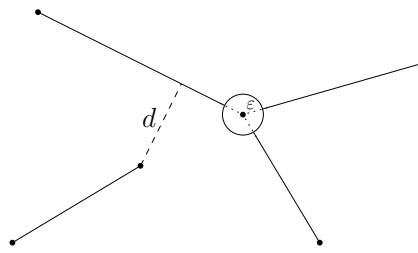
► **Observation 3.** *A set  $S$  of  $k$  centers is a solution to Problem 1 of radius  $r$  if and only if  $S$  is a hitting set of size  $k$  for  $\mathcal{C} \oplus B(r)$ . This holds since for any  $C \in \mathcal{C}$  and  $s \in S$ ,  $B(s, r) \cap C \neq \emptyset$  if and only if  $s \in C \oplus B(r)$ .*

In the geometric hitting set problem we are given a set  $\mathcal{R}$  of  $n$  regions and a set  $P$  of  $m$  points in the plane, and the goal is to select a minimum sized hitting set for  $\mathcal{R}$  using points from  $P$ . The above observation implies we can reduce any given instance  $\mathcal{C}, k$  of Problem 1 to multiple instances of geometric hitting set. Specifically, by Lemma 2, in  $O(n^3 \log n)$  time we can compute a set  $R$  of  $O(n^3)$  values, one of which must be the optimal radius  $r_{opt}$ . Then for each  $r \in R$  we construct a hitting set instance where  $\mathcal{R} = \mathcal{C} \oplus B(r)$ , and  $P$  is the set of points from Lemma 2. By the above observation, if  $r < r_{opt}$ , then the hitting set instance requires more than  $k$  points, and if  $r \geq r_{opt}$  then it requires at most  $k$  points. Therefore, given an algorithm for geometric hitting set we can use it to binary search for  $r_{opt}$ .

While hitting set is in general NP-hard to approximate within logarithmic factors [30], in our case there is a PTAS as the regions are nicely behaved. A collection of regions in the plane is called a set of *pseudo-disks* if the boundaries of any two distinct regions in the set cross at most twice. Mustafa and Ray [29] showed that there is an  $nm^{O(1/\varepsilon^2)}$  time PTAS for geometric hitting set when  $\mathcal{R}$  is a collection of  $n$  pseudo-disks and  $P$  is a set of  $m$  points. It is known that if we take the Minkowski sum of a single convex object with each member of a set of disjoint convex objects, then the resulting set is a collection of pseudo-disks (see for example [2]). Thus  $\mathcal{C} \oplus B(r)$  is a collection of pseudo-disks. Therefore, by the above discussion, we have the following theorem. As the decision procedure is now approximate, the binary search must be modified to look at larger radii when the hitting set algorithm returns  $> (1 + \varepsilon)k$  points, and smaller radii otherwise. (This yields an adjacent pair  $r < r'$  such that  $r < r_{opt}$ , implying  $r' \leq r_{opt}$ , and an  $r'$ -cover of the input using  $\leq (1 + \varepsilon)k$  points.)

► **Theorem 4.** *There is a PTAS for Problem 1 for approximating the optimal solution size. That is, for any fixed  $\varepsilon > 0$ , there is a  $(1 + \varepsilon)$ -size-approximation with running time  $n^{O(1/\varepsilon^2)}$ .*

We remark that the PTAS of [29] implicitly assumes the objects are in general position, that is if two objects intersect then they properly intersect (i.e. their interiors intersect). While  $\mathcal{C}$  satisfies this property, it may not after we take the Minkowski sum with a given radius. However, as we can compute distances between our objects, this is easily overcome by computing the smallest non-zero distance  $d$  between two objects in  $\mathcal{C} \oplus B(r)$ , and instead running the hitting set algorithm on  $\mathcal{C} \oplus B(r + \alpha)$ , where  $\alpha$  is some infinitesimal value less than  $d/2$ . This ensures any objects which intersected in  $\mathcal{C} \oplus B(r)$  now properly intersect, and there are no new intersections.



■ **Figure 4.1** Reducing planar Vertex Cover to Problem 1 for segments.

## 4 Radius Approximation Hardness

In this section we argue that for Problem 1 it is hard to approximate the radius within any factor, even when  $\mathcal{C}$  is restricted to being a set of line segments. Moreover, for the case when  $\mathcal{C}$  is a set of disks, i.e. the case considered in Section 5, we argue the problem is APX-Hard. Our hardness results use a construction similar to the one from [12], where they reduce from the problem of planar vertex cover where the maximum degree of a vertex is three, which is known to be NP-complete [18]. We denote this problem as P3VC.

### 4.1 Line Segments

Here we argue that it is hard to radius-approximate Problem 1 within any factor, even when  $\mathcal{C}$  is a set of line segments. We remark that the following reduction works for any instance of planar vertex cover (i.e. regardless of the degree), but the reduction for disks in the next subsection uses that the degree is at most three.

► **Theorem 5.** *Problem 1 cannot in polynomial time be radius-approximated within any factor that is computable in polynomial time unless  $P = NP$ , even when restricting to the set of instances in which  $\mathcal{C}$  is a set of disjoint line segments.*

**Proof.** Let  $G, k$  be an instance of P3VC. Consider a straight line embedding of  $G$ , and let  $d$  denote the distance between the closest pair of non-adjacent segment edges.<sup>2</sup> Let  $\varepsilon > 0$  be a value strictly smaller than  $d/2$  and strictly smaller than half the length of any segment edge. The set  $\mathcal{C}$  of segments in our instance of Problem 1 will be the segment edges from the embedding, but where each segment has an  $\varepsilon$  amount removed from each end, i.e. we remove all portions of segments in  $\varepsilon$  balls around the vertices, see Figure 4.1. We use the same value of  $k$  in our Problem 1 instance as in the P3VC instance.

If there is a vertex cover of size at most  $k$  then if we place balls of radius  $\varepsilon$  at each of the  $k$  corresponding vertices of the embedding, then these balls will intersect all segments in  $\mathcal{C}$ , i.e. we have a solution to Problem 1 of radius  $\varepsilon$ . On the other hand, by the definition of  $d$ , any ball of radius  $< d/2$  cannot simultaneously intersect two segments from  $\mathcal{C}$  if they correspond to non-adjacent edges from  $G$ . (Note when we shrunk the edges by  $\varepsilon$  this could only have made them further apart.) Thus if the minimum vertex cover requires  $> k$  vertices, then our instance of Problem 1 requires  $> k$  centers if we limit to balls with radius  $< d/2$ .

<sup>2</sup> In  $O(n \log n)$  time one can compute a straight line embedding of  $G$  where the vertices are on an  $(2n - 4) \times (n - 2)$  grid [14]. This implies a lower bound on  $d$  with a polynomial number of bits.

Therefore, if we could approximate the minimum radius of our Problem 1 instance within any factor less than  $d/(2\varepsilon)$  then we can determine whether the corresponding vertex cover instance had a solution with  $\leq k$  vertices. However, we are free to make  $\varepsilon > 0$  as small as we want and thus  $d/(2\varepsilon)$  as large as we want, so long as this quantity (or more precisely a lower bound on it) is computable in polynomial time.  $\blacktriangleleft$

## 4.2 Disks

Here we argue that it is hard to radius-approximate Problem 1 within a constant factor when  $\mathcal{C}$  is restricted to be a set of unit disks. The following reduction from P3VC is similar to the one given in [12], which embeds the graph such that edges are replaced by odd length sequences of points. In our case, these odd length sequences of points are instead replaced with odd length sequences of appropriately spaced disks.

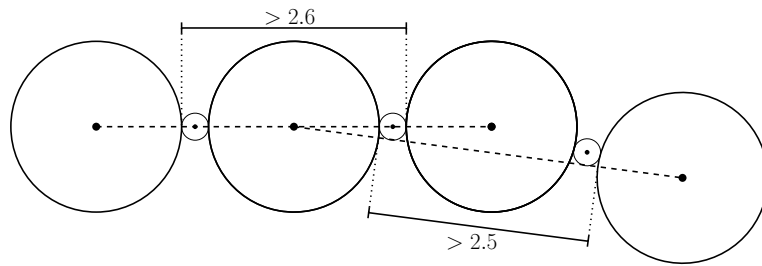
**► Theorem 6.** *For the set of instances in which  $\mathcal{C}$  is a set of disjoint unit disks, Problem 1 cannot be radius-approximated to any factor less than  $\frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}}$  in polynomial time unless  $P = NP$ .*

**Proof.** To simplify our construction description, instead of requiring the disks be disjoint, we allow them to intersect at their boundaries, but not their interiors. Later we remark how this easily implies the result for the disjoint disk case.

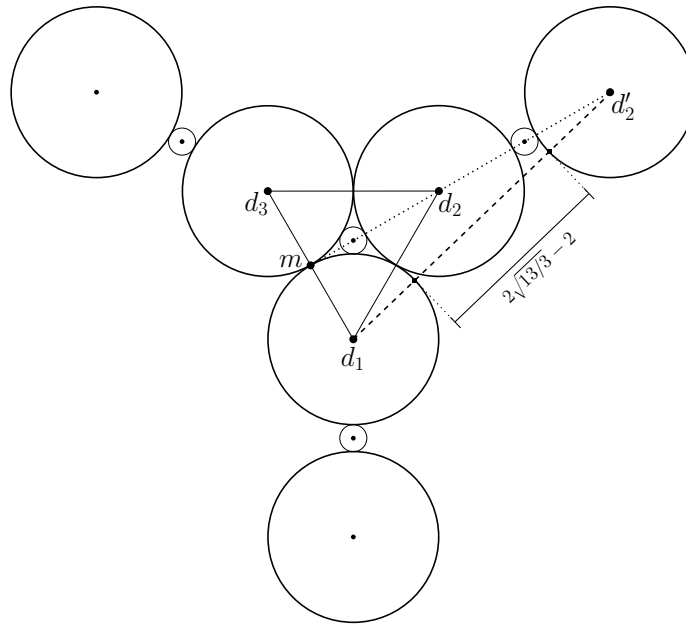
So let  $G = (V, E)$ ,  $k$  be an instance of P3VC. For every edge in  $E$  we create a sequence of an odd number (greater than 1) of unit disks, where consecutive disks in the sequence are spaced  $2(2/\sqrt{3} - 1)$  apart from one another. (Note  $2(2/\sqrt{3} - 1)$  is the distance between the disks, not their centers.) For a vertex  $v$  of degree two, we place the disks corresponding to the  $v$  end of the adjacent edges again at distance  $2(2/\sqrt{3} - 1)$  apart. For a vertex  $v$  of degree three, we place the disks corresponding to the  $v$  end of the adjacent edges such that they all just touch one another at their boundaries, see Figure 4.3. Thus the centers of these disks form an equilateral triangle, and let the center point of this triangle be  $t$ . For any one of the adjacent edges, we further require that the centers of the first two disks (on the  $v$  end of the edge) lie on a straight line containing  $t$ , in other words the edges leaving  $v$  do not bend until several disks away from  $v$ . As  $G$  is a planar graph with maximum degree three such an embedding of polynomial size is possible, similar to the case in [12]. Doing so requires using different numbers of disks for each edge and allowing the edges to bend (i.e. the centers of three consecutive disks of an edge may not lie on a line). However, we will require these bends to be gradual. Specifically, observe that if the centers of three consecutive disks of an edge were on a straight line, the distance between the two non-consecutive disks would be  $2(1 + 2(2/\sqrt{3} - 1)) > 2.6$ , see Figure 4.2. We then require that the bends are shallow enough such that two non-consecutive disks of an edge are more than 2.5 apart. We also require this for disks from edges adjacent to a degree two vertex (when they are not both the disks immediately adjacent at the vertex), or a degree three vertex when neither disk is one of corresponding three touching disks of the vertex. Finally, for disks that come from edges that are not adjacent, we easily enforce that they are again more than 2.5 apart. (This is similar to the value  $d$  from Theorem 5.)

So given an instance  $G, k$  of P3VC, we construct an instance  $\mathcal{C}, \kappa$  of Problem 1 where  $\mathcal{C}$  is determined from  $G$  as described above and  $\kappa = k + (|\mathcal{C}| - |E|)/2$ . We first argue if  $G$  has a vertex cover of size  $k$  then for our instance of Problem 1 there is a solution of radius  $2/\sqrt{3} - 1$ . First, for any vertex  $v$  in the vertex cover we create a center, and roughly speaking place it at the location of  $v$  in the embedding. Namely, if  $v$  had degree two then we place the center at the midpoint of the centers of the disks at the ends of the edges adjacent to  $v$ , which





■ **Figure 4.2** Consecutive disks along an edge.



■ **Figure 4.3** Three touching disks corresponding to a degree three vertex.

by construction are exactly  $2(2/\sqrt{3} - 1)$  apart and thus a ball at the midpoint with radius  $2/\sqrt{3} - 1$  intersects both. If  $v$  has degree three then we place a center at the center point  $t$  of the equilateral triangle determined by three touching disks of the adjacent edges. An easy calculation<sup>3</sup> shows that since our disks have unit radius, that  $B(t, 2/\sqrt{3} - 1)$  intersects the three touching disks. We now cover the remaining disks with  $(|\mathcal{C}| - |E|)/2$  centers. For any edge  $e \in E$  let  $n_e$  be the number of disks used for  $e$  in the above construction. Observe that as we already placed centers at vertices corresponding to a vertex cover of the edges, at least one disk at the end of each edge is already covered, and so there are at most  $n_e - 1$  consecutive disks that need to be covered. (Note  $n_e - 1$  is even.) However, as consecutive disks are exactly  $2(2/\sqrt{3} - 1)$  apart on each edge, these  $n_e - 1$  disks can be covered with  $(n_e - 1)/2$  balls of radius  $(2/\sqrt{3} - 1)$  by covering the disks in pairs. Thus the total number of centers used is  $k + \sum_{e \in E} (n_e - 1)/2 = k + (|\mathcal{C}| - |E|)/2 = \kappa$ .

Now suppose the minimum vertex cover of  $G$  requires  $> k$  vertices. In this case we argue that our instance of Problem 1 requires more than  $\kappa$  centers if we limit to balls with radius  $< \sqrt{13}/3 - 1$ . Call any two disks in  $\mathcal{C}$  neighboring if they are consecutive on an

<sup>3</sup> For an equilateral triangle with edge length 2, the distance from an edge to the center point of the triangle is  $1/\sqrt{3}$ , thus the distance from the center point to any one of the unit balls is  $2/\sqrt{3} - 1$ .

## 6:10 Clustering with Neighborhoods

edge or if they are disks on the  $v$  end of two edges adjacent to a vertex  $v$ . By construction, neighboring disks have distance  $\leq 2/\sqrt{3} - 1$  from each other. For a pair of disks which are not neighboring we now argue their distance is at least  $2\sqrt{13}/3 - 2$ . Specifically, if these disks come from the same edge but are not consecutive along that edge, or if they are from distinct edges that are either non-adjacent or are adjacent to a degree two vertex (but not the two disks of that vertex), then by construction their distance is  $> 2.5 > 2\sqrt{13}/3 - 2$ . The remaining case is when the disks are from distinct edges adjacent to a degree three vertex, but they are not both from the three touching disks of the vertex. It is easy to see that the closest two such disks can be is when one of the disks is one of the three touching disks, and the other is the second disk on another edge. We now calculate the distance between two such disks, see Figure 4.3. Let the three touching disks be denoted  $D_1$ ,  $D_2$ , and  $D_3$ , with centers  $d_1$ ,  $d_2$ , and  $d_3$ , respectively. Let  $D'_2$  denote the second disk on the edge containing  $D_2$ , and let its center be  $d'_2$ . We wish to compute  $\|D_1 - D'_2\| = \|d_1 - d'_2\| - 2$ , as these are unit disks. Let  $m$  denote the midpoint of  $d_1$  and  $d_3$ , and observe that the line through  $d_2$  and  $d'_2$  passes through  $m$  and is orthogonal to the line through  $d_1$  and  $d_3$ , as the points  $d_1$ ,  $d_2$ , and  $d_3$  form an equilateral triangle. Thus by the Pythagorean theorem we have  $\|d_1 - d'_2\|^2 = 1^2 + (1 + 2(2/\sqrt{3} - 1) + 1 + \sqrt{3})^2 = 1 + (4/\sqrt{3} + \sqrt{3})^2 = 52/3$ , where the  $+\sqrt{3}$  term is the height of an equilateral triangle of side length 2. Thus  $\|D_1 - D'_2\| = \|d_1 - d'_2\| - 2 = 2\sqrt{13}/3 - 2$ .

Now we finish the argument that when the minimum vertex cover of  $G$  requires  $> k$  vertices, our instance of Problem 1 requires more than  $\kappa$  centers if we limit to balls with radius  $< \sqrt{13}/3 - 1$ . By the above, limiting to radius  $< \sqrt{13}/3 - 1$  implies that any ball either covers just a single disk, or a pair of neighboring disks. An edge  $e$  with  $n_e$  disks thus requires at least  $\lceil n_e/2 \rceil = 1 + (n_e - 1)/2$  disks to cover it. Moreover, a ball can only cover both a disk of  $e$  and  $e'$  if those disks are on the  $v$  end of two edges adjacent to  $v$ . Let  $E_z$  be the subset of edges with at least one disk covered by such a ball (i.e. a ball corresponding to a vertex), and let  $z$  be the number of such balls. Then the total number of balls required is

$$\begin{aligned} &\geq z + \sum_{e \in E_z} (n_e - 1)/2 + \sum_{e \in E \setminus E_z} (1 + (n_e - 1)/2) \\ &= z + (|\mathcal{C}| - |E|)/2 + |E \setminus E_z| = z + (\kappa - k) + |E \setminus E_z|, \end{aligned}$$

which is more than  $\kappa$  when  $z + |E \setminus E_z| > k$ . Notice, however, there is a vertex cover of  $G$  of size  $z + |E \setminus E_z|$ , consisting of the vertices that  $z$  counted, and one vertex from either end of each edge in  $E \setminus E_z$ . Thus as the minimum vertex cover has size  $> k$ , we have  $z + |E \setminus E_z| > k$  as desired.

Therefore, if we could approximate the minimum radius of our Problem 1 instance within any factor less than  $\frac{\sqrt{13/3}-1}{2/\sqrt{3}-1} = \frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}}$  then we can determine whether the corresponding vertex cover instance had a solution with  $\leq k$  vertices. In the above analysis the boundaries of the circles were allowed to intersect, but we can enforce that all disks are disjoint without changing the approximation hardness factor since we showed the problem is hard for any factor that is less than  $\frac{\sqrt{13}-\sqrt{3}}{2-\sqrt{3}}$ . Specifically, rather than having the disks for a degree three vertex touch, we can instead make them arbitrarily close to touching. ◀

## 5 Constant Factor Radius Approximation for Disks

In this section we argue that when  $\mathcal{C}$  is a set of disjoint disks (of possibly differing radii), that there is a constant factor radius-approximation for Problem 1.



► **Lemma 7.** *Let  $\mathcal{C}$  be a set of pairwise disjoint disks such that for all  $C \in \mathcal{C}$ , the radius of  $C$  is  $\geq r$ . If there is a point  $s \in \mathbb{R}^2$  where  $\|s - C\| \leq (2/\sqrt{3} - 1)r$  for all  $C \in \mathcal{C}$ , then  $|\mathcal{C}| \leq 2$ .*

**Proof.** We give a proof by contradiction. So suppose there exists a point  $s$  such that there are three disjoint disks in  $\mathcal{C}$ , each with radius  $\geq r$ , and all of which intersect the ball  $B(s, (\frac{2}{\sqrt{3}} - 1)r)$ . Observe that if any one of these three disjoint disks  $C$  has radius  $> r$ , then it can be replaced by a disk  $C'$  of radius  $r$  such that  $C' \subset C$  and  $C'$  still intersects  $B(s, (\frac{2}{\sqrt{3}} - 1)r)$ . As these new disks are all still disjoint and intersect  $B(s, (\frac{2}{\sqrt{3}} - 1)r)$ , it suffices to argue we get a contradiction when all three disks have radius exactly  $r$ . Let the centers of these three disks be denoted  $x, y$ , and  $z$ . Now, at least one of the angles  $\angle xsy$ ,  $\angle ysz$ , and  $\angle zsx$  is  $\leq 2\pi/3$ . Without loss of generality assume it is  $\angle xsy$ , and let  $\gamma = \angle xsy$ .

Consider the triangle  $\triangle sxy$ , and let its side lengths be denoted  $a = \|x - s\|, b = \|y - s\|, c = \|x - y\|$ . Since  $\gamma \leq 2\pi/3$ , by the Law of Cosines we thus have  $c^2 = a^2 + b^2 - 2ab \cos(\gamma) \leq a^2 + b^2 + ab$ . As the  $r$  radius disks with centers  $x$  and  $y$  are disjoint, we know that  $2r < c$ . Combining these two inequalities we get  $4r^2 < a^2 + b^2 + ab$ . As  $B(s, (\frac{2}{\sqrt{3}} - 1)r)$  intersects the  $r$  radius disks centered at both  $x$  and at  $y$ , we also have that  $a, b \leq (\frac{2}{\sqrt{3}} - 1)r + r = \frac{2r}{\sqrt{3}}$ . Combining this with the previous inequality gives  $4r^2 < a^2 + b^2 + ab \leq 4r^2/3 + 4r^2/3 + 4r^2/3 = 4r^2$ , which is a clear contradiction and thus the number of disks in  $\mathcal{C}$  is at most 2. ◀

For any constant  $c \geq 1$ , we call an algorithm a  $c$ -decider for Problem 1, if for a given instance with optimal radius  $r_{opt}$ , and for any given query radius  $r$ , if  $r \geq r_{opt}$  then the algorithm returns a solution  $S$  of radius  $\leq cr$ , and if  $r < r_{opt}/c$  it returns False (for  $r_{opt}/c \leq r < r_{opt}$  either answer is allowed).

► **Lemma 8.** *There is an  $O(n^{2.5})$  time  $(5 + 2\sqrt{3})$ -decider for Problem 1, when restricted to instances where  $\mathcal{C}$  is a set of disjoint disks.*

**Proof.** Let  $r$  be the given query radius. We build a set  $S$  of centers as follows, where initially  $S = \emptyset$ . Let  $P$  be the set of center points of all disks in  $\mathcal{C}$  with radius  $< (3 + 2\sqrt{3})r$ . Until  $P$  is empty repeatedly add an arbitrary point  $p \in P$  to the set  $S$ , remove all disks from  $\mathcal{C}$  which intersect  $B(p, (5 + 2\sqrt{3})r)$ , and remove all center points from  $P$  corresponding to disks removed from  $\mathcal{C}$ . Let  $S_1$  refer to the resulting set of centers. For the remaining set of disks  $\mathcal{C}'$ , define the subset  $\mathcal{C}'' = \{C \in \mathcal{C}' \mid \exists D \in \mathcal{C}' \setminus \{C\} \text{ s.t. } \|C - D\| \leq 2r\}$ . First, for every disk  $C$  in  $\mathcal{C}' \setminus \mathcal{C}''$  we add an arbitrary point from  $C$  to  $S$ . Let this set of added centers be denoted  $S_2$ . Now for the set  $\mathcal{C}''$  we construct a graph  $G = (V, E)$  where  $V = \mathcal{C}''$  and there is an edge from  $C$  to  $D$  if and only if  $\|C - D\| \leq 2r$ . Let  $\mathcal{E}$  be a minimum edge cover of  $G$ . (Note every vertex in  $G$  has an adjacent edge by the definition of  $\mathcal{C}''$  and thus  $\mathcal{E}$  exists.) For every edge  $(C, D) \in \mathcal{E}$ ,  $\|C - D\| \leq 2r$  and thus there is a point  $p \in \mathbb{R}^2$  such that  $\|p - C\|, \|p - D\| \leq r$ . So finally, for each  $(C, D) \in \mathcal{E}$  we add this corresponding point  $p$  to  $S$ . Let this final set of added centers be denoted  $S_3$ . If  $|S| \leq k$  we return  $S$  (which is the disjoint union of  $S_1, S_2$ , and  $S_3$ ) and otherwise we return False.

To prove the above algorithm is a  $(5 + 2\sqrt{3})$ -decider, first we argue that if  $r < r_{opt}/(5 + 2\sqrt{3})$  then it returns False. To do so we prove the contrapositive. So assume  $|S| \leq k$ . Let  $S_1, S_2$ , and  $S_3$ , and  $\mathcal{C}'' \subseteq \mathcal{C}' \subseteq \mathcal{C}$  be as defined above. As we used balls of radius  $(5 + 2\sqrt{3})r$ , all  $C \in \mathcal{C} \setminus \mathcal{C}'$  are within distance  $(5 + 2\sqrt{3})r$  of points in  $S_1$ . All  $C \in \mathcal{C}' \setminus \mathcal{C}''$  have distance zero to a point in  $S_2$ . Finally, all  $C \in \mathcal{C}''$  have distance  $\leq r$  to a point in  $S_3$ . As  $S$  is the disjoint union of  $S_1, S_2$ , and  $S_3$ , we thus have that all  $C \in \mathcal{C}$  are within distance  $(5 + 2\sqrt{3})r$  to a set  $S$  with  $\leq k$  points, which by the definition of Problem 1 means that  $r_{opt} \leq (5 + 2\sqrt{3})r$ .

Now suppose  $r \geq r_{opt}$ , where  $r_{opt}$  is the optimal radius for the given instance  $\mathcal{C}, k$  of Problem 1. In order to prove the algorithm is a  $(5 + 2\sqrt{3})$ -decider, in this case we must argue it returns a  $\leq (5 + 2\sqrt{3})r$  radius solution. As already shown above, if the algorithm

returns a solution then it has radius  $\leq (5 + 2\sqrt{3})r$ , thus all we must argue is that a solution is returned, namely that  $|S| \leq k$ . So fix an optimal solution  $S^*$  for the original input instance  $\mathcal{C}, k$ . We argue that there are disjoint subsets  $S_1^*, S_2^*$ , and  $S_3^*$  of  $S^*$  such that  $|S_1^*| \geq |S_1|$ ,  $|S_2^*| \geq |S_2|$ , and  $|S_3^*| \geq |S_3|$ , and therefore  $|S| \leq |S^*| = k$ .

Let the points in  $S_1 = \{t_1, \dots, t_{|S_1|}\}$  be indexed in the order they were selected. Consider the point  $t_i$ , which is the center of some disk  $C_i \in \mathcal{C}$  with radius  $\leq (3 + 2\sqrt{3})r$ . Let  $U_i = \cup_{j \leq i} B(t_j, (5 + 2\sqrt{3})r)$ . Define  $S_1^*$  as the centers  $s$  of  $S^*$  such that  $B(s, r) \subseteq U_{|S_1|}$ . To argue  $|S_1^*| \geq |S_1|$ , it suffices to argue that for all  $i$  there exists some  $s \in S^*$  such that  $B(s, r) \not\subseteq U_{i-1}$  while  $B(s, r) \subseteq U_i$  (i.e.  $s$  gets charged uniquely to  $t_i$ ). Now there must be some center  $s \in S^*$  such that  $B(s, r_{opt}) \cap C_i \neq \emptyset$ , as  $S^*$  covers  $\mathcal{C}$  with radius  $r_{opt}$ . Moreover, since  $r \geq r_{opt}$ , we have  $B(s, r) \not\subseteq U_{i-1}$ , since otherwise it implies  $U_{i-1} \cap C_i \neq \emptyset$  and thus  $t_i$  could not have been selected in the  $i$ th round as the algorithm had already removed it from  $P$ . Conversely,  $B(s, r) \subseteq U_i$ , since  $B(s, r)$  intersects  $C_i$  and  $C_i$  has radius  $\leq (3 + 2\sqrt{3})r$ , and thus  $B(s, r) \subseteq B(t_i, (5 + 2\sqrt{3})r) \subseteq U_i$ . Therefore  $|S_1^*| \geq |S_1|$ .

For any  $s \in S_1^*$ ,  $B(s, r) \subseteq U_{|S_1|}$ , and since the disks of  $\mathcal{C}'$  do not intersect  $U_{|S_1|}$ , in the optimal solution  $\mathcal{C}'$  must be  $r_{opt}$ -covered only using centers from  $S^* \setminus S_1^*$ . Let  $S_2^*$  be the subset of centers from  $S^* \setminus S_1^*$  which  $r_{opt}$  covers  $\mathcal{C}' \setminus \mathcal{C}''$ . Since any disk  $C \in (\mathcal{C}' \setminus \mathcal{C}'')$  has distance  $> 2r$  to its nearest neighbor in  $\mathcal{C}' \setminus \{C\}$  and  $r_{opt} \leq r$ , the optimal solution must use a distinct center to cover each disk in  $\mathcal{C}' \setminus \mathcal{C}''$ , i.e.  $|S_2^*| \geq |S_2|$ , and moreover,  $\mathcal{C}''$  must be covered in the optimal solution by  $S^* \setminus (S_1^* \cup S_2^*)$ . So finally, let  $S_3^*$  be the subset of centers from  $S^* \setminus (S_1^* \cup S_2^*)$  which  $r_{opt}$  covers  $\mathcal{C}''$ . By construction, the radius of each  $C \in \mathcal{C}''$  is  $\geq (3 + 2\sqrt{3})r$ . Thus, by Lemma 7 any point from  $S_3^*$  can  $(2/\sqrt{3} - 1) \cdot (3 + 2\sqrt{3})r = r \geq r_{opt}$  cover at most 2 disks from  $\mathcal{C}''$ . Now the graph  $G$ , for which our algorithm computes a minimum edge cover  $\mathcal{E}$ , contains an edge for every pair of disks which can be simultaneously covered with a single  $r$  radius ball. Therefore  $|S_3^*| \geq |\mathcal{E}| = |S_3|$ .

For the running time, computing the set  $P$  takes  $O(n)$  time. Selecting a new point  $p \in P$  and removing all disks from  $\mathcal{C}$  which intersect  $B(p, (5 + 2\sqrt{3})r)$  can be done in  $O(n)$  time, and thus repeating this till  $P$  is empty takes  $O(n^2)$  time. Determining the subset  $\mathcal{C}''$ , and hence the graph  $G$ , can naively be done in  $O(n^2)$  by checking the distances between all pairs in  $\mathcal{C}'$ . Selecting a point from each  $C \in (\mathcal{C}' \setminus \mathcal{C}'')$  takes  $O(n)$  time. Finally, since computing a minimum edge cover can be reduced to computing a maximum matching,  $\mathcal{E}$  can be found in  $O(n^{2.5})$  time (see [28]).  $\blacktriangleleft$

We remark that it should be possible to improve the running time of the above decision procedure, by arguing that the graph  $G$  it constructs is sparse. However, ultimately that will not improve the running time of the following optimization procedure, as it searches over the  $O(n^3)$  sized set of Lemma 2.

► **Theorem 9.** *There is an  $O(n^3 \log n)$  time  $(5 + 2\sqrt{3})$ -radius-approximation algorithm for Problem 1, when restricted to instances where  $\mathcal{C}$  is a set of disjoint disks.*

**Proof.** By Lemma 2, in  $O(n^3 \log n)$  time we can compute an  $O(n^3)$  sized set  $R$  of values, such that  $r_{opt} \in R$ , where  $r_{opt}$  is the optimal radius. So sort the values in  $R$ , and then binary search over them using the  $(5 + 2\sqrt{3})$ -decider of Lemma 8, which we denote `decider`( $r$ ). Specifically, if `decider` returns False we recurse to the right, and if it returns a solution (i.e. True) then we recurse on the left. Note that since our decision procedure is approximate, the values for which it returns True or for which it returns False may not be contiguous in the sorted order of  $R$ . Regardless, however, our binary search allows us to find a pair  $r' < r$  which are consecutive in  $R$  and such that `decider`( $r'$ ) is False, and `decider`( $r$ ) is True. (Unless `decider` always returns True, in which case it returns the smallest value in  $R$ .) By Lemma 8

decider is a  $(5 + 2\sqrt{3})$ -decider, and thus since  $\text{decider}(r')$  is False by definition we have that  $r' < r_{opt}$ . However, as  $r' < r$  are consecutive in the sorted order of  $R$  and since  $r_{opt} \in R$ , this implies  $r_{opt} \geq r$ . On the other hand, again by the definition of a  $(5 + 2\sqrt{3})$ -decider,  $\text{decider}(r)$  outputs a solution with radius at most  $(5 + 2\sqrt{3})r \leq (5 + 2\sqrt{3})r_{opt}$ , thus giving us a  $(5 + 2\sqrt{3})$ -approximation as claimed.

By Lemma 2, computing and sorting the  $O(n^3)$  values in  $R$  takes  $O(n^3 \log n)$  time. By Lemma 8 each call to  $\text{decider}$  takes  $O(n^{2.5})$  time, and since we are binary searching over  $O(n^3)$  values, the time for all calls to  $\text{decider}$  is  $O(n^{2.5} \log(n^3)) = O(n^{2.5} \log n)$ . Thus the total time is  $O(n^3 \log n)$  as claimed.  $\blacktriangleleft$

Our focus in this paper is on the planar case, however, in the full version [22] we remark how the above decision procedure works in higher dimensions. The above optimization procedure does not immediately extend as it makes use of Lemma 2, however, in the full version we informally sketch how one can approximately recover the same result.

## 6 An Efficient FPTAS for Bounded $k$

By Lemma 2, we can compute a set of  $O(n^3)$  points which contains a subset of size  $k$  that is an optimal  $k$ -center solution. Thus, for constant  $k$ , enumerating all  $O(n^{3k})$  possible subsets, and taking the minimum cost found, yields a polynomial time algorithm. In this section, we argue that for constant  $k$ , we can achieve a  $(1 + \varepsilon)$ -radius-approximation for unit disks, whose running time depends only linearly on  $n$ . Contrast this with Theorem 6, where we argued that when  $k$  is not assumed to be constant, that the problem is hard to approximate for unit disks within a given constant factor. We use the following from Agarwal and Procopiuc [3].

► **Theorem 10** ([3]). *Given a set  $P$  of  $n$  points in the plane, there is an  $O(n \log k) + (k/\varepsilon)^{O(\sqrt{k})}$  time  $(1 + \varepsilon)$ -radius-approximation algorithm for  $k$ -center, denoted  $\mathbf{kCenter}(\varepsilon, P)$ .*

► **Theorem 11.** *There is an  $O(n \log k) + (k/\varepsilon)^{O(k)}$  time  $(1 + \varepsilon)$ -radius-approximation algorithm for Problem 1, when restricted to instances where  $\mathcal{C}$  is a set of disjoint unit disks.*

**Proof.** Let  $P$  denote the set of center points of the disks in  $\mathcal{C}$ . For any given set  $S$  of  $k$  points in the plane, let  $r_P(S) = \max_{p \in P} \|p - S\|$  and  $r_{\mathcal{C}}(S) = \max_{C \in \mathcal{C}} \|C - S\|$ . Observe that  $r_{\mathcal{C}}(S) \leq r_P(S) \leq r_{\mathcal{C}}(S) + 1$ . Specifically,  $r_{\mathcal{C}}(S) \leq r_P(S)$  since any ball (in particular one centered at a point from  $S$ ) which contains a center point from  $P$  also intersects the corresponding disk in  $\mathcal{C}$ . On the other hand,  $r_P(S) \leq r_{\mathcal{C}}(S) + 1$  since for any ball intersecting a disk in  $\mathcal{C}$ , if we increase its radius by 1 then it will contain the center point of that disk, as  $\mathcal{C}$  consists of unit disks.

Let  $r_{opt}$  denote the optimum radius for the given instance  $\mathcal{C}, k$  of Problem 1. We consider two cases based on the value of  $r_{opt}$ . First, suppose that  $r_{opt} > 2/\varepsilon$ . Let  $S'$  denote the solution returned by  $\mathbf{kCenter}(\varepsilon/3, P)$ . By the above inequalities and Theorem 10,

$$\begin{aligned} r_{\mathcal{C}}(S') &\leq r_P(S') \leq (1 + \varepsilon/3) \min_{S \subset \mathbb{R}^2, |S|=k} r_P(S) \leq (1 + \varepsilon/3) \left(1 + \min_{S \subset \mathbb{R}^2, |S|=k} r_{\mathcal{C}}(S)\right) \\ &= (1 + \varepsilon/3)(1 + r_{opt}) < (1 + \varepsilon/3)(\varepsilon r_{opt}/2 + r_{opt}) = (1 + \varepsilon/3)(1 + \varepsilon/2)r_{opt} \leq (1 + \varepsilon)r_{opt}, \end{aligned}$$

where the last inequality assumed  $\varepsilon \leq 1$ . Thus  $S'$  is  $(1 + \varepsilon)$ -approximation for Problem 1.

Now suppose that  $r_{opt} \leq 2/\varepsilon$ . In this case observe that for any point  $x \in \mathbb{R}^2$ , the ball  $B(x, r_{opt})$  can intersect only  $O(1/\varepsilon^2)$  disks from  $\mathcal{C}$  as they are disjoint and all have radius 1. Thus any center from the optimal solution can cover at most  $O(1/\varepsilon^2)$  disks within the optimal radius, and so it must be that  $n = O(k/\varepsilon^2)$ .

The algorithm is now straightforward. If  $n \leq \gamma k/\varepsilon^2$ , for some sufficiently large constant  $\gamma$ , then by Lemma 2 in  $O((k/\varepsilon^2)^3 \log(k/\varepsilon))$  time we can compute a set  $P$  of  $O((k/\varepsilon^2)^3)$  points such that  $P$  contains an optimal set of  $k$  centers. We try all possible subsets of  $P$  of size  $k$  and take the best one. There are  $O((k/\varepsilon^2)^{3k})$  such subsets, and for each subset its cost can be determined in  $O(kn) = O((k/\varepsilon)^2)$  time. Thus in this case we can compute the optimal solution in  $O((k/\varepsilon)^2 \cdot (k/\varepsilon^2)^{3k}) = (k/\varepsilon)^{O(k)}$  time.

On the other hand, if  $n > \gamma k/\varepsilon^2$  then the above implies  $r_{opt} > 2/\varepsilon$ . In this case it was argued above that  $\mathbf{kCenter}(\varepsilon/3, P)$  returns a  $(1 + \varepsilon)$ -approximation, and by Theorem 10 it does so in  $O(n \log k) + (k/\varepsilon)^{O(\sqrt{k})}$  time. In either case, we have a  $(1 + \varepsilon)$ -approximation (or better) and the total time is  $\max\{(k/\varepsilon)^{O(k)}, O(n \log k) + (k/\varepsilon)^{O(\sqrt{k})}\}$ . ◀

## 7 One Dimensional Clustering with Neighborhoods

In this section we show that despite clustering with neighborhoods being hard to radius approximate within any factor in the plane, we can solve the one dimensional variant exactly in  $O(n \log n)$  time, even when object intersections are allowed. First, we argue the decision problem can be solved in linear time. Then we argue that we can use a scheme similar to that in [15] to search for the optimal radius.

In one dimension, a convex object is just a closed interval. Thus we have the following one dimensional version of Problem 1, where intersections are no longer prohibited.

► **Problem 12** (One Dimensional Clustering with Neighborhoods). *Given a set  $\mathcal{C}$  of  $n$  closed intervals on the real line, and an integer parameter  $k \geq 0$ , find a set of  $k$  points  $S$  (called centers) which minimize the maximum distance to an interval in  $\mathcal{C}$ . That is,*

$$S = \arg \min_{S' \subset \mathbb{R}, |S'|=k} \max_{C \in \mathcal{C}} \|C - S'\|.$$

The following decision procedure is similar in spirit to various folklore results for interval problems in one dimension (for example, see the discussion in [13] on interval stabbing). The challenge is turning this decision procedure into an efficient optimization procedure, for which as discussed below we make use of [15].

We first sort the intervals in increasing order both by their left and by their right endpoints. We maintain cross links between the two sorted lists so that if we remove an interval from one list, its copy in the other list can be removed in constant time.

► **Lemma 13.** *Given an instance  $\mathcal{C}, k$  of Problem 12, where the intervals have been presorted, for any query radius  $r$ , in  $O(n)$  time one can decide whether  $r \geq r_{opt}$ .*

**Proof.** We build a set  $S$  of centers as follows, where initially  $S = \emptyset$ . Let  $[\alpha, \beta]$  denote the interval with the leftmost right endpoint (i.e.  $\beta$  is smallest among all intervals). We place a center at  $\beta + r$  and add it  $S$ . Next we remove all intervals which intersect the ball  $B(\beta + r, r)$ . Note that these intersecting intervals are precisely those whose left endpoint is  $\leq \beta + 2r$ , as this condition is clearly necessary to intersect  $B(\beta + r, r)$ , but also sufficient as all intervals have right end point  $\geq \beta$ . We then repeat this process until all intervals are removed. If  $|S| \leq k$  we return True and otherwise we return False.

Observe that every time we place a center, we remove intervals it covers within distance  $r$ . Thus the final set  $S$  is a set of centers of radius  $r$ , and so if  $|S| \leq k$ , then  $r \geq r_{opt}$  and the algorithm correctly returns True. Moreover, we now argue that  $S$  is a minimum cardinality set of centers of radius  $r$ , and thus if  $|S| > k$  then the algorithm correctly returns False. Adopting notation from above, let  $[\alpha, \beta]$  be the interval with leftmost right endpoint, and

let  $c$  be the center our algorithm places at  $\beta + r$ . Now in the minimum cardinality solution, there must be at least one center  $c'$  within distance  $r$  from  $[\alpha, \beta]$ , implying the location of  $c'$  is  $\leq \beta + r$ . Thus  $c'$  can only  $r$ -cover intervals with left endpoint  $\leq \beta + 2r$ . However, as described above,  $c$   $r$ -covers all intervals with left endpoint  $\leq \beta + 2r$ , and thus  $c'$   $r$ -covers a subset of those  $c$  does. Conversely, the subset of intervals not  $r$ -covered by  $c$  is a subset of those not  $r$ -covered by  $c'$ . By induction our algorithm uses the smallest possible number of centers to  $r$ -cover the intervals not  $r$ -covered by  $c$ , which therefore is at most the number centers the global minimum solution uses to  $r$ -cover the superset of intervals not  $r$ -covered by  $c'$ . Thus overall our set of centers was an  $r$ -cover of minimum cardinality.

For the running time, observe that determining the location of the next center takes constant time since it only depends on the leftmost right endpoint, and we assumed we have the sorted ordering of the intervals by right endpoint. Moreover, we can remove all of the intervals intersecting the  $r$  radius ball at the new center in time linear in the number of intersecting intervals, since as discussed above these intersecting intervals are a prefix of the sorted ordering by left endpoint. As we spend constant time per interval removed, overall this is an  $O(n)$  time algorithm. ◀

Lemma 13 gives us a decision procedure for Problem 12 which we now wish to utilize to search for the optimum radius. We use the following lemma to reduce the search space, which can be seen as a simplification of Lemma 2 for the one dimensional case, where here we only need to consider distances from bisecting points rather than bisecting curves.

► **Lemma 14.** *Let  $\mathcal{C}$  be a set of closed intervals. Then for any value  $k$ , the optimal radius for the instance  $\mathcal{C}, k$  of Problem 12 is either 0 or  $\|C - C'\|/2$  for some pair  $C, C' \in \mathcal{C}$ .*

**Proof.** For any value  $k$ , let  $S$  be an optimal solution with optimal radius  $r_{opt}$ . Consider an arbitrary center  $s \in S$ , and let  $\mathcal{C}'$  be the subset of  $\mathcal{C}$  which intersects the ball  $B(s, r_{opt})$ . We can assume that  $|\mathcal{C}'| \geq 1$ , as otherwise  $B(s, r_{opt})$  does not intersect any interval and so  $s$  can be thrown out. If  $|\mathcal{C}'| = 1$ , then  $s$  intersects only one interval, and thus without loss of generality  $s$  can be placed inside this interval, i.e. at distance 0 from it. So assume  $|\mathcal{C}'| > 1$ , and let  $C$  be the furthest interval from  $s$  in  $\mathcal{C}'$ . As we move  $s$  towards  $C$ , so long as  $C$  remains the furthest interval from  $s$  in  $\mathcal{C}'$ ,  $B(s, \|s - C\|)$  will continue to intersect all intervals in  $\mathcal{C}'$ . If  $C$  always remains the furthest, when  $s$  eventually reaches  $C$ , its distance to  $C$  and hence all of  $\mathcal{C}'$  will be 0. Otherwise, if before we reach  $C$ ,  $s$  is no longer the furthest from  $s$ , then we must have crossed the bisector point between  $C$  and some other interval in  $\mathcal{C}'$ . In this case, we can place  $s$  on this bisector point and  $B(s, \|s - C\|)$  will intersect all intervals in  $\mathcal{C}'$ , and moreover  $\|s - C\| \leq r_{opt}$  since  $\|s - C\|$  monotonically decreased as we moved  $s$  towards  $C$ . Modifying all centers in  $S$  in this way thus produces a solution whose radius is  $\leq r_{opt}$  and is either 0 or the distance from a bisector point to either interval in the pair it bisects. ◀

Given a set  $\mathcal{C}$  of  $n$  intervals, let  $P(\mathcal{C})$  denote the set of all  $2n$  left and right endpoints of the intervals in  $\mathcal{C}$ . To find the optimal solution to an instance  $\mathcal{C}, k$  of Problem 12, by Lemma 14, we can binary search over the interpoint distances of points in  $P(\mathcal{C})$  using our decider from Lemma 13. (When we call the decider we divide the interpoint distance by two as Lemma 14 actually tells us it is a bisector distance.) As there are  $\Theta(n^2)$  interpoint distances, naively this approach takes  $O(n^2 \log n)$  time. However, [15] previously showed that in the abstract setting where one is given a linear time decider, and the optimal solution is an interpoint distance, one can find the optimal solution in  $O(n \log n)$  time. This is achieved by reducing the problem to searching in an implicitly defined sorted matrix, which we describe in the full version [22]. Below is the summarizing theorem.

► **Theorem 15.** *Problem 12 can be solved in  $O(n \log n)$  time, where  $n = |\mathcal{C}|$ .*



## References

- 1 P. K. Agarwal, J. Matousek, and M. Sharir. On range searching with semialgebraic sets II. In *53rd Annual IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 420–429, 2012. doi:10.1109/FOCS.2012.32.
- 2 P. K. Agarwal, J. Pach, and M. Sharir. State of the union (of geometric objects). In J.E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, volume 453 of *Contemp. Math.*, pages 9–48. AMS, 2008.
- 3 P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.
- 4 H.-K. Ahn, S.-S. Kim, C. Knauer, L. Schlipf, C.-S. Shin, and A. Vigneron. Covering and piercing disks with two centers. *Comput. Geom.*, 46(3):253–262, 2013.
- 5 G. Anegg, H. Angelidakis, A. Kurpisz, and R. Zenklusen. A technique for obtaining true approximations for  $k$ -center with covering constraints. In *21st Integer Programming and Combinatorial Optimization (IPCO)*, volume 12125 of *LNCS*, pages 52–65. Springer, 2020. doi:10.1007/978-3-030-45771-6\_5.
- 6 M. Badoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 250–257. ACM, 2002. doi:10.1145/509907.509947.
- 7 S. Bandyapadhyay, T. Inamdar, S. Pai, and K. R. Varadarajan. A constant approximation for colorful  $k$ -center. In *27th Annual European Symposium on Algorithms (ESA)*, volume 144 of *LIPICs*, pages 12:1–12:14, 2019. doi:10.4230/LIPICs.ESA.2019.12.
- 8 M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *12th Annual Symposium on Discrete Algorithms (SODA)*, pages 642–651. ACM/SIAM, 2001. URL: <http://dl.acm.org/citation.cfm?id=365411.365555>.
- 9 O. Cheong, H. Everett, M. Glisse, J. Gudmundsson, S. Hornus, S. Lazard, M. Lee, and H.-S. Na. Farthest-polygon Voronoi diagrams. *Comput. Geom.*, 44(4):234–247, 2011.
- 10 A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003. doi:10.1016/S0196-6774(03)00047-6.
- 11 D. Eisenstat, P. N. Klein, and C. Mathieu. Approximating  $k$ -center in planar graphs. In *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 617–627. SIAM, 2014. doi:10.1137/1.9781611973402.47.
- 12 T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444. ACM, 1988. doi:10.1145/62212.62255.
- 13 S. P. Fekete, K. Huang, J. S. B. Mitchell, O. Parekh, and C. A. Phillips. Geometric hitting set for segments of few orientations. *Theory Comput. Syst.*, 62(2):268–303, 2018.
- 14 H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Comb.*, 10(1):41–51, 1990. doi:10.1007/BF02122694.
- 15 G. N. Frederickson. Parametric search and locating supply centers in trees. In *2nd Workshop on Algorithms and Data Structures (WADS)*, pages 299–319, 1991. doi:10.1007/BF0028271.
- 16 J. Gao, M. Langberg, and L. J. Schulman. Analysis of incomplete data and an intrinsic-dimension helly theorem. *Discrete and Computational Geometry*, 40(4):537–560, 2008. doi:10.1007/s00454-008-9107-5.
- 17 J. Gao, M. Langberg, and L. J. Schulman. Clustering lines in high-dimensional space: Classification of incomplete data. *ACM Trans. Algorithms*, 7(1):8:1–8:26, 2010. doi:10.1145/1868237.1868246.
- 18 M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977. doi:10.1137/0132071.
- 19 T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
- 20 D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the  $k$ -center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985. doi:10.1287/moor.10.2.180.

- 21 W. Hsu and G. L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979. doi:10.1016/0166-218X(79)90044-1.
- 22 H. Huang, G. Klimenko, and B. Raichel. Clustering with neighborhoods, September 2021. arXiv:2109.13302.
- 23 X. Jia, K. Sheth, and O. Svensson. Fair colorful  $k$ -center clustering. In *21st Integer Programming and Combinatorial Optimization (IPCO)*, volume 12125 of *LNCS*, pages 209–222. Springer, 2020. doi:10.1007/978-3-030-45771-6\_17.
- 24 M. I. Karavelas and M. Yvinec. The Voronoi diagram of planar convex objects. In *11th Annual European Symposium on Algorithms (ESA)*, volume 2832 of *LNCS*, pages 337–348. Springer, 2003. doi:10.1007/978-3-540-39658-1\_32.
- 25 R. Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *LNCS*. Springer, 1989. doi:10.1007/3-540-52055-4.
- 26 E. Lee and L. J. Schulman. Clustering affine subspaces: Hardness and algorithms. In *24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 810–827. SIAM, 2013. doi:10.1137/1.9781611973105.58.
- 27 Y. Marom and D. Feldman.  $k$ -means clustering of lines for big data. In *32nd Annual Advances in Neural Information Processing Systems (NeurIPS)*, pages 12797–12806, 2019. URL: <http://papers.nips.cc/paper/9442-k-means-clustering-of-lines-for-big-data>.
- 28 S. Micali and V. V. Vazirani. An  $O(\sqrt{|V||E|})$  algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.
- 29 N. H. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44(4):883–895, 2010. doi:10.1007/s00454-010-9285-9.
- 30 R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *29th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 475–484. ACM, 1997. doi:10.1145/258533.258641.
- 31 G. Xu and J. Xu. Efficient approximation algorithms for clustering point-sets. *Computational Geometry*, 43(1):59–66, 2010. doi:10.1016/j.comgeo.2007.12.002.





# Approximating Longest Spanning Tree with Neighborhoods

Ahmad Biniáz  

School of Computer Science, University of Windsor, Canada

---

## Abstract

We study the following maximization problem in the Euclidean plane: Given a collection of neighborhoods (polygonal regions) in the plane, the goal is to select a point in each neighborhood so that the longest spanning tree on selected points has maximum length. It is not known whether or not this problem is NP-hard. We present an approximation algorithm with ratio 0.548 for this problem. This improves the previous best known ratio of 0.511.

The presented algorithm takes linear time after computing a diameter. Even though our algorithm itself is fairly simple, its analysis is rather involved. In some part we deal with a minimization problem with multiple variables. We use a sequence of geometric transformations to reduce the number of variables and simplify the analysis.

**2012 ACM Subject Classification** Theory of computation → Computational geometry; Theory of computation → Approximation algorithms analysis

**Keywords and phrases** Euclidean maximum spanning tree, spanning tree with neighborhoods, approximation algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.7

**Funding** Supported by NSERC.

## 1 Introduction

The spanning tree is a well-studied and fundamental structure in graph theory and combinatorics. The well-known minimum spanning tree (Min-ST) problem asks for a spanning tree with minimum total edge-weight. In contrast, the maximum spanning tree (Max-ST) problem asks for a spanning tree with maximum total edge-weight. In the context of abstract graphs, the two problems are algorithmically equivalent in the sense that an algorithm that finds a Min-ST can also find a Max-ST within the same time bound (by simply negating the edge weights), and vice versa. The situation is quite different in the context of geometric graphs where vertices are points in the plane and edge-weights are Euclidean distances between points. In geometric graphs, an algorithm that exploits geometry for finding a Min-ST is not necessarily useful for finding a Max-ST because there is no known geometric transformation between the “nearest” and “farthest” relations among points [22]. The existing geometric algorithms, for solving these two problems, exploit different sets of techniques.

Problems related to maximum configurations in the plane (also known as *long configurations*) have received considerable attention after the seminal work of Alon, Rajagopalan, and Suri [2]. In this paper we study the *longest spanning tree with neighborhoods* (Max-ST-NB) problem. We are given a collection of  $n$  neighborhoods (polygonal regions) in the Euclidean plane and we want to find a maximum-length tree that connects  $n$  representative points, one point from each neighborhood, as in Figure 1(a). The *length* of the tree is the total Euclidean length of its edges. Each *neighborhood* is the union of simple polygons, and the neighborhoods are not necessarily disjoint. The neighborhoods are assumed to be *colored* by  $n$  different colors. The classical Euclidean Max-ST problem is a special case of the Max-ST-NB in which each neighborhood consists of exactly one point, as in Figure 1(b). Although the Euclidean Max-ST problem can be solved in  $O(n \log n)$  time [22], it is not known whether or



© Ahmad Biniáz;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 7; pp. 7:1–7:11

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

not the Max-ST-NB problem can be solved in polynomial time. The difficulty lies in choosing representative points from neighborhoods; once these points are selected, the problem is reduced to the Euclidean Max-ST problem.



■ **Figure 1** (a) Longest spanning tree with four polygonal neighborhoods that are colored red, green, blue, and purple. (b) Euclidean maximum spanning tree.

It is easily seen (see Section 2) that the longest star (which connects a point in one neighborhood to a point in each of the other neighborhoods) achieves a 0.5-approximate solution for the Max-ST-NB problem. Recently, Chen and Dumitrescu [10] present an approximation algorithm with ratio 0.511, which is the first improvement beyond 0.5.

Although any optimal solution for the Max-ST problem contains a diametral pair (two points with maximum distance) as an edge, an optimal solution for the Max-ST-NB problem does not necessarily contain any bichromatic diametral pair (two points with maximum distance that belong to different neighborhoods). Another result of Chen and Dumitrescu [10] shows that any algorithm, that always includes a bichromatic diametral pair in the solution, cannot achieve an approximation ratio better than  $\sqrt{2} - \sqrt{3} \approx 0.517$ . This somehow breaks the hope of getting a good approximation ratio by using greedy techniques. Thus, to improve the ratio beyond 0.517 one needs to employ some nontrivial ideas.

## 1.1 Our contributions and approach

We present an approximation algorithm for the Max-ST-NB problem with improved ratio  $\frac{\sqrt{7}-1}{3} \approx 0.548$ . Our algorithm is not complicated: We compute a double-star (a tree of diameter 3) centered at a bichromatic diametral pair, and compute up to three stars (trees of diameter 2) centered at points on the smallest enclosing circle, and then report the longest one. Our algorithm takes linear time after computing a bichromatic diameter. Our analysis involves a minimization problem with multiple variables. We employ a sequence of geometric transformations to reduce the number of variables and simplify the analysis. The following theorem summarizes our main contribution.

► **Theorem 1.** *A 0.548-approximation for the longest spanning tree with neighborhoods can be computed in linear time after computing a bichromatic diameter.*

As a minor result we improve the upper bound 0.517 on the approximation ratio of algorithms that always include a bichromatic diameter in their solutions. We show that the ratio of such algorithms cannot be better than 0.5. This upper bound is tight because there exists a 0.5-approximation algorithm that always includes a bichromatic diameter (see Section 2). Therefore, to obtain a ratio of better than 0.5 one should take into account also spanning trees that do not contain any bichromatic diameter.

## 1.2 Related problems and applications

The Max-ST-NB problem has the same flavor as the Euclidean group Steiner tree problem in which we are given  $n$  groups of points in the plane and the goal is to find a shortest tree that contains “at least” one point from each group. The group Steiner tree problem in graphs is NP-hard and cannot be approximated by a factor  $O(\log^{2-\epsilon} n)$  for any  $\epsilon > 0$  [16]. The Max-ST-NB also lies in the concept of *imprecision* in computational geometry where each input point is provided as a region of uncertainty and the exact position of the point may be anywhere in the region; see e.g. [13, 18]. Analogous problems have been studied for other structures, e.g., minimum spanning tree with neighborhoods [8, 13, 25], traveling salesman tour with neighborhoods [3, 20, 21] (which is APX-hard [12]), and convex hulls [18, 23], to name a few. We refer the interested readers to the thesis of Löffler [17].

The maximum spanning tree and related problems, in addition to their fundamental nature, find applications in worst-case analysis of various heuristics in combinatorial optimization [2], and in approximating maximum triangulations [5, pp. 338]. They also appear in clustering algorithms where one needs to partition a set of entities into well-separated and homogeneous clusters [4, 22]. Maximum spanning trees are directly related to computing diameter and farthest neighbors which are fundamental problems in computational geometry, with many applications [1].

## 2 Preliminaries for the algorithm

The output of our algorithm is either a star or a double-star. A *star*, centered at a vertex  $p$ , is a tree in which every edge is incident to  $p$ . A *double-star*, centered at two vertices  $p$  and  $q$ , is a tree that contains the edge  $pq$  and its every other edge is incident to either  $p$  or  $q$ .

Let  $P$  be a set of points in the Euclidean plane. The *smallest enclosing disk* for  $P$  is the smallest disk that contains all the points of  $P$ . A *diametral pair* of  $P$  is a pair of points in  $P$  that attain the maximum distance. If the points in  $P$  are colored, then a *bichromatic diametral pair* of  $P$  is defined as a pair of points in  $P$  with different colors that attain the maximum distance. The *center of mass* of  $P$  (also known as the *centroid*) is a point  $m$  in the plane such that for any arbitrary point  $u$  in the plane we have

$$\sum_{p \in P} \vec{up} = |P| \cdot \vec{um}. \quad (1)$$

The intersection of two disks is called a *lens*. We denote the Euclidean distance between two points  $p$  and  $q$  in the plane by  $|pq|$ . In our context, a *geometric graph* is a graph whose vertices are points in the plane and whose edges are straight line segments. The length of a geometric graph  $G$ , denoted by  $\text{len}(G)$ , is the total Euclidean length of its edges.

### A simple 0.5-approximation algorithm

Chen and Dumitrescu [10] pointed out the following simple 0.5-approximation algorithm for the Max-ST-NB problem (a similar approach was previously used in [2] and [14]). Take a bichromatic diametral pair  $(a, b)$  from the given  $n$  neighborhoods;  $a$  and  $b$  belong to two different neighborhoods. Choose an arbitrary point from each of the other  $n-2$  neighborhoods. Let  $S_a$  be the star obtained by connecting  $a$  to  $b$  and to all chosen points. Define  $S_b$  analogously on the same point set. Every edge of any optimal solution  $T^*$  has length at most  $|ab|$ , and thus  $\text{len}(T^*) \leq (n-1)|ab|$ . By the triangle inequality  $\text{len}(S_a) + \text{len}(S_b) \geq n|ab| \geq \text{len}(T^*)$ . Therefore the longer of  $S_a$  and  $S_b$  is a 0.5-approximate solution for the problem.

### 3 The approximation algorithm

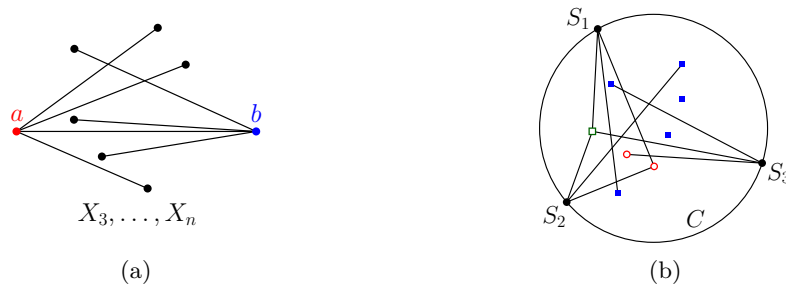
In this section we prove Theorem 1. Put  $\delta = \frac{\sqrt{7}-1}{3} \approx 0.548$ .

To facilitate comparisons we use the same notation as of Chen and Dumitrescu [10]. Let  $\mathcal{X} = \{X_1, X_2, \dots, X_n\}$  be the given collection of  $n$  polygonal neighborhoods of total  $N$  vertices. We assume that each  $X_i$  is colored by a unique color. Our algorithm selects representative points only from boundary vertices of polygonal neighborhoods. Thus, in the algorithm (but not in the analysis) we consider each polygonal neighborhood  $X_i$  as the set of its boundary vertices, and consequently we consider  $\mathcal{X}$  as a collection of  $N$  points colored by  $n$  colors. Define the *longest spanning star* centered at a point  $p \in X_i$  as the star connecting  $p$  to its farthest point in every other neighborhood.

#### The algorithm

The main idea of the algorithm is simple: we compute a spanning double-star  $D$  and (at most) three spanning stars  $S_1, S_2, S_3$ , and then report the longest one.

We compute  $D$  as follows. Let  $(a, b)$  be a bichromatic diametral pair of  $\mathcal{X}$ . After a suitable relabeling assume that  $a \in X_1$  and  $b \in X_2$ . Add the edge  $ab$  to  $D$ . For each  $X_i$ , with  $i \in \{3, \dots, n\}$ , find a vertex  $p_i \in X_i$  that is farthest from  $a$  and find a vertex  $q_i \in X_i$  that is farthest from  $b$  (it might be the case that  $p_i = q_i$ ). If  $|ap_i| \geq |bq_i|$  then add  $ap_i$  to  $D$  otherwise add  $bq_i$  to  $D$ . Observe that  $D$  spans all neighborhoods in  $\mathcal{X}$ , and each edge of  $D$  has length at least  $|ab|/2$ , as in Figure 2(a). Now we introduce the stars. Let  $C$  be the smallest enclosing disk for  $\mathcal{X}$ . Notice that the boundary of  $C$  contains at least two points of  $\mathcal{X}$ . If it contains exactly two points then we define  $S_1$  and  $S_2$  as the two longest spanning stars that are centered at these points (in this case we do not have  $S_3$ ). If it contains three or more points then there exist three of them such that the triangle formed by those points contains the center of  $C$  [11, Chapter 4, Section 4.7]. In this case we define  $S_1, S_2$ , and  $S_3$  as the three longest spanning stars that are centered at those three points, as in Figure 2(b).



■ **Figure 2** Illustration of the algorithm: (a) the double-star  $D$ , and (b) the stars  $S_1, S_2$ , and  $S_3$ .

#### Running time analysis

The smallest enclosing disk  $C$  for  $\mathcal{X}$  can be computed in  $O(N)$  time [9, 19, 24]. The result of [7], that computes a maximum spanning tree on multicolored points, implies that a bichromatic diametral pair  $(a, b)$  for  $\mathcal{X}$  can be found in  $O(N \log N \log n)$  time (the algorithm of Bhattacharya and Toussaint [6] also computes a bichromatic diameter, but only for two-colored points). The rest of our algorithm (finding farthest points from  $a, b$ , and from the points on the boundary of  $C$ ) takes  $O(N)$  time.

### 3.1 Analysis of the approximation ratio

Our main plan for analysis works as follows: we show that if the radius of  $C$  is at least  $\delta$  then one of the stars  $S_i$  is a desired tree, otherwise the double-star  $D$  is a desired tree.

For the analysis we consider  $\mathcal{X}$  as the initial collection of polygonal neighborhoods. Let  $T^*$  denote a longest spanning tree with neighborhoods in  $\mathcal{X}$ . It is not hard to see that for any point in the plane, its farthest point in a polygon  $P$  must be a vertex of  $P$  (see also [11, Chapter 7, Section 7.4]). Thus, any bichromatic diameter of  $\mathcal{X}$  is introduced by two vertices of polygons in  $\mathcal{X}$ . Hence the pair  $(a, b)$ , selected in the algorithm, is a bichromatic diameter of the initial collection  $\mathcal{X}$ . Therefore,  $|ab|$  is an upper bound for the length of edges in  $T^*$ . After a suitable scaling assume that  $|ab| = 1$ . Since  $T^*$  has  $n - 1$  edges,

$$\text{len}(T^*) \leq (n - 1)|ab| = n - 1. \tag{2}$$

Recall the smallest enclosing disk  $C$  from the algorithm. Let  $c$  denote the center of  $C$  and  $r$  denote its radius. If the boundary of  $C$  has exactly two points of  $\mathcal{X}$  then denote them by  $c_1$  and  $c_2$ . In this case the segment  $c_1c_2$  is a diameter of  $C$ ; see [11, Chapter 4, Section 4.7]. If the boundary of  $C$  has three or more points of  $\mathcal{X}$  then denote the three points (that are chosen in the algorithm) by  $c_1, c_2$ , and  $c_3$ . In this case the triangle  $c_1c_2c_3$  is acute and it contains the center  $c$ ; see [11, Chapter 4, Section 4.7]. Recall the longest spanning stars  $S_1, S_2, S_3$  from the algorithm. After a suitable relabeling assume that the star  $S_i$  is centered at the point  $c_i$ .

► **Lemma 2.** *If  $r \geq \delta$  and the boundary of  $C$  contains exactly two points of  $\mathcal{X}$  then*

$$\max\{\text{len}(S_1), \text{len}(S_2)\} \geq \delta \cdot \text{len}(T^*).$$

**Proof.** In this case  $c_1c_2$  is a diameter of  $C$ , and thus  $|c_1c_2| = 2r \geq 2\delta$ . We consider two cases depending on similarity of colors of  $c_1$  and  $c_2$ .

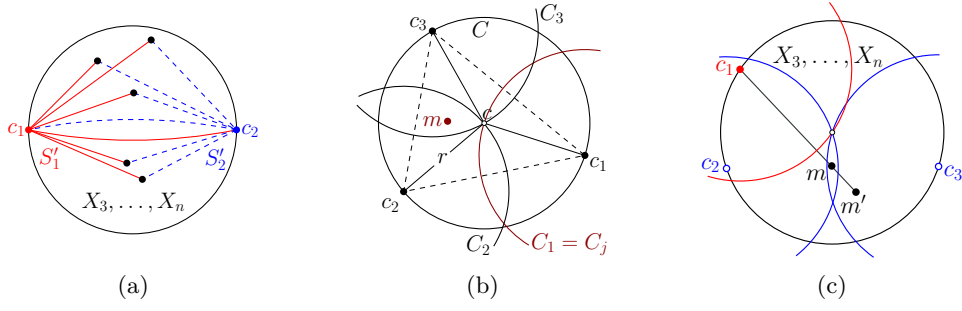
- The points  $c_1$  and  $c_2$  have different colors. This case is depicted in Figure 3(a). After a suitable relabeling assume that  $c_1 \in X_1$  and  $c_2 \in X_2$ . Pick an arbitrary point  $p_i$  from each  $X_i$  with  $i \in \{3, \dots, n\}$ . Let  $S'_1$  be the spanning star with center  $c_1$  that connects  $c_1$  to all  $p_i$ s and to  $c_2$ . Let  $S'_2$  be the spanning star with center  $c_2$  that connects  $c_2$  to all  $p_i$ s and to  $c_1$ . Since  $S_1$  and  $S_2$  are longest spanning stars centered at  $c_1$  and  $c_2$ , it holds that  $\text{len}(S'_1) \leq \text{len}(S_1)$  and  $\text{len}(S'_2) \leq \text{len}(S_2)$ . By bounding the maximum with the average, then using the triangle inequality and (2) we get:

$$\begin{aligned} \max\{\text{len}(S_1), \text{len}(S_2)\} &\geq \max\{\text{len}(S'_1), \text{len}(S'_2)\} \geq \frac{1}{2}(\text{len}(S'_1) + \text{len}(S'_2)) \\ &= \frac{1}{2} \left( |c_1c_2| + |c_1c_2| + \sum_{i=3}^n (|c_1p_i| + |p_ic_2|) \right) \\ &\geq \frac{1}{2} \left( 2|c_1c_2| + \sum_{i=3}^n |c_1c_2| \right) = \frac{|c_1c_2|}{2} \cdot n \geq \delta \cdot n \geq \delta \cdot \text{len}(T^*). \end{aligned}$$

- The points  $c_1$  and  $c_2$  have the same color. Assume that  $c_1, c_2 \in X_1$ . Pick an arbitrary point  $p_i$  from each  $X_i$  with  $i \in \{2, \dots, n\}$ . Let  $S'_1$  be the spanning star that connects  $c_1$  to all  $p_i$ s. Let  $S'_2$  be the spanning star that connects  $c_2$  to all  $p_i$ s. Notice that  $\text{len}(S'_1) \leq \text{len}(S_1)$  and  $\text{len}(S'_2) \leq \text{len}(S_2)$ . Similar to the previous case we have:

$$\max\{\text{len}(S_1), \text{len}(S_2)\} \geq \frac{1}{2} \sum_{i=2}^n (|c_1p_i| + |p_ic_2|) \geq \frac{|c_1c_2|}{2} \cdot (n - 1) \geq \delta \cdot \text{len}(T^*). \quad \blacktriangleleft$$

7:6 Approximating Longest Spanning Tree with Neighborhoods



■ **Figure 3** Illustration of the proofs of (a) Lemma 2, (b) Lemma 3, and (c) Lemma 4.

► **Lemma 3.** *If the boundary of  $C$  contains three or more points of  $\mathcal{X}$  then for any point  $m$  in the plane there exists a point  $c_j \in \{c_1, c_2, c_3\}$  such that  $|c_j m| \geq r$ .*

**Proof.** Let  $C_i$  be the disk of radius  $r$  centered at each  $c_i$ . The boundary of each  $C_i$  passes through the center  $c$  of  $C$ , as in Figure 3(b). Since the triangle  $c_1 c_2 c_3$  contains  $c$  it holds that  $C_1 \cap C_2 \cap C_3 = c$ . Therefore there exists a disk  $C_j$  that does not have  $m$  in its interior, and thus  $|c_j m| \geq r$ . ◀

► **Lemma 4.** *If  $r \geq \delta$  and the boundary of  $C$  contains three or more points of  $\mathcal{X}$  then*

$$\max\{\text{len}(S_1), \text{len}(S_2), \text{len}(S_3)\} \geq \delta \cdot \text{len}(T^*).$$

**Proof.** In this case the triangle  $c_1 c_2 c_3$  contains the center  $c$  of  $C$ . We consider three cases depending on similarity of colors of  $c_1$ ,  $c_2$ , and  $c_3$ .

- The points  $c_1, c_2, c_3$  have pairwise distinct colors. Thus, they belong to three different neighborhoods. After a suitable relabeling assume that  $c_1 \in X_1, c_2 \in X_2$ , and  $c_3 \in X_3$ . Pick an arbitrary point from each  $X_i$  with  $i \in \{4, \dots, n\}$ . Denote the selected points by  $P$ . Let  $m$  be the center of mass of  $P$ . By Lemma 3 there exists a point  $c_j \in \{c_1, c_2, c_3\}$  where  $|c_j m| \geq r$ . After a suitable relabeling assume that  $c_j = c_1$ , and thus  $|c_1 m| \geq r$ . By (1) we get

$$\sum_{p \in P} |c_1 p| \geq |P| \cdot |c_1 m| \geq (n-3) \cdot r \geq (n-3) \cdot \delta.$$

Let  $S'_1$  be the star that connects  $c_1$  to all points of  $P$  and to  $c_2$  and  $c_3$ . Since  $S_1$  is the longest spanning star centered at  $c_1$ , we have that  $\text{len}(S_1) \geq \text{len}(S'_1)$ . Since the triangle  $c_1 c_2 c_3$  contains  $c$ , we have  $|c_2 c_1| + |c_3 c_1| \geq |c_2 c| + |c_3 c| = 2r \geq 2\delta$ . These inequalities and (2) give

$$\text{len}(S_1) \geq \text{len}(S'_1) = |c_1 c_2| + |c_1 c_3| + \sum_{p \in P} |c_1 p| \geq 2\delta + (n-3) \cdot \delta = (n-1) \cdot \delta \geq \delta \cdot \text{len}(T^*).$$

- All points  $c_1, c_2, c_3$  have the same color. Assume that  $c_1, c_2, c_3 \in X_1$ . Pick an arbitrary point from each  $X_i$  with  $i \in \{2, \dots, n\}$ . Denote the selected points by  $P$ . Let  $m$  be the centroid of  $P$ , and let  $c_1$  be the point in  $\{c_1, c_2, c_3\}$  for which  $|c_1 m| \geq r$  (by Lemma 3). Let  $S'_1$  be the star that connects  $c_1$  to all points of  $P$ . Similar to the previous case by using (1) we get

$$\text{len}(S_1) \geq \text{len}(S'_1) \geq \sum_{p \in P} |c_1 p| \geq |P| \cdot |c_1 m| \geq (n-1) \cdot r \geq (n-1) \cdot \delta \geq \delta \cdot \text{len}(T^*).$$

- Only two of  $c_1, c_2, c_3$  have the same color. This case is depicted in Figure 3(c). Assume that  $c_2$  and  $c_3$  have the same color and they belong to  $X_2$ . Also assume that  $c_1 \in X_1$ . We handle this case in a slightly different way; this is because if the point  $c_j$  from Lemma 3 (which would have distance at least  $r$  to the centroid) belongs to  $\{c_2, c_3\}$  then there is no guarantee that  $|c_j c_1| \geq \delta$ , and hence we may not be able to establish the lower bound  $\delta \cdot (n - 1)$ .

Pick an arbitrary point from each  $X_i$  with  $i \in \{3, \dots, n\}$ . Let  $P$  be the set containing all selected points together with the point  $c_1$ . Let  $m$  be the centroid of  $P$ . Consider the point  $c_j$  (from Lemma 3) for which  $|c_j m| \geq r$ . If  $c_j = c_2$  then let  $S'_2$  be the star that connects  $c_2$  to all points of  $P$ . In this case

$$\text{len}(S_2) \geq \text{len}(S'_2) = \sum_{p \in P} |c_2 p| \geq |P| \cdot |c_2 m| \geq (n - 1) \cdot r \geq (n - 1) \cdot \delta \geq \delta \cdot \text{len}(T^*).$$

If  $c_j = c_3$  then by a similar argument we can show that  $\text{len}(S_3) \geq \delta \cdot \text{len}(T^*)$ .

Now assume that  $c_j = c_1$ , and thus  $|c_1 m| \geq r$ . Let  $P' = P \setminus \{c_1\}$ , and let  $m'$  be the centroid of  $P'$ . Using the recursive definition of centroid [15] (based on the Euclidean rule of the lever) the point  $m$  lies on the segment  $c_1 m'$ , as in Figure 3(c). Informally speaking, if we remove  $c_1$  from  $P$  then its (new) centroid moves away from  $c_1$ . Therefore  $|c_1 m'| \geq |c_1 m| \geq r$ . Let  $S'_1$  be the star obtained by connecting  $c_1$  to all points of  $P'$  and to the one of  $c_2$  and  $c_3$  that is farther from  $c_1$ . Assume that  $c_2$  is the farther one, and notice that  $|c_1 c_2| \geq r$ . Then,

$$\begin{aligned} \text{len}(S_1) &\geq \text{len}(S'_1) = |c_1 c_2| + \sum_{p \in P'} |c_1 p| \geq r + |P'| \cdot |c_1 m'| \\ &\geq r + (n - 2) \cdot r \geq \delta \cdot \text{len}(T^*). \end{aligned} \quad \blacktriangleleft$$

Lemmas 2 and 4 take care of our analysis for the case where the radius  $r$  of  $C$  is at least  $\delta$ . The next lemma takes care of the case where  $r \leq \delta$  by showing that in this case the double-star  $D$  is a desired tree. We employ a collection of geometric transformations to simplify the proof.

► **Lemma 5.** *If  $r \leq \delta$  then  $\text{len}(D) \geq \delta \cdot \text{len}(T^*)$ .*

**Proof.** Recall  $(a, b)$  as a bichromatic diametral pair of  $\mathcal{X}$ . Also recall our assumptions that  $a \in X_1$ ,  $b \in X_2$ , and that  $|ab| = 1$ .

One challenge that we face here is that the vertices of our double-star  $D$  could be different from the vertices of the optimal tree  $T^*$ ; this could make it difficult to obtain a lower bound for the length of  $D$  in terms of the length of  $T^*$ . But we know that the vertices of both  $D$  and  $T^*$  come from the same ground sets  $X_1, \dots, X_n$ . Our plan is to compare the length of  $D$  with the length of  $T^*$  by comparing the lengths of their edges separately. For each  $i \in \{1, \dots, n\}$  let  $p_i^*$  and  $p_i$  be the vertices of  $T^*$  and  $D$  that belong to  $X_i$ , respectively (it might be that  $p_i^* = p_i$ ). Notice that  $a = p_1$  and  $b = p_2$ . Direct all edges of  $T^*$  towards  $p_1^*$  and direct all edges of  $D$  towards  $p_1$ . To each vertex of  $T^*$  and  $D$  (except  $p_1^*$  and  $p_1$ ) assign its unique outgoing edge. For each  $i \in \{2, \dots, n\}$  let  $\text{len}(p_i^*)$  and  $\text{len}(p_i)$  be the length of edges that are assigned to  $p_i^*$  and  $p_i$ , respectively. We already know that  $\text{len}(p_2) = |ab| = 1$  and  $\text{len}(p_i^*) \leq |ab| = 1$  for all  $i$ . Thus, in order to show that  $\text{len}(D) \geq \delta \cdot \text{len}(T^*)$  it suffices to show that  $\text{len}(p_i) \geq \delta \cdot \text{len}(p_i^*)$  for each  $i \in \{3, \dots, n\}$ . From the optimization point of view, we are interested in the minimum value of the ratio

$$\frac{\text{len}(p_i)}{\text{len}(p_i^*)} \tag{3}$$

over all pairs  $(p_i, p_i^*)$ . In particular we want this value to be at least  $\delta$ .

## 7:8 Approximating Longest Spanning Tree with Neighborhoods

From now on we consider a fixed value of  $i \in \{3, \dots, n\}$ . For brevity we write  $p$  for  $p_i$ ,  $p^*$  for  $p_i^*$ , and  $X$  for  $X_i$ . In the rest of this section we will show that  $\text{len}(p) \geq \delta \cdot \text{len}(p^*)$ . Recall from the algorithm that  $p$  is connected to the farther of  $a$  and  $b$ , and thus  $\text{len}(p) = \max\{|pa|, |pb|\}$ . Let  $D(a, \delta)$  and  $D(b, \delta)$  be the disks of radii  $\delta$  that are centered at  $a$  and  $b$ , respectively. If  $p$  is outside  $D(a, \delta)$  then  $\text{len}(p) \geq |pa| \geq \delta \geq \delta \cdot \text{len}(p^*)$ . Likewise, if  $p$  is outside  $D(b, \delta)$  then  $\text{len}(p) \geq \delta \cdot \text{len}(p^*)$ , and we are done.

In the rest of this section we assume that  $p$  is in the lens  $L = D(a, \delta) \cap D(b, \delta)$  which is depicted in Figure 4(a). In the current setting, the neighborhood  $X$  (which contains  $p$ ) lies entirely in  $L$  because otherwise our algorithm would have picked a point of  $X$  outside  $L$ . Therefore, the point  $p^*$  (which also belongs to  $X$ ) lies in  $L$ . Moreover  $\max\{|ap|, |bp|\} \geq \max\{|ap^*|, |bp^*|\}$  because otherwise our algorithm would have picked  $p^*$  instead of  $p$ . Thus, in view of (3), it suffices to show that

$$\frac{\max\{|ap^*|, |bp^*|\}}{\text{len}(p^*)} \geq \delta. \quad (4)$$

For any point  $q$  in disk  $C$  let  $q_C$  be the intersection point of the boundary of  $C$  with the ray emanating from  $q$  and passing through the center  $c$ ; Figure 4(a) depicts this for point  $q = p^*$ . The point  $q_C$  is the farthest point of  $C$  from  $q$ . Thus the largest possible length of the edge of  $T^*$  that is assigned to  $p^*$  is  $|p^*p_C^*|$ , that is,  $\text{len}(p^*) \leq |p^*p_C^*|$ . Thus, in view of (4), it suffices to show that

$$\frac{\max\{|ap^*|, |bp^*|\}}{|p^*p_C^*|} \geq \delta. \quad (5)$$

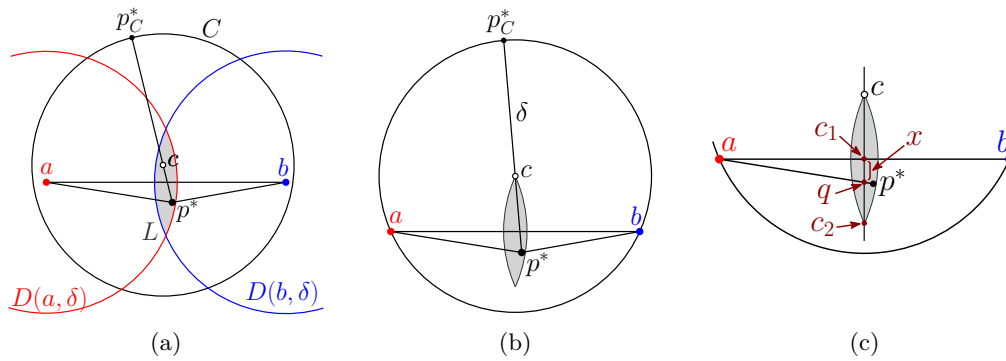
Inequality (5) deals with a minimization problem which has multiple variables, including the coordinates of  $a$ ,  $b$ ,  $p^*$ , and  $c$ . We use a sequence of geometric transformations to reduce the number of variables and simplify the analysis. Our transformations will not increase the ratio in (5).

If we increase the radius of  $C$  (while fixing its center  $c$ ) then  $|p^*p_C^*|$  would increase but  $|ap^*|$  and  $|bp^*|$  remain unchanged. Thus, for the purpose of (5) we can assume that  $C$  has maximum possible radius which is  $\delta$ . Then, for any point  $q \in C$  it holds that  $|qq_C| = |qc| + |cq_C| = |qc| + \delta$ .

Let  $\ell(a, b)$  be the line through  $a$  and  $b$ . If  $p^*$  lies in the same side of  $\ell(a, b)$  as  $c$  does, then let  $\bar{p}^*$  be the reflection of  $p^*$  with respect to  $\ell(a, b)$ . Notice that  $\bar{p}^*$  also lies in lens  $L$ . Moreover  $|\bar{a}\bar{p}^*| = |ap^*|$  and  $|\bar{b}\bar{p}^*| = |bp^*|$ , but  $|p^*p_C^*| \leq |\bar{p}^*\bar{p}_C^*|$ . Thus, in view of (5), the point  $\bar{p}^*$  achieves a smaller ratio than  $p^*$ . Therefore we can assume that  $p^*$  lies in a different side of  $\ell(a, b)$  than  $c$  does.

Let  $\mathcal{L}$  denote the configuration that is the union of the lens  $L$ , the segment  $ab$ , and the point  $p^*$ . Notice that any translation, rotation, and reflection of  $\mathcal{L}$  will not change  $|ap^*|$  and  $|bp^*|$ . Move  $\mathcal{L}$  along the ray, that is emanating from  $c$  and passing through  $p^*$ , and stop as soon as one of  $a$  and  $b$  lies on the boundary of  $C$ . Assume that  $b$  is the point that lies on  $C$ . This translation can only increase  $|p^*p_C^*|$ , but not decrease. Now fix  $\mathcal{L}$  at  $b$  and rotate it in the direction, that moves  $p^*$  away from  $c$ , until  $a$  also lies on the boundary of  $C$ . The lens  $L$  is small enough and does not intersect the boundary of  $C$  after rotation. This rotation can only increase  $|p^*p_C^*|$ , but not decrease. (Such a rotation moves  $p_C^*$  on the boundary of  $C$ , but that does not affect the argument because the value  $|cp_C^*|$ , which is equal to the radius of  $C$ , remains unchanged.) Therefore, above transformations do not increase the ratio in (5). After these transformations assume, without loss of generality, that  $ab$  is horizontal,  $a$  is to the left of  $b$ , and  $c$  lies above  $ab$ . The current setting is depicted in Figure 4(b). Notice that





■ **Figure 4** Illustration of (a) the lens  $L$  and the point  $p_C^*$  associated with  $p^*$ , (b) the configuration  $\mathcal{L}$  after translation and rotation, (c) the points  $q, c_1, c_2$ .

$|p^*p_C^*| = |cp^*| + |cp_C^*| = |cp^*| + \delta$ . Due to symmetry we may assume that  $p^*$  lies to the right side of the vertical line through  $c$ , and thus  $|ap^*| \geq |bp^*|$ , as in Figure 4(c). In view of (5), in the current setting our goal is to show that

$$\frac{|ap^*|}{|cp^*| + \delta} \geq \delta. \tag{6}$$

Let  $q$  be the intersection point of  $ap^*$  with the vertical line through  $c$ , as in Figure 4(c). Then  $|ap^*| = |aq| + |qp^*|$  and  $|cp^*| \leq |cq| + |qp^*|$ . Thus,

$$\frac{|ap^*|}{|cp^*| + \delta} \geq \frac{|aq| + |qp^*|}{|cq| + |qp^*| + \delta} \geq \frac{|aq|}{|cq| + \delta},$$

where the second inequality is valid because we subtract the same amount  $|qp^*|$  from the numerator and denominator of a fraction which is smaller than 1 (notice that  $|aq| < |cq| + \delta$ ). Thus, for showing (6) it suffices to show that

$$\frac{|aq|}{|cq| + \delta} \geq \delta. \tag{7}$$

Recall the definition of  $L$ , and notice that its topmost point lies on the center  $c$ . Let  $c_1$  be the intersection point of  $ab$  with the vertical line through  $c$ , and let  $c_2$  be the lowest point of  $L$ ; see Figure 4(c). Then  $|cc_1| = |c_1c_2|$ ,  $|ac| = |ac_2| = \delta$ , and  $|ac_1| = 1/2$ . Notice that  $q$  lies on the segment  $c_1c_2$ , and  $|cq| = |cc_1| + |qc_1|$ . Denote the length  $|qc_1|$  by  $x$ . Then  $0 \leq x \leq |c_1c_2|$ . Using the Pythagorean theorem we get  $|aq| = \sqrt{x^2 + 1/4}$  and  $|c_1c_2| = \sqrt{\delta^2 - 1/4}$ . Thus we can write the ratio in (7) as a function  $f$  which depends only on  $x$ :

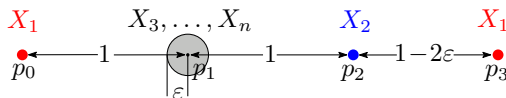
$$f(x) = \frac{|aq|}{|cq| + \delta} = \frac{\sqrt{x^2 + 1/4}}{x + \delta + \sqrt{\delta^2 - 1/4}},$$

where  $x \in [0, \sqrt{\delta^2 - 1/4}]$ . The function  $f(x)$  is decreasing on this interval of  $x$  and thus its minimum value is attained at  $\sqrt{\delta^2 - 1/4}$ . Plugging this into  $f$  we get  $f(\sqrt{\delta^2 - 1/4}) = \frac{\sqrt{7}-1}{3} = \delta$ . This verifies (7) and finishes the proof of the lemma. ◀

The cases considered in Lemmas 2, 4, and 5 ensure that the length of one of  $S_1, S_2, S_3$ , and  $D$  is at least  $\delta \cdot \text{len}(T^*)$ . This concludes our analysis and finishes the proof of Theorem 1.

### 3.2 Inclusion of bichromatic diameter

Here we show that the approximation ratio of an algorithm, that always includes a bichromatic diametral pair in its solution, cannot be larger than 0.5.



■ **Figure 5** Illustration of the upper bound 0.5 for inclusion of a bichromatic diametral pair.

We introduce an input instance with  $n$  neighborhoods. Consider four points  $p_0 = (0, 0)$ ,  $p_1 = (1, 0)$ ,  $p_2 = (2, 0)$ , and  $p_3 = (3 - 2\varepsilon, 0)$  for arbitrary small  $\varepsilon > 0$ , e.g.  $\varepsilon = 1/n$ . Our input consists of neighborhoods  $X_1, \dots, X_n$  where  $X_1 = \{p_0, p_3\}$ ,  $X_2 = \{p_2\}$ , and each of  $X_3, \dots, X_n$  has exactly one point that is placed at distance at most  $\varepsilon$  from  $p_1$ ; see Figure 5. In this setting,  $(p_0, p_2)$  is the unique bichromatic diametral pair. Consider any tree  $T$  that contains the bichromatic diameter  $p_0p_2$  (this means that  $p_3$  is not in  $T$ ). Any edge of  $T$  incident to  $X_3, \dots, X_n$  has length at most  $1 + \varepsilon$ . Therefore  $\text{len}(T) \leq 2 + (1 + \varepsilon)(n - 2) < n + 1$ . Now consider the tree  $T^*$  that does not contain  $p_0p_2$  but connects each of  $X_2, \dots, X_n$  to  $p_3$ . The length of  $T^*$  is at least  $(1 - 2\varepsilon) + (2 - 3\varepsilon)(n - 2) > 2n - 6$ . Then, the ratio

$$\frac{\text{len}(T)}{\text{len}(T^*)} < \frac{n + 1}{2n - 6}$$

tends to  $1/2$  in the limit. This establishes the upper bound 0.5 on the approximation ratio.

## 4 Conclusions

A natural open problem is to further improve the approximation ratio for the Max-ST-NB problem. We believe that our algorithm has better approximation guarantee, however this requires more detailed analysis. We obtained the ratio of 0.548 by analyzing the stars  $S_1, S_2, S_3$  and the double-star  $D$  separately. One might be able to improve the ratio by analyzing the stars and the double-star together and then taking the longest one.

---

### References

- 1 Pankaj K. Agarwal, Jiri Matousek, and Subhash Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom.*, 1:189–201, 1991.
- 2 Noga Alon, Sridhar Rajagopalan, and Subhash Suri. Long non-crossing configurations in the plane. *Fundam. Inform.*, 22(4):385–394, 1995. Also in *SoCG* 1993.
- 3 Esther M. Arkin and Refael Hassin. Approximation algorithms for the geometric covering salesman problem. *Discret. Appl. Math.*, 55(3):197–218, 1994.
- 4 Tetsuo Asano, Binay K. Bhattacharya, J. Mark Keil, and F. Frances Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the 4th Annual Symposium on Computational Geometry (SoCG)*, pages 252–257, 1988.
- 5 Marshall Bern and David Eppstein. Approximation algorithms for geometric problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 296–345. PWS Publishing, 1996.
- 6 Binay K. Bhattacharya and Godfried T. Toussaint. Efficient algorithms for computing the maximum distance between two finite planar sets. *J. Algorithms*, 4(2):121–136, 1983.
- 7 Ahmad Biniiaz, Prosenjit Bose, David Eppstein, Anil Maheshwari, Pat Morin, and Michiel Smid. Spanning trees in multipartite geometric graphs. *Algorithmica*, 80(11):3177–3191, 2018.

- 8 Víctor Blanco, Elena Fernández, and Justo Puerto. Minimum spanning trees with neighborhoods: Mathematical programming formulations and solution methods. *Eur. J. Oper. Res.*, 262(3):863–878, 2017.
- 9 Bernard Chazelle and Jirí Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21(3):579–597, 1996.
- 10 Ke Chen and Adrian Dumitrescu. On the longest spanning tree with neighborhoods. *Discrete Mathematics, Algorithms and Applications*, 12(5), 2020. Also in FAW’18.
- 11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.
- 12 Mark de Berg, Joachim Gudmundsson, Matthew J. Katz, Christos Levkopoulos, Mark H. Overmars, and A. Frank van der Stappen. TSP with neighborhoods of varying size. *J. Algorithms*, 57(1):22–36, 2005. Also in *ESA* 2002.
- 13 Reza Dorigiv, Robert Fraser, Meng He, Shahin Kamali, Akitoshi Kawamura, Alejandro López-Ortiz, and Diego Seco. On minimum- and maximum-weight minimum spanning trees with neighborhoods. *Theory Comput. Syst.*, 56(1):220–250, 2015.
- 14 Adrian Dumitrescu and Csaba D. Tóth. Long non-crossing configurations in the plane. *Discret. Comput. Geom.*, 44(4):727–752, 2010. Also in *STACS* 2010.
- 15 G. A. Galperin. A concept of the mass center of a system of material points in the constant curvature spaces. *Communications in Mathematical Physics*, 154(1):63–84, 1993.
- 16 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 585–594, 2003.
- 17 Maarten Löffler. *Data Imprecision in Computational Geometry*. Phd thesis, Utrecht University, 2009.
- 18 Maarten Löffler and Marc J. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.
- 19 Nimrod Megiddo. Linear-time algorithms for linear programming in  $\mathbb{R}^3$  and related problems. *SIAM J. Comput.*, 12(4):759–776, 1983. Also in *FOCS 1982*.
- 20 Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 11–18, 2007.
- 21 Joseph S. B. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *Proceedings of the 26th ACM Symposium on Computational Geometry (SoCG)*, pages 183–191, 2010.
- 22 Clyde L. Monma, Mike Paterson, Subhash Suri, and F. Frances Yao. Computing Euclidean maximum spanning trees. *Algorithmica*, 5(3):407–419, 1990.
- 23 Marc J. van Kreveld and Maarten Löffler. Approximating largest convex hulls for imprecise points. *J. Discrete Algorithms*, 6(4):583–594, 2008.
- 24 Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *Maurer H. (eds) New Results and New Trends in Computer Science*, pages 359–370. LNCS 555, 1991.
- 25 Yang Yang, Mingen Lin, Jinhui Xu, and Yulai Xie. Minimum spanning tree with neighborhoods. In *Proceedings of the 3rd International Conference Algorithmic Aspects in Information and Management (AAIM)*, pages 306–316, 2007.



# Self-Improving Voronoi Construction for a Hidden Mixture of Product Distributions

Siu-Wing Cheng ✉

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

Man Ting Wong ✉

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

---

## Abstract

We propose a self-improving algorithm for computing Voronoi diagrams under a given convex distance function with constant description complexity. The  $n$  input points are drawn from a hidden mixture of product distributions; we are only given an upper bound  $m = o(\sqrt{n})$  on the number of distributions in the mixture, and the property that for each distribution, an input instance is drawn from it with a probability of  $\Omega(1/n)$ . For any  $\varepsilon \in (0, 1)$ , after spending  $O(mn \log^{O(1)}(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$  time in a training phase, our algorithm achieves an  $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} n 2^{O(\log^* n)} + \frac{1}{\varepsilon} H)$  expected running time with probability at least  $1 - O(1/n)$ , where  $H$  is the entropy of the distribution of the Voronoi diagram output. The expectation is taken over the input distribution and the randomized decisions of the algorithm. For the Euclidean metric, the expected running time improves to  $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} H)$ .

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** entropy, Voronoi diagram, convex distance function, hidden mixture of product distributions

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.8

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.13460>

**Funding** Research of Cheng and Wong are supported by Research Grants Council, Hong Kong, China (project no. 16200317).

## 1 Introduction

Self-improving algorithms, proposed by Ailon et al. [1], is a framework for studying algorithmic complexity beyond the worst case. There is a *training phase* that allows some auxiliary structures about the input distribution to be constructed. In the *operation phase*, these auxiliary structures help to achieve an expected running time, called the *limiting complexity*, that may surpass the worst-case optimal time complexity.

Self-improving algorithms have been designed for *product distributions* [1, 11]. Let  $n$  be the input size. A product distribution  $\mathcal{D} = (D_1, \dots, D_n)$  consists of  $n$  distributions  $D_i$  such that the  $i$ th input item is drawn independently from  $D_i$ . It is possible that  $D_i = D_j$  for some  $i \neq j$ , but the draws of the  $i$ th and  $j$ th input items are independent. No further information about  $\mathcal{D}$  is given. Sorting, Delaunay triangulation, 2D maxima, and 2D convex hull have been studied for product distributions. For all four problems, the training phase uses  $O(n^\varepsilon)$  input instances, and the space complexity is  $O(n^{1+\varepsilon})$ . The limiting complexities of sorting and Delaunay triangulation are  $O(\frac{1}{\varepsilon} n + \frac{1}{\varepsilon} H_{\text{out}})$  for any  $\varepsilon \in (0, 1)$ , where  $H_{\text{out}}$  is the entropy of the output distribution [1]. The limiting complexities for 2D maxima and 2D convex hull are  $O(\text{OptM} + n)$  and  $O(\text{OptC} + n \log \log n)$  respectively, where  $\text{OptM}$  and  $\text{OptC}$  are the expected depths of the optimal linear decision trees for the two problems [11].

Extensions that allow dependence among input items have been developed. One extension is that there is a *hidden partition* of  $[n]$  into groups. The input items with indices in the  $k$ th group follow some *hidden functions* of a common parameter  $u_k$ . The parameters  $u_1, u_2, \dots$  follow a product distribution. The partition of  $[n]$  is not given though. If the hidden functions



© Siu-Wing Cheng and Man Ting Wong;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 8; pp. 8:1–8:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

are known to be linear, sorting can be solved in a limiting complexity of  $O(\frac{1}{\varepsilon}n + \frac{1}{\varepsilon}H_{\text{out}})$  after a training phase that takes  $O(n^2 \log^3 n)$  time [8]. If it is only known that each hidden function has  $O(1)$  extrema and the graphs of two functions intersect in  $O(1)$  places (without knowing any of the functions, or any of these extrema and intersections), sorting can be solved in a limiting complexity of  $O(n + H_{\text{out}})$  after an  $\tilde{O}(n^3)$ -time training phase [7]. For the Delaunay triangulation problem, if it is known that the hidden functions are bivariate polynomials of  $O(1)$  degree (without knowing the polynomials), a limiting complexity of  $O(n\alpha(n) + H_{\text{out}})$  can be achieved after a polynomial-time training phase [7].

Another extension is that the input instance  $I$  is drawn from a *hidden mixture* of at most  $m$  product distributions. That is, there are at most  $m$  product distributions  $\mathcal{D}_1, \mathcal{D}_2, \dots$  such that  $\Pr[I \sim \mathcal{D}_a] = \lambda_a$  for some fixed positive value  $\lambda_a$ . The upper bound  $m$  is given, but no information about the  $\lambda_a$ 's and the  $\mathcal{D}_a$ 's is provided. Sorting can be solved in a limiting complexity of  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H_{\text{out}})$  after a training phase that takes  $O(mn \log^2(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$  time [8].

In this paper, we present a self-improving algorithm for constructing Voronoi diagrams under a convex distance function  $d_Q$  in  $\mathbb{R}^2$ , assuming that the input distribution is a hidden mixture of at most  $m$  product distributions. The convex distance function  $d_Q$  is induced by a given convex polygon  $Q$  of  $O(1)$  size. The upper bound  $m$  is given, and we assume that  $m = o(\sqrt{n})$ . We also assume that for each product distribution  $\mathcal{D}_a$  in the mixture,  $\lambda_a = \Omega(1/n)$ . Let  $\varepsilon \in (0, 1)$  be a parameter fixed beforehand. The training phase uses  $O(mn \log(mn))$  input instances and takes  $O(mn \log^{O(1)}(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$  time. In the operation phase, given an input instance  $I$ , we can construct its Voronoi diagram  $\text{Vor}_Q(I)$  under  $d_Q$  in a limiting complexity of  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}n 2^{O(\log^* n)} + \frac{1}{\varepsilon}H)$ , where  $H$  denotes the entropy of the distribution of the Voronoi diagram output. Note that  $\Omega(H)$  is a lower bound of the expected running time of any comparison-based algorithm. Our algorithm also works for the Euclidean case, and the limiting complexity improves to  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H)$ .

For simplicity, we will assume throughout the rest of this paper that the hidden mixture has exactly  $m$  product distributions. We give an overview of our method in the following.

We follow the strategy in [1] for computing a Euclidean Delaunay triangulation. The idea is to form a set  $S$  of sample points and build  $\text{Del}(S)$  and some auxiliary structures in the training phase so that any future input instance  $I$  can be merged quickly into  $\text{Del}(S)$  to form  $\text{Del}(S \cup I)$ , and then  $\text{Del}(I)$  can be split off in  $O(n)$  expected time. Merging  $I$  into  $\text{Del}(S)$  requires locating the input points in  $\text{Del}(S)$ . The location distribution is gathered in the training phase so that distribution-sensitive point location can be used to avoid the logarithmic query time as much as possible. Modifying  $\text{Del}(S)$  efficiently into  $\text{Del}(S \cup I)$  requires that only  $O(1)$  points in  $I$  fall into the same neighborhood in  $\text{Del}(S)$  in expectation.

In our case, since there are  $m$  product distributions, we will need a larger set  $S$  of  $mn$  sample points in order to ensure that only  $O(1)$  points in  $I$  fall into the same neighborhood in  $\text{Vor}_Q(S)$  in expectation. But then merging  $I$  into  $\text{Vor}_Q(S)$  in the operation phase would be too slow because scanning  $\text{Vor}_Q(S)$  already requires  $\Theta(mn)$  time. We need to extract a subset  $R \subseteq S$  such that  $R$  has  $O(n)$  size and  $R$  contains all points in  $S$  whose Voronoi cells conflict with the input points.

Still, we cannot afford to construct  $\text{Vor}_Q(R)$  in  $O(n \log n)$  time. In the training phase, we form a metric  $d$  related to  $d_Q$  and construct a *net-tree*  $T_S$  for  $S$  under  $d$  [15]. In the operation phase, after finding the appropriate  $R \subseteq S$ , we use nearest common ancestor queries [20] to compress  $T_S$  in  $O(n \log \log m)$  time to a subtree  $T_R$  for  $R$  that has  $O(n)$  size. Next, we use  $T_R$  to construct a well-separated pair decomposition of  $R$  under  $d$  in  $O(n)$  time [15], use the decomposition to compute the nearest neighbor graph of  $R$  under  $d$  in  $O(n)$  time, and then

construct  $\text{Vor}_Q(R)$  from the nearest neighbor graph in  $O(n)$  expected time. The merging of  $I$  into  $\text{Vor}_Q(R)$  to form  $\text{Vor}_Q(R \cup I)$ , and the splitting of  $\text{Vor}_Q(R \cup I)$  into  $\text{Vor}_Q(I)$  and  $\text{Vor}_Q(R)$  are obtained by transferring their analogous results in the Euclidean case [1, 6].

We have left out the expected time to locate the input points in  $\text{Vor}_Q(S)$ . It is bounded by  $O(1/\varepsilon)$  times the sum of the entropies of the point location outcomes. We show that  $\text{Vor}_Q(I)$  allows us to locate the input points in  $\text{Vor}_Q(S)$  in  $O(n \log m + n2^{O(\log^* n)})$  time. Then, a result in [1] implies that the sum of the entropies of the point location outcomes is  $O(n \log m + n2^{O(\log^* n)} + H)$ . The expected running time is thus  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}n2^{O(\log^* n)} + \frac{1}{\varepsilon}H)$ , which dominates the limiting complexity. In the Euclidean case,  $\text{Vor}(I)$  allows us to locate the input points in  $O(n \log m)$  time, so the limiting complexity improves to  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H)$ .

Details and proofs that are omitted due to space constraint can be found in the full version of this paper [9].

## 2 Preliminaries

Let  $Q$  be a convex polygon that has  $O(1)$  complexity and contains the origin in its interior. Let  $\partial$  and  $\text{int}(\cdot)$  be the boundary and interior operators, respectively. So  $Q$ 's boundary is  $\partial Q$  and its interior is  $\text{int}(Q)$ . Let  $d_Q$  be the distance function induced by  $Q$ :  $\forall x, y \in \mathbb{R}^2$ ,  $d_Q(x, y) = \min\{\lambda \in [0, \infty) : y \in \lambda Q + x\}$ . As  $Q$  may not be centrally symmetric (i.e.,  $x \in Q \iff -x \in Q$ ),  $d_Q$  may not be a metric.

The bisector of two points  $p$  and  $q$  is  $\{x \in \mathbb{R}^2 : d_Q(p, x) = d_Q(q, x)\}$ , which is an open polygonal curve of  $O(1)$  size. The Voronoi diagram of a set  $\Sigma$  of  $n$  points,  $\text{Vor}_Q(\Sigma)$ , is a partition of  $\mathbb{R}^2$  into interior-disjoint cells  $V_p(\Sigma) = \{x \in \mathbb{R}^2 : \forall q \in \Sigma, d_Q(p, x) \leq d_Q(q, x)\}$  for all  $p \in \Sigma$ . There are algorithms for constructing  $\text{Vor}_Q(\Sigma)$  in  $O(n \log n)$  time [10, 18].

$V_p(\Sigma)$  is simply connected and star-shaped with respect to  $p$  [10]. We use  $N_p(\Sigma)$  to denote the set of Voronoi neighbors of  $p$  in  $\text{Vor}_Q(\Sigma)$ . The Voronoi edges of  $\text{Vor}_Q(\Sigma)$  form a planar graph of  $O(|\Sigma|)$  size. Each Voronoi edge is a polygonal line, and we call its internal vertices *Voronoi edge bends*. We use  $V_\Sigma$  to denote the set of Voronoi edge bends and Voronoi vertices in  $\text{Vor}_Q(\Sigma)$ . For the infinite Voronoi edges, their endpoints at infinity are included in  $V_\Sigma$ .

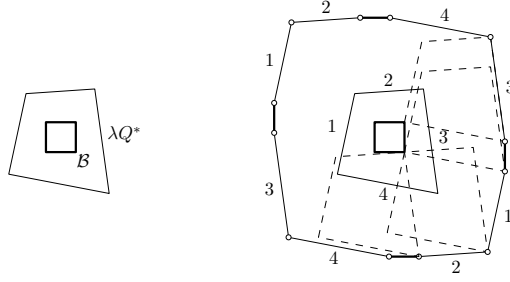
Define  $Q^* = \{-x : x \in Q\}$ . For any points  $x, y \in \mathbb{R}^2$ ,  $d_{Q^*}(x, y) = d_Q(y, x)$ . At any point  $x$  on a Voronoi edge of  $\text{Vor}_Q(\Sigma)$  defined by  $p, q \in \Sigma$ , there exists  $\lambda \in (0, \infty)$  such that  $d_{Q^*}(x, p) = d_Q(p, x) = d_Q(q, x) = d_{Q^*}(x, q) = \lambda$  and  $d_{Q^*}(x, s) = d_Q(s, x) \geq \lambda$  for all  $s \in \Sigma$ . Hence,  $\{p, q\} \subset \partial(\lambda Q^* + x)$  and  $\text{int}(\lambda Q^* + x) \cap \Sigma = \emptyset$ , i.e., an ‘‘empty circle property’’.

Take a point  $x$ . Consider the largest homothetic<sup>1</sup> copy  $Q_x^*$  of  $Q^*$  centered at  $x$  such that  $\text{int}(Q_x^*) \cap \Sigma = \emptyset$ . If we insert a new point  $q$  to  $\Sigma$ , we say that  $q$  *conflicts with*  $x$  if  $q \in Q_x^*$ . We say that  $q$  *conflicts with* a cell  $V_p(\Sigma)$  if  $q$  conflicts with some point in  $V_p(\Sigma)$ . Clearly,  $V_p(\Sigma)$  must be updated by the insertion of  $q$ . We use  $V_\Sigma|_q$  to denote the subset of  $V_\Sigma$  that conflict with  $q$ . The Voronoi edge bends and Voronoi vertices in  $V_\Sigma|_q$  will be destroyed by the insertion of  $q$ .

We make three general position assumptions. First, no two sides of  $Q$  are parallel. Second, for every pair of input points, their support line is not parallel to any side of  $Q$ . Third, no four input points lie on the boundary of any homothetic copy of  $Q^*$ , which implies that every Voronoi vertex has degree three.

It is much more convenient if all Voronoi cells of the input points are bounded. We assume that all possible input points appear in some fixed bounding square  $\mathcal{B}$  centered at the origin. We place  $O(1)$  dummy points outside  $\mathcal{B}$  so that all Voronoi cells of the input

<sup>1</sup> A homothetic copy of a shape is a scaled and translated copy of it.



■ **Figure 1** The left image shows the bounding square  $\mathcal{B}$  and the large enclosing  $\lambda Q^*$ . In the right image, we slide a copy of  $\lambda Q^*$  around  $\mathcal{B}$  to generate the outer convex polygon. The dashed polygon demonstrates the sliding of  $\lambda Q^*$  around  $\mathcal{B}$ . The bold edges on this convex polygon are translates of the boundary edges of  $\mathcal{B}$ . Every edge of  $\lambda Q^*$  has two translational copies too as labelled.

points are bounded, and their portions inside  $\mathcal{B}$  remain the same as before. Refer to Figure 1. Take  $\lambda Q^*$  for some large enough  $\lambda \in \mathbb{R}$  such that for every point  $x \in \mathcal{B}$ ,  $\lambda Q^* + x$  contains  $\mathcal{B}$ . Refer to the left image in Figure 1. We slide a copy of  $\lambda Q^*$  around  $\mathcal{B}$  to generate the outer convex polygon. The dashed polygon demonstrates the sliding of  $\lambda Q^*$  around  $\mathcal{B}$ . This outer polygon contains a translational copy of every edge of  $\mathcal{B}$  and two translational copies of every edge of  $\lambda Q^*$ . We add the vertices of this outer polygon as dummy points. Any homothetic copy of  $Q^*$  that intersects  $\mathcal{B}$  cannot be expanded indefinitely without containing some of these dummy points. So all Voronoi cells of input points are bounded. For each point  $x \in \mathcal{B}$ , since the dummy points lie outside  $\lambda Q^* + x$  and  $\mathcal{B} \subseteq \lambda Q^* + x$  (i.e.,  $\lambda Q^* + x$  is not empty of the input points), the portion of the Voronoi diagram inside  $\mathcal{B}$  is unaffected by the dummy points.

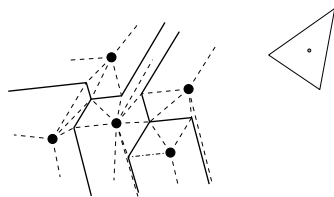
### 3 Training phase

**Sample set  $\mathcal{S}$ .** Take  $mn \ln(mn)$  instances  $I_1, I_2, \dots, I_{mn \ln(mn)}$ . Define  $x_1, \dots, x_{mn \ln(mn)}$  by taking the  $p_1$ 's in  $I_1, \dots, I_{m \ln(mn)}$  to be  $x_1, \dots, x_{m \ln(mn)}$ ,  $p_2$ 's in  $I_{m \ln(mn)+1}, \dots, I_{2m \ln(mn)}$  to be  $x_{m \ln(mn)+1}, \dots, x_{2m \ln(mn)}$ , and so on. The set  $\mathcal{S}$  of sample points includes a  $\frac{1}{mn}$ -net of the  $x_i$ 's with respect to the family of homothetic copies of  $Q^*$ , as well as the  $O(1)$  dummy points. The set  $\mathcal{S}$  has  $O(mn)$  points and can be constructed in  $O(mn \log^{O(1)}(mn))$  time as homothetic copies of  $Q^*$  are pseudo-disks [1, 19].

**Point location.** Compute  $\text{Vor}_Q(\mathcal{S})$  and triangulate it by connecting each  $p \in \mathcal{S}$  to  $V_S \cap \partial V_p(S)$ , i.e., the Voronoi edge bends and Voronoi vertices in  $\partial V_p(S)$ . For unbounded Voronoi cells, we view the infinite Voronoi edges as leading to some vertices at infinity; an extra triangulation edge that goes between two infinite Voronoi edges also leads to a vertex at infinity, giving rise to unbounded triangles. Figure 2 shows an example.

Construct a point location structure  $L_S$  for the triangulated  $\text{Vor}_Q(\mathcal{S})$  with  $O(\log(mn))$  query time [13]. Take another  $m^\epsilon n^\epsilon$  input instances and use  $L_S$  to locate the points in these input instances in the triangulated  $\text{Vor}_Q(\mathcal{S})$ . For every  $i \in [n]$  and every triangle  $t$ , we compute  $\tilde{\pi}_{i,t}$  to be the ratio of the frequency of  $t$  hit by  $p_i$  to  $m^\epsilon n^\epsilon$ , which is an estimate of  $\Pr[p_i \in t]$ . For each  $i \in [n]$ , form a subdivision  $\mathcal{S}_i$  that consists of triangles with positive  $\tilde{\pi}_{i,t}$ 's, triangulate the exterior of  $\mathcal{S}_i$ , and give these new triangles a zero estimated probability. Set the weight of each triangle in  $\mathcal{S}_i$  to be the maximum of  $(mn)^{-\epsilon}$  and its estimated probability. Construct a distribution-sensitive point location structure  $L_i$  for  $\mathcal{S}_i$  based on the triangle weights [2, 16]. Note that  $L_i$  has  $O(m^\epsilon n^\epsilon)$  size, and locating a point in a triangle  $t \in \mathcal{S}_i$  takes  $O(\log \frac{W_i}{w_t})$  time, where  $w_t$  is the weight of  $t$  and  $W_i$  is the total weight in  $\mathcal{S}_i$ .





■ **Figure 2** Part of the triangulation of a Voronoi diagram induced by the triangle shown with a gray center. The solid edges form the Voronoi diagram. The dashed edges refine it into a triangulation.

For any input instance  $(p_1, \dots, p_n)$  in the operation phase, we will query  $L_i$  to locate  $p_i$  in the triangulated  $\text{Vor}_Q(S)$ , which may fail if  $p_i$  falls into a triangle with zero estimated probability. If the search fails, we query  $L_S$  to locate  $p_i$ .

**Net-tree.** We first define a metric that is induced by a centrally symmetric convex polygon. Define  $\hat{Q} = \{x - y : x, y \in Q^*\}$ , i.e., the Minkowski sum of  $Q^*$  and  $-Q^*$ , or equivalently  $Q^*$  and  $Q$ . It is centrally symmetric by definition. It can be visualized as the region covered by all possible placements of  $Q^*$  that has the origin in the polygon boundary. Since  $\hat{Q}$  is a Minkowski sum, its number of vertices is within a constant factor of the total number of vertices of  $Q^*$  and  $-Q^*$ , which is  $O(1)$ .

Let  $d$  be the metric induced by the centrally symmetric convex polygon  $\hat{Q}$ , which is a *doubling metric* – there is a constant  $\lambda > 0$  such that for any point  $x \in \mathbb{R}^2$  and any positive number  $r$ , the ball with respect to  $d$  centered at  $x$  with radius  $r$  can be covered by  $\lambda$  balls with respect to  $d$  of radius  $r/2$ .

Given a set of points  $P$ , a *net-tree* for  $P$  with respect to  $d$  [15] is an analog of the well-separated pair decomposition for Euclidean spaces [4]. It is a rooted tree whose leaves are the points in  $P$ . For each node  $v$ , let  $\text{parent}(v)$  denote its parent, and let  $P_v$  denote the subset of  $P$  at the leaves that descend from  $v$ . Every tree node  $v$  is given a representative point  $p_v$  and an integer level  $\ell_v$ . Let  $\tau \geq 11$  be a fixed constant. Let  $B(x, h)$  denote the ball  $\{y \in \mathbb{R}^2 : d(x, y) \leq h\}$ . By the results in [15] (Definition 2.1 and the remark that follows Proposition 2.2), the following properties are satisfied by a net-tree:

- (a)  $p_v \in P_v$ .
- (b) For every non-root node  $v$ ,  $\ell_v < \ell_{\text{parent}(v)}$ , and if  $v$  is a leaf, then  $\ell_v = -\infty$ .
- (c) Every internal node has at least two and at most a constant number of children.
- (d) For every node  $v$ ,  $B(p_v, \frac{2\tau}{\tau-1} \cdot \tau^{\ell_v})$  contains  $P_v$ .
- (e) For every non-root node  $v$ ,  $B(p_v, \frac{\tau-5}{2\tau-2} \cdot \tau^{\ell_{\text{parent}(v)}-1}) \cap P \subset P_v$ .
- (f) For every internal node  $v$ , there is a child  $w$  of  $v$  such that  $p_w = p_v$ .

**Clusters.** We construct a *net-tree*  $T_S$  for  $S$  in  $O(mn \log(mn))$  expected time [15]. We define *clusters* as follows. Label all leaves of  $T_S$  as unclustered initially. Select the leftmost  $m$  unclustered leaves of  $T_S$ ; if there are fewer than  $m$  such leaves, select them all. Find the subtree rooted at a node  $v$  of  $T_S$  that contains the selected unclustered leaves, but no child subtree of  $v$  contains them all. We call the subtree rooted at  $v$  a cluster and label all its leaves clustered. Then, we repeat the above until all leaves of  $T_S$  are clustered. By construction, the clusters are disjoint, each cluster has  $O(m)$  nodes, and there are  $O(n)$  clusters in  $T_S$ .

We assign nodes in each cluster a unique cluster index in the range  $[1, O(n)]$ . We also assign each node of a cluster three indices from the range  $[1, O(m)]$  according to its rank in the preorder, inorder, and postorder traversals of that cluster. The preorder and postorder indices allow us to tell in  $O(1)$  time whether two nodes are an ancestor-descendant pair.

We keep an initially empty van Emde Boas tree  $EB_c$  [21] with each cluster  $c$ . The universe for  $EB_c$  is the set of leaves in the cluster  $c$ , and the in-order of these leaves in  $c$  is the total order for  $EB_c$ . We also build a nearest common ancestor query data structure for each cluster [20]. The nearest common ancestor query of any two nodes can be reported in  $O(\log \log m)$  time.

**Planar separator.**  $\text{Vor}_Q(S)$  is a planar graph of  $O(mn)$  size with all Voronoi edge bends and Voronoi vertices as graph vertices. By a recursive application of the planar separator theorem, one can produce an  $m^2$ -division of  $\text{Vor}_Q(S)$ : it is divided into  $O(n/m)$  regions, each region contains  $O(m^2)$  vertices, and the boundary of each region contains  $O(m)$  vertices [14].

Extract the subset  $B \subset S$  of points whose Voronoi cell boundaries contain some region boundary vertices in the  $m^2$ -division. So  $|B| = O(m \cdot n/m) = O(n)$ . Compute  $\text{Vor}_Q(B)$  and triangulate it as in triangulating  $\text{Vor}_Q(S)$ . By our choice of  $B$ , the region boundaries in the  $m^2$ -division of  $\text{Vor}_Q(S)$  form a subgraph of  $\text{Vor}_Q(B)$ . Label in  $O(n)$  time the Voronoi edge bends and Voronoi vertices in  $\text{Vor}_Q(B)$  whether they exist in  $\text{Vor}_Q(S)$ .

We construct point location data structures for every region  $\Pi$  in the  $m^2$ -division as follows. For every boundary vertex  $w$  of  $\Pi$ , let  $Q_w^*$  be the largest homothetic copy of  $Q^*$  centered at  $w$  such that  $\text{int}(Q_w^*) \cap B = \emptyset$ . These  $Q_w^*$ 's form an arrangement of  $O(m^2)$  complexity, and we construct a point location data structure that allows a point to be located in this arrangement in  $O(\log m)$  time. We also construct a point location data structure for the portion of the triangulated  $\text{Vor}_Q(S)$  inside  $\Pi$ . Since the region has  $O(m^2)$  complexity, this point location data structure can return in  $O(\log m)$  time the triangle in the triangulated  $\text{Vor}_Q(S)$  that contains a point inside  $\Pi$ .

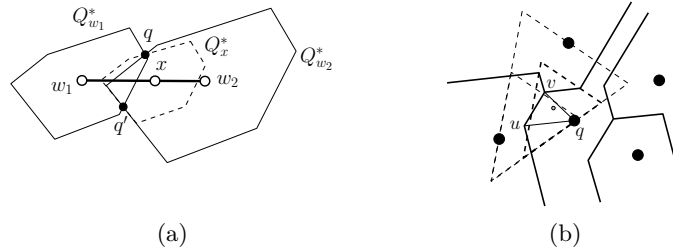
**Output and performance.** The following result summarizes the output and performance of the training phase. The proof of Lemma 1(a) is similar to an analogous result for sorting in [8].

► **Lemma 1.** *Let  $\mathcal{D}_a$ ,  $a \in [m]$ , be the distributions in the hidden mixture. The training phase computes the following structures in  $O(mn \log^{O(1)}(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$  time.*

- (i) *A set  $S$  of  $O(mn)$  points and  $\text{Vor}_Q(S)$ . It holds with probability at least  $1 - 1/n$  that for any  $a \in [1, m]$  and any  $v \in V_S$ ,  $\sum_{i=1}^n \Pr[X_{iv} | I \sim \mathcal{D}_a] = O(1/m)$ , where  $X_{iv} = 1$  if  $p_i \in I$  conflicts with  $v$  and  $X_{iv} = 0$  otherwise.*
- (ii) *Point location structures  $L_S$  and  $L_i$  for each  $i \in [n]$  that allow us to locate  $p_i$  in the triangulated  $\text{Vor}_Q(S)$  in  $O(\frac{1}{\varepsilon} H(t_i))$  expected time, where  $t_i$  is the random variable that represents the point location outcome, and  $H(t_i)$  is the entropy of the distribution of  $t_i$ .*
- (iii) *A net-tree  $T_S$  for  $S$ , the  $O(n)$  clusters in  $T_S$ , the initially empty van Emde Boas trees for the clusters, and the nearest common ancestor data structures for the clusters.*
- (iv) *An  $m^2$ -division of  $\text{Vor}_Q(S)$ , the subset  $B \subseteq S$  of  $O(n)$  points whose Voronoi cell boundaries contain some region boundary vertices in the  $m^2$ -division,  $\text{Vor}_Q(B)$ , and the point location data structures for the regions in the  $m^2$ -division.*

Lemma 1(a) leads to Lemma 2 below, which implies that for any  $v \in V_S$ , if we feed the input points that conflict with  $v$  to a procedure that runs in quadratic time in the worst case, the expected running time of this procedure over all points in  $V_S$  is  $O(n)$ . The proof of Lemma 2 is just an algebraic manipulation of the probabilities.

► **Lemma 2.** *For every  $v \in V_S$ , let  $Z_v$  be the subset of input points that conflict with  $v$ . It holds with probability at least  $1 - O(1/n)$  that  $\sum_{v \in V_S} \mathbb{E}[|Z_v|^2] = O(n)$ .*



■ **Figure 3** (a) The points  $q$  and  $q'$  define a Voronoi edge, and  $w_1$  and  $w_2$  are two adjacent Voronoi edge bends or Voronoi vertices on this edge. At any point  $x$  between  $w_1$  and  $w_2$ , the polygon  $Q_x^*$  (shown dashed) is a subset of  $Q_{w_1}^* \cup Q_{w_2}^*$ . (b) A triangle  $quv$  in the triangulated Voronoi diagram in Figure 2 is shown. If a point  $p$  conflicts with the white dot (i.e., lies inside the bold dashed triangle), then  $p$  conflicts with  $u$  or  $v$  (i.e., lies inside one of the two light dashed circles).

We state two technical results. Figure 3(a) and (b) illustrate these two lemmas.

► **Lemma 3.** Consider  $\text{Vor}_Q(Y)$  for some point set  $Y$ . For any point  $x \in \mathbb{R}^2$ , let  $Q_x^*$  be the largest homothetic copy of  $Q^*$  centered at  $x$  such that  $\text{int}(Q_x^*) \cap Y = \emptyset$ . Let  $w_1$  and  $w_2$  be two adjacent Voronoi edge bends or Voronoi vertices in  $\text{Vor}_Q(Y)$ . For any point  $x \in w_1 w_2$ ,  $Q_x^* \subseteq Q_{w_1}^* \cup Q_{w_2}^*$ . The same property holds if  $w_1$  and  $w_2$  are Voronoi vertices connected by a Voronoi edge, and  $x$  lies on that Voronoi edge.

► **Lemma 4.** Let  $q$  be a point in some point set  $Y$ . Let  $quv$  be a triangle in the triangulated  $\text{Vor}_Q(Y)$ . If a point  $p \notin Y$  conflicts with a point in  $quv$ , then  $p$  conflicts with  $u$  or  $v$ . Hence, if  $p$  conflicts with  $V_q(Y)$ ,  $p$  conflicts with a Voronoi edge bend or Voronoi vertex in  $\partial V_q(Y)$ .

## 4 Operation phase

Given an instance  $I = (p_1, \dots, p_n)$ , we construct  $\text{Vor}_Q(I)$  using the pseudocode below.

### ■ Algorithm 1 OPERATION PHASE.

1. For each  $i \in [n]$ , query  $L_i$  to find the triangle  $t_i$  in the triangulated  $\text{Vor}_Q(S)$  that contains  $p_i$ , and if the search fails, query  $L_S$  to find  $t_i$ .
2. For each  $i \in [n]$ , search  $\text{Vor}_Q(S)$  from  $t_i$  to find  $V_S|_{p_i}$ , i.e., the subset of  $V_S$  that conflict with  $p_i$ . This also gives the subset of  $S$  whose Voronoi cells conflict with the input points. Let  $R$  be the union of this subset of  $S$  and the set of representative points of all cluster roots in  $T_S$ .
3. Compute the compression  $T_R$  of  $T_S$  to  $R$ .
4. Construct the nearest neighbor graph  $1\text{-NN}_R$  under the metric  $d$  from  $T_R$ .
5. Compute  $\text{Vor}_Q(R)$  from  $1\text{-NN}_R$ .
6. Modify  $\text{Vor}_Q(R)$  to produce  $\text{Vor}_Q(R \cup I)$ .
7. Split  $\text{Vor}_Q(R \cup I)$  to produce  $\text{Vor}_Q(I)$  and  $\text{Vor}_Q(R)$ . Return  $\text{Vor}_Q(I)$ .

We analyze step 1 in Section 4.1, steps 2 and 3 in Section 4.2, steps 4 and 5 in Section 4.3, and steps 6 and 7 in Section 4.4. Step 1 is the most time-consuming; all other steps run in  $O(n)$  expected time.

#### 4.1 Point location

By Lemma 1(b), step 1 runs in  $O(\sum_{i=1}^n \frac{1}{\varepsilon} H(t_i))$  expected time, which is  $O(\frac{1}{\varepsilon} n \log m + \frac{1}{\varepsilon} H(t_1, \dots, t_n))$  as we will show later. By Lemma 5 below, if there is an algorithm that can use  $\text{Vor}_Q(I)$  to determine  $t_1, \dots, t_n$  in  $c(n)$  expected time, then  $H(t_1, \dots, t_n) = O(c(n) + H)$ , implying that step 1 takes  $O(\frac{1}{\varepsilon}(n \log m + c(n) + H))$  expected time. Any preprocessing cost of  $S$  is excluded from  $c(n)$ . We present such an algorithm.

► **Lemma 5** (Lemma 2.3 in [1]). *Let  $\mathcal{D}$  be a distribution on a universe  $\mathcal{U}$ . Let  $X : \mathcal{U} \rightarrow \mathcal{X}$ , and let  $Y : \mathcal{U} \rightarrow \mathcal{Y}$  be two random variables. Suppose that there is a comparison-based algorithm that computes a function  $f : (I, X(I)) \rightarrow Y(I)$  in  $C$  expected comparisons over  $\mathcal{D}$  for every  $I \in \mathcal{U}$ . Then  $H(Y) = C + O(H(X))$ .*

Recall that we have computed in the training phase the subset  $B \subseteq S$  whose Voronoi cell boundaries contain some region boundary vertices in the  $m^2$ -division of  $\text{Vor}_Q(S)$ . Note that  $|B| = O(n)$ . We have also computed  $\text{Vor}_Q(B)$  and point location data structures associated with the regions in the  $m^2$ -division. We use  $\text{Vor}_Q(B)$  and these point location data structures determines  $t_1, \dots, t_n$  as follows.

- Task 1: Merge  $\text{Vor}_Q(B)$  with  $\text{Vor}_Q(I)$  to form the triangulated  $\text{Vor}_Q(B \cup I)$ .
- Task 2: Use  $\text{Vor}_Q(S)$ ,  $\text{Vor}_Q(B)$ , and  $\text{Vor}_Q(B \cup I)$  to find the triangles  $t_1, \dots, t_n$ .

We discuss these two tasks in the following.

**Task 1.** For every point  $p \in B$ , define a polygonal cone surface  $C_p = \{(a, b, d_Q(p, (a, b))) : (a, b) \in \mathbb{R}^2\}$ . Each horizontal cross-section of  $C_p$  is a scaled copy of  $Q$  centered at  $p$ . The triangulated  $\text{Vor}_Q(B)$  is the vertical projection of the lower envelope of  $\{C_p : p \in B\}$ , denoted by  $\mathcal{L}(B)$ . Similarly,  $\mathcal{L}(I)$  projects to  $\text{Vor}_Q(I)$ . We take the lower envelope of  $\mathcal{L}(B)$  and  $\mathcal{L}(I)$  to form  $\mathcal{L}(B \cup I)$  which projects to  $\text{Vor}_Q(B \cup I)$ . We do so in  $O(n2^{O(\log^* n)})$  expected time with a randomized algorithm that is based on an approach proposed and analyzed by Chan [5, Section 4].

**Task 2.** Suppose that for an input point  $p_i \in I$ , we have determined some subset  $B_i$  that satisfies  $B \subseteq B_i \subseteq S$ , and we have computed a Voronoi edge bend or Voronoi vertex  $v_i$  in  $\text{Vor}_Q(B_i)$  that conflicts with  $p_i$  and is known to be in  $V_S$  or not.

If  $v_i \in V_S$ , we search  $\text{Vor}_Q(S)$  from  $v_i$  to find  $V_S|_{p_i}$  (i.e., the subset of  $V_S$  that conflict with  $p_i$ ), which by Lemma 3 also gives the triangle  $t_i$  in the triangulated  $\text{Vor}_Q(S)$  that contains  $p_i$ . By Lemma 2, the expected total running time of this procedure over all input points is  $O(n)$ .

Suppose that  $v_i \notin V_S$ . So  $v_i$  is not a region boundary vertex in the  $m^2$ -division of  $\text{Vor}_Q(S)$ , i.e.,  $v_i$  lies inside a region in the  $m^2$ -division of  $\text{Vor}_Q(S)$ , say  $\Pi$ . For each boundary vertex  $w$  of  $\Pi$ , let  $Q_w^*$  be the largest homothetic copy of  $Q^*$  centered at  $w$  such that  $\text{int}(Q_w^*) \cap B = \emptyset$ . These  $Q_w^*$ 's form an arrangement of  $O(m^2)$  complexity, and we locate  $p_i$  in this arrangement in  $O(\log m)$  time. It tells us whether  $p_i \in Q_w^*$  for some boundary vertex  $w$  of  $\Pi$ . If so, then  $p_i$  conflicts with  $w$ , which belongs to  $V_S$ , and we search  $\text{Vor}_Q(S)$  from  $w$  to find  $V_S|_{p_i}$  and hence the triangle  $t_i$  in the triangulated  $\text{Vor}_Q(S)$  that contains  $p_i$ . Otherwise,  $p_i$  must lie inside  $\Pi$  in order to conflict with  $v_i$  inside  $\Pi$  without conflicting with any boundary vertex of  $\Pi$ . So we do a point location in  $O(\log m)$  time to locate  $p_i$  in the portion of the triangulated  $\text{Vor}_Q(S)$  inside  $\Pi$ . This gives  $t_i$ .

How do we compute  $v_i$  for  $p_i$ ? We discuss this computation and provide more details of Step 2 in the full version [9]. The following lemma summarizes the result that follows from the discussion above.

► **Lemma 6.** *Given  $\text{Vor}_Q(I)$ , the triangles  $t_1, \dots, t_n$  in the triangulated  $\text{Vor}_Q(S)$  that contain  $p_1, \dots, p_n \in I$  can be computed in  $O(n \log m + n2^{O(\log^* n)})$  expected time.*

► **Lemma 7.** *Step 1 of the operation phase takes  $O(\frac{1}{\varepsilon}(n \log m + n2^{O(\log^* n)} + H))$  expected time, where  $H$  is the entropy of the distribution of  $\text{Vor}_Q(I)$ .*

**Proof.** Let  $A \in [1, m]$  be a random variable that indicates which distribution in the mixture generates the input instance. By the chain rule for conditional entropy [22, Proposition 2.23],  $H(t_i) \leq H(t_i) + H(A|t_i) = H(t_i, A) = H(A) + H(t_i|A)$ . It is known that  $H(A) \leq \log_2(\text{domain size of } A) = \log_2 m$  [22, Theorem 2.43]. Thus,  $\sum_{i=1}^n H(t_i) \leq n \log_2 m + \sum_{i=1}^n H(t_i|A)$ . The variables  $t_1|A, \dots, t_n|A$  are mutually independent. So  $\sum_{i=1}^n H(t_i|A) = H(t_1, \dots, t_n|A)$ . Since entropy is not increased by conditioning [22, Theorem 2.38], we get  $\sum_{i=1}^n H(t_i|A) = H(t_1, \dots, t_n|A) \leq H(t_1, \dots, t_n)$ . By Lemma 6, we can determine  $t_1, \dots, t_n$  using  $\text{Vor}_Q(I)$  in  $O(n \log m + n2^{O(\log^* n)})$  expected time. So  $H(t_1, \dots, t_n) = O(n \log m + n2^{O(\log^* n)} + H)$  by Lemma 5, where  $H$  is the entropy of the distribution of  $\text{Vor}_Q(I)$ . ◀

In the Euclidean metric, merging  $\text{Vor}(B)$  and  $\text{Vor}(I)$  into  $\text{Vor}(B \cup I)$  can be reduced to finding the intersection of two convex polyhedra of  $O(n)$  size in  $\mathbb{R}^3$ , which can be solved in  $O(n)$  time [5]. So the expected running time of step 1 improves to  $O(\frac{1}{\varepsilon}(n \log m + H))$ .

## 4.2 Construction of $R$

Step 1 determines the triangle  $t_i$  in the triangulated  $\text{Vor}_Q(S)$  that contains  $p_i \in I$ . We search  $\text{Vor}_Q(S)$  from  $t_i$  to find  $V_S|_{p_i}$ , which takes  $O(|V_S|_{p_i}|)$  time [18]. This search also gives the Voronoi cells that conflict with  $p_i$ . The total time over all  $i \in [n]$  is  $O(\sum_{v \in V_S} |Z_v|)$ , where  $Z_v$  is the subset of input points that conflict with  $v$ . Since  $R$  includes all sites whose cells conflict with the input points and the representative points of all cluster roots in  $T_S$ , we have  $|R| \leq \sum_{v \in V_S} |Z_v| + O(n)$ . The following result follows from Lemma 2.

► **Lemma 8.** *The set  $R$  has  $O(n)$  expected size. Step 2 of the operation phase constructs  $R$  in  $O(n)$  expected time.*

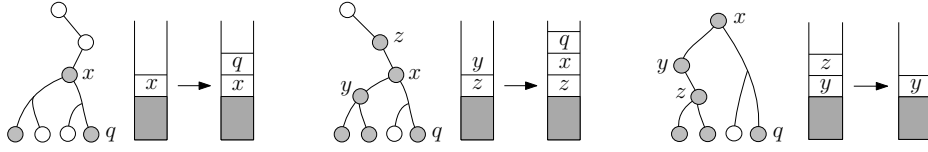
## 4.3 Extraction of $\text{Vor}_Q(R)$

### 4.3.1 Construction of $T_R$

We define a *compression* of a net-tree  $T$ . Select a subset  $U$  of leaves in  $T$ . Let  $T' \subseteq T$  be the minimal subtree that spans  $U$ . Bypass all internal nodes in  $T'$  that have only one child. *The resulting tree is the compression of  $T$  to  $U$ .* The following result is an easy observation.

► **Lemma 9.** *Let  $T$  be a net-tree. Let  $T_1$  be the compression of  $T$  to a subset  $U_1$  of leaves. The compression of  $T_1$  to any subset  $U_2$  of leaves in  $T_1$  can also be obtained by a compression of  $T$  to  $U_2$ .*

Conceptually,  $T_R$  is defined as follows. Select all leaves of  $T_S$  that are points in  $R$ , and  $T_R$  is the compression of  $T_S$  to these selected leaves. Since  $R$  includes the representative points of all cluster roots, all ancestors of the cluster roots in  $T_S$  will survive the compression and exist as nodes in  $T_R$ . The compression affects the clusters only. More precisely, for each



■ **Figure 4** Three different cases in the manipulation of the stack. The tree shown is a part of  $T_S$ . The gray nodes are nodes in  $T_c$ . The gray leaves are leaves in  $R_c$ .

cluster  $c$  in  $T_S$ , we select its leaves that are points in  $R$  and compute the compression  $T_c$  of the cluster  $c$  to these selected leaves. Substituting every cluster  $c$  in  $T_S$  by  $T_c$  gives the desired  $T_R$ . It remains to discuss how to compute the  $T_c$ 's.

We divide  $R$  in  $O(n)$  expected time into sublists  $R_1, R_2, \dots$  such that  $R_c$  consists of the points that are leaves in cluster  $c$ . Recall that every cluster  $c$  has an initially empty van Emde Boas tree  $EB_c$  for its leaves in left-to-right order. For each  $R_c$ , we insert all leaves in  $R_c$  into  $EB_c$  and then repeatedly perform extract-min on  $EB_c$ . This gives in  $O(|R_c| \log \log m)$  time a sorted list  $R'_c$  of the leaves in  $R_c$  according to their left-to-right order in the cluster  $c$ .

If  $|R'_c| = 1$ , then  $T_c$  consists of the single leaf in  $R_c$ . Suppose that  $|R'_c| \geq 2$ . We construct  $T_c$  using a stack. Initially,  $T_c$  is a single node which is the first leaf in  $R'_c$ . The stack stores the nodes on the rightmost root-to-leaf path in the current  $T_c$ , with the root at the stack bottom and the leaf at the stack top. When we scan the next leaf  $q$  in  $R'_c$ , we find in cluster  $c$  the nearest common ancestor  $x$  of  $q$  and  $q$ 's predecessor in  $R'_c$ . This takes  $O(\log \log m)$  time [20]. If we see  $x$  at the stack top, we add  $q$  as a new leaf to  $T_c$  with  $x$  as its parent, and then we push  $q$  onto the stack. Refer to the left image in Figure 4. If we see an ancestor  $z$  of  $x$  at the stack top, let  $y$  be the node that was immediately above  $z$  in the stack and was just popped, we make  $x$  the rightmost child of  $z$  in  $T_c$  (which was  $y$  previously), we also make  $y$  and  $q$  the left and right children of  $x$  respectively, and then we push  $x$  and  $q$  in this order onto the stack. Refer to the middle image in Figure 4. If neither of the two conditions above happens and the stack is not empty, we pop the stack and repeat. Refer to the right image in Figure 4. If the stack becomes empty, we make  $x$  the new root of  $T_c$ , we also make the old root of  $T_c$  and  $q$  the left and right children of  $x$  respectively, and then we push  $x$  and  $q$  in this order onto the stack. The construction of  $T_c$  takes  $O(|R_c| \log \log m)$  time.

► **Lemma 10.** *The compression  $T_R$  of  $T_S$  to  $R$  can be computed in  $O(n \log \log m)$  time.*

### 4.3.2 Construction of the $k$ -nearest neighbor graph

Let  $X$  be any subset of  $S$ . Assume that the compression  $T_X$  of  $T_S$  to  $X$  is available. We show how to use  $T_X$  to construct in  $O(k|X|)$  time the  $k$ -nearest neighbor graph of  $X$  under the metric  $d$ . We denote this graph by  $k\text{-NN}_X$ . We will use the *well-separated pair decomposition* or WSPD for short. For any  $c \geq 1$ , a set  $\{\{A_1, B_1\}, \dots, \{A_s, B_s\}\}$  is a  $c$ -WSPD of  $X$  under  $d$  if the following properties are satisfied:

- $\forall i, A_i, B_i \subseteq X$ .
- $\forall$  distinct  $x, y \in X, \exists i$  such that  $\{x, y\} \in \{\{a, b\} : a \in A_i \wedge b \in B_i\}$ .
- $\forall i$ , the maximum of the diameters of  $A_i$  and  $B_i$  under  $d$  is less than  $\frac{1}{c} \cdot d(A_i, B_i)$ . It implies that  $A_i \cap B_i = \emptyset$ .

It is known that a  $c$ -WSPD has  $O(c^{O(1)}|X|)$  size and can be constructed in  $O(c^{O(1)}|X|)$  time from a net-tree for  $X$  [15]. The same method works for a compression  $T_X$  of  $T_S$  to  $X$ , giving a  $c$ -WSPD of  $O((c+1)^{O(1)}|X|)$  size in  $O((c+1)^{O(1)}|X|)$  time. To compute  $k\text{-NN}_X$ , we transfer a strategy in [4] for constructing a Euclidean  $k$ -nearest neighbor graph using a WSPD.



► **Lemma 11.** *Given the compression  $T_X$  of  $T_S$  to any subset  $X \subseteq S$ , the  $k$ - $\text{NN}_X$  can be constructed in  $O(k|X|)$  time.*

The next result shows that the vertex degree of  $1\text{-NN}_X$  is  $O(1)$ .

► **Lemma 12.** *For any subset  $X \subseteq S$ , every vertex in  $1\text{-NN}_X$  has  $O(1)$  degree, and adjacent vertices in  $1\text{-NN}_X$  are Voronoi neighbors in  $\text{Vor}_Q(X)$ .*

### 4.3.3 $\text{Vor}_Q(R)$ from the nearest neighbor graph

We show how to construct  $\text{Vor}_Q(R)$  in  $O(n)$  expected time using  $1\text{-NN}_R$ . We use the following recursive routine which is similar to the one in [3] for constructing an Euclidean Delaunay triangulation from the Euclidean nearest neighbor graph. The top-level call is  $\text{VorNN}(R, T_R)$ .

■ **Algorithm 2**  $\text{VorNN}(Y, T_Y)$ .

1. If  $|Y| = O(1)$ , compute  $\text{Vor}_Q(Y)$  directly and return.
2. Compute  $1\text{-NN}_Y$  under the metric  $d$  using  $T_Y$ .
3. Let  $X \subseteq Y$  be a random sample such that  $X$  meets every connected component of  $1\text{-NN}_Y$ , and  $\Pr[p \in X] = 1/2$  for every  $p \in Y$ .
4. Compute the compression  $T_X$  of  $T_Y$  to  $X$ .
5. Call  $\text{VorNN}(X, T_X)$  to compute  $\text{Vor}_Q(X)$ .
6. Using  $1\text{-NN}_Y$  as a guide, insert the points in  $Y \setminus X$  into  $\text{Vor}_Q(X)$  to form  $\text{Vor}_Q(Y)$ .

There are two differences from [3]. First, we use a compression  $T_Y$  of  $T_S$  to compute  $1\text{-NN}_Y$  in step 2, which takes  $O(|Y|)$  time by Lemma 11. Second, we need to compress  $T_Y$  to  $T_X$  in step 4. This compression works in almost the same way as described in Section 4.3.1 except that we can afford to traverse  $T_Y$  in  $O(|Y|)$  time to answer all nearest common ancestor queries required for constructing  $T_X$ . Thus, step 4 runs in  $O(|Y|)$  time.

Step 3 is implemented as follows [3]. Form an arbitrary maximal matching of  $1\text{-NN}_Y$ . By the definition of  $1\text{-NN}_Y$ , each connected component of  $1\text{-NN}_Y$  contains at least one matched pair. Randomly select one point from every matched pair. Then, among those unmatched points in  $1\text{-NN}_Y$ , select each one with probability  $1/2$  uniformly at random. The selected points form the subset  $X$  required in step 3. The time needed is  $O(|Y|)$ .

In step 6, for each  $p \in Y \setminus X$  that is connected to some point  $q \in X$  in  $1\text{-NN}_Y$ ,  $p$  and  $q$  are Voronoi neighbors in  $\text{Vor}_Q(Y)$  by Lemma 12. So  $p$  conflicts with a point in  $V_q(X)$ . By Lemma 4,  $p$  conflicts with a Voronoi edge bend or Voronoi vertex in  $\partial V_q(X)$ , which can be found in  $O(|\partial V_q(X)|)$  time. After finding a Voronoi edge bend or Voronoi vertex  $v$  in  $\partial V_q(X)$  that conflicts with  $p$ , we search  $\text{Vor}_Q(X)$  from  $v$  to find all Voronoi edge bends and Voronoi vertices that conflict with  $p$ . In the same search of  $\text{Vor}_Q(X)$ , we modify  $\text{Vor}_Q(X)$  into  $\text{Vor}_Q(X \cup \{p\})$  as in a randomized incremental construction [18]. By the Clarkson-Shor analysis [12], the expected running time of the search of  $\text{Vor}_Q(X)$  and the Voronoi diagram modification over the insertions of all points in  $Y \setminus X$  is  $O(|Y|)$ . We spend  $O(|\partial V_q(X)|)$  time to find  $v$ . It translates to an  $O(1)$  charge at each vertex of  $V_q(X)$ . This charging happens only for  $q$ 's neighbors in  $1\text{-NN}_Y$ . By Lemma 12, there are  $O(1)$  such neighbors of  $q$ , so the charge at each vertex of  $V_q(X)$  is  $O(1)$ . Moreover, if a vertex of  $V_q(X)$  is destroyed by the insertion of a point from  $Y \setminus X$ , that vertex will not reappear. So the  $O(|\partial V_q(X)|)$  cost is absorbed by the structural changes which is already taken care of by the Clarkson-Shor analysis. Unwinding the recursion gives a total expected running time of  $O(|R| + |R|/2 + |R|/4 + \dots) = O(|R|)$ .

► **Lemma 13.**  *$\text{VorNN}(R, T_R)$  computes  $\text{Vor}_Q(R)$  in  $O(|R|)$  expected time.*

#### 4.4 Computing $\text{Vor}_Q(I)$ from $\text{Vor}_Q(R)$ and $I$

Let  $q$  be a point in  $R$ . Let  $v_1, v_2, \dots$  be the vertices of  $V_q(R)$ , in clockwise order, which may be Voronoi edge bends or Voronoi vertices. Let  $Q_{v_i}^*$  denote the largest homothetic copy of  $Q^*$  centered at  $v_i$  such that  $\text{int}(Q_{v_i}^*) \cap R = \emptyset$ . Let  $Z_{v_i} = Q_{v_i}^* \cap I$  where  $I$  is an input instance.

► **Lemma 14.** *The portions of  $\text{Vor}_Q(R \cup I)$  and  $\text{Vor}_Q(\{q\} \cup Z_{v_i} \cup Z_{v_{i+1}})$  inside the triangle  $qv_i v_{i+1}$  are identical.*

**Proof.** Let  $p$  be a point in  $(R \cup I) \setminus \{q\}$  that contributes to  $\text{Vor}_Q(R \cup I)$  inside  $qv_i v_{i+1}$ . As  $qv_i v_{i+1} \subseteq V_q(R)$ ,  $p \notin R$ . So  $p \in I$ . By Lemma 4,  $p$  conflicts with  $v_i$  or  $v_{i+1}$ . ◀

Step 2 of the operation phase has found  $V_S|_{p_i}$  for each  $p_i \in I$ .  $V_S|_{p_i}$  and the portions of the Voronoi edges of  $\text{Vor}_Q(S)$  among the points in  $V_S|_{p_i}$  are preserved in  $\text{Vor}_Q(R)$  because  $R$  includes the subset of  $S$  whose Voronoi cells conflict with the input points. Hence,  $\bigcup_{i=1}^n V_S|_{p_i}$  is the set  $U_R$  of Voronoi edge bends and Voronoi vertices in  $\text{Vor}_Q(R)$  that conflict with the input points. By Lemma 14, we locally compute pieces of  $\text{Vor}_Q(R \cup I)$  and stitch them together. The running time is  $O(\sum_{u,v} (|Z_u| + |Z_v|) \log(|Z_u| + |Z_v|))$ , where the sum is over all pairs  $\{u, v\}$  of adjacent Voronoi edge bends and Voronoi vertices in  $\text{Vor}_Q(R)$  such that  $\{u, v\} \cap U_R \neq \emptyset$ . Since the degrees of Voronoi edge bends and Voronoi vertices are two and three respectively, this running time can be bounded by  $O(\sum_{v \in U_R} |Z_v| \log |Z_v|)$ . Since  $U_R \subseteq V_S$ , by Lemma 2, step 6 of the operation phase computes  $\text{Vor}_Q(R \cup I)$  in  $O(n)$  expected time.

In step 7, the splitting of  $\text{Vor}_Q(R \cup I)$  into  $\text{Vor}_Q(R)$  and  $\text{Vor}_Q(I)$  can be performed in  $O(n)$  expected time by using the algorithm in [6] for splitting a Euclidean Delaunay triangulation. That algorithm is combinatorial in nature. It relies on the Voronoi diagram being planar and of  $O(n)$  size, all points having  $O(1)$  degrees in the nearest neighbor graph, and that one can delete a site from a Voronoi diagram in time proportional to its number of Voronoi neighbors. The first two properties hold in our case, and it is known how to delete a site from an abstract Voronoi diagram so that the expected running time is proportional to its number of Voronoi neighbors [17].

► **Lemma 15.** *Step 6 of the operation phase computes  $\text{Vor}_Q(R \cup I)$  in  $O(n)$  expected time, and step 7 splits  $\text{Vor}_Q(R \cup I)$  into  $\text{Vor}_Q(I)$  and  $\text{Vor}_Q(R)$  in  $O(n)$  expected time.*

In summary, since steps 2-7 of the operation phase take  $O(n)$  expected time, the limiting complexity is dominated by the  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}n2^{O(\log^* n)} + \frac{1}{\varepsilon}H)$  expected running time of step 1. In the Euclidean case, step 1 runs faster in  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H)$  time.

► **Theorem 16.** *Let  $Q$  be a convex polygon with  $O(1)$  complexity. Let  $n$  be the input size. For any  $\varepsilon \in (0, 1)$  and any hidden mixture of at most  $m = o(\sqrt{n})$  product distributions such that each distribution contributes an instance with a probability of  $\Omega(1/n)$ , there is a self-improving algorithm for constructing a Voronoi diagram under  $d_Q$  with a limiting complexity of  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}n2^{O(\log^* n)} + \frac{1}{\varepsilon}H)$ . For the Euclidean metric, the limiting complexity is  $O(\frac{1}{\varepsilon}n \log m + \frac{1}{\varepsilon}H)$ . The training phase runs in  $O(mn \log^2(mn) + m^\varepsilon n^{1+\varepsilon} \log(mn))$  time. The success probability is at least  $1 - O(1/n)$ .*

## 5 Conclusion

It is open whether one can get rid of the requirement that each distribution in the mixture contributes an instance with a probability of  $\Omega(1/n)$ , which is not needed for self-improving sorting [8]. Eliminating the  $n2^{O(\log^* n)}$  term from the limiting complexity might require solving the question raised in [5] that whether there is an  $O(n)$ -time algorithm for computing



the lower envelope of pseudo-planes. As a Voronoi diagram can be interpreted as the lower envelope of some appropriate surfaces, a natural question is what surfaces admit a self-improving lower envelope algorithm.

---

### References

---

- 1 N. Ailon, B. Chazelle, K. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. *SIAM Journal on Computing*, 40:350–375, 2011.
- 2 S. Arya, T. Malamatos, D.M. Mount, and K.C. Wong. Optimal expected-case planar point location. *SIAM Journal on Computing*, 37:584–610, 2007.
- 3 K. Buchin and W. Mulzer. Delaunay triangulations in  $o(\text{sort}(n))$  time and more. *Journal of the ACM*, 58:6:1–6:27, 2011.
- 4 P.B. Callahan and S.R. Kosaraju. A decomposition of multidimensional point sets with applications to  $k$ -nearest-neighbors and  $n$ -body potential fields. *Journal of the ACM*, 42:67–90, 1995.
- 5 T.M. Chan. A simpler linear-time algorithm for intersecting two convex polyhedra in three dimensions. *Discrete and Computational Geometry*, 56:860–865, 2016.
- 6 B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristan, and M. Teillaud. Splitting a delaunay triangulation in linear time. *Algorithmica*, 34:39–46, 2002.
- 7 S.-W. Cheng, M.-K. Chiu, K. Jin, and M.T. Wong. A generalization of self-improving algorithms. In *Proceedings of the International Symposium on Computational Geometry*, pages 29:1–29:13. Full version: arXiv:2003.08329v2 [cs.CG], 2020, 2020.
- 8 S.-W. Cheng, K. Jin, and L. Yan. Extensions of self-improving sorters. *Algorithmica*, 82:88–106, 2020.
- 9 S.-W. Cheng and M.T. Wong. Self-improving voronoi construction for a hidden mixture of product distributions. arXiv:2109.13460 [cs.CG], 2021.
- 10 L. Paul Chew and R.L. Scot Drysdale. Voronoi diagrams based on convex distance functions. In *Proceedings of the 1st Annual Symposium on Computational Geometry*, pages 235–244, 1985.
- 11 K.L. Clarkson, W. Mulzer, and C. Seshadhri. Self-improving algorithms for coordinatewise maxima and convex hulls. *SIAM Journal on Computing*, 43:617–653, 2014.
- 12 K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.
- 13 H. Edelsbrunner, L.J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.
- 14 G.N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.
- 15 S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35:1148–1184, 2006.
- 16 J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry: Theory and Applications*, 29:19–22, 2004.
- 17 K. Junginer and E. Papadopoulou. Deletion in abstract voronoi diagram in expected linear time. In *Proceedings of the 34th International Symposium on Computational Geometry*, pages 50:1–50:14, 2018.
- 18 R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract voronoi diagrams. *Computational Geometry: Theory and Applications*, 3:157–184, 1993.
- 19 E. Pyrga and S. Ray. New existence proofs for  $\epsilon$ -nets. In *Proceedings of the 24th Annual Symposium on Computational Geometry*, pages 199–207, 2008.
- 20 A.K. Tsakalides and J. van Leeuwen. An optimal pointer machine algorithm for finding nearest common ancestors. Technical Report RUU-CS-88-17, Department of Computer Science, University of Utrecht, 1988.
- 21 P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.
- 22 R.W. Yeung. *A First Course in Information Theory*. Kluwer Academic/Plenum Publishers, 2002.



# Connected Coordinated Motion Planning with Bounded Stretch

Sándor P. Fekete  

Department of Computer Science, TU Braunschweig, Germany

Phillip Keldenich  

Department of Computer Science, TU Braunschweig, Germany

Ramin Kosfeld  

Department of Computer Science, TU Braunschweig, Germany

Christian Rieck  

Department of Computer Science, TU Braunschweig, Germany

Christian Scheffer  

Faculty of Electrical Engineering and Computer Science,  
Bochum University of Applied Sciences, Bochum, Germany

---

## Abstract

We consider the problem of coordinated motion planning for a swarm of simple, identical robots: From a given start grid configuration of robots, we need to reach a desired target configuration via a sequence of parallel, continuous, collision-free robot motions, such that the set of robots induces a connected grid graph at all integer times. The objective is to minimize the *makespan* of the motion schedule, i.e., to reach the new configuration in a minimum amount of time. We show that this problem is NP-hard, even for deciding whether a makespan of 2 can be achieved, while it is possible to check in polynomial time whether a makespan of 1 can be achieved.

On the algorithmic side, we establish simultaneous constant-factor approximation for two fundamental parameters, by achieving *constant stretch* for *constant scale*. Scaled shapes (which arise by increasing all dimensions of a given object by the same multiplicative factor) have been considered in previous seminal work on self-assembly, often with unbounded or logarithmic scale factors; we provide methods for a generalized scale factor, bounded by a constant. Moreover, our algorithm achieves a *constant stretch factor*: If mapping the start configuration to the target configuration requires a maximum Manhattan distance of  $d$ , then the total duration of our overall schedule is  $\mathcal{O}(d)$ , which is optimal up to constant factors.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry; Computing methodologies  $\rightarrow$  Motion path planning

**Keywords and phrases** Motion planning, parallel motion, bounded stretch, scaled shape, makespan, connectivity, swarm robotics

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.9

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.12381> [11]

**Acknowledgements** We thank Linda Kleist and Arne Schmidt for helpful algorithmic discourse and suggestions that improved the presentation of this paper.

## 1 Introduction

Coordinating the motion of a set of objects is a fundamental problem that occurs in a large spectrum of theoretical and practical contexts. This problem was also the subject of the 2021 CG Challenge [12], highlighting the high relevance for the algorithmic community.



© Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer; licensed under Creative Commons License CC-BY 4.0

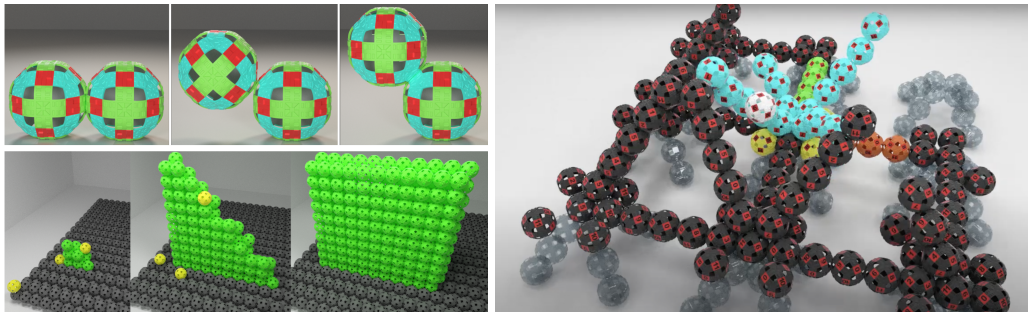
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 9; pp. 9:1–9:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** (Top left) An autonomous, sphere-shaped catom, changing location by rotating around a second catom used as a pivot [17]. (Bottom left) A swarm of catoms building a wall [17]. (Right) A configuration of catoms in the process of building a scaffold structure [26].

In this paper, we consider *connected* swarm reconfiguration: transform a set of mobile agents from a given start into a desired target configuration by a sequence of parallel, continuous, collision-free motions that keeps the overall arrangement connected at all integer times. Problems of this type occur for assemblies in space, where disconnected pieces cannot regain connectivity, or for small-scale swarm robots (such as *catoms* in *claytronics* [13]) which need connectivity for local motion, electric power and communication; see Figure 1.

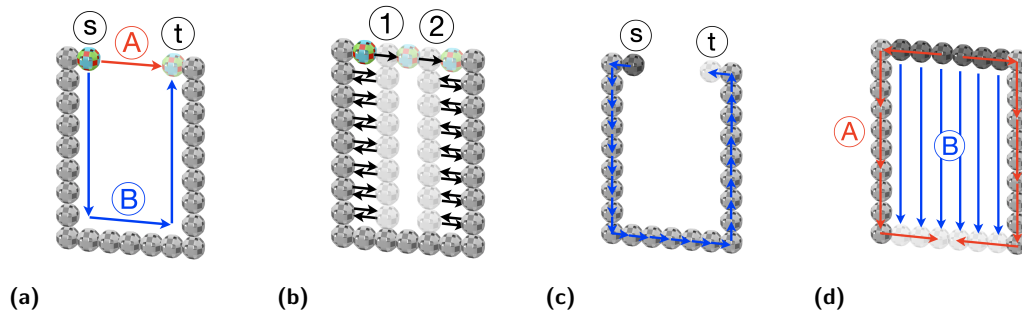
A crucial algorithmic aspect is *efficiency*: How can we coordinate the robot motions, such that a target configuration is reached in timely or energy-efficient manner? Most previous work has largely focused on sequential schedules, where one robot moves at a time, with objectives such as minimizing the number of moves. In practice, however, robots usually move simultaneously, so we desire a *parallel* motion schedule, with a natural objective of minimizing the time until completion, called *makespan*. How well can we exploit parallelism in a robot swarm to achieve an efficient schedule? As illustrated in Figure 2, this is where the connectivity constraints make a tremendous difference.

A critical parameter in self-assembly is the robustness of the involved shapes, corresponding to sufficient local connectivity to prevent fragility. This leads to the concept of *scaled shapes*; intuitively, a scale factor of  $c$  corresponds to replacing each pixel of a polyomino shape by a quadratic  $c \times c$  array of pixels. This has fundamental connections to Kolmogorov and runtime complexity, as shown by Soloveichik and Winfree [22]: “Furthermore, the independence of scale in self-assembly theory appears to play the same crucial role as the independence of running time in the theory of computability. . . [we] show that the running-time complexity, with respect to Turing machines, is polynomially equivalent to the scale complexity of the same function implemented via self-assembly by a finite set of tile types.” As a consequence, limiting scale has received considerable attention, as sketched in the related work section.

As we demonstrate in this paper, achieving optimal makespan for connected reconfiguration is provably hard, even in relatively basic cases. On the positive side, we present methods that are capable of achieving a constant-factor approximation, assuming not more than a generalization of constant scale of configurations. As can be seen from Figure 2, this is considerably more intricate than in a non-connected setting, even in very basic instances.

## 1.1 Our Results

We provide a spectrum of new results for questions arising from efficiently reconfiguring a connected, unlabeled swarm of robots from a start configuration  $C_s$  into a target configuration  $C_t$ , aiming for minimizing the overall makespan and maintaining connectivity in each step.



■ **Figure 2** Reconfiguration with and without connectivity constraints. (a) Relocating the colored particle from  $s$  to  $t$ , without (red trajectory A) and with connectivity constraint (blue trajectory B). (b) Coordinating many particles to quickly deliver a specific particle to a desired location, while preserving connectivity. (c) Reconfiguring an arrangement of *identical* particles in a single, parallel, connected step. (d) Reconfiguring an arch-shaped arrangement of identical particles into a U-shaped one, without (motion plan A, shown in red) and with connectivity (motion plan B, shown in blue).

- Deciding whether there is a schedule with a makespan of 1 transforming  $C_s$  into  $C_t$  can be done in polynomial time, see Theorem 1.
- Deciding whether there is a schedule with a makespan of 2 transforming  $C_s$  into  $C_t$  is NP-hard, see Theorem 2. This implies NP-hardness of approximating the minimum makespan within a constant of  $(\frac{3}{2} - \varepsilon)$ , for any  $\varepsilon > 0$ , see Corollary 3.
- As our main algorithmic result, we show that there is a constant  $c^*$  such that for any pair of start and target configurations with a (generalized) scale of at least  $c^*$ , a schedule with constant stretch can be computed in polynomial time, see Theorem 4 and Corollary 11. This implies that there is a constant-factor approximation for the problem of computing schedules with minimal makespan restricted to pairs of start and target configurations with a scale of at least  $c^*$ , see Corollary 12.

## 1.2 Related Work

In the following, we provide a sketch of the wide spectrum of related work; see the full version of our paper [11] for a more detailed overview, as well as [7].

The basic question of coordinating the motion of many agents in an efficient manner arises in many applications, such as ground swarm robotics [18, 19], aerial swarm robotics [3, 28], air traffic control [5], and vehicular traffic networks [10, 20]. Multi-robot coordination dates back to the seminal work by Schwartz and Sharir [21] from the 1980s. In both discrete and geometric variants of the problem, the objects can be *labeled*, *colored* or *unlabeled*. In the *labeled* case, the objects are all distinguishable and each object has its own, uniquely defined target position. In the *colored* case, the objects are partitioned into  $k$  groups and each target position can only be covered by an object with the right color; see Solovey and Halperin [23]. In the *unlabeled* case, objects are indistinguishable and target positions can be covered by any object; see Kloder and Hutchinson [15], Turpin et al. [27], Adler et al. [1], and Solovey et al. [25]. On the negative side, Solovey and Halperin [24] prove that the unlabeled multiple-object motion planning problem is PSPACE-hard.

For an instance of parallel reconfiguration, a lower bound for the time required for *all* robots to reach their destinations is the maximum distance between a robot's origin and destination. This motivates the *stretch factor*, i.e., the ratio of the makespan of a parallel motion plan divided by the maximum distance. In recent work, Demaine et al. [2, 7] were

able to develop algorithms that can achieve *constant* stretch factors that are independent of the number of robots; however, these approaches do not satisfy the crucial connectivity constraint, so different algorithmic methods are required.

The concept of scale complexity has received considerable attention in self-assembly; achieving constant scale has required special cases or operations. Soloveichik and Winfree [22] showed that the minimal number of distinct tile types necessary to self-assemble a shape, at some scale, can be bounded both above and below in terms of the shape’s Kolmogorov complexity, leading to unbounded scale in general. Demaine et al. [9] showed that allowing to destroy tiles can be exploited to achieve a scale that is only bounded by a logarithmic factor, beating the linear bound without such operations. In a setting of recursive, multi-level *staged* assembly with a logarithmic number of stages (i.e., “hands” for handling subassemblies), Demaine et al. [6] achieved logarithmic scale, and constant scale for more constrained classes of polyomino shapes; this was later improved by Demaine et al. [8] to constant scale for a logarithmic number of stages. More recently, Luchsinger et al. [16] employed repulsive forces between tiles to achieve constant scale in two-handed self-assembly.

## 2 Preliminaries

We consider *robots* at integer grid positions. A set of  $n$  unlabeled robots forms a *configuration*  $C$ , corresponding to a vertex-induced subgraph  $H$  of the infinite integer grid, with an edge between two grid vertices  $v_1, v_2 \in C$  if and only if  $v_1$  and  $v_2$  are on adjacent grid positions, i.e., a distance of 1 apart. A configuration is *connected*, if  $H$  is connected. Two configurations  $C_1$  and  $C_2$  *overlap*, if they have at least one position in common. A configuration  $C$  is *c-scaled*, if it is the union of  $c \times c$  squares of vertices. The *scale* of a configuration  $C$  is the maximal  $c$  such that  $C$  is  $c$ -scaled. This corresponds to objects being composed of pixels at a certain resolution; note that this is a generalization of the uniform pixel scaling studied in previous literature (which considers a  $c$ -grid-based partition instead of an arbitrary union), so it supersedes that definition and leads to a more general set of results. Two robots are *adjacent* if their positions  $v_1, v_2$  are adjacent, i.e.,  $(v_1, v_2) \in E(H)$ ; and *diagonally adjacent* if their positions are adjacent with a common vertex  $v$  such that  $(v_1, v)$  and  $(v, v_2)$  lie orthogonal.

A robot can move in discrete time steps by changing its location from a grid position  $v$  to an adjacent grid position  $w$ ; denoted by  $v \rightarrow w$ . Two moves  $v_1 \rightarrow w_1$  and  $v_2 \rightarrow w_2$  are called *collision-free* if  $v_1 \neq v_2$  and  $w_1 \neq w_2$ . A *transformation* between two configurations  $C_1 = \{v_1, \dots, v_n\}$  and  $C_2 = \{w_1, \dots, w_n\}$  is a set of collision-free moves  $\{v_i \rightarrow w_i \mid i = 1, \dots, n\}$ . Note that a robot is allowed to hold its position. For  $M \in \mathbb{N}$ , a *schedule* is a sequence  $C_1 \rightarrow \dots \rightarrow C_{M+1}$  (also denoted as  $C_1 \rightrightarrows C_{M+1}$ ) of transformations, with a *makespan* of  $M$ . A *stable* schedule  $C_1 \rightrightarrows_{\chi} C_{M+1}$  uses only connected configurations. Let  $C_s, C_t$  be two connected configurations with equally many robots called *start* and *target configuration*, respectively. A *matching* is a one-to-one mapping between vertices from  $C_s$  and  $C_t$ . The *diameter* of a matching is the maximal Manhattan distance between two matched vertices. A *bottleneck matching* is a matching with a minimal diameter. The *diameter*  $d$  of  $(C_s, C_t)$  is the diameter of a bottleneck matching. The *stretch (factor)* of a (stable) schedule is the ratio between the makespan  $M$  of the schedule and the diameter  $d$  of  $(C_s, C_t)$ .



### 3 Makespan 1 and 2

As a first observation we note that it can be decided in polynomial time whether there is a schedule  $C_s \rightarrow C_t$  with a makespan of 1 between a start and a target configuration.

► **Theorem 1.** *For a pair of configurations  $C_s$  and  $C_t$ , each with  $n$  vertices, it can be decided in polynomial time whether there is a schedule with a makespan of 1 transforming  $C_s$  into  $C_t$ .*

**Proof.** Given two connected configurations  $C_s$  and  $C_t$ , each with  $n$  vertices. We compute the bipartite graph  $G_{C_s, C_t} = (V_s \cup V_t, E)$ , where  $V_s$  and  $V_t$  consist of all occupied positions in  $C_s$  and  $C_t$ . For  $E$ , we add an edge if and only if an occupied position in  $C_t$  is adjacent (or identical) to an occupied position in  $C_s$ . Consider a perfect matching in  $G_{C_s, C_t}$ . Because edges only connect positions which are at most one unit step apart, all robots can move along their respective matching edges without crossing the path of another robot. If there is no perfect matching in  $G_{C_s, C_t}$ , at least one robot would have to move to a position further away. Thus, a makespan of 1 would not be achievable. So, there is a schedule of makespan 1 if and only if  $G_{C_s, C_t}$  admits a perfect matching. Because the graph is sparse, this can be checked in  $\mathcal{O}(n^{3/2})$  time, using the method of Hopcroft and Karp [14]. ◀

Note that, because  $C_s$  and  $C_t$  have to be connected, a schedule with a makespan of 1 is always stable. Even for a makespan of 2, the same problem becomes provably difficult.

► **Theorem 2.** *For a pair of configurations  $C_s$  and  $C_t$ , each with  $n$  vertices, deciding whether there is a stable schedule with a makespan of 2 transforming  $C_s$  into  $C_t$  is NP-hard.*

The proof is based on a reduction from the NP-hard problem PLANAR MONOTONE 3SAT [4], which asks to decide whether a Boolean 3-CNF formula  $\varphi$  is satisfiable, for which in each clause the literals are either all unnegated or all negated.

The reduction considers an instance  $\varphi$  of PLANAR MONOTONE 3SAT and constructs an instance  $I_\varphi$  with start configuration  $C_s$  and target configuration  $C_t$ ; see Figure 3, with start configuration (red), target configuration (dark cyan), and positions in both configurations (gray) indicated by colors. We consider a rectilinear planar embedding of the variable-clause incidence graph  $G_\varphi$  of  $\varphi$ , with variable vertices placed horizontally in a row, and clauses with unnegated and negated literals placed above and below, respectively. Variables of  $\varphi$  are represented by horizontal *variable gadgets* (light red). Two additional *auxiliary gadgets* (light blue) are positioned at the top and at the bottom boundary of the instance, connected to the variable gadget via bridges at the right boundary, and a *separation gadget* (yellow) between each adjacent and nested pair of *clause gadgets* (blue). All clause gadgets are connected via bridges to separation gadgets and possibly to the auxiliary gadgets. Further, there are bridges from a clause gadget to the respectively contained variables.

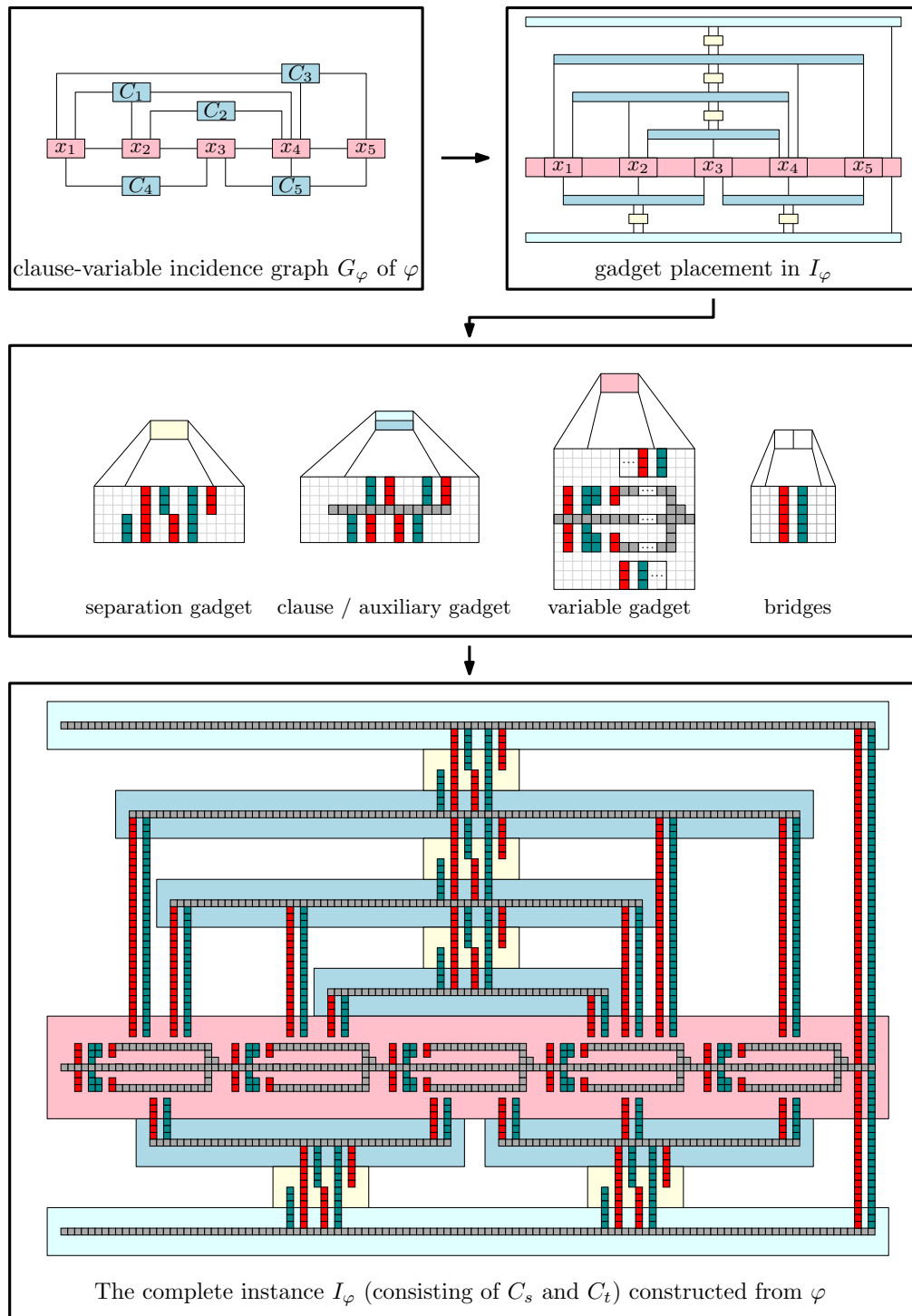
Now a stable schedule for  $I_\varphi$  transforming the start configuration  $C_s$  into the target configuration  $C_t$  with a makespan of 2 requires some robots of the variable gadget to move in a very particular way to ensure connectivity between the variable gadget and the clause gadgets via corresponding bridges. As shown in technical detail in the full paper, this allows variable robots to connect either to the negated or the unnegated literal of involved clauses, inducing a satisfying variable assignment for  $\varphi$ .

Technical details of the proof of Theorem 2 are given in the full version [11].

As a consequence of Theorem 2, even approximating the makespan is NP-hard.

► **Corollary 3.** *It is NP-hard to compute for a pair of configurations  $C_s$  and  $C_t$ , each with  $n$  vertices, a stable schedule that transforms  $C_s$  into  $C_t$  within a constant of  $(\frac{3}{2} - \varepsilon)$  (for any  $\varepsilon > 0$ ) of the minimum makespan.*

9:6 Connected Coordinated Motion Planning with Bounded Stretch



■ **Figure 3** Symbolic overview of the NP-hardness reduction. The depicted instance is due to the PLANAR MONOTONE 3SAT formula  $\varphi = (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_3 \vee x_4 \vee x_5)$ . We use three different colors to indicate occupied positions in the start configuration (red), in the target configuration (dark cyan), and in both configurations (gray).



## 4 Bounded Stretch for Arbitrary Makespan

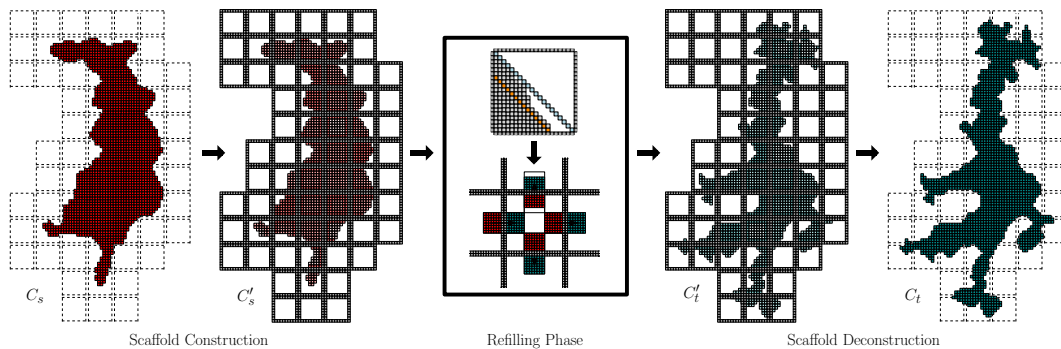
Now we describe our algorithm for computing stable schedules with constant stretch.

► **Theorem 4.** *There is a constant  $c^*$  such that for any pair of overlapping start and target configurations with a scale of at least  $c^*$ , there is a stable schedule of constant stretch.*

For clearer presentation, we do not focus on the specific value of the constant  $c^*$ , but only argue its existence.

### 4.1 Algorithm Overview and Preliminaries

#### 4.1.1 Informal Outline



■ **Figure 4** Overview of the computed schedule: (Left) Constructing the scaffold of  $cd$ -tiles, (middle) the refilling phase, and (right) deconstructing the scaffold.

See Figure 4 for an overview. In two preprocessing phases, we first ensure that the pair  $(C_s, C_t)$  overlaps in at least one position. For this, we move  $C_s$  towards  $C_t$  along a bottleneck matching such that the respective positions that realize the bottleneck distance, coincide. The overlap is necessary to successfully construct the auxiliary structure in the third phase of our approach. Afterwards, we use another bottleneck matching algorithm for mapping the start configuration  $C_s$  to the target configuration  $C_t$ , minimizing the maximum distance  $d$  between a start and a target location. Furthermore, we establish the scale in both configurations, set  $c$  to be the minimum of both scale values, and compute a suitable tiling whose tile size is  $c \cdot d$ , and that contain both  $C_s$  and  $C_t$ .

In a third phase, we build a scaffolding structure around  $C_s$  and  $C_t$ , based on the boundaries of  $cd$ -tiles of the specific tiling, see Figures 4 (left) and 5. This provides connectivity throughout the actual reconfiguration. Restricting robot motion to their current and adjacent tiles also ensures constant stretch. Note that, as the size of the tiles is related to  $d$ , the scaffolding structure is connected.

In a fourth phase, we perform the actual reconfiguration of the arrangement. This consists of refilling the tiles of the scaffold structure, achieving the proper number of robots within each tile, based on elementary flow computations. As a subroutine, we transform the robots inside each tile into a canonical “triangle” configuration, see Figures 4 (middle), 6, and 7.

In a fifth and final phase, we disassemble the scaffolding structure and move the involved robots to their proper destinations, see Figures 4 (right) and 5.

### 4.1.2 Technical Key Components

On a technical level, the five phases can be summarized as follows; again, refer to Figure 4.

- (1) **Guaranteeing Overlap:** Move the configurations towards each other along a bottleneck matching to ensure that the pair  $(C_s, C_t)$  overlaps in at least one position.
- (2) **Preprocessing:** Apply the following three preprocessing steps: (2.1) Set  $c$  to be the minimum of  $c^*$  and the minimum scale values of  $C_s$  and  $C_t$ . (2.2) Compute the diameter  $d$  of  $(C_s, C_t)$ . (2.3) Compute the tiling  $\mathcal{T}$  of  $(C_s, C_t)$ .

The algorithmic core of our algorithm consists of the following three phases.

- (3) **Scaffold Construction:** Reconfigure the start configuration  $C_s$  to a tiled configuration  $C'_s$  such that the interior of  $C'_s$  is a subset of the start configuration  $C_s$ , see Figure 5.
- (4) **Refilling Tiles:** Reconfigure  $C'_s$  to a tiled configuration  $C'_t$ , such that the interior of  $C'_t$  is a subset of the target configuration  $C_t$ , see Figures 6 and 7.
- (5) **Scaffold Deconstruction:** Reconfigure  $C'_t$  to  $C_t$ , see Figure 5.

Note that the scaffold deconstruction is inverse to the scaffold construction.

### 4.1.3 Preliminaries for the Algorithm

Let  $c, d \in \mathbb{N}$  be the scale and the diameter of the pair  $(C_s, C_t)$ , respectively. For  $x, y \in \mathbb{N}$ , a  $cd$ -tile  $T$ , or *tile*  $T$  for short, with *anchor vertex*  $(x \cdot cd, y \cdot cd) \in V(G)$  is a set of  $(cd)^2$  vertices from the grid  $G$  with  $x$ -coordinates from the range between  $x \cdot cd$  and  $x \cdot cd + cd - 1$  and  $y$ -coordinates from the range between  $y \cdot cd$  and  $y \cdot cd + cd - 1$ . The *boundary* of  $T$  is the set of vertices from  $T$  with an  $x$ -coordinate equal to  $x \cdot cd$  or equal to  $x \cdot cd + cd - 1$ , or with a  $y$ -coordinate equal to  $y \cdot cd$  or equal to  $y \cdot cd + cd - 1$ . The *interior* of  $T$  is  $T$  without its boundary. The *right, top, left, and bottom sides* of  $T$  are the sets of vertices from the boundary of  $T$  with maximum  $x$ -coordinates, maximum  $y$ -coordinates, minimum  $x$ -coordinates, and minimum  $y$ -coordinates, respectively. The left and right sides of a tile are *vertical sides* and the top and bottom sides are *horizontal sides*. Two tiles  $T_1, T_2$  are *horizontal (vertical) neighbors* if they have two vertical (horizontal) sides  $s_1 \subset T_1$  and  $s_2 \subset T_2$ , such that each vertex from  $s_1$  is adjacent in  $G$  to a vertex from  $s_2$ . Two tiles  $T_1$  and  $T_2$  are *diagonal neighbors* if there is another tile  $T$ , such that  $T$  and  $T_1$  are horizontal neighbors and  $T$  and  $T_2$  are vertical neighbors. The *neighborhood* of a tile  $T$  is the set of all neighbors of  $T$ . A configuration in the interior of a tile  $T$  is called *monotone*, if and only if for every robot  $r$  in the interior of  $T$  all positions to the left and to the bottom are occupied.

A *start tile* is a tile containing a vertex from the start configuration. A *target tile* is a tile containing a vertex from the target configuration. The *cd-tiling*  $\mathcal{T}$  of  $(C_s, C_t)$  is the union of all start tiles including their neighborhoods and all target tiles. The *scaffold* of  $\mathcal{T}$  is the union of all boundaries of tiles from  $\mathcal{T}$ . A *cd-tiled configuration*  $C$ , or *tiled configuration*  $C$  for short, is a configuration that is a subset of  $\mathcal{T}$  and a superset of the scaffold of  $\mathcal{T}$ . The *interior* of a tiled configuration  $C$  is the set of all vertices from  $C$  not lying on the scaffold. The *filling level* of a tile  $T \in \mathcal{T}$  is the number of robots in the interior of  $T$ . The *filling level* of a tiled configuration  $C$  is the mapping of each tile onto its filling level in  $C$ .

In the following we give the technical description of our algorithm and the corresponding correctness analysis. In particular, we first assume that the start and target configurations overlap in at least one position, resulting in an algorithm guaranteeing constant stretch, and adapt this to the case in that an overlap initially does not exist, afterwards.

## 4.2 Scaffold Construction

► **Lemma 5.** *For any configuration  $C_s$  of scale  $c$  there is a stable schedule of makespan  $\mathcal{O}(d)$ , transforming  $C_s$  into a tiled configuration  $C'_s$ , with the interior of  $C'_s$  being a subset of  $C_s$ .*

**Outline of the Construction.** For the construction we consider  $5 \cdot 5$  different classes, based on  $x$ - and  $y$ -coordinates modulo  $5cd$ ; see Figure 5. We process a single class as follows. For each tile  $T$  we consider its *indirect* neighborhood  $N[T]$  consisting of all neighbors of  $T$  and all neighbors of neighbors of  $T$ , i.e., a  $5 \times 5$  arrangement of tiles centered at  $T$ . For constructing the boundary of  $T$ , we make use of robots from the interior of a tile in  $N[T]$ .

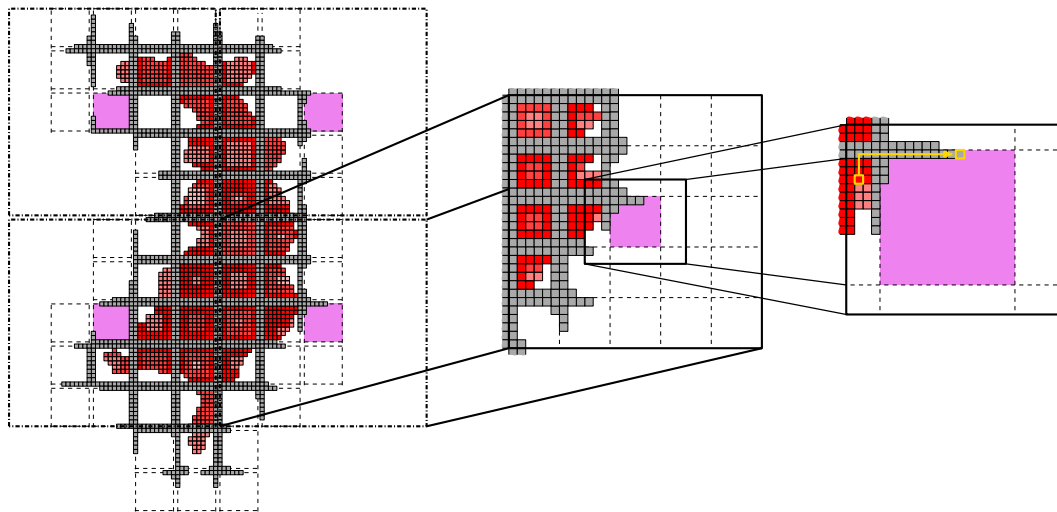
**Constructing the Boundary of  $T$  works in Two Phases.**

(3.1) Constructing the boundaries of all start tiles.

(3.2) Constructing the boundaries of all neighbors of start tiles.

Note that it suffices to construct all boundaries of the start tiles and their neighboring tiles, because each target tile shares a side with a start tile or a side with a tile adjacent to a start tile. Furthermore, the scale condition is only necessary for the construction of the scaffold, i.e., we need to ensure that enough robots are available to build the scaffolding structure. Each additional step of the algorithm works independently from this condition. A very rough estimate on the scale is that  $c^* = 400$  is sufficient.

For details of the construction and the proof, we refer to the full version [11].



■ **Figure 5** Constructing the scaffold. Tiles with currently constructed boundaries are marked in purple. The zoom into the start configuration  $C_s$  shows the indirect neighborhood  $N[T]$  of a tile  $T$  (middle) for which its boundary is currently constructed and a further zoom into  $T$  with an associated robot motion (right). In each transformation step a robot from the interior of a tile  $T' \in N[T]$  is swapped with a free position on the boundary of  $T$  based on a path  $P$  on a BFS-tree.

## 4.3 Refilling Tiles

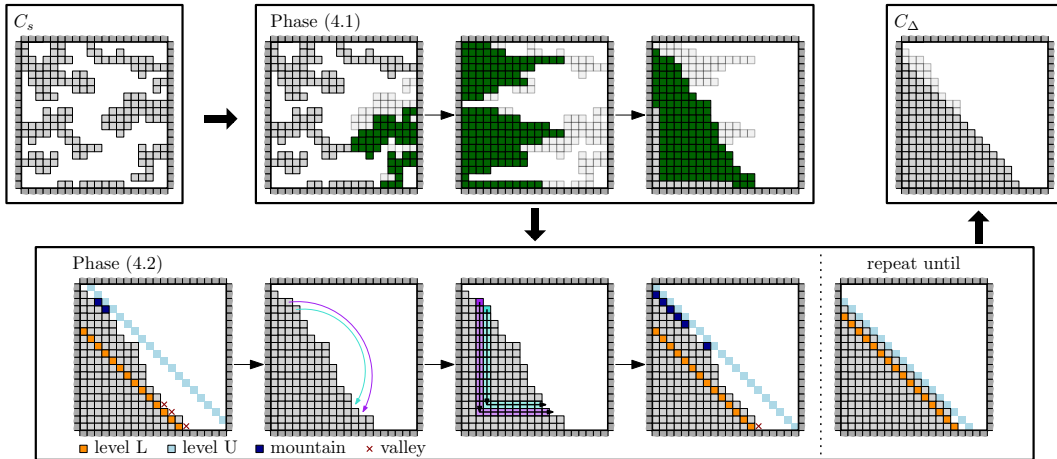
It remains to modify configurations within and between tiles. To this end, we first establish how to efficiently perform reconfigurations between any two tiled configurations *with the same numbers of robots* in the interior of respective tiles; see Section 4.3.1. As a second step, we describe how to relocate robots between tiles such that efficient reconfigurations between any two tiled configurations *with different numbers of robots* in the interior of respective tiles are achieved; see Section 4.3.2.

### 4.3.1 Reconfiguration Maintaining the Number of Robots inside Tiles

► **Lemma 6.** *Let  $C'_s, C'_t$  be two tiled configurations such that for all tiles  $T$ ,  $C'_s$  and  $C'_t$  have the same filling levels, i.e., for any tile, the corresponding start and target configurations consist of the same respective numbers of robots. Then there is a stable schedule transforming  $C'_s$  into  $C'_t$  within a makespan of  $\mathcal{O}(d)$ .*

In the following we describe reconfigurations that leave all robot movements within the interior of their respective tiles  $T$ ; thus, all tiles can be reconfigured in parallel. Therefore, we only have to describe the approach for a start configuration  $C_s$  and a target configuration  $C_t$  within the interior of a single tile  $T$  of a tiled configuration  $C'_s$ .

**Outline of the Reconfiguration.** First compute two stable schedules  $C_s \Rightarrow_\chi C_s^m$  and  $C_t \Rightarrow_\chi C_t^m$ , where  $C_s^m$  and  $C_t^m$  are monotone configurations. These reconfigurations are achieved by a sequence of down and left movements, maintaining connectivity after each move (see Figure 6 (Phase 4.1)). Proceeding from these monotone configurations, the robots are arranged into a triangular configuration  $C_\Delta$  that occupies the lower left positions (defined by a diagonal line with a slope of  $-1$ ) of the interior of  $T$ . This is achieved by swapping pairs of occupied and empty positions within a carefully defined area in several one-step moves along L-shaped paths (see Figure 6 (Phase 4.2)). The property of  $C_\Delta$  is that it is the same for all initial configurations with equally many robots. Thus, to get the stable schedule  $C_s \Rightarrow_\chi C_\Delta \Rightarrow_\chi C_t$ , we can simply revert  $C_t \Rightarrow_\chi C_\Delta$  and combine the result with  $C_s \Rightarrow_\chi C_\Delta$ .



■ **Figure 6** Turning arrangement  $C_s$  (top) into the canonical triangle configuration  $C_\Delta$  (bottom). Phase (4.1) achieves a monotonic arrangement; light gray indicates previous positions of moved robots (shown in green). Phase (4.2) transforms the monotonic configuration into  $C_\Delta$ .

Technically, the approach consists of the following four phases, see Figure 6.

- (4.1) **Monotone Start Configuration:** Reconfigure  $C_s$  into  $C_s^m$ .
- (4.2) **Canonical Triangle:** Reconfigure  $C_s^m$  into  $C_\Delta$ .
- (4.3) **Monotone Target Configuration:** Reconfigure  $C_\Delta$  into  $C_t^m$ .
- (4.4) **Target Configuration:** Reconfigure  $C_t^m$  into  $C_t$ .

Phase (4.4) corresponds to a reversal of Phase (4.1), and Phase (4.3) to one of Phase (4.2), so we only have to describe the first two phases. We analyze them individually, leading to a proof of Lemma 6. Note that we exclude the corners of a tile, so the robots on the tile’s side now form four *non-adjacent* sides. Furthermore, only robots in a tile’s interior move.

**Constructing the Monotone Start Configuration (Phase (4.1)).** In the first step, we only consider robots for which the right side of their tile  $T$  is the only one to which they are connected through the interior of  $T$ . We iteratively move these robots down until further movement is blocked, i.e., any further down move is not collision-free. In the second and third steps, we move all robots left, followed by moving all robots down, each time until further movement is blocked.

For Phase (4.2), we use the following terminology. The *level* of a position in the tile's interior is the sum of its coordinates. A level is *filled*, if all of its positions are occupied by a robot, and *empty*, if none is occupied by a robot. The highest filled level is denoted by  $L$ , the lowest empty level by  $U$ . Let  $\mathcal{M}$  be the set of all positions on level  $U - 1$  occupied by a robot, and  $\mathcal{V}$  be the set of all positions on level  $L + 1$  that are not occupied by a robot (see Figure 6 (Phase 4.2 left)); we call the positions of  $\mathcal{M}$  and  $\mathcal{V}$  *mountains* and *valleys*, respectively.

**Constructing the Canonical Triangle (Phase (4.2)).** Choose two equally sized subsets  $\mathcal{M}' \subseteq \mathcal{M}$  and  $\mathcal{V}' \subseteq \mathcal{V}$  and push each robot from  $\mathcal{M}'$  to a different position in  $\mathcal{V}'$  along an L-shaped path; this can be done simultaneously in one parallel move for all paths. To determine the paths, simply match mountains and valleys, iteratively, in a way that no pair of paths cross each other. This results in reducing  $U$ , and raising  $L$ , i.e., the two levels move towards each other. We distinguish two cases.

- $U - L > 2$ : If  $|\mathcal{M}| \geq |\mathcal{V}|$ , choose an arbitrary subset  $\mathcal{M}' \subset \mathcal{M}$  with  $|\mathcal{M}'| = |\mathcal{V}|$ . Otherwise, choose an arbitrary subset  $\mathcal{V}' \subset \mathcal{V}$  with  $|\mathcal{V}'| = |\mathcal{M}|$ .
- $U - L = 2$ : Note that mountains and valleys are on the same level, and  $|\mathcal{V}| \geq |\mathcal{M}|$  hold. Choose  $\mathcal{V}' \subset \mathcal{V}$  to be the subset of size  $|\mathcal{M}|$  with *smallest*  $x$ -coordinates and set  $\mathcal{M}' = \mathcal{M}$ .

Due to space constraints, the details can be found in the full version [11].

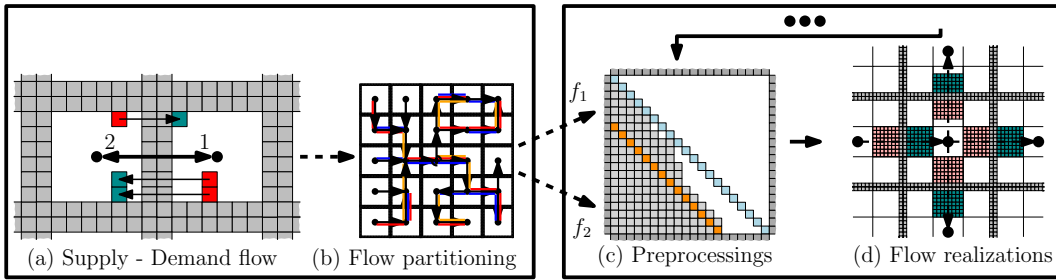
### 4.3.2 Refilling Tiled Configurations

Now we describe the final step for reconfiguring a tiled start configuration  $C'_s$  into a tiled target configuration  $C'_t$ ; because no robots are destroyed or created, this hinges on shifting robots between adjacent tiles, such that the required filling levels are achieved.

► **Lemma 7.** *We can efficiently compute a stable schedule transforming  $C'_s$  into  $C'_t$  within a makespan of  $\mathcal{O}(d)$ .*

**Outline of the Refilling Phase.** To compute the schedule of Lemma 7, we transfer robots between tiles, so that each tile  $T$  contains the desired number in  $C'_t$ . We model this robot transfer by a *supply and demand flow*, see Figure 7, followed by partitioning the flow into  $\mathcal{O}(1)$  subflows, such that each subflow can be realized within a makespan of  $\mathcal{O}(d)$ . For realizing a single subflow, we use the approach of Section 4.3.1 as a preprocessing step, i.e., to rearrange robots participating in a specific subflow and place them at suitable positions.

**Modeling Transfer of Robots via a Supply and Demand Flow.** We model the transfer of robots between tiles as a flow  $F : E(G) \rightarrow \mathbb{N}$ , using the directed graph  $G = (\mathcal{T}, E)$  which is dual to the tiling  $\mathcal{T}$ . Let  $B$  be the bottleneck matching between vertices from the original (non-tiled)  $C_s$  and vertices from the final (non-tiled)  $C_t$ . In  $G$  we have an edge  $(u, v) \in E$ , if there is at least one matching edge  $(r_u, r_v) \in B$ , such that  $r_u$  lies in the interior of the tile  $u$  in configuration  $C'_s$ , and  $r_v$  lies in the interior of the tile  $v$  in configuration  $C'_t$ . The flow value  $F((u, v))$  of  $(u, v)$  is equal to the number of such edges  $(r_u, r_v) \in B$ . A vertex  $v \in V(G)$



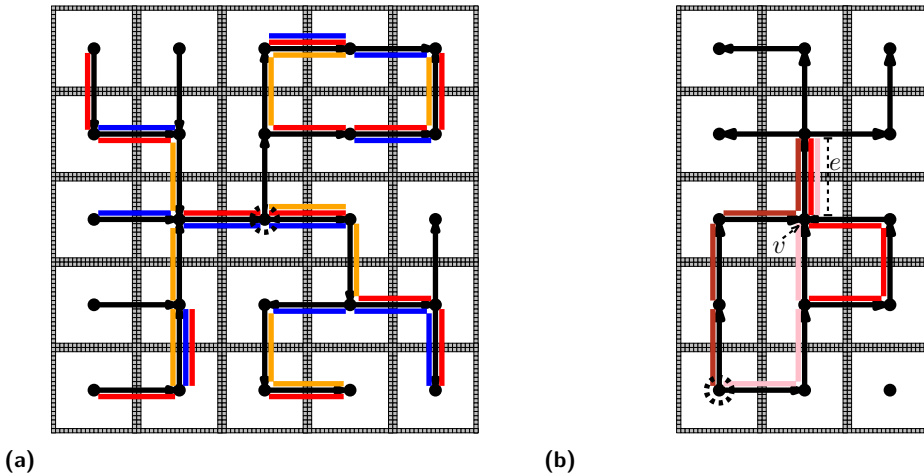
■ **Figure 7** An overview of the schedule refilling tiles: transforming  $C'_s$  into  $C'_t$  by realizing a partition of a supply and demand flow that is computed in advance.

has a *demand* of  $a > 0$  if the sum of the flow values of outgoing edges from  $v$  plus  $a$  is equal to the sum of the flows of incoming edges to  $v$ . Analogously,  $v$  has a *supply* of  $a > 0$  if the sum of flow values incoming to  $v$  plus  $a$  equals the sum of flow values outgoing from  $v$ .

**Flow Partition and Algorithmic Computation.** Now we define a *flow partition* of  $F$ .

► **Definition 8.** For  $k \in \mathbb{N}$ , a  $k$ -subflow of  $F$  is a supply and demand flow  $f : E(G) \rightarrow \mathbb{N}$  on  $G$  with  $f(e) \leq \min\{k, F(e)\}$ . A  $k$ -partition of the flow  $F$  is a set  $\{f_1, \dots, f_\ell\}$  of  $k$ -subflows of  $F$ , such that  $\sum_{i=1, \dots, \ell} f_i(e) = F(e)$  holds for all edges  $e \in E(G)$ .

We describe our approach for computing a  $\phi$ -partition of  $F$ , with  $\phi := \lfloor \frac{(cd-2)^2}{9} \rfloor$ ; the value  $\phi$  arises from partitioning the interior (made up of  $(cd - 2)^2$  pixels) of each tile into 9 almost equally sized *subtiles* that are used for realizing a single set of paths as described below, see Figure 7(d).



■ **Figure 8** (a) We model the movements of robots between tiles as paths forming a tree. By greedily assigning these paths to sets (here highlighted by different colors), such that inside each set each edge is contained in no more than 3 paths, we obtain that at most  $\Theta(d^2)$  sets are needed. (b) An example of a set containing three paths (dark red, pink, and red; assigned in that order to  $S_j$ ) having a common edge  $e$  caused by a vertex  $v$  of  $e$  with an incoming degree of 3.

We compute a 1-partition of  $G$ , with each 1-subflow being either a cycle or a path that connects a supply vertex with a demand vertex. Because the robots are unlabeled, we can simplify  $G$  by eliminating all cyclic 1-subflows, as they are not necessary to realize this



specific transfer of robots; note that this also applies to bidirectional edges. Furthermore, we replace diagonal edges  $(v, w) \in E(G)$  by a pair of adjacent edges  $(v, u), (u, w) \in E(G)$ . After all cyclic subflows are removed,  $G$  is a planar, directed forest consisting of 1-subflows that are paths, see Figure 8(a). We process each tree  $A \subset G$  that is made up of paths  $P_1, P_2, \dots$  separately as follows: We choose an arbitrary vertex of  $A$  as its *root* and consider the *link distance* of  $P_i$  as the minimal length of the path between a vertex from  $P_i$  and the root of  $A$ . Let  $P_1, P_2, \dots \subseteq G$  be sorted by increasing link distances, which is important for our next argument, regarding that we can partition these paths into constant many subflows each of which is realizable in time linear in  $d$ . We greedily assign each path  $P_i = P_1, P_2, \dots$  to a set  $S_j$ , such that the first edge  $e_1$  of  $P_i$  is not part of another path inside  $S_j$ . If no such a set  $S_j$  exists, we create a new set  $S \leftarrow \{P_i\}$ . For each tree we use the same sets  $S_1, S_2, \dots$  of collected paths, because different trees of  $G$  are disjoint. Note, that the construction of the sets  $S_j$  allow that an edge is part of at most three paths inside  $S_j$ . This is due to the fact that the income degree of a head vertex of a directed edge is at most three in the setting of a grid graph, resulting in at most three outgoing edges.

Finally, we greedily partition  $\{S_1, S_2, \dots\}$  into subsets  $G_1, G_2, \dots$  called *groups*, made up of  $\frac{(cd-2)^2}{9 \cdot 3}$  sets. For each group  $G_i$ , we define a subflow  $f_i$  by setting  $f_i(e)$  as the number of paths from  $G_i$  containing the edge  $e$ . As for each set  $S_i$  and each edge  $e$ , there are at most three paths inside  $S_i$  containing  $e$ , each resulting subflow  $f_i$  is a  $\left(\frac{(cd-2)^2}{9 \cdot 3} \cdot 3\right) = \phi$ -subflow. Finally, we have to upper-bound the number of resulting subflows, i.e., the number of groups.

► **Lemma 9.** *The constructed  $\phi$ -partition  $\{f_1, f_2, \dots\}$  consists of at most 28 subflows.*

**Realizing a  $\phi$ -Partition.** Now we describe how to reconfigure a tiled configuration, such that a  $\phi$ -subflow is removed from  $G$ .

► **Definition 10.** *A  $\phi$ -subflow  $f_i$  is realized by transforming the current configuration into another configuration, such that for each edge  $e$  with  $f_i((T, T')) > 0$ , the number of robots in the interior of tile  $T$  is decreased by  $f_i((T, T'))$  and the number of the robots in the interior of tile  $T'$  is increased by  $f_i((T, T'))$ .*

Next we realize a specific  $\phi$ -subflow within a makespan of  $\mathcal{O}(d)$ . In particular, we partition the interior of each tile  $T$  into 9 *subtiles* with equal side lengths (up to rounding), see Figure 7(d). For each subflow  $f_i$ , we place  $f_i((T, T'))$  robots inside the middle subtile of  $T$  that shares an edge with the boundary of  $T$  adjacent to  $T'$ . In particular, robots placed in the same subtile are arranged in layers of width  $\lfloor \frac{cd-2}{9} \rfloor$  as close as possible to the boundary of the tile, see Figure 7(d). The resulting arrangement of robots inside the subtile of  $T$  is a *cluster* and  $T'$  the *target tile* of the cluster. By a single application of the approach from Section 4.3.1, all clusters of all tiles are arranged simultaneously within a makespan of  $\mathcal{O}(d)$ . Finally, simultaneously pushing all clusters of all tiles into the direction of their target tiles realizes  $S_i$ , see Figure 7(d). Note that not all robots are pushed into the target tile  $T'$  but some replace robots on the boundaries between  $T$  and  $T'$ , see Figure 7(d).

Repeating this approach for each subflow  $f_i$  leads to a stable schedule that realizes the entire flow  $F$  within a makespan linear in  $d$ , i.e., transforms  $C'_s$  into  $C'_t$  within  $\mathcal{O}(d)$  transformation steps, see Figure 7.

The omitted proofs can be found in the full version [11].

This concludes the proof of Theorem 4. Finally, we adapt the result to the general case, for which overlap of the start and target configurations is not guaranteed.

► **Corollary 11.** *There is a constant  $c^*$  such that for any pair of start and target configurations with a scale of at least  $c^*$ , there is a stable schedule of constant stretch.*

**Proof.** In case of a pair  $(C_s, C_t)$ , consisting of a start and a target configuration, that does not overlap, our algorithm computes in a first step a minimum bottleneck matching mapping  $C_s$  to  $C_t$  resulting in a bottleneck distance  $\bar{d}$ , and translates  $C_s$  into a configuration  $\bar{C}_s$  overlapping the target configuration within a makespan of  $\bar{d}$ . This results in a bottleneck distance  $d$  between  $\bar{C}_s$  and  $C_t$  which is at most  $2\bar{d}$ . As Theorem 4 guarantees a makespan linear in  $d$ , we obtain a makespan linear in  $\bar{d} + d$ , i.e., linear in  $\bar{d}$  for the overall algorithm. ◀

As the diameter of the pair  $(C_s, C_t)$  is a lower bound for the makespan of any schedule transforming  $C_s$  into  $C_t$ , we obtain the following.

► **Corollary 12.** *There is a constant-factor approximation for computing stable schedules with minimal makespan between pairs of start and target configurations with a scale of at least  $c^*$ , for some constant  $c^*$ .*

## 5 Conclusion

We have shown that connected coordinated motion planning is challenging even in relatively simple cases, such as unlabeled robots that have to travel a distance of at most 2 units. On the other hand, we have shown that (assuming sufficient scale of the swarm), it is possible to compute efficient reconfiguration schedules with constant stretch.

It is straightforward to extend our approach to other scenarios, e.g., to three-dimensional configurations. Other questions appear less clear. Is it possible to achieve constant stretch for arrangements with very small scale factor? We believe that this may hinge on the ability to perform synchronized shifts on long-distance “chains” of robots without delay, which is not a valid assumption for many real-world scenarios. (A well-known example is a line of cars when a traffic light turns green.) As a consequence, the answer may depend on crucial assumptions on motion control; we avoid this issue in our approach. Can we provide alternative approaches with either weaker scale assumptions or better stretch factors? Can we extend our methods to the labeled case? All these questions are left for future work.

---

## References

- 1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient Multi-Robot Motion Planning for Unlabeled Discs in Simple Polygons. *IEEE Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015. doi:10.1109/TASE.2015.2470096.
- 2 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. Coordinated Motion Planning: The Video. In *Symposium on Computational Geometry (SoCG)*, pages 74:1–74:6, 2018. Video at <https://www.ibr.cs.tu-bs.de/users/fekete/Videos/CoordinatedMotionPlanning.mp4>. doi:10.4230/LIPIcs.SoCG.2018.74.
- 3 Soon-Jo Chung, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar. A Survey on Aerial Swarm Robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018. doi:10.1109/TR0.2018.2857475.
- 4 Mark de Berg and Amirali Khosravi. Optimal Binary Space Partitions for Segments in the Plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012. doi:10.1142/S0218195912500045.
- 5 Daniel Delahaye, Stéphane Puechmorel, Panagiotis Tsiotras, and Eric Féron. Mathematical Models for Aircraft Trajectory Design: A Survey. In *Air Traffic Management and Systems*, pages 205–247, 2014. doi:10.1007/978-4-431-54475-3\_12.
- 6 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane Souvaine. Staged Self-assembly: Nanomanufacture of Arbitrary Shapes with  $O(1)$  Glues. *Natural Computing*, 7(3):347–370, 2008. doi:10.1007/s11047-008-9073-0.



- 7 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.
- 8 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New Geometric Algorithms for Fully Connected Staged Self-Assembly. *Theoretical Computer Science*, 671:4–18, 2017. doi:10.1016/j.tcs.2016.11.020.
- 9 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-Assembly of Arbitrary Shapes Using RNase Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor. In *Symposium on Theoretical Aspects of Computer Science*, volume 9, pages 201–212, 2011. doi:10.4230/LIPIcs.STACS.2011.201.
- 10 Sándor P. Fekete, Björn Hendriks, Christopher Tessars, Axel Wegener, Horst Hellbrück, Stefan Fischer, and Sebastian Ebers. Methods for Improving the Flow of Traffic. In *Organic Computing — A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*. Springer, 2011. doi:10.1007/978-3-0348-0130-0\_29.
- 11 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected Coordinated Motion Planning with Bounded Stretch, 2021. arXiv:2109.12381.
- 12 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing Coordinated Motion Plans for Robot Swarms: The CG:SHOP Challenge 2021, 2021. arXiv:2103.15381.
- 13 Seth C. Goldstein and Todd C. Mowry. Claytronics: A scalable basis for future robots. In *Robosphere 2004*, 2004. URL: <http://www.cs.cmu.edu/~claytronics/papers/goldstein-robosphere04.pdf>.
- 14 John E. Hopcroft and Richard M. Karp. An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.
- 15 Stephen Kloder and Seth Hutchinson. Path planning for permutation-invariant multi-robot formations. *IEEE Transactions on Robotics and Automation*, 22(4):650–665, 2006. doi:10.1109/TR0.2006.878952.
- 16 Austin Luchsinger, Robert T. Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, 18(1):93–105, 2019. doi:10.1007/s11047-018-9707-9.
- 17 Florian Pescher, Nils Napp, Benoît Piranda, and Julien Bourgeois. GAPCoD: A Generic Assembly Planner by Constrained Disassembly. In *Autonomous Agents and MultiAgent Systems*, pages 1028–1036, 2020. URL: <https://dl.acm.org/doi/abs/10.5555/3398761.3398881>.
- 18 Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014. doi:10.1126/science.1254295.
- 19 Erol Şahin and Alan F. T. Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2):69–72, 2008. doi:10.1007/s11721-008-0020-6.
- 20 Michael Schreckenberg and Reinhard Selten. *Human Behaviour and Traffic Networks*. Springer, 2013. doi:10.1007/978-3-662-07809-9.
- 21 Jacob T. Schwartz and Micha Sharir. On the piano movers’ problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.
- 22 David Soloveichik and Erik Winfree. Complexity of Self-Assembled Shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. doi:10.1137/S0097539704446712.
- 23 Kiril Solovey and Dan Halperin.  $k$ -Color Multi-Robot Motion Planning. *International Journal of Robotics Research*, 33(1):82–97, 2014. doi:10.1177/0278364913506268.
- 24 Kiril Solovey and Dan Halperin. On the Hardness of Unlabeled Multi-Robot Motion Planning. *International Journal of Robotics Research*, 35(14):1750–1759, 2016. doi:10.1177/0278364916672311.

## 9:16 Connected Coordinated Motion Planning with Bounded Stretch

- 25 Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion Planning for Unlabeled Discs with Optimality Guarantees. In *Robotics: Science and Systems*, 2015. doi:10.15607/RSS.2015.XI.011.
- 26 Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure. In *Autonomous Agents and MultiAgent Systems*, pages 140–148, 2019. URL: <https://dl.acm.org/doi/abs/10.5555/3306127.3331685>.
- 27 Matthew Turpin, Nathan Michael, and Vijay Kumar. Trajectory planning and assignment in multirobot systems. In *Algorithmic Foundations of Robotics X*, pages 175–190. Springer, 2013. doi:10.1007/978-3-642-36279-8\_11.
- 28 Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*, 37(4):401–415, 2014. doi:10.1007/s10514-014-9412-1.

# Enclosing Depth and Other Depth Measures

Patrick Schneider   

Department of Mathematical Sciences, University of Copenhagen, Denmark

---

## Abstract

We study families of depth measures defined by natural sets of axioms. We show that any such depth measure is a constant factor approximation of Tukey depth. We further investigate the dimensions of depth regions, showing that the *Cascade conjecture*, introduced by Kalai for Tverberg depth, holds for all depth measures which satisfy our most restrictive set of axioms, which includes Tukey depth. Along the way, we introduce and study a new depth measure called *enclosing depth*, which we believe to be of independent interest, and show its relation to a constant-fraction Radon theorem on certain two-colored point sets.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** Depth measures, Tukey depth, Tverberg theorem, Combinatorial Geometry

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.10

**Related Version** *Full Version:* <https://arxiv.org/abs/2103.08421>

**Funding** The author has received funding from the European Research Council under the European Unions Seventh Framework Programme ERC Grant agreement ERC StG 716424 – CAsE.

**Acknowledgements** Thanks to Emo Welzl, Karim Adiprasito and Uli Wagner for the helpful discussions.

## 1 Introduction

Medians are an important tool in the statistical analysis and visualization of data. Due to the fact that medians only depend on the order of the data points, and not their exact positions, they are very robust against outliers. However, in many applications, data sets are multidimensional, and there is no clear order of the data set. For this reason, various generalizations of medians to higher dimensions have been introduced and studied, see e.g. [1, 17, 21] for surveys. Many of these generalized medians rely on a notion of *depth* of a query point within a data set, a median then being a query point with the highest depth among all possible query points. Several such depth measures have been introduced over time, most famously Tukey depth [28] (also called halfspace depth), simplicial depth [16], or convex hull peeling depth (see, e.g., [1]). In particular, just like the median, all of these depth measures only depend on the relative positions of the involved points. More formally, let  $S^{\mathbb{R}^d}$  denote the family of all finite sets of points in  $\mathbb{R}^d$ . A depth measure is a function  $\varrho : (S^{\mathbb{R}^d}, \mathbb{R}^d) \rightarrow \mathbb{R}_{\geq 0}$  which assigns to each pair  $(S, q)$  consisting of a finite set of data points  $S$  and a query point  $q$  a value, which describes how deep the query point  $q$  lies within the data set  $S$ . A depth measure  $\varrho$  is called *combinatorial* if it depends only on the order type of  $S \cup \{q\}$ , that is, if it only depends on the orientations of the simplices spanned by the points, but not on their actual positions. In this paper, we consider general classes of combinatorial depth measures, defined by a small set of axioms, and prove relations between them and concrete depth measures, such as *Tukey depth* (TD) and *Tverberg depth* (TvD). Let us first briefly discuss these two depth measures.

► **Definition 1.** *Let  $S$  be a finite point set in  $\mathbb{R}^d$  and let  $q$  be a query point. Then the Tukey depth of  $q$  with respect to  $S$ , denoted by  $\mathbf{TD}(S, q)$ , is the minimum number of points of  $S$  in any closed half-space containing  $q$ .*



© Patrick Schneider;

licensed under Creative Commons License CC-BY 4.0

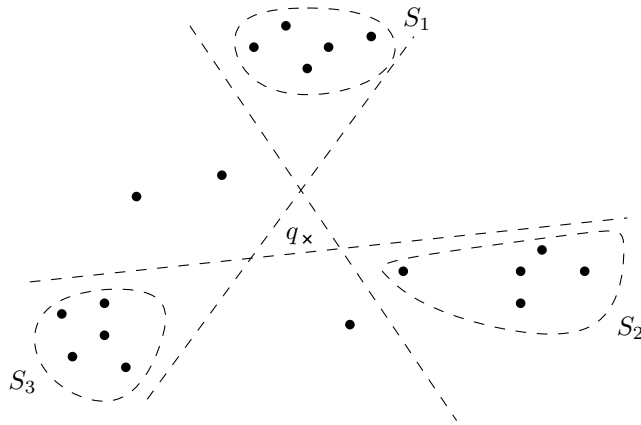
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 10; pp. 10:1–10:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** The point  $q$  has enclosing depth 5.

Tukey depth, also known as *halfspace depth*, was independently introduced by Joseph L. Hodges in 1955 [11] and by John W. Tukey in 1975 [28] and has received significant attention since, both from a combinatorial as well as from an algorithmic perspective, see e.g. Chapter 58 in [27] and the references therein. Notably, the *centerpoint theorem* states that for any point set  $S \subset \mathbb{R}^d$ , there exists a point  $q \in \mathbb{R}^d$  for which  $\text{TD}(S, q) \geq \frac{|S|}{d+1}$  [22].

In order to define Tverberg depth, we need a preliminary definition: given a point set  $S$  in  $\mathbb{R}^d$ , an  $r$ -partition of  $S$  is a partition of  $S$  into  $r$  pairwise disjoint subsets  $S_1, \dots, S_r \subset S$  with  $\bigcap_{i=1}^r \text{conv}(S_i) \neq \emptyset$ , where  $\text{conv}(S_i)$  denotes the convex hull of  $S_i$ . We call  $\bigcap_{i=1}^r \text{conv}(S_i)$  the *intersection* of the  $r$ -partition.

► **Definition 2.** Let  $S$  be a finite point set in  $\mathbb{R}^d$  and let  $q$  be a query point. Then the Tverberg depth of  $q$  with respect to  $S$ , denoted by  $\text{TvD}(S, q)$ , is the maximum  $r$  such that there is an  $r$ -partition of  $S$  whose intersection contains  $q$ .

Tverberg depth is named after Helge Tverberg who proved in 1966 that any set of  $(d + 1)(r - 1) + 1$  points in  $\mathbb{R}^d$  allows an  $r$ -partition [29]. In particular, this implies that there is a point  $q$  with  $\text{TvD}(S, q) \geq \frac{|S|}{d+1}$ . Just as for Tukey depth, there is an extensive body of work on Tverberg's theorem, see the survey [4] and the references therein.

In  $\mathbb{R}^1$ , both Tukey and Tverberg depth give a very natural depth measure: it counts the number of points of  $S$  to the left and to the right of  $q$  and then returns the minimum of the two numbers. We call this measure the *standard depth* in  $\mathbb{R}^1$ . In particular, for all of them there is always a point  $q \in \mathbb{R}^1$  for which we have  $\varrho(S, q) \geq \frac{|S|}{2}$ , that is, a median.

Another depth measure that is important in this paper is called enclosing depth. For an illustration of this depth measure, see Figure 1. We say that a point set  $S$  of size  $(d + 1)k$  in  $\mathbb{R}^d$   $k$ -encloses a point  $q$  if  $S$  can be partitioned into  $d + 1$  pairwise disjoint subsets  $S_1, \dots, S_{d+1}$ , each of size  $k$ , in such a way that for every transversal  $p_1 \in S_1, \dots, p_{d+1} \in S_{d+1}$ , the point  $q$  is in the convex hull of  $p_1, \dots, p_{d+1}$ . Intuitively, the points of  $S$  are centered around the vertices of a simplex with  $q$  in its interior.

► **Definition 3.** Let  $S$  be a finite point set in  $\mathbb{R}^d$  and let  $q$  be a query point. Then the enclosing depth of  $q$  with respect to  $S$ , denoted by  $\text{ED}(S, q)$ , is the maximum  $k$  such that there exists a subset of  $S$  which  $k$ -encloses  $q$ .

It is straightforward to see that enclosing depth also gives the standard depth in  $\mathbb{R}^1$ . The centerpoint theorem [22] and Tverberg's theorem [29] show that both for Tukey as well as Tverberg depth, there are deep points in any dimension. The question whether a depth

measure enforces deep points is a central question in the study of depth measures. We will show that this also holds for enclosing depth. In fact, we will show that enclosing depth can be bounded from below by a constant fraction of Tukey depth. We will further show that all depth measures considered in this paper can be bounded from below by enclosing depth. From this we get one of the main results of this paper: all depth measures that satisfy the axioms given later are a constant factor approximation of Tukey depth.

Another area of study in depth measures are *depth regions*, also called depth contours. For some depth measure  $\varrho$  and  $\alpha \in \mathbb{R}$ , we define the  $\alpha$ -region of a point set  $S \subset \mathbb{R}^d$  as the set of all points in  $\mathbb{R}^d$  that have depth at least  $\alpha$  with respect to  $S$ . We denote the  $\alpha$ -region of  $S$  by  $D_\varrho^S(\alpha) := \{q \in \mathbb{R}^d \mid \varrho(S, q) \geq \alpha\}$ . Note that for  $\alpha < \beta$  we have  $D_\varrho^S(\alpha) \supset D_\varrho^S(\beta)$ , that is, the depth regions are nested. The structure of depth regions has been studied for several depth measures, see e.g. [20, 32] In particular, depth regions in  $\mathbb{R}^2$  have been proposed as a tool for data visualization [28]. From a combinatorial point of view, Gil Kalai introduced the following conjecture [13]

► **Conjecture 4** (Cascade Conjecture). *Let  $S$  be a point set of size  $n$  in  $\mathbb{R}^d$ . For each  $i \in \{1, \dots, n\}$ , denote by  $d_i$  the dimension of  $D_{T_v D}^S(i)$ , where we set  $\dim(\emptyset) = -1$ . Then*

$$\sum_{i=1}^n d_i \geq 0.$$

The conjecture is known to be true when  $S$  is in so-called *strongly* general position [23], for general position in some dimensions [24, 25, 26] (see also [4] for more information), and without any assumption of general position for  $d \leq 2$  in an unpublished M. Sc thesis in Hebrew by Akiva Kadari (see [15]).

While Kalai's conjecture is specifically about Tverberg depth, the sum of dimensions of depth regions can be computed for any depth measure, and thus the conjecture can be generalized to other depth measures. In fact, in a talk Kalai conjectured that the Cascade conjecture is true for Tukey depth, mentioning on his slides that "this should be doable" [14]. In this work, we will prove the conjecture to be true for a family of depth measures that includes Tukey depth.

## Structure of the paper

We start the technical part by introducing a first set of axioms in Section 2, defining what we call *super-additive* depth measures. For these depth measures, we show that they lie between Tukey and Tverberg depth. In Section 3 we then prove the cascade conjecture for additive depth measures whose depth regions are convex. We then give a second set of axioms in Section 4, defining *central* depth measures, and show how to bound them from below by enclosing depth. Finally, in Section 5, we give a lower bound for enclosing depth in terms of Tukey depth. In order to prove this bound, we notice a close relationship of enclosing depth with a version of Radon's theorem on certain two-colored point sets.

## 2 A first set of axioms

The first set of depth measures that we consider are *super-additive* depth measures<sup>1</sup>. A combinatorial depth measure  $\varrho : (S^{\mathbb{R}^d}, \mathbb{R}^d) \rightarrow \mathbb{R}_{\geq 0}$  is called super-additive if it satisfies the following conditions:

<sup>1</sup> We name both our families of depth measures after one of the conditions they satisfy. The reason for this is that the condition they are named after is the condition which separates this family from the other one.

## 10:4 Enclosing Depth and Other Depth Measures

- (i) for all  $S \in S^{\mathbb{R}^d}$  and  $q, p \in \mathbb{R}^d$  we have  $|\varrho(S, q) - \varrho(S \cup \{p\}, q)| \leq 1$  (sensitivity),
- (ii) for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) = 0$  for  $q \notin \text{conv}(S)$  (locality),
- (iii) for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) \geq 1$  for  $q \in \text{conv}(S)$  (non-triviality),
- (iv) for any disjoint subsets  $S_1, S_2 \subseteq S$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) \geq \varrho(S_1, q) + \varrho(S_2, q)$  (super-additivity).

It is not hard to show that a one-dimensional depth measure which satisfies these conditions has to be the standard depth measure (in fact, the arguments are generalized to higher dimensions in the following two observations) and that no three conditions suffice for this. Further, it can be shown that both Tukey depth and Tverberg depth are super-additive.

We first note that the first two axioms suffice to give an upper bound:

► **Observation 5.** *For every depth measure  $\varrho$  satisfying (i) sensitivity and (ii) locality and for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) \leq \text{TD}(S, q)$ .*

**Proof.** By the definition of Tukey depth,  $\text{TD}(S, q) = k$  implies that we can remove a subset  $S'$  of  $k$  points from  $S$  so that  $q$  is not in the convex hull of  $S \setminus S'$ . In particular,  $\varrho(S \setminus S', q) = 0$  by locality. By sensitivity we further have  $\varrho(S \setminus S', q) \geq \varrho(S, q) - k$ , which implies the claim. ◀

Further, the last two axioms can be used to give a lower bound:

► **Observation 6.** *For every depth measure  $\varrho$  satisfying (iii) non-triviality and (iv) super-additivity and for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) \geq \text{TvD}(S, q)$ .*

**Proof.** Let  $\text{TvD}(S, q) = k$  and consider a  $k$ -partition  $S_1, \dots, S_k$  with  $q$  in its intersection. By non-triviality we have  $\varrho(S_i, q) \geq 1$  for each  $S_i$ . Using super-additivity and induction we conclude that  $\varrho(\bigcup_{i=1}^k S_i, q) \geq \sum_{i=1}^k \varrho(S_i, q) \geq k$ . ◀

Finally, it is not too hard to show that  $\text{TvD}(S, q) \geq \frac{1}{d} \text{TD}(S, q)$ , see e.g. [10] for an argument. Combining these observations, we thus get the following.

► **Corollary 7.** *Let  $\varrho$  be a super-additive depth measure. Then for every point set  $S$  and query point  $q$  in  $\mathbb{R}^d$  we have*

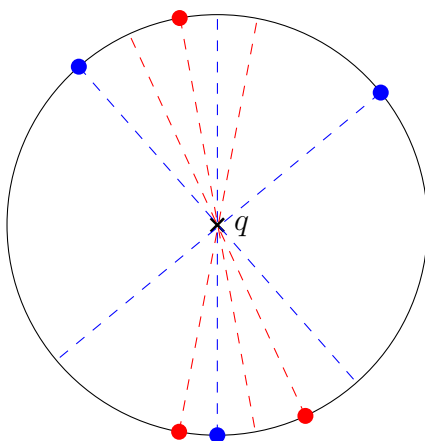
$$\text{TD}(S, q) \geq \varrho(S, q) \geq \text{TvD}(S, q) \geq \frac{1}{d} \text{TD}(S, q).$$

Let us note here that it could be that the factor  $\frac{1}{d}$  in the last inequality could be improved. Indeed, in the plane, we have that  $\text{TvD} = \min\{\text{TD}, \lceil \frac{|S|}{3} \rceil\}$  [23]. This fails already in dimension 3 [3]. It would be interesting to see how much the factor  $\frac{1}{d}$  can be improved.

From Corollary 7 it follows that for any super-additive depth measure and any point set there is always a point of depth at least  $\frac{|S|}{d+1}$ , for example any Tverberg point. On the other hand, there are depth measures which give the standard depth in  $\mathbb{R}^1$  which are not super-additive, for example convex hull peeling depth or enclosing depth.

► **Observation 8.** *Enclosing depth satisfies conditions (i)–(iii) and (v), but not the super-additivity condition (iv).*

**Proof.** It follows straight from the definition that enclosing depth satisfies the conditions (i)–(iii) and (v). To see that the super-additivity condition is not satisfied, consider the example in Figure 2. The point  $q$  has enclosing depth 1 with respect to both the set of blue points and the set of red points. However, it can be seen that the enclosing depth of  $q$  with respect to both the red and the blue points is still 1. ◀



■ **Figure 2** Enclosing depth does not satisfy the super-additivity condition: the point  $q$  has enclosing depth 1 with respect to both the blue and the red points, but its enclosing depth with respect to the union of the two sets is still 1.

### 3 The Cascade Conjecture

In this section we prove the cascade conjecture for super-additive depth measures whose depth regions are convex. In fact, we will prove the cascade conjecture for the case of *weighted point sets*. A weighted point set is a point set  $S$  together with a weight function  $w : S \rightarrow \mathbb{R}_{\geq 0}$  which assigns a weight  $w(p)$  to each  $p \in S$ . We say that a weighted point set  $S'$  is a strict subset of  $S$ , denoted by  $S' \subset S$ , if the underlying point set of  $S'$  is a strict subset of the underlying point set of  $S$ , and  $w'(p) \leq w(p)$  for every  $p \in S'$ , where  $w'$  is the weight function on  $S'$ . In particular, if  $S' \subset S$ , there is a point which is in  $S$  but not in  $S'$ . For two weighted point sets  $A$  and  $B$  with weight functions  $w_A$  and  $w_B$ , respectively, the weight function on their union  $A \cup B$  is defined as the sum of the respective weight functions. That is, we have  $w(p) = w_A(p)$  for  $p \in A \setminus B$ ,  $w(p) = w_B(p)$  for  $p \in B \setminus A$  and  $w(p) = w_A(p) + w_B(p)$  for  $p \in A \cap B$ . Further, for a set  $S$  of points we define the weight of  $S$  as  $w(S) := \sum_{p \in S} w(p)$ . Similarly, by a partition of a weighted point set  $S$  into parts  $A$  and  $B$  we mean two weight functions  $w_A$  and  $w_B$ , such that  $w(p) = w_A(p) + w_B(p)$  for  $p \in S$ , and by a partition into strict subsets  $A$  and  $B$ , we mean that both weighted point sets  $A$  and  $B$  must be strict subsets of  $S$ , that is, there are points  $p_A, p_B$  in  $S$  for which  $w_A(p_A) = 0$  and  $w_B(p_B) = 0$ . The axioms for super-additive depth measures extend to weighted point sets in the following way:

- (i) for all  $S \in S^{\mathbb{R}^d}$  and  $q, p \in \mathbb{R}^d$  we have  $|\varrho(S, q) - \varrho(S \cup \{p\}, q)| \leq w(p)$  (sensitivity),
- (ii) for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) = 0$  for  $q \notin \text{conv}(S)$  (locality),
- (iii) for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) \geq \min\{w(p) : p \in S\}$  for  $q \in \text{conv}(S)$  (non-triviality),
- (iv) for any disjoint subsets  $S_1, S_2 \subseteq S$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) \geq \varrho(S_1, q) + \varrho(S_2, q)$  (super-additivity).

Clearly, each point set can be considered as a weighted point set by assigning weight 1 to each point. On the other hand, by placing several points at the same location, normalizing and using the fact that  $\mathbb{Q}$  is dense in  $\mathbb{R}$ , each depth measure defined on point sets can be extended to weighted point sets. Further, we can again define depth regions  $D_\varrho^S(\alpha) := \{q \in \mathbb{R}^d \mid \varrho(S, q) \geq \alpha\}$ . We will also use a special depth region, called the *median region*,



## 10:6 Enclosing Depth and Other Depth Measures

denoted by  $M_\varrho(S)$ , which is the deepest non-empty depth region. More formally, let  $\alpha_0$  be the supremum value for which  $D_\varrho^S(\alpha_0) \neq \emptyset$ . Then  $M_\varrho(S) := D_\varrho^S(\alpha_0)$ . In the setting of weighted point sets, the cascade condition translates to

$$\int_0^{w(S)} d_\alpha d\alpha \geq 0.$$

Note that the cascade conjecture for a depth measure on weighted point sets implies the cascade conjecture for that depth measure on unweighted point sets. If for a depth measure  $\varrho$  the above integral is non-negative for any weighted point set  $S$ , we say that  $\varrho$  is *cascading*.

In the following, we will show that super-additive depth measures whose depth regions are convex are cascading in two steps. First we will show that if we partition a weighted point set into two parts whose median regions intersect and the cascade condition holds for both parts, then the cascade condition holds for the whole set. In a second step, we prove that we can always partition a point set in such a way, further enforcing that none of the parts contains all points, that is, each part is a strict subset. The claim then follows by induction.

► **Lemma 9.** *Let  $\varrho$  be a super-additive depth measure whose depth regions are convex and let  $S_1$  and  $S_2$  be two weighted point sets in  $\mathbb{R}^d$  whose median regions intersect. Assume that the cascade condition holds for  $S_1$  and  $S_2$ . Then the cascade condition holds for  $S_1 \cup S_2$ .*

Before we prove this, let us describe a way to compute  $\int_0^{w(S)} d_\alpha d\alpha$ . Consider some depth region  $D_\varrho^S(\alpha)$  of dimension  $k$ . Being convex, this depth region lies in some  $k$ -dimensional affine subspace  $H \subset \mathbb{R}^d$ . Considering all depth regions, they lie in a sequence of nested affine subspaces, also known as a *flag*. Assuming that the origin lies in the median region, we can find a basis  $F = \{f_1, \dots, f_d\}$  of  $\mathbb{R}^d$  such that each relevant affine subspace is spanned by a subset of the basis vectors. In fact, there are many choices of bases. Further, we can assign to each basis vector  $f_i$  a *survival time*  $\alpha_i$  defined by the following property: for each  $\alpha \in \mathbb{R}$ , the affine subspace in which  $D_\varrho^S(\alpha)$  lies is spanned by the subset  $\{f_i \in F \mid \alpha_i \geq \alpha\}$ . As above, we let  $\alpha_0$  be the supremum value for which  $D_\varrho^S(\alpha_0) \neq \emptyset$ , that is, we view  $\alpha_0$  as the survival time of the origin. Using this formulation, we note that

$$\int_0^{w(S)} d_\alpha d\alpha = \sum_{i=0}^d \alpha_i - w(S),$$

see Figure 3 for an illustration.

**Proof of Lemma 9.** We may assume without loss of generality that the origin is in both median regions. Further, we can choose a basis  $F = \{f_1, \dots, f_d\}$  of  $\mathbb{R}^d$  such that all relevant affine subspaces both of  $S_1$  and  $S_2$ , and thus also of  $S_1 \cup S_2$ , are spanned by subsets of  $F$ . Let  $\alpha_i$ ,  $\beta_i$  and  $\gamma_i$  denote the survival times of  $f_i$  for  $S_1$ ,  $S_2$  and  $S_1 \cup S_2$ , respectively. It follows from the super-additivity condition that  $\gamma_i \geq \alpha_i + \beta_i$ . Thus we get

$$\begin{aligned} \sum_{i=0}^d \gamma_i - w(S_1 \cup S_2) &\geq \sum_{i=0}^d (\alpha_i + \beta_i) - (w(S_1) + w(S_2)) \\ &\geq \sum_{i=0}^d \alpha_i - w(S_1) + \sum_{i=0}^d \beta_i - w(S_2) \geq 0. \end{aligned} \tag{1}$$

◀



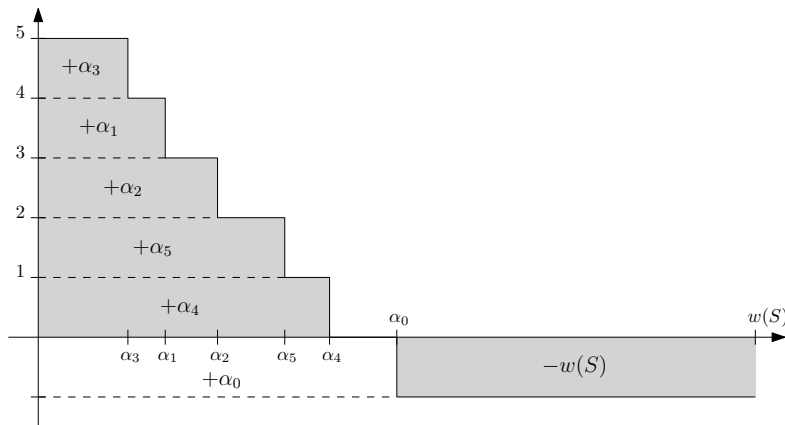


Figure 3  $\int_0^{w(S)} d_\alpha d\alpha = \sum_{i=0}^d \alpha_i - w(S)$ .

► **Lemma 10.** *Let  $\varrho$  be a super-additive depth measure whose depth regions are convex and let  $S$  be a weighted point set in  $\mathbb{R}^d$  with  $|S| \geq d + 2$ . Then there exists a partition of  $S$  into strict subsets  $S_1$  and  $S_2$  whose median regions intersect.*

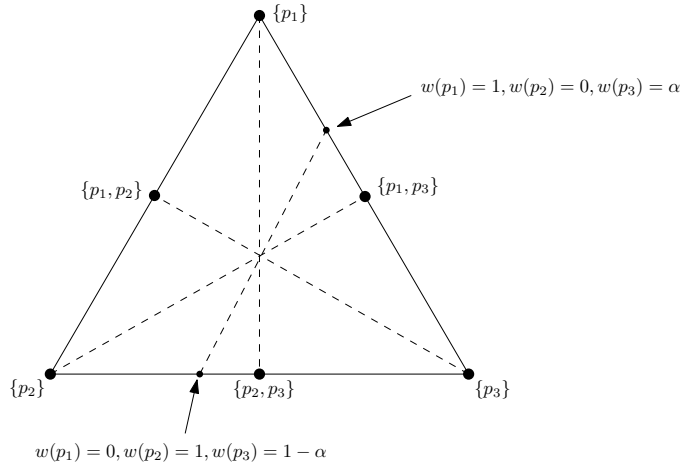
**Proof.** Assume without loss of generality that  $w(p) = 1$  for every  $p \in S$  (otherwise just multiply the weights of  $p$  in  $S_1$  and  $S_2$  with  $w(p)$  after finding the partition). Consider the barycentric subdivision  $B$  of the boundary  $\partial\Delta$  of the simplex with vertices  $S$ . There is a natural identification of the vertices of  $B$  with strict subsets of  $S$  (see Figure 4). Linearly extending this assignment to  $\partial\Delta$  defines a map which assigns to each point  $b$  on  $\partial\Delta$  a strict weighted subset  $S(b)$  of  $S$ . Further, under the natural antipodality on  $\partial\Delta$ , we get complements of the weighted subsets, that is,  $S(-b) = S(b)^C$ .

We claim that for some point  $b$  on  $\partial\Delta$  we have that the median regions of  $S(b)$  and  $S(-b)$  intersect. If this is true, our claim follows by setting  $S_1 = S(b)$  and  $S_2 = S(-b)$ . Using Proposition 1 from [31], for each  $b$  we may assume that the median region of  $S(b)$  is a single point  $m(b)$  in  $\mathbb{R}^d$  and that the map  $m$  which sends  $b$  to  $m(b)$  is continuous. We thus want to find a point  $b$  for which  $m(b) = m(-b)$ . Further,  $\partial\Delta$  is homeomorphic to the sphere  $S^{|S|-2}$ , and the antipodality on  $\partial\Delta$  corresponds to the standard antipodality on the sphere. As  $|S| \geq d + 2$ , the existence of a point  $b$  for which  $m(b) = m(-b)$  thus follows from the Borsuk-Ulam theorem. ◀

While we have only shown that there is a partition, Bourgin-Yang-type theorems [6, 30] tell us, that the space of possible partitions has to be large. In particular, it has dimension at least  $|S| - d - 2$ . Depending on the application, this might be used to enforce other conditions on the partitions.

► **Theorem 11.** *Let  $\varrho$  be a super-additive depth measure whose depth regions are convex. Then  $\varrho$  is cascading.*

**Proof.** Let  $S$  be a weighted point set in  $\mathbb{R}^d$  and assume without loss of generality that its affine hull is  $\mathbb{R}^d$  (otherwise, we can just consider  $S$  to be a weighted point set in some lower dimensional space). We want to show that the cascade condition holds for  $S$ . We prove this by induction on  $|S|$ . If  $|S| \leq d + 1$ , then  $S$  must be the vertices of a simplex, and in this case it is not hard to check that the cascade condition holds. So, assume now that  $|S| \geq d + 2$ . By Lemma 10, we can partition  $S$  into  $S_1$  and  $S_2$  whose median regions intersect. Note that  $|S_1|, |S_2| < |S|$ , so by the induction hypothesis the cascade condition holds for both  $S_1$  and  $S_2$ . Thus, by Lemma 9, the cascade condition also holds for  $S$ . ◀



■ **Figure 4** Vertices of the barycentric subdivision correspond to strict subsets.

As noted above, an example of a super-additive depth measure with convex depth regions is Tukey depth. Thus, we get the following.

► **Corollary 12.** *Tukey depth is cascading.*

On the other hand, while Tverberg depth is super-additive, its depth regions are in general not convex; in fact, they are not even connected. A weak version of Kalai’s cascade conjecture claims that the cascade condition holds for the convex hull of Tverberg depth regions. These depth regions are convex by definition, but the resulting depth measure is in general not super-additive anymore. So while our approach proves the cascade conjecture for an entire family of depth measures, solving Kalai’s cascade conjecture even in its weak form likely requires additional ideas. As every super-additive depth measure is bounded from below by Tverberg depth, solving the strong version of Kalai’s cascade conjecture would imply that all super-additive depth measures are cascading. Further, it can be seen that any cascading depth measure must enforce deep points. More precisely, if  $\varrho$  is a cascading depth measure and  $S$  is a point set in  $\mathbb{R}^d$ , then there must be a point  $q \in \mathbb{R}^d$  for which  $\varrho(S, q) \geq \frac{|S|}{d+1}$ . Indeed, if there was no such point, we would have  $d_{|S|/(d+1)} = -1$ , and even if  $d_i = d$  for all  $i < \frac{|S|}{d+1}$ , the sum  $\sum_{i=1}^{|S|} d_i$  would still be negative. The existence of deep points is the main feature of the next family of depth measures that we study.

**4 A second set of axioms**

The second family of depth measures we consider are *central* depth measures. A combinatorial depth measure  $\varrho : (S^{\mathbb{R}^d}, \mathbb{R}^d) \rightarrow \mathbb{R}_{\geq 0}$  is called central if it satisfies the following conditions:

- (i) for all  $S \in S^{\mathbb{R}^d}$  and  $q, p \in \mathbb{R}^d$  we have  $|\varrho(S, q) - \varrho(S \cup \{p\}, q)| \leq 1$  (sensitivity),
- (ii) for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $\varrho(S, q) = 0$  for  $q \notin \text{conv}(S)$  (locality),
- (iii’) for every  $S \in S^{\mathbb{R}^d}$  there is a  $q \in \mathbb{R}^d$  for which  $\varrho(S, q) \geq \frac{1}{d+1}|S|$  (centrality).
- (iv’) for all  $S \in S^{\mathbb{R}^d}$  and  $q, p \in \mathbb{R}^d$  we have  $\varrho(S \cup \{p\}, q) \geq \varrho(S, q)$  (monotonicity),

Note that conditions (i) and (ii) are the same as for super-additive depth measures, so by Observation 5 we have  $\varrho(S, q) \leq \text{TD}(S, q)$  for every central depth measure. Further, the centrality condition (iii’) is stronger than the non-triviality condition (iii) for super-additive depth measures. On the other hand, the super-additivity condition (iv) is stronger than

the monotonicity condition (iv'), so at first glance, the families of super-additive depth measures and central depth measures are not comparable. However, we have seen before that any super-additive depth measure indeed satisfies the centrality condition, so central depth measures are a superset of super-additive depth measures. It is actually a strict superset, as for example the depth measure whose depth regions are defined as the convex hulls of Tverberg depth regions is central but not super-additive.

While central depth measures enforce deep points by definition, they might still differ a lot locally. In the following, we will show that we can bound by how much they differ locally, showing that every central depth measure is a constant factor approximation of Tukey depth.

► **Theorem 13.** *Let  $\varrho$  be a central depth measure in  $\mathbb{R}^d$ . Then there exists a constant  $c = c(d)$ , which depends only on the dimension  $d$ , such that*

$$TD(S, q) \geq \varrho(S, q) \geq ED(S, q) - (d + 1) \geq c \cdot TD(S, q) - (d + 1).$$

Here the first inequality is just Observation 5. As for the second inequality, we would like to argue that if  $S$   $k$ -encloses  $q$  then  $\varrho(S, q) = k$ . By centrality, there must indeed be a point  $q'$  with  $\varrho(S, q') = k$  (note that  $|S| = k(d + 1)$  by definition of  $k$ -enclosing), but this point can lie anywhere in the centerpoint region of  $S$  and not every point in the centerpoint region is  $k$ -enclosed by  $S$ . However, by adding  $d + 1$  points very close to  $q$ , we can ensure that  $q$  is the only possible centerpoint in the new point set, and the second inequality then follows from sensitivity and monotonicity after removing these points again.

This argument can be generalized even to a relaxation of central depth measures: We say that a combinatorial depth measure is  $\alpha$ -central if it satisfies conditions (i), (ii) and (iv'), and the following weak version of condition (iii'): for every  $S \in \mathcal{S}^{\mathbb{R}^d}$  there is a  $q \in \mathbb{R}^d$  for which  $\varrho(S, q) \geq \alpha|S|$  ( $\alpha$ -centrality)

► **Lemma 14.** *Let  $\alpha > \frac{1}{d+2}$ , and let  $\varrho$  be an  $\alpha$ -central depth measure. Then there exists a constant  $c_1 = c_1(d)$  such that*

$$\varrho(S, q) \geq c_1 \cdot ED(S, q) - (d + 1).$$

**Proof.** Let  $ED(S, q) = k$  and let  $S'$  be a witness subset. Recall that by monotonicity, we have  $\varrho(S, q) \geq \varrho(S', q)$ . Further, note that  $TD(S', q) = k$  and  $TD(S', q') \leq k$  for all  $q' \in \mathbb{R}^d$ . Let  $\alpha' := (d + 1)\alpha$  and let  $m := \lfloor \frac{1-\alpha'}{\alpha'}k + 1 \rfloor$ . Add  $(d + 1)m$  points very close to  $q$  such that the new point set  $P$   $(k + m)$ -encloses  $q$ . The new point set  $P$  has  $(d + 1)(k + m)$  many points, and we have

$$\alpha|P| = \alpha'(k + m) > \alpha'(k + \frac{1-\alpha'}{\alpha'}k) = \alpha'k + (1 - \alpha')k = k.$$

In particular, the only points  $q'$  for which  $\varrho(P, q') \geq \alpha|P|$  is possible are by construction very close to  $q$ . As they were in the same cell as  $q$  before adding the new points, we can assume without loss of generality that we have  $\varrho(P, q) \geq \alpha|P|$ . By sensitivity we now have

$$\begin{aligned} \varrho(S', q) &\geq \varrho(P, q) - (d + 1)m \\ &\geq \alpha'(k + m) - (d + 1)m \\ &\geq \alpha'k - (d + 1 - \alpha')m \\ &\geq \alpha'k - (d + 1 - \alpha')(\frac{1-\alpha'}{\alpha'} + 1) \\ &= \alpha'k - \frac{(d + 1 - \alpha')(1 - \alpha')}{\alpha'}k - (d + 1) + \alpha \end{aligned}$$

## 10:10 Enclosing Depth and Other Depth Measures

$$\begin{aligned} &\geq (\alpha'^2 - (d+1) + \alpha' + (d+1)\alpha' - \alpha'^2) \frac{k}{\alpha'} - (d+1) \\ &= \frac{(d+2)\alpha' - (d+1)}{\alpha'} k - (d+1). \quad (2) \end{aligned}$$

Plugging in  $\alpha' := (d+1)\alpha$  we get

$$\varrho(S, q) \geq \frac{(d+2)(d+1)\alpha - (d+1)}{(d+1)\alpha} k - (d+1) = (d+2 - \frac{1}{\alpha})k - (d+1).$$

As  $(d+2 - \frac{1}{\alpha}) > 0$  for  $\alpha > \frac{1}{d+2}$ , the claim follows.  $\blacktriangleleft$

The most involved part of Theorem 13 is the last inequality, which we will prove in the next section.

### 5 A lower bound for enclosing depth

In this section, we will prove a lower bound on the enclosing depth in terms of Tukey depth:

► **Theorem 15** ( $E(d)$ ). *There is a constant  $c_1 = c_1(d)$  such that for all  $S \in S^{\mathbb{R}^d}$  and  $q \in \mathbb{R}^d$  we have  $ED \leq c_1 \cdot TD(S, q)$ .*

We will denote this statement in dimension  $d$  by  $E(d)$ . Note that  $E(1)$  is true and  $c_1(1) = 1$ . The general result could be proved using the semi-algebraic same type lemma due to Fox, Pach and Suk [9], combined with the first selection lemma (see e.g. [19]). Here we will give a different proof for two reasons: first, the bounds on  $c_1$  that our proof gives are better than the bounds we would get from the proof using the semi-algebraic same type lemma. Second, our proof shows an intimate relation of enclosing depth to a positive fraction Radon theorem on certain bichromatic point sets.

Let  $P = R \cup B$  be a bichromatic point set with color classes  $R$  (red) and  $B$  (blue). We say that  $B$  *surrounds*  $R$  if for every halfspace  $h$  we have  $|B \cap h| \geq |R \cap h|$ . Note that this in particular implies  $|B| \geq |R|$ . The positive fraction Radon theorem is now the following:

► **Theorem 16** ( $R(d)$ ). *Let  $P = R \cup B$  be a bichromatic point set where  $B$  surrounds  $R$ . Then there is a constant  $c_2 = c_2(d)$  such that there are integers  $a$  and  $b$  and pairwise disjoint subsets  $R_1, \dots, R_a \subseteq R$  and  $B_1, \dots, B_b \subseteq B$  with*

1.  $a + b = d + 2$ ,
2.  $|R_i| \geq c_2 \cdot |R|$  for all  $1 \leq i \leq a$ ,
3.  $|B_i| \geq c_2 \cdot |R|$  for all  $1 \leq i \leq b$ ,
4. for every transversal  $r_1 \in R_1, \dots, r_a \in R_a, b_1 \in B_1, \dots, b_b \in B_b$ , we have

$$\text{conv}(r_1, \dots, r_a) \cap \text{conv}(b_1, \dots, b_b) \neq \emptyset.$$

In other words, the Radon partition respects the color classes. We will denote the above statement in dimension  $d$  by  $R(d)$ .

► **Lemma 17.**  $R(1)$  can be satisfied choosing  $a = 1$ ,  $b = 2$  and  $c_2(1) = \frac{1}{3}$ .

**Proof.** Consider two points  $x_1$  and  $x_2$  such that there are exactly  $\frac{|R|}{3}$  blue points to the left of  $x_1$  and to the right of  $x_2$ , respectively. Define  $B_1$  as the set of blue points left of  $x_1$  and  $B_2$  as the set of blue points right of  $x_2$ . We then have  $|B_1| = |B_2| = \frac{1}{3}|R|$ . Further, as  $B$  surrounds  $R$ , we have at most  $\frac{|R|}{3}$  red points to the left of  $x_1$ , and also to the right of  $x_2$ . In particular, there are at least  $\frac{|R|}{3}$  red points between  $x_1$  and  $x_2$ . Let now  $R_1$  be any subset of  $\frac{|R|}{3}$  red points between  $x_1$  and  $x_2$ . It follows from the construction that  $\text{conv}(R_1) \cap \text{conv}(B_1, B_2) \neq \emptyset$ .  $\blacktriangleleft$

In the following, we will prove that  $R(d - 1) \Rightarrow E(d)$  and that  $E(d - 1) \Rightarrow R(d)$ . By induction, these two claims then imply the above theorems.

► **Lemma 18.**  $R(d - 1) \Rightarrow E(d)$ .

**Proof.** Assume that  $\text{TD}(S, q) = k$  and let  $h$  be a witnessing hyperplane which contains  $q$  but no points of  $S$ . Without loss of generality, assume that  $q$  is the origin and that  $h$  is the hyperplane through the equator on  $S^{d-1} \subseteq \mathbb{R}^d$ , with exactly  $k$  points below. Color the points below  $h$  red and the points above  $h$  blue. Now, for every point  $p \in S$ , consider the line through  $p$  and  $q$  and let  $p'$  be the intersection of that line with the tangent hyperplane to the north pole of  $S^{d-1}$ . Color  $p'$  the same color as  $p$ . This gives a bichromatic point set  $S' = R \cup B$  in  $\mathbb{R}^{d-1}$ . Further, in  $S'$ , we have that  $B$  surrounds  $R$ : Assume there is a hyperplane  $\ell$  (in  $\mathbb{R}^{d-1}$ ) with  $r$  red points and  $b$  blue points on its positive side, where  $r > b$ . In  $\mathbb{R}^d$ , this lifts to a hyperplane containing  $q$  with  $k - r$  red points and  $b$  blue points on its positive side (note that there are exactly  $k$  red points). However,  $k - r + b < k$ , whenever  $r > b$ , thus we would have  $\text{TD}(s, q) < k$ , which is a contradiction.

As we now have a point set in  $\mathbb{R}^{d-1}$ , in which  $B$  surrounds  $R$ , we can apply  $R(d - 1)$  to find families of  $d + 2$  subsets of  $S'$ , each of size  $c_2 \cdot k$ , some red and some blue, such that in each transversal the color classes form a Radon partition. We claim that the corresponding subsets of  $S$   $c_2 \cdot k$ -enclose  $q$ . Pick some transversal (which we call the original red and blue points) and consider the corresponding subset in  $S'$ . Let  $z$  be a point in the intersection of the convex hulls of the two color classes, and let  $g$  be the line through  $z$  and  $q$ . As  $z$  is in the convex hull of the blue points, there is a point  $z^+$  on  $g$  which is in the convex hull of the original blue points, and thus above  $h$ . Similarly, there is a point  $z^-$  on  $g$  which is in the convex hull of the original red points, and thus below  $h$ . As  $q$  is in the convex hull of  $z^+$  and  $z^-$ , it is thus in the convex hull of the original blue and red points. ◀

In particular, this proof shows that  $c_1(d) = c_2(d - 1)$ .

For the proof of the second implication, we need to recall a few results, starting with the *Same Type Lemma* by Bárány and Valtr [5].

► **Theorem 19** (Theorem 2 in [5]). *For every two natural numbers  $d$  and  $m$  there is a constant  $c_3(d, m) > 0$  with the following property: Given point sets  $X_1, \dots, X_m \subseteq \mathbb{R}^d$  such that  $X_1 \cup \dots \cup X_m$  is in general position, there are subsets  $Y_i \subseteq X_i$  with  $|Y_i| \geq c_3 \cdot |X_i|$  such that all transversals of the  $Y_i$  have the same order type.*

From the proof in [5], we get  $c_3(d, m) = 2^{-m^{O(d)}}$ . This bound has been improved in [9] to  $c_3(d, m) = 2^{-O(d^3 m \log m)}$ .

The second result that we will need is the *Center Transversal Theorem*, proved independently by Dol'nikov [8] as well as Zivaljević and Vrećica [31]. We will only need the version for two colors, so we state it in this restricted version:

► **Theorem 20** (Center Transversal for two colors). *Let  $\mu_1$  and  $\mu_2$  be two finite Borel measures on  $\mathbb{R}^d$ . Then there exists a line  $\ell$  such that for every closed halfspace  $H$  which contains  $\ell$  and every  $i \in \{1, 2\}$  we have  $\mu_i(H) \geq \frac{\mu_i(\mathbb{R}^d)}{d}$ .*

Such a line  $\ell$  is called a *center transversal*. By a standard argument (replacing points with balls of small radius, see e.g. [18]), the same result also holds for two point sets  $P_1, P_2$  in general position, where  $\mu_i(H)$  is replaced by  $|P_i \cap H|$ . As we will need similar ideas later, we will briefly sketch a proof of the above Theorem. Consider some  $(d - 1)$ -dimensional linear subspace  $F$ , i.e., a hyperplane through the origin, and project both measures to it.

## 10:12 Enclosing Depth and Other Depth Measures

For each projected measure, consider the centerpoint region (i.e., the region of Tukey depth  $\geq \frac{\mu_i(\mathbb{R}^d)}{(d-1)+1}$ ). This is a non-empty, convex set, so it has a unique center of mass, which we will denote by  $c_i(F)$ . Rotating the subspace  $F$  in continuous fashion, these centers of mass also move continuously, so the  $c_i(F)$  are two continuous assignments of points to the set of all  $(d-1)$ -dimensional linear subspaces. The result then follows from the following Lemma, again proved independently by Dol'nikov [8] as well as Zivaljević and Vrećica [31]:

► **Lemma 21.** *Let  $g_1$  and  $g_2$  be two continuous assignments of points to the set of all  $(d-1)$ -dimensional linear subspaces of  $\mathbb{R}^d$ . Then there exists such a subspace  $F$  in which  $g_1(F) = g_2(F)$ .*

Note that in order to apply this Lemma, we had to choose in a continuous way a centerpoint. If the two measures can be separated by a hyperplane, we can do something similar with the center transversal:

► **Lemma 22.** *Let  $\mu_1$  and  $\mu_2$  be two finite Borel measures on  $\mathbb{R}^d$ , which can be separated by a hyperplane. Then there is a unique canonical choice of a center transversal.*

**Proof.** Let  $x_1, \dots, x_d$  be the basis vectors of  $\mathbb{R}^d$  and assume without loss of generality that the hyperplane  $H : x_d = 0$  separates the two measures  $\mu_1, \mu_2$ . For any  $d-1$ -dimensional linear subspace  $F$ , consider the projection  $\pi_F : \mathbb{R}^d \rightarrow F$ . Note that if  $F$  is orthogonal to  $H$ , then  $\pi_F(H)$  separates  $\pi_F(\mu_1)$  and  $\pi_F(\mu_2)$ , so there is no center transversal parallel to  $H$ . It thus suffices to consider only (oriented) subspaces which point upwards (in the sense that the  $x_d$ -component in their normal vector is  $> 0$ ). The space of these subspaces is homeomorphic to the upper hemisphere  $S^+$  of  $S^{d-1}$ . Let now  $C$  be the set of all such subspaces in which we have  $g_1(F) = g_2(F)$ . We claim that  $C$  is a convex set in  $S^+$ . Consider two subspaces  $F_1$  and  $F_2$  with  $g_1(F_1) = g_2(F_1)$  and  $g_1(F_2) = g_2(F_2)$ . The shortest path between  $F_1$  and  $F_2$  corresponds to a rotation around a  $(d-2)$ -dimensional axis. Rotate from  $F_1$  to  $F_2$  with constant speed and consider a point in the support of a measure. The projection of this point moves along a line in the projection. In fact, all points in move along parallel lines with direction  $\vec{d}$ , and the points in the support of  $\mu_1$  move in the opposite direction of the points in the support of  $\mu_2$ . Further, for any points  $p_1$  in the support of  $\mu_1$  and  $p_2$  in the support of  $\mu_2$ , their projections move towards one another, until they are on a common hyperplane with normal vector  $\vec{d}$ , and the away from one another. The same arguments hold for the centerpoint regions of the projections and their centers of mass, which shows that if  $g_1(F_1) = g_2(F_1)$  and  $g_1(F_2) = g_2(F_2)$  then  $g_1(F) = g_2(F)$  for every subspace  $F$  along the rotation. Thus, the set  $C$  is indeed convex, and we can choose the unique solution corresponding to the center of mass of  $C$ . ◀

Again, the same statement holds for point sets in general position. With these tools at hand, we are now ready to prove the second part of the induction.

► **Lemma 23.**  $E(d-1) \Rightarrow R(d)$ .

**Proof.** Let  $\ell$  be a line through the origin. Sweep a hyperplane orthogonal to  $\ell$  from one side to the other (without loss of generality from left to right). Let  $h_1$  be a sweep hyperplane with exactly  $\frac{|R|}{3}$  blue points to the left, and let  $A_1$  be the set of these blue points. Similarly, let  $A_2$  be a set of exactly  $\frac{|R|}{3}$  blue points to the right of a sweep hyperplane  $h_2$ . Let  $c$  be the unique center transversal of  $A_1$  and  $A_2$  given by Lemma 22 and let  $g$  be the  $(d-1)$ -dimensional linear subspace which is orthogonal to  $c$ . Note that it follows from the proof of Lemma 22 that  $g$  cannot be orthogonal to the sweep hyperplanes. We denote the projection of  $c$  to  $g$  as  $c_A$ . Note that  $c_A$  is a centerpoint of the projections of  $A_1$  and of  $A_2$  to  $g$ . Now, consider the

set  $M$  of all red points between  $h_1$  and  $h_2$  and note that as the blue points surround the red points we have  $|M| \geq \frac{|R|}{3}$ . Project  $M$  to  $g$  and denote by  $c_M$  the center of mass of the centerpoint region of the projected point set. We claim that there exists a choice of a line  $\ell$ , such that  $c_M = c_A$ . Indeed, as  $g$  is not orthogonal to a sweep hyperplane, there is a unique shortest rotation which rotates  $g$  to a hyperplane orthogonal to  $\ell$ , thus the space of all  $g$ 's is homeomorphic to the space of all  $(d - 1)$ -dimensional linear subspaces. Further,  $c_A$  and  $c_M$  are continuous assignments of points, thus the above claim follows from Lemma 21.

So assume now that  $c_M = c_A$ . In particular,  $c$  is a center transversal for  $A_1, A_2$  and  $M$ . Project  $A_1$  to  $g$ . The projection of  $c$  is a centerpoint of the projection of  $A_1$  in  $g$  and  $g$  has dimension  $d - 1$ , thus by the statement  $E(d - 1)$  there are three subsets  $A_{1,1}, \dots, A_{1,d}$  of  $A_1$ , each of size  $c_1 \cdot |A_1|$  whose projections enclose the projection of  $c$ . The analogous arguments gives subsets  $A_{2,1}, \dots, A_{2,d}$  of  $A_2$  and  $M_1, \dots, M_d$  of  $M$ . Consider now these  $3d$  subsets. By Theorem 19 there are subsets  $A'_{1,1}, \dots, M'_d$ , each of size linear in the size of the original subset, such that each transversal of the subsets has the same order type. Consider such a transversal. By construction, the  $d$  points of  $A_1$  contain in their convex hull a point on  $c$  which is to the left of  $h_1$ . Similarly, the  $d$  points of  $A_2$  contain in their convex hull a point on  $c$  to the right of  $h_2$ . Finally, the  $d$  points of  $M$  contain in their convex hull a point on  $c$  between  $h_1$  and  $h_2$ . Thus, the convex hulls of the blue points (from  $A_1$  and  $A_2$ ) and the red points (from  $M$ ) intersect. In particular, there is a subset of  $d + 2$  red and blue points, which form a Radon partition. By choosing the subsets from which these points were selected, we now get the subsets required for  $R(d)$ . ◀

This proof show that  $c_2(d) = \frac{c_3(d,d+2)}{3^d} c_1(d - 1)$ . Using the bound on  $c_3$  from [9] and  $c_1(d) = c_2(d - 1)$ , we thus get  $c_2(d) = \Omega(\frac{c_2(d-2)}{3^d \cdot 2^{d^4 \log d}}) = \dots = \Omega(\frac{1}{3^{d/2} d! \cdot 2^{d^5 \log d}})$ , and as  $c_1(d) = c_2(d - 1)$  we get the same asymptotics for  $c_1$ .

Combining this with the results from Section 4, we get that any central depth measure is an approximation of Tukey depth. In fact, by Lemma 14 this even holds for many  $\alpha$ -central depth measures.

▶ **Corollary 24.** *Let  $\varrho$  be an  $\alpha$ -central depth measure on  $\mathbb{R}^d$  where  $\alpha > \frac{1}{d+2}$ . Then there exists a constant  $c = c(d)$  such that for every point set  $S$  and query point  $q$  in  $\mathbb{R}^d$  we have*

$$TD(S, q) \geq \varrho(S, q) \geq c \cdot TD(S, q).$$

## 6 Conclusion

We have introduced two families of depth measures, called super-additive depth measures and central depth measures, where the first is a strict subset of the second. We have shown that all these depth measures are a constant-factor approximation of Tukey depth.

It is known that Tukey depth is coNP-hard to compute when both  $|S|$  and  $d$  is part of the input [12], and it is even hard to approximate [2] (see also [7]). Our result is thus an indication that central depth measures are hard to compute. However, this does not follow directly, as our constant has a doubly exponential dependence on  $d$ . It is an interesting open problem whether the approximation factor can be improved.

Further, we have introduced a new depth measure called enclosing depth, which is neither super-additive nor central, but still is a constant-factor approximation of Tukey depth. As it turns out, this depth measure is intimately related to a constant fraction Radon theorem on bi-colored point sets. Finally, we have shown that any super-additive depth measure whose depth regions are convex is cascading.



This last result is motivated by Kalai’s cascade conjecture, which, in the terminology of this paper, states that Tverberg depth is cascading. While this conjecture remains open, we hope that our results might be useful for an eventual proof.

There is a depth measure which has attracted a lot of research, which does not fit into our framework: simplicial depth (SD). The reason for this is that while the depth studied in this paper are linear in the size of the point set, simplicial depth has values of size  $O(|S|^{d+1})$ . However, after the right normalization, simplicial depth can be reformulated to satisfy all conditions except super-additivity and centrality. It would be interesting to see whether there is some function  $g$  depending on point sets and query points such that the depth measure  $\frac{SD(S,q)}{g(S,q)}$  is super-additive. Such a function, if it exists, could potentially be used to improve bounds for the first selection lemma (see e.g. [19]).

---

### References

- 1 Greg Aloupis. Geometric measures of data depth. In *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, pages 147–158, 2003. doi:10.1090/dimacs/072/10.
- 2 Edoardo Amaldi and Viggo Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Theoretical Computer Science*, 147(1):181–210, 1995.
- 3 David Avis. The  $m$ -core properly contains the  $m$ -divisible points in space. *Pattern recognition letters*, 14(9):703–705, 1993.
- 4 Imre Bárány and Pablo Soberón. Tverberg’s theorem is 50 years old: a survey. *Bulletin of the American Mathematical Society*, 55(4):459–492, 2018.
- 5 Imre Bárány and Pavel Valtr. A positive fraction Erdős-Szekeres theorem. *Discrete & Computational Geometry*, 19(3):335–342, 1998.
- 6 DG Bourgin. On some separation and mapping theorems. *Commentarii Mathematici Helvetici*, 29(1):199–214, 1955.
- 7 Dan Chen, Pat Morin, and Uli Wagner. Absolute approximation of Tukey depth: Theory and experiments. *Computational Geometry*, 46(5):566–573, 2013. Geometry and Optimization.
- 8 VL Dol’nikov. Transversals of families of sets in  $\mathbb{R}^n$  and a connection between the Helly and Borsuk theorems. *Russian Academy of Sciences. Sbornik Mathematics*, 79(1):93, 1994.
- 9 Jacob Fox, János Pach, and Andrew Suk. A polynomial regularity lemma for semialgebraic hypergraphs and its applications in geometry and property testing. *SIAM Journal on Computing*, 45(6):2199–2223, 2016.
- 10 Sariel Har-Peled and Timothy Zhou. Improved approximation algorithms for tverberg partitions. *arXiv preprint*, 2020. arXiv:2007.08717.
- 11 Joseph L Hodges. A bivariate sign test. *The Annals of Mathematical Statistics*, 26(3):523–527, 1955.
- 12 D.S. Johnson and F.P. Preparata. The densest hemisphere problem. *Theoretical Computer Science*, 6(1):93–107, 1978.
- 13 GIL KALAI. Combinatorics with a geometric flavor. *Visions in Mathematics: GAFA 2000 Special Volume, Part II*, page 742, 2011.
- 14 Gil Kalai. Problems in geometric and topological combinatorics. Lecture at FU Berlin, 2011.
- 15 Gil Kalai. Problems for imre bárány’s birthday. *Discrete Geometry and Convexity in Honour of Imre Bárány*, page 59, 2017.
- 16 R. Y. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, 18(1):405–414, 1990.
- 17 Regina Y. Liu, Jesse M. Parelius, and Kesar Singh. Multivariate analysis by data depth: descriptive statistics, graphics and inference. *Ann. Statist.*, 27(3):783–858, June 1999.
- 18 Jiří Matoušek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*. Springer Publishing Company, Incorporated, 2007.
- 19 Jivri Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate texts in mathematics*. Springer, 2002.



- 20 Kim Miller, Suneeta Ramaswami, Peter Rousseeuw, J. Antoni Sellarès, Diane Souvaine, Ileana Streinu, and Anja Struyf. Efficient computation of location depth contours by methods of computational geometry. *Statistics and Computing*, 13(2):153–162, 2003.
- 21 Karl Mosler. *Depth Statistics*, pages 17–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- 22 Richard Rado. A theorem on general measure. *Journal of the London Mathematical Society*, 21:291–300, 1947.
- 23 John R Reay. Several generalizations of tverberg’s theorem. *Israel Journal of Mathematics*, 34(3):238–244, 1979.
- 24 Jean-Pierre Roudneff. Partitions of points into simplices with  $k$ -dimensional intersection. part i: The conic tverberg’s theorem. *European Journal of Combinatorics*, 22(5):733–743, 2001.
- 25 Jean-Pierre Roudneff. Partitions of points into simplices with  $k$ -dimensional intersection. part ii: Proof of reay’s conjecture in dimensions 4 and 5. *European Journal of Combinatorics*, 22(5):745–765, 2001.
- 26 Jean-Pierre Roudneff. New cases of reay’s conjecture on partitions of points into simplices with  $k$ -dimensional intersection. *European Journal of Combinatorics*, 30(8):1919–1943, 2009.
- 27 Csaba D Toth, Joseph O’Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017.
- 28 John W. Tukey. Mathematics and the picturing of data. In *Proc. International Congress of Mathematicians*, pages 523–531, 1975.
- 29 Helge Tverberg. A generalization of Radon’s theorem. *Journal of the London Mathematical Society*, 1(1):123–128, 1966.
- 30 Chung-Tao Yang. On theorems of Borsuk-Ulam, Kakutani-Yamabe-Yujobô and Dyson, I. *Annals of Mathematics*, pages 262–282, 1954.
- 31 Rade T. Zivaljević and Siniša T Vrećica. An extension of the ham sandwich theorem. *Bulletin of the London Mathematical Society*, 22(2):183–186, 1990.
- 32 Yijun Zuo and Robert Serfling. Structural properties and convergence results for contours of sample statistical depth functions. *Ann. Statist.*, 28(2):483–499, April 2000.



# Illuminating the $x$ -Axis by $\alpha$ -Floodlights

**Bengt J. Nilsson** ✉ 

Department of Computer Science and Media Technology, Malmö University, Sweden

**David Orden** ✉ 

Physics and Mathematics Department, Universidad Alcalá, Spain

**Leonidas Palios** ✉ 

Department of Computer Science and Engineering, University of Ioannina, Greece

**Carlos Seara** ✉ 

Department of Mathematics, Universitat Politècnica de Catalunya, Barcelona, Spain

**Paweł Żyliński** ✉

Institute of Informatics, University of Gdańsk, Poland

---

## Abstract

Given a set  $S$  of regions with piece-wise linear boundary and a positive angle  $\alpha < 90^\circ$ , we consider the problem of computing the locations and orientations of the minimum number of  $\alpha$ -floodlights positioned at points in  $S$  which suffice to illuminate the entire  $x$ -axis. We show that the problem can be solved in  $O(n \log n)$  time and  $O(n)$  space, where  $n$  is the number of vertices of the set  $S$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases** Computational Geometry, Visibility, Art Gallery Problems, Floodlights

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.11

**Funding** *Bengt J. Nilsson*: Grant 2018-04001 from the Swedish Research Council.

*David Orden*: H2020-MSCA-RISE project 734922-CONNECT and project PID2019-104129GB-I00 funded by MCIN/AEI/10.13039/501100011033.

*Carlos Seara*: H2020-MSCA-RISE project 734922-CONNECT and projects PID2019-104129GB-I00/AEI/10.13039/501100011033 and Gen. Cat. DGR 2017SGR1640.

## 1 Introduction

An  $\alpha$ -floodlight is a two-dimensional floodlight whose illumination cone angle is equal to a positive angle  $\alpha$ . We are interested in using the minimum number of  $\alpha$ -floodlights positioned at points of a given set  $S$  in the plane in order to illuminate the entire  $x$ -axis; in particular, we consider that  $S$  is a collection of regions with piece-wise linear boundary which may degenerate into a point. We assume that no point of  $S$  lies on the  $x$ -axis (otherwise, at most two floodlights would suffice for any value of  $\alpha$ ) and that the entire  $S$  lies in the halfplane above the  $x$ -axis (any point of  $S$  below the  $x$ -axis can be equivalently reflected about the  $x$ -axis into the halfplane above the  $x$ -axis). Next, regarding the angle  $\alpha$  of the  $\alpha$ -floodlights, we consider that  $\alpha < 90^\circ$  because for  $\alpha \geq 90^\circ$  the problem admits a trivial solution: if  $90^\circ \leq \alpha < 180^\circ$  then two floodlights are necessary and sufficient to illuminate the entire  $x$ -axis, and if  $\alpha \geq 180^\circ$  then one floodlight is necessary and sufficient. Thus, in this paper we focus on the following problem.

### The Axis $\alpha$ -Illumination Problem

Given a set  $S$  of regions with piece-wise linear boundary above the  $x$ -axis and a positive angle  $\alpha < 90^\circ$ , compute the locations and orientations of the minimum number of  $\alpha$ -floodlights positioned at points in  $S$  which suffice to illuminate the entire  $x$ -axis.



© Bengt J. Nilsson, David Orden, Leonidas Palios, Carlos Seara, and Paweł Żyliński; licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 11; pp. 11:1–11:12

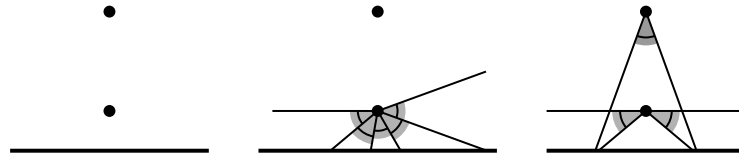
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 11:2 Illuminating the $x$ -Axis by $\alpha$ -Floodlights

As the number of required  $\alpha$ -floodlights can be very large even for a set  $S$  of small descriptive size, we designate that a solution to an instance of the Axis  $\alpha$ -floodlight Problem is a set of pairs  $(t_i, R_i)$  where  $t_i \in S$  and  $R_i$  is a maximal continuous range of the  $x$ -axis to be illuminated by  $\alpha$ -floodlights positioned at  $t_i$  such that (i) the union of all the ranges  $R_i$  is equal to the  $x$ -axis and (ii) over all pairs, the sum  $\sum_i \lceil \frac{\text{angle}(R_i)}{\alpha} \rceil$  is minimized where  $\text{angle}(R_i)$  is the angle subtended by the range  $R_i$  from  $t_i$ ; this sum is precisely the total number of  $\alpha$ -floodlights needed to illuminate the entire  $x$ -axis. We note that there may be more than one pair associated with a location  $t_i \in S$  but if it is so, then the corresponding  $x$ -axis ranges do not intersect; see Figure 1(right). As we show (Corollary 12), the size of such a solution is at most linear in the number of vertices of the given set  $S$ .

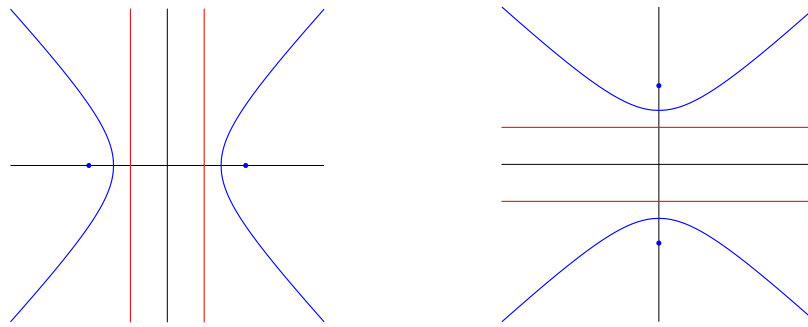


■ **Figure 1** (left) An instance of the Axis  $\alpha$ -Illumination Problem for two floodlight locations and  $\alpha = 40^\circ$ ; (middle) A solution with five floodlights; (right) A solution with three floodlights.

Clearly, any instance of the Axis  $\alpha$ -floodlight Problem admits a solution since a single point in the set  $S$  can illuminate the entire  $x$ -axis by using  $\lceil \frac{180^\circ}{\alpha} \rceil$   $\alpha$ -floodlights; see Figure 1(middle) where five  $40^\circ$ -floodlights can be used to illuminate the  $x$ -axis. Yet, the minimum number of needed floodlights may be much smaller; in Figure 1(right), for the given set  $S$  containing two locations, three  $40^\circ$ -floodlights suffice to illuminate the entire  $x$ -axis.

**Our Contribution.** In this paper, we present an algorithm to solve the Axis  $\alpha$ -Illumination Problem. Our algorithm runs in  $O(n \log n)$  time where  $n$  is the number of vertices of the given set  $S$  of potential floodlight locations. Our algorithm can be used to illuminate arbitrary lines as well as line segments.

**Related Work.** Floodlight illumination problems are considered a prominent class in Computational Geometry [14, 20] and find applications in the field of directional sensor networks [19]. The seminal *Stage Illumination Problem* [3] was posed by Urrutia in 1992 and later proved to be NP-complete even with some restrictions by Ito et al. [11]. This problem takes as inputs a line segment and a set of floodlights  $F_i$  with angles  $\alpha_i$  and apexes at predetermined locations  $p_i$  on the same side of the segment, the goal being to rotate the floodlights around their positions in such a way that the segment is completely illuminated. Even more related to our work is the problem of the *Optimal Floodlight Illumination of the Real Line* [5], which takes as inputs a line and a set  $S$  of  $n$  points, with the goal being to determine a finite set of floodlights  $F_j$  of arbitrary angles  $\alpha_j$  and with apexes at  $p_j \in S$  (more than one floodlight can have the same point as apex) such that the line is illuminated and the sum of the angles  $\alpha_j$  is the smallest possible. This problem was shown to be solvable in  $\Theta(n \log n)$  time [5]. Further, the *Optimal Floodlight Illumination of a Stage*, similar to the previous one but considering a segment instead of a line, has also been considered and solved in  $\Theta(n \log n)$  time even if no more than one floodlight are allowed to have the same point as apex [7]. On the other hand, the problem of whether a polygon can be illuminated by a given number of  $\alpha$ -floodlights is NP-hard and APX-hard [1].



■ **Figure 2** (left) A hyperbola with its foci (shown in blue) on a horizontal line; (right) A hyperbola with its foci (shown in blue) on a vertical line. The red lines are the directrices of the hyperbola.

Finally, a wider perspective locates our problem as a variant of the Art Gallery Problem, originally posed by Klee in 1973 as the question of determining the minimum number of guards sufficient to see every point of the interior of a simple polygon; for more details, see the book by O’Rourke [13], the survey article by Shermer [15], and the book chapter by Urrutia [20]. Among all these variants, we only mention just a few, chronologically, those the most related to our problem: the searchlight problem in polygons [18], floodlight illumination of the plane [3], floodlight illumination of polygons [8], the two-floodlight illumination problem [9], floodlight illumination of wedges [17], continuous surveillance of points by rotating floodlights [2], and monitoring the plane with rotating radars [4].

## 2 Preliminaries

Our algorithm relies on the use of hyperbola arcs and of the farthest-point Voronoi diagram of a point set in the plane. So, we present the definition and useful properties of these two notions.

**Hyperbola.** A *hyperbola* with foci  $f_1$  and  $f_2$  is the locus of the points in the plane such that the absolute value of the difference of their distances from  $f_1$  and  $f_2$  is constant (and less than the distance of the foci) [21]; see Figure 2(left). Most commonly, the foci are located at  $(x_0 - c, y_0)$  and  $(x_0 + c, y_0)$ , where  $c > 0$ . Then, if the absolute value of the difference of the distances from the foci is equal to  $2a$  with  $0 < a < c$ , the expression of the hyperbola is

$$\frac{(x - x_0)^2}{a^2} - \frac{(y - y_0)^2}{c^2 - a^2} = 1. \quad (1)$$

Such a hyperbola consists of two branches separated by any vertical line  $x = x_0 + \delta$  where  $-a < \delta < a$ . In fact, these branches can also be defined in terms of the two *directrices* of the hyperbola which, in this case, are vertical lines; the directrices are symmetrically positioned about the *center*  $(x_0, y_0)$  of the hyperbola at distance  $a^2/c$  from it. Then, each *hyperbola branch* is the locus of points whose distance from one of the foci divided by the (perpendicular) distance from the corresponding directrix is greater than 1; for the hyperbola satisfying Equation (1), this ratio of distances is equal to  $c/a > 1$ .

If the foci are located at  $(x_0, y_0 - c)$  and  $(x_0, y_0 + c)$  (i.e., the foci are on a *vertical* line), we have a symmetric case by “exchanging” the  $x$ - and  $y$ -axis, as shown in Figure 2(right).

**Farthest-point Voronoi diagram.** The farthest-point Voronoi diagram of a planar point set will also be exploited in our algorithm. For a given set of points  $S = \{p_1, p_2, \dots, p_n\}$  in the plane, typically called *sites*, the *farthest-point Voronoi diagram* of  $S$  divides the plane into cells such that each cell contains all the points of the plane with the same farthest site among the sites in  $S$  [6]. It is well known that:

## 11:4 Illuminating the $x$ -Axis by $\alpha$ -Floodlights

► **Lemma 1** ([6]).

- (i) A point of a point-set  $S$  has a cell in the farthest-point Voronoi diagram of  $S$  if and only if it is a vertex of the convex hull of  $S$ .
- (ii) The farthest-point Voronoi diagram of  $n$  points in the plane has  $O(n)$  vertices, edges, and cells.

As a result of Lemma 1(i), all cells of a planar farthest-point Voronoi diagram are unbounded and its vertices and edges form a tree-like structure (Figure 9). Additionally, the definition of the farthest-point Voronoi diagram readily implies the following corollary.

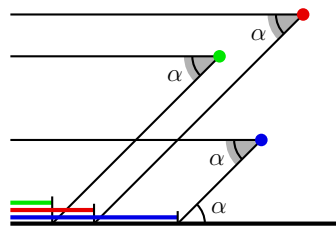
► **Corollary 2.** Let  $fVD(S)$  be the farthest-point Voronoi diagram of a planar point-set  $S$  and let  $V(p)$  be the cell of  $p \in S$  in  $fVD(S)$ . Then, a point  $r$  belongs to the closure of  $V(p)$  if and only if the entire  $S$  is enclosed by the circle with center  $r$  and radius the distance of  $r$  to  $p$ .

### 3 Illuminating unbounded and bounded ranges of the $x$ -axis

In this section, we present how to efficiently illuminate unbounded and bounded ranges of the  $x$ -axis and we introduce some useful notation. We first consider unbounded ranges of the  $x$ -axis, i.e., a range  $(-\infty, \chi]$  or a range  $[\chi', +\infty)$ . Then, the definition of an  $\alpha$ -floodlight implies the following observation.

► **Observation 3.** Let  $t$  be a two-dimensional point above the  $x$ -axis. The illumination cone of any  $\alpha$ -floodlight positioned at  $t$  and illuminating the maximum range  $(-\infty, \chi]$  of the  $x$ -axis (i.e.,  $\chi$  is maximized) is delimited by the  $t$ -originating leftward-pointing horizontal ray and the  $t$ -originating downward-pointing ray that forms an angle equal to  $\alpha$  with the positive  $x$ -axis; see Figure 3. Symmetrically, the illumination cone of any  $\alpha$ -floodlight positioned at  $t$  and illuminating the maximum range  $[\chi', +\infty)$  of the  $x$ -axis (i.e.,  $\chi'$  is minimized) is delimited by the  $t$ -originating rightward-pointing horizontal ray and the  $t$ -originating downward-pointing ray that forms an angle equal to  $180^\circ - \alpha$  with the positive  $x$ -axis.

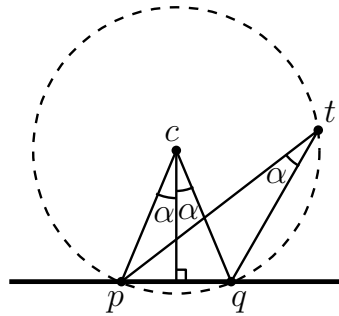
In light of this observation, in the next lemma we show how to efficiently illuminate the unbounded ranges of the  $x$ -axis.



■ **Figure 3** Illuminating a maximum  $x$ -axis range  $(-\infty, \chi]$ .

► **Lemma 4.** For the given set  $S$  of potential floodlight locations, let  $t \in S$  lie on the line supporting  $S$  from below and forming angle  $\alpha$  with the positive  $x$ -axis. Then, the maximum range  $(-\infty, \chi_t]$  of the  $x$ -axis illuminated by an  $\alpha$ -floodlight positioned at  $t$  is no smaller than the maximum range  $(-\infty, \chi]$  of the  $x$ -axis illuminated by an  $\alpha$ -floodlight positioned at any other point in  $S$ ; see Figure 3. A symmetric result holds for the illumination of a range  $[\chi', +\infty)$  of the  $x$ -axis where the points in  $S$  that maximize the illuminated range are those that lie on the line supporting  $S$  from below and forming angle  $180^\circ - \alpha$  with the positive  $x$ -axis.

Next, let  $(-\infty, \chi_{left}]$  and  $[\chi_{right}, +\infty)$  be the maximum such ranges of the  $x$ -axis illuminated by an  $\alpha$ -floodlight positioned at any point in the given location set  $S$ ; the values  $\chi_{left}$  and  $\chi_{right}$  can be computed as described in Lemma 4. Note that  $\chi_{left} < \chi_{right}$  because  $\alpha < 90^\circ$ .

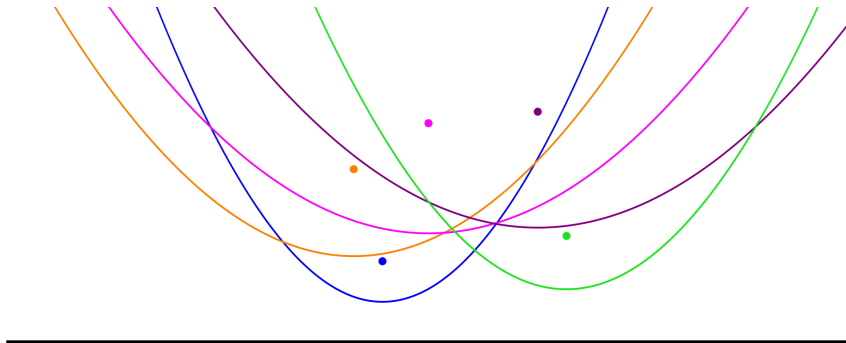


■ **Figure 4** Illuminating a bounded range of the  $x$ -axis with an  $\alpha$ -floodlight positioned at  $t$ .

Now, let us consider the illumination of bounded ranges of the  $x$ -axis. If an  $\alpha$ -floodlight positioned at a point  $t$  above the  $x$ -axis illuminates the range  $[\chi_L, \chi_R]$  of the  $x$ -axis with  $-\infty < \chi_L < \chi_{right}$ , we write that  $illum(t, \chi_L) = [\chi_L, \chi_R]$ ; the range  $illum(t, \chi_L)$  is uniquely defined as the range of the  $x$ -axis swept by the counterclockwise rotation by an angle  $\alpha$  of the  $t$ -originating ray that goes through the point  $(\chi_L, 0)$  on the  $x$ -axis. In addition, we extend this notation and use  $illum(t, -\infty)$  to denote the maximum range of the form  $(-\infty, \chi]$  illuminated by an  $\alpha$ -floodlight at  $t$ ; see Observation 3.

Assuming that  $illum(t, \chi_L) = [\chi_L, \chi_R]$ , let  $p, q$  be the points  $p = (\chi_L, 0)$  and  $q = (\chi_R, 0)$  and let  $r$  be the radius of the circle through  $t, p, q$  (see Figure 4). Then, the distance of the center  $c$  of this circle from the  $x$ -axis is  $r \cos \alpha$ , that is, the ratio of its distance to  $t$  over its distance to the  $x$ -axis is  $1/\cos \alpha$ , which is a constant greater than 1 for any fixed  $\alpha < 90^\circ$ . Therefore, from Section 2, the center  $c$  belongs to a hyperbola branch which we formally define next.

► **Definition 5.** For any positive angle  $\alpha < 90^\circ$  and any point  $t$  above the  $x$ -axis, we define the hyperbola branch  $H_t$  (with the point  $t$  as focus and the  $x$ -axis as directrix) that is the locus of points whose distance from the point  $t$  divided by the (perpendicular) distance from the  $x$ -axis is equal to  $1/\cos \alpha > 1$ .



■ **Figure 5** A set of five point locations and the corresponding hyperbola branches  $H_i$  for  $\alpha = 10^\circ$ .

## 11:6 Illuminating the $x$ -Axis by $\alpha$ -Floodlights

Figure 5 shows the hyperbola branches  $H_t$  for a set of five points and  $\alpha = 10^\circ$ . Then, from the above discussion and from Figure 4, we have:

- **Lemma 6.** *Let  $t$  be a point above the  $x$ -axis and  $\alpha$  a positive angle where  $\alpha < 90^\circ$ . Consider an  $\alpha$ -floodlight positioned at  $t$  which illuminates a range  $[\chi_L, \chi_R]$  of the  $x$ -axis, and let  $p, q$  be the points  $p = (\chi_L, 0)$  and  $q = (\chi_R, 0)$  on the  $x$ -axis. If  $C$  is the circle defined by  $t, p, q$ , then:*
- (i) *The center  $c$  of the circle  $C$  lies on the hyperbola branch  $H_t$  (see Definition 5).*
  - (ii) *The line through  $p$  and  $c$  forms a fixed angle equal to  $90^\circ - \alpha$  with the positive  $x$ -axis. Symmetrically, the line through  $q$  and  $c$  forms a fixed angle equal to  $90^\circ + \alpha$  with the positive  $x$ -axis.*

As we are interested in minimizing the total number of  $\alpha$ -floodlights used, it is important to use floodlights positioned at points of the given set  $S$  of locations such that the illuminated range is maximized. Thus, for  $-\infty \leq \chi < \chi_{right}$ , we denote by  $loc\_max(\chi) \subseteq S$  the set of locations  $t \in S$  such that  $illum(t, \chi)$  is maximized; Lemma 4 implies that if  $\chi = -\infty$ , all these locations belong to a line forming angle  $\alpha$  with the positive  $x$ -axis whereas Figure 4 implies that for  $-\infty < \chi < \chi_{right}$ , they belong to an arc of a circle with endpoints being the endpoints of  $illum(t, \chi)$ .

In order to determine the points in  $loc\_max(\chi)$  for  $\chi$  such that  $\chi_{left} \leq \chi < \chi_{right}$ , we use the properties stated in the following lemma.

- **Lemma 7.** *Let  $t$  be a point in the location set  $S$ . For any real number  $\chi$  such that  $\chi_{left} \leq \chi < \chi_{right}$ , assume that  $illum(t, \chi) = [\chi, \chi_R]$  and let  $p, q$  be the points  $p = (\chi, 0)$  and  $q = (\chi_R, 0)$  on the  $x$ -axis. If  $c$  is the center of the circle  $C$  through  $t, p, q$ , then:*
- (i) *The point  $t$  belongs to  $loc\_max(\chi)$  if and only if the circle  $C$  encloses the entire set  $S$ .*
  - (ii) *The point  $t$  belongs to  $loc\_max(\chi)$  if and only if  $c$  belongs to the upper envelope of the hyperbola branches  $H_u$  for all  $u \in S$ .*
  - (iii) *If  $t$  belongs to  $loc\_max(\chi)$ , then no range  $[\chi'_L, \chi'_R]$  with  $\chi'_L \leq \chi$  and  $\chi'_R > \chi_R$  can be illuminated by an  $\alpha$ -floodlight located at any point of  $S$ .*

### 4 The Algorithm

First, we show that the Axis  $\alpha$ -Illumination Problem admits solutions in which we use one floodlight to illuminate the range  $(-\infty, \chi_{left}]$  and another one to illuminate the range  $[\chi_{right}, +\infty)$  (Figure 6) where  $\chi_{left}, \chi_{right}$  are as defined in Section 3.

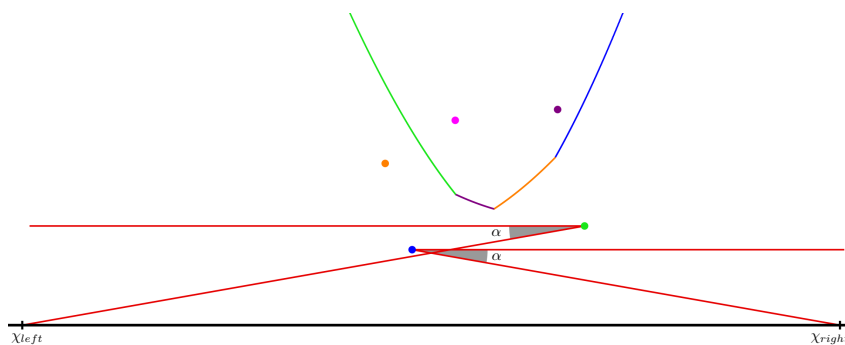
- **Lemma 8.** *Let  $S$  be a given set of locations and let  $\alpha < 90^\circ$ . Then, there exists a solution of the Axis  $\alpha$ -Illumination Problem on  $S$  in which one floodlight is used to illuminate the range  $(-\infty, \chi_{left}]$  and another one to illuminate the range  $[\chi_{right}, +\infty)$  of the  $x$ -axis.*

Therefore, next, we concentrate on the illumination of the range  $[\chi_{left}, \chi_{right}]$  of the  $x$ -axis. First, we note that, as is the case in [5], any instance of the Axis  $\alpha$ -Illumination Problem on a (possibly continuous) location set  $S$  can be reduced into an instance on the set of the convex hull vertices of  $S$ , that is, on a discrete location set.

- **Lemma 9.** *The Axis  $\alpha$ -Illumination Problem on a set  $S$  of planar regions with piece-wise linear boundary has the same solution as the Axis  $\alpha$ -Illumination Problem on the vertices of the convex hull  $CH(S)$  of  $S$ .*

Because of Lemma 9, in the following, we can assume that  $S$  is a discrete set, with  $|S| = n$ .





■ **Figure 6** The upper envelope  $H$  of the hyperbola branches in Figure 5 and the  $\alpha$ -floodlights (shown in red) illuminating the ranges  $(-\infty, \chi_{left}]$  and  $[\chi_{right}, +\infty)$  of the  $x$ -axis.

For the illumination of the range  $[\chi_{left}, \chi_{right}]$  of the  $x$ -axis, Lemma 7(iii) suggests that it is advantageous to use locations that belong to the corresponding  $loc\_max(\cdot)$  set. This is what we do in First Algorithm. The algorithm determines a location in the current  $loc\_max(\chi)$  set (for  $\chi_{left} \leq \chi < \chi_{right}$ ) from the intersection of the upper envelope  $H$  of the hyperbola branches with a line forming angle  $90^\circ - \alpha$  with the positive  $x$ -axis that goes through point  $(\chi, 0)$  on the  $x$ -axis (see Lemma 6(ii)); any such line intersects a hyperbola branch  $H_t$  at exactly one point.

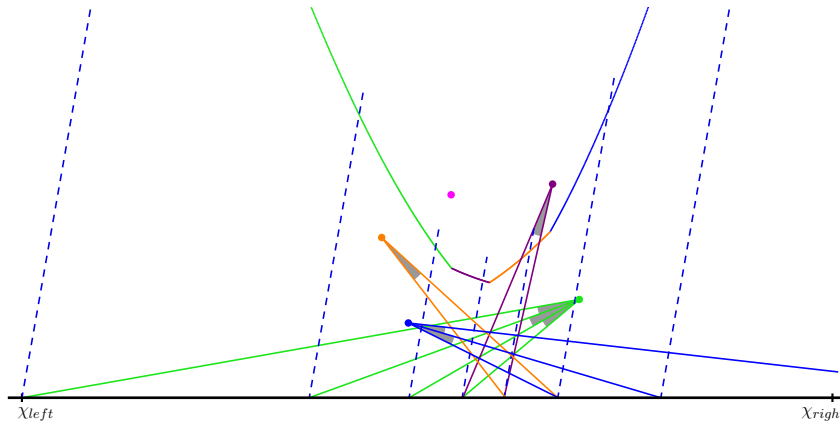
#### First Algorithm

*Input:* a positive angle  $\alpha < 90^\circ$ , a set  $S$  of regions with piece-wise linear boundary above the  $x$ -axis, and the range  $[\chi_{left}, \chi_{right}]$  of the  $x$ -axis to be illuminated

*Output:* a set  $F$  of  $\alpha$ -floodlights at points in  $S$  illuminating the entire  $[\chi_{left}, \chi_{right}]$ , and the corresponding illuminated ranges

1.  $F \leftarrow \emptyset$ ;     { $F$  will store a solution}  
 $current\_x \leftarrow \chi_{left}$ ;  
**while**  $current\_x < \chi_{right}$  **do**  
 $p \leftarrow$  the point  $(current\_x, 0)$  on the  $x$ -axis;  
 $v \leftarrow$  a vertex in  $loc\_max(current\_x)$ ;  
 $q \leftarrow$  point on the  $x$ -axis to the right of  $p$  such that the angle  $\widehat{pvq}$  is equal to  $\alpha$ ;  
 $\chi' \leftarrow$  the  $x$ -coordinate of  $q$ ;  
 $F \leftarrow F \cup \{ (v, [current\_x, \chi']) \}$ ;  
 $current\_x \leftarrow \chi'$ ;  
**return** the resulting  $F$ ;

Clearly, the algorithm can be used to illuminate any subset of the range  $[\chi_{left}, \chi_{right}]$  of the  $x$ -axis, and in general any line segment in the plane. First Algorithm places an  $\alpha$ -floodlight at a point in  $loc\_max(\chi_{left})$ , which is computed from the arc in the upper envelope  $H$  that is intersected by the line through the point  $(\chi_{left}, 0)$  and forming angle  $90^\circ - \alpha$  with the positive  $x$ -axis (Lemma 6 and Lemma 7(ii)), and if the illuminated range is  $[\chi_{left}, \chi_1]$ , in a similar fashion, places an  $\alpha$ -floodlight at a point in  $loc\_max(\chi_1)$ , and if the new illuminated range is  $[\chi_1, \chi_2]$ , it places an  $\alpha$ -floodlight at a point in  $loc\_max(\chi_2)$ , and so on so forth until  $\chi_{right}$  gets illuminated. Figure 7 shows how Step 2 of the First Algorithm works in order to illuminate the range  $[\chi_{left}, \chi_{right}]$  of the  $x$ -axis for the set of five point locations of Figure 5.



■ **Figure 7** Illustration of the operation of First Algorithm on the hyperbola branches of Figure 5 and the set of floodlights produced by the algorithm to optimally illuminate the range  $[\chi_{left}, \chi_{right}]$  of the  $x$ -axis.

If the upper envelope  $H$  of the hyperbola branches  $H_t$  of the vertices of the convex hull  $CH(S)$  of the location set  $S$  is given as a left-to-right sequence of hyperbola arcs along with the associated vertices of  $CH(S)$ , First Algorithm takes  $O(n + k)$  time where  $k$  is the number of  $\alpha$ -floodlights that are eventually used. If the output consists of a list of the required floodlights, this algorithm is output-size sensitive. However, the number  $k$  may be very large even for a location set  $S$  of small descriptive size.

In the following, we propose a modified approach which, in one fell swoop, computes a number of illuminated ranges corresponding to many  $\alpha$ -floodlights placed at the same location in  $S$ ; once we reach an arc  $A$  of the upper envelope  $H$ , we determine all consecutive  $\alpha$ -floodlights that need to be placed at the convex-hull vertex corresponding to the arc  $A$  by using the right endpoint of  $A$ . This approach is used in Step 2 of Second Algorithm for the general Axis  $\alpha$ -Illumination Problem, which we give below.

**Second Algorithm**

*Input:* a positive angle  $\alpha < 90^\circ$  and a set  $S$  of regions with piece-wise linear boundary above the  $x$ -axis

*Output:* a set  $F$  of  $\alpha$ -floodlights located at points in  $S$  illuminating the entire  $x$ -axis, and the corresponding illuminated ranges on the  $x$ -axis

1. compute the convex hull  $CH(S)$  of the given location set  $S$ ;  
 compute the upper envelope  $H$  of the hyperbola branches (each defined by a vertex in  $CH(S)$  and the  $x$ -axis as directrix) and store it as a left-to-right sequence of hyperbola arcs, each associated with the corresponding vertex in  $CH(S)$ ;  
 $v_{first} \leftarrow$  a vertex of  $CH(S)$  as described in Lemma 4 in order to place an  $\alpha$ -floodlight to illuminate the range  $(-\infty, \chi_{left}]$  of the  $x$ -axis;  
 $v_{last} \leftarrow$  a vertex of  $CH(S)$  as described in Lemma 4 in order to place an  $\alpha$ -floodlight to illuminate the range  $[\chi_{right}, +\infty)$  of the  $x$ -axis;  
 $F \leftarrow \{ (v_{first}, (-\infty, \chi_{left})) \}; \quad \{F \text{ will store a solution} \}$

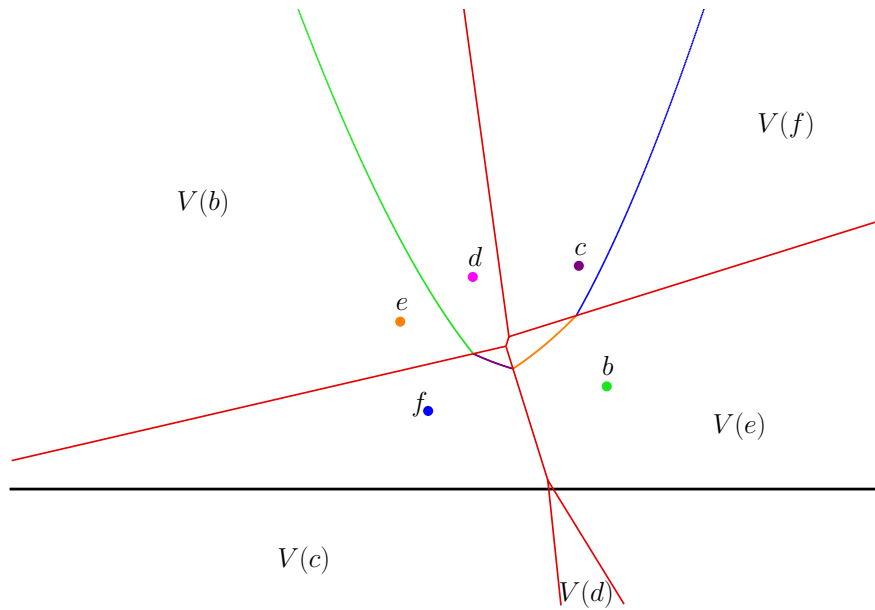
*continued on next page...*



### 4.1 Complexity of Second Algorithm

The upper envelope  $H$  of the hyperbola branches  $H_t$  for  $t \in S$  can be efficiently computed by using the farthest-point Voronoi diagram  $fVD(S)$  of the vertices of the convex hull of the set  $S$ , which coincides with the farthest point Voronoi diagram of the vertices of  $S$  (see Lemma 1(i)). The following lemma and the corollary give the relationship of the  $fVD(S)$  with the arcs in the upper envelope  $H$ .

► **Lemma 10.** *Let  $H$  be the upper envelope of the hyperbola branches of all the vertices in the convex hull  $CH(S)$  of  $S$  and let  $fVD(S)$  be the farthest-point Voronoi diagram of the vertices of  $CH(S)$ . Moreover, let  $H_v$  be the hyperbola branch of a vertex  $v$  of the convex hull of  $S$  and let  $t$  be a point of  $H_v$ . Then, the point  $t$  belongs to  $H$  if and only if  $t$  belongs to the closure of the cell of  $v$  in  $fVD(S)$ .*



■ **Figure 9** The upper envelope  $H$  of the five points shown in Figure 5 and their farthest-point Voronoi diagram (shown in red).

Lemma 10 implies the following corollary; see Figure 9.

- **Corollary 11.** *Let  $H$ ,  $fVD(S)$ ,  $v$ , and  $H_v$  be as in Lemma 10. Then, the following hold.*
- (i) *The part of the hyperbola branch  $H_v$  of a vertex  $v \in CH(S)$  that belongs to  $H$  is precisely the intersection of  $H_v$  with the cell of  $v$  in  $fVD(S)$ .*
  - (ii) *A point  $t \in H$  is a vertex of  $H$  if and only if  $t$  either lies on an edge or is a vertex of  $fVD(S)$ .*
  - (iii) *The size (number of vertices or hyperbola arcs) of the upper envelope  $H$  of the hyperbola branches of all the vertices in the convex hull of the set  $S$  is  $O(|CH(S)|)$ .*

Since each arc of the upper envelope  $H$  produces at most one pair in the solution  $F$  in Step 2 of the Second Algorithm (see Figure 8), Corollary 11(iii) implies that:

► **Corollary 12.** *The size of the solution computed by the Second Algorithm is  $O(n)$  where  $n$  is the total number of vertices of the location set  $S$ . Hence, the same holds for any solution to the Axis  $\alpha$ -Illumination Problem as it is described in Section 1.*

Now we are ready to estimate the complexity of the Second Algorithm. Let  $n$  be the number of vertices of the set  $S$ . The computation of the convex hull of the vertices of  $S$ , which coincides with the convex hull of  $S$ , takes  $O(n \log n)$  time [6]. The computation of the upper envelope of  $H$  can be done by computing first the farthest-point Voronoi diagram of the convex hull vertices of  $S$ , and then the vertices of  $H$  by taking advantage of Corollary 11(ii), the proof of Corollary 11(iii), and the appropriate hyperbola arcs of  $H$  in accordance with Corollary 11(i); the farthest point Voronoi diagram of  $O(n)$  points can be computed in  $O(n \log n)$  time [16] and has  $O(n)$  size (see Lemma 1(ii)), while the remaining work can be done in  $O(n)$  time. Finally, computing  $v_{first}$  and  $v_{last}$  takes  $O(n)$  time and the initialization of  $F$  takes constant time. In total, Step 1 can be completed in  $O(n \log n)$  time.

Each iteration of the **while** loop in Step 2 of the algorithm requires constant time for everything but determining the arc  $A$  intersected by the line  $L$ . Finding the arc  $A$  can be done in  $O(\log n)$  time by using binary search on the  $x$ -monotone upper envelope  $H$ ; in fact, all the arcs  $A$  needed in the different iterations of the **while** loop can be found in  $O(n)$  total time by walking along  $H$  from left to right as needed by the algorithm. Because each iteration of the **while** loop involves a different arc in  $H$  and the total number of arcs is  $O(n)$  (Corollary 11(iii)), the total number of iterations is  $O(n)$ . In addition to the **while** loop, Step 2 contains operations that require constant total time; thus, Step 2 can be completed in  $O(n \log n)$  time. Step 3 takes  $O(n)$  time because we can efficiently merge the appropriate pairs in the set  $F$  computed after Step 2 by processing them in the order they are produced.

Overall, the algorithm takes  $O(n \log n)$  time. The space for computing and storing the upper envelope  $H$  of the hyperbola branches, the farthest-point Voronoi diagram (Lemma 1(ii)), and the solution  $F$  (Corollary 12) is  $O(n)$ . Thus, we conclude with the following theorem.

► **Theorem 13.** *The Second Algorithm correctly computes a solution to the Axis  $\alpha$ -Illumination Problem and requires  $O(n \log n)$  time and  $O(n)$  space, where  $n$  is the number of vertices of the given location set.*

## 5 Concluding Remarks

We proved that the Axis  $\alpha$ -Illumination Problem admits an  $O(n \log n)$ -time and  $O(n)$ -space algorithm where  $n$  is the number of vertices of the given location set. The obvious open question is whether there exists a matching lower bound for that problem or if not, to find a faster algorithm.

A natural extension of our Axis  $\alpha$ -Illumination Problem is the case of a set  $S$  of regions (with piece-wise linear boundary) lying in a polygon and we want to illuminate the boundary of that polygon. This problem may be thought of as a variant of the Art Gallery polygon where the purpose is to guard only the boundary of the input polygon [12]. We believe that the Second Algorithm can be extended to the case where  $S$  lies in a circle or in a convex polygon and we want to illuminate the boundary of that circle/polygon. However, in the case of a simple polygon, the problem is NP-hard [13] and APX-hard [10].

---

## References

- 1 Ahmed Abdelkader, Ahmed Saeed, Khaled A. Harras, and Amr Mohamed. The inapproximability of illuminating polygons by  $\alpha$ -floodlights. In *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, Kingston, Ontario, Canada, August 10-12, 2015*. Queen's University, Ontario, Canada, 2015. URL: <http://research.cs.queensu.ca/cccg2015/CCCG15-papers/25.pdf>.
- 2 Sergey Bereg, José Miguel Díaz-Báñez, Marta Fort, Mario Alberto López, Pablo Pérez-Lantero, and Jorge Urrutia. Continuous surveillance of points by rotating floodlights. *Int. J. Comput. Geom. Appl.*, 24(3):183–196, 2014. doi:10.1142/S0218195914600024.

- 3 Prosenjit Bose, Leonidas J. Guibas, Anna Lubiw, Mark H. Overmars, Diane L. Souvaine, and Jorge Urrutia. The floodlight problem. *Int. J. Comput. Geom. Appl.*, 7(1/2):153–163, 1997. doi:10.1142/S0218195997000090.
- 4 Jurek Czyzowicz, Stefan Dobrev, Benson L. Joeris, Evangelos Kranakis, Danny Krizanc, Ján Manuch, Oscar Morales-Ponce, Jaroslav Opatrny, Ladislav Stacho, and Jorge Urrutia. Monitoring the plane with rotating radars. *Graphs Comb.*, 31(2):393–405, 2015. doi:10.1007/s00373-015-1543-4.
- 5 Jurek Czyzowicz, Eduardo Rivera-Campo, and Jorge Urrutia. Optimal floodlight illumination of stages. In *Proceedings of the 5th Canadian Conference on Computational Geometry, Waterloo, Ontario, Canada, August 1993*, pages 393–398. University of Waterloo, 1993.
- 6 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. URL: <https://www.worldcat.org/oclc/227584184>.
- 7 Jana Dietel, Hans-Dietrich Hecker, and Andreas Spillner. A note on optimal floodlight illumination of stages. *Inf. Process. Lett.*, 105(4):121–123, 2008. doi:10.1016/j.ipl.2007.08.009.
- 8 Vladimir Estivill-Castro, Joseph O’Rourke, Jorge Urrutia, and Dianna Xu. Illumination of polygons with vertex lights. *Inf. Process. Lett.*, 56(1):9–13, 1995. doi:10.1016/0020-0190(95)00129-Z.
- 9 Vladimir Estivill-Castro and Jorge Urrutia. Two-floodlight illumination of convex polygons. In Selim G. Akl, Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 4th International Workshop, WADS ’95, Kingston, Ontario, Canada, August 16-18, 1995, Proceedings*, volume 955 of *Lecture Notes in Computer Science*, pages 62–73. Springer, 1995. doi:10.1007/3-540-60220-8\_51.
- 10 Christodoulos Fragoudakis, Euripides Markou, and Stathis Zachos. Maximizing the guarded boundary of an art gallery is apx-complete. *Comput. Geom.*, 38(3):170–180, 2007. doi:10.1016/j.comgeo.2006.12.001.
- 11 Hiro Ito, Hideyuki Uehara, and Mitsuo Yokoyama. Np-completeness of stage illumination problems. In Jin Akiyama, Mikio Kano, and Masatsugu Urabe, editors, *Discrete and Computational Geometry, Japanese Conference, JCDCG’98, Tokyo, Japan, December 9-12, 1998, Revised Papers*, volume 1763 of *Lecture Notes in Computer Science*, pages 158–165. Springer, 1998. doi:10.1007/978-3-540-46515-7\_12.
- 12 Aldo Laurentini. Guarding the walls of an art gallery. *Vis. Comput.*, 15(6):265–278, 1999. doi:10.1007/s003710050177.
- 13 Joseph O’Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., USA, 1987.
- 14 Joseph O’Rourke. Visibility. In Jacob E. Goodman and Joseph O’Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 643–663. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.ch28.
- 15 T.C. Shermer. Recent results in art galleries (geometry). *Proceedings of the IEEE*, 80(9):1384–1399, 1992. doi:10.1109/5.163407.
- 16 Sven Skyum. A simple algorithm for computing the smallest enclosing circle. *Inf. Process. Lett.*, 37(3):121–125, 1991. doi:10.1016/0020-0190(91)90030-L.
- 17 William L. Steiger and Ileana Streinu. Illumination by floodlights. *Comput. Geom.*, 10(1):57–70, 1998. doi:10.1016/S0925-7721(97)00027-8.
- 18 Kazuo Sugihara, Ichiro Suzuki, and Masafumi Yamashita. The searchlight scheduling problem. *SIAM J. Comput.*, 19(6):1024–1040, 1990. doi:10.1137/0219070.
- 19 Dan Tao and Tin-Yu Wu. A survey on barrier coverage problem in directional sensor networks. *IEEE Sensors Journal*, 15(2):876–885, 2015. doi:10.1109/JSEN.2014.2310180.
- 20 Jorge Urrutia. Art gallery and illumination problems. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. North Holland / Elsevier, 2000. doi:10.1016/b978-044482537-7/50023-1.
- 21 Izu Vaisman. *Analytical Geometry*. World Scientific, 1997. doi:10.1142/3494.


# On Geometric Priority Set Cover Problems

Aritra Banik 

National Institute of Science Education and Research, HBNI, Bhubaneswar, India

Rajiv Raman<sup>1</sup> 

Université Clermont Auvergne, Clermont Auvergne INP, CNRS, Mines Saint-Etienne, LIMOS, F-63000, Clermont-Ferrand, France

Saurabh Ray 

New York University, Abu Dhabi, UAE

---

## Abstract

We study the priority set cover problem for simple geometric set systems in the plane. For pseudo-halfspaces in the plane we obtain a PTAS via local search by showing that the corresponding set system admits a planar support. We show that the problem is APX-hard even for unit disks in the plane and argue that in this case the standard local search algorithm can output a solution that is arbitrarily bad compared to the optimal solution. We then present an LP-relative constant factor approximation algorithm (which also works in the weighted setting) for unit disks via quasi-uniform sampling. As a consequence we obtain a constant factor approximation for the capacitated set cover problem with unit disks. For arbitrary size disks, we show that the problem is at least as hard as the vertex cover problem in general graphs even when the disks have nearly equal sizes. We also present a few simple results for unit squares and orthants in the plane.

**2012 ACM Subject Classification** Theory of computation → Packing and covering problems; Theory of computation → Computational geometry

**Keywords and phrases** Approximation algorithms, geometric set cover, local search, quasi-uniform sampling

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.12

**Funding** *Rajiv Raman*: This work has been partially supported by the French government research program “Investissements d’Avenir” through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20–25).

## 1 Introduction

The priority set cover problem is defined as follows. Given a ground set  $X$  and a set  $\mathcal{S}$  of subsets of  $X$ , where each element  $x \in X$  has an associated priority  $\pi(x)$  and each set  $S \in \mathcal{S}$  has an associated priority  $\pi(S)$ , the goal is to pick the smallest cardinality subset  $\mathcal{S}' \subseteq \mathcal{S}$  s.t. for each  $x \in X$ , there is some  $S \in \mathcal{S}'$  contain  $x$  s.t.  $\pi(S) \geq \pi(x)$ .

We study the priority set cover problem for simple geometric regions in the plane. It is a natural generalization of the set cover problem, and is interesting in its own right. It is also related to the capacitated covering problem via the work of Chakarabarty et al. [11]. Another motivation for studying such problems is that it leads to a better understanding of the limitations of the current techniques and forces us to extend them.

By treating the priority as an additional dimension, these problems can be seen as special cases of three dimensional set cover problems. However, these problems turn out to be significantly harder than the corresponding problems without priority. For example, while the set cover problem with disks in the plane admits a PTAS, the same problem with priorities surprisingly turns out to be APX-hard even for unit disks. This is one of the few problems known that is APX-hard for unit disks. One standard technique that yields a constant factor

---

<sup>1</sup> On leave from IIIT-Delhi, India.





approximation for many geometric covering problems is quasi-uniform sampling [28, 13]. However this fails for the priority set cover problem for unit disks since the *shallow cell complexity* (defined in Section 2) of the corresponding set system can be quadratic. Another common technique used to obtain approximation algorithms is *local search*. The analysis of local search requires showing the existence of a *support*, or the existence of a *local search graph* that come from a hereditary family with sublinear size separators (see [26] and the references therein). However, even for the priority set cover problem with unit disks, such graphs do not exist. In fact we show that the standard local search algorithm may produce solutions that are arbitrarily bad compared to an optimal solution.

We develop techniques to obtain the first  $O(1)$ -factor approximation algorithms for the priority set cover problem with unit disks. The algorithm relies on tools we develop to study the priority set cover problem defined by points and *pseudo-halfspaces* in the plane. For the latter problem we show that the shallow cell complexity of the corresponding set system is linear. For the set cover problem without priorities (or equivalently the priority set cover problem with uniform priorities), the problem is known to be solvable in polynomial time [21]. However, we show that the introduction of priorities renders the problem NP-hard.

The proof is non-trivial and uses a novel approach that might be useful in other settings. We also obtain a PTAS for the priority set cover problem for pseudo-halfspaces via local search. For this, we prove that the corresponding set system admits a planar support. Again due to priorities, the proof is much more subtle than for pseudo-halfspaces without priorities.

An identical proof also yields an  $O(1)$ -approximation for the priority set cover problem with unit squares. In this case, we do not know if the problem is APX-hard, and this remains an intriguing open question.

As a consequence of our results for the priority problem, we immediately obtain  $O(1)$ -approximation algorithms for the capacitated covering problems when combined with the results of Bansal and Pruhs [6], and Chakrabarty et al. [11]. We start with the necessary definitions and results in Section 2, and describe related work in Section 3. In Section 4 we present our results for pseudo-halfspaces. We present our results for disks in Section 5. We conclude with open problems in Section 6.

## 2 Preliminaries

Let  $P$  be a set of points and let  $\mathcal{R}$  be a set of regions in the plane and let  $\pi : P \cup \mathcal{R} \rightarrow \mathbb{R}$  be a function that assigns a priority to each point and each region. We say that a region  $R$  *covers* a point  $p$  if  $R$  contains  $p$  and  $\pi(R) \geq \pi(p)$ . We use the notation  $p \prec R$  for “ $R$  covers  $p$ ”. For any region  $R$ , we denote by  $R(P)$ , the set of points in  $P$  covered by  $R$ . We denote the set system  $(P, \{R(P) : R \in \mathcal{R}\})$  by  $(P, \mathcal{R}, \pi)$  and call it the “set system defined by  $P$  and  $\mathcal{R}$ ”. For any point  $p$ , we denote by  $p(\mathcal{R})$  the set of regions in  $\mathcal{R}$  covering  $p$  and we denote the set system  $(\mathcal{R}, \{p(\mathcal{R}) : p \in P\})$  by  $(\mathcal{R}, P, \pi)$  and call it the “dual set system defined by  $P$  and  $\mathcal{R}$ ”.

The *Priority Set Cover problem* defined by  $P$ ,  $\mathcal{R}$  and  $\pi$  is the set cover problem on the set system  $(P, \mathcal{R}, \pi)$ . In other words, the goal is to find the smallest subset  $\mathcal{R}' \subseteq \mathcal{R}$  s.t. each point in  $P$  is covered by at least one of the regions in  $\mathcal{R}'$ . In the *weighted* variant of this problem, we have a weight  $w_R$  with each region  $R$  and the goal is to minimize the total weight of the regions in  $\mathcal{R}'$  instead of its cardinality. We also consider the *Capacitated Set Cover* problem studied by Chakrabarty et al. [11]. In this problem, we are given a set system  $(X, \mathcal{S})$ , where  $X$  is a set of elements,  $\mathcal{S}$  is a collection of subsets of  $X$  with a weight function  $w : \mathcal{S} \rightarrow \mathbb{R}_+$ , and a capacity function  $c : \mathcal{S} \rightarrow \mathbb{R}_+$ . Each element  $x \in X$  has a demand  $d(x) > 0$ . The objective is to select the smallest weight sub-collection  $\mathcal{S}' \subseteq \mathcal{S}$  such that the total capacity of the sets in  $\mathcal{S}'$  containing any element  $x$  is at least  $d(x)$ . A special case of this problem is the *Set Multicover problem* where the capacity of each set in  $\mathcal{S}'$  is 1.



A finite collection of unbounded  $x$ -monotone curves is called a *family of pseudolines* if every pair of curves intersect in at most one point and at this point the curves cross [3]. Pseudoline arrangements have a rich history and a rich combinatorial structure. See the book [20] for further results on pseudolines. A *family of pseudo-halfspaces* is a collection of closed unbounded regions in the plane whose boundaries form a family of pseudolines.

Given a set system  $(X, \mathcal{S})$ , a *support* is a graph  $G = (X, E)$  s.t. any set  $S \in \mathcal{S}$  induces a connected subgraph of  $G$ . A *planar support* is a support that is planar.

A *plane graph* is a drawing of a planar graph in the plane where the vertices are drawn as points and edges are drawn as interior disjoint simple Jordan curves connecting the points corresponding to the incident vertices. A *triangulation* is a plane graph in which all faces have three vertices.

Given a set system  $(X, \mathcal{S})$ , let  $x(\mathcal{S}) = \{S \in \mathcal{S} : x \in S\}$ . The set system  $(X, \mathcal{S})$  has *shallow cell complexity* [13] function  $f(\cdot, \cdot)$  if for any subset  $\mathcal{S}' \subseteq \mathcal{S}$ , and any  $k \in \mathbb{N}$ , the number of sets of size at most  $k$  in  $\{x(\mathcal{S}') : x \in X\}$  is at most  $f(|\mathcal{S}'|, k)$ . If  $f(|\mathcal{S}'|, k) \leq \phi(|\mathcal{S}'|) \cdot k^c$  for some constant  $c$  where  $\phi(\cdot)$  is a linear function of its argument, we say that the shallow cell complexity of the set system is “linear”. Similarly, if  $\phi(\cdot)$  is a quadratic function of its argument, we say that the shallow cell complexity is “quadratic”.

### 3 Related Work

Packing and covering problems are central topics in computational geometry literature, and studied intensively over several decades. Broadly, there are three main algorithmic techniques: LP-rounding, local search and separator based methods.

The technique of Bronimann and Goodrich [9] reduces any covering problem to an  $\epsilon$ -net question so that if the set system admits an  $\epsilon$ -net of size  $\frac{1}{\epsilon} \cdot f(\frac{1}{\epsilon})$ , then we obtain an LP-relative approximation algorithm with approximation factor  $f(\text{OPT})$  where OPT is the size of the optimal solution. Since set systems of finite VC- dimension admit  $\epsilon$ -nets of size  $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ , this implies an  $O(\log \text{OPT})$  approximation algorithm for covering problems involving such set systems. Similarly, for set systems with low shallow cell complexity, we obtain algorithms with correspondingly small approximation factors (see [28, 13]). In particular if the shallow cell complexity is linear, we obtain constant factor approximation algorithms. Varadarajan [28] showed via the *quasi-uniform* sampling technique how these results can be made to work in the weighted setting. His technique was optimized by Chan et al. [13] who also introduced the notion of *shallow cell complexity* generalizing the notion of *union complexity* from geometric set systems to abstract set systems. Some of these algorithms have also been extended to work in the multicover setting (see [15], [6]). One limitation of the approach in [9] is that even for simple set systems with linear shallow cell complexity, the lower bound on the size of the  $\epsilon$ -net may involve a large constant factor which then translates into a lower bound on the approximation ratio of the corresponding rounding algorithm. For simple set systems, such as that for points and halfspaces in the plane, Har-Peled and Lee [21] gave a polynomial time dynamic programming algorithm. Bringmann et al. [8] show a tight  $O(n^{\sqrt{k}})$  exact algorithm to check if there is a set cover of size at most  $k$ , while for dimensions larger than 3, they show that under ETH, it is not possible to improve upon brute force enumeration.

The local search framework, where one starts with any feasible solution and tries to improve the solution by only making constant size *swaps* (i.e., adding/removing a constant number of elements from the solution), yields a PTAS for several packing and covering problems (see e.g. [25, 14, 5, 19, 26, 7]). This framework also has major limitations: it is not as broadly applicable as the LP-rounding technique (in particular it works only for the

unweighted setting so far), often hard to analyse, and the PTASes they yield have a running time like  $n^{O(1/\epsilon^2)}$  with large constants in the exponent, making them irrelevant for practical applications.

The third type of algorithms consists of separator based methods where some kind of *separator* is used to split the problem instance into smaller problems which can be solved independently and combined to obtain an approximate solution. Hochbaum and Maass [22] used this idea to obtain PTASes for several packing and covering problems. More recently, Adamaszek and Wiese [1, 2] have used this kind of idea for obtaining a QPTAS for independent set problems. Mustafa et al. [24] extend the idea of Adamaszek and Wiese to obtain a QPTAS for weighted set cover problem with pseudodisks in the plane and halfspaces in  $\mathbb{R}^3$ . For unit disks, there have also been attempts to obtain better approximation algorithms that run fast. See [17] for an 18-approximation algorithm that runs in  $O(mn)$  for  $m$  points and  $n$  unit disks.

The priority set cover problem was introduced by Chakrabarty et al. [11] as an approach to solve the capacitated set cover problem. In particular, they showed that an LP-relaxation for the capacitated covering problem with *knapsack cover inequalities*, introduced by Carr et al. [10] in the context of approximation algorithms has an  $O(1)$ -approximation algorithm if there is an LP-relative  $O(1)$ -approximation for the set multicovering problem, and an  $O(1)$  LP-relative approximation for the priority set cover problem.

#### 4 Pseudo-halfspaces

In this section we study the priority set cover problem for pseudo-halfspaces in the plane. For the set cover problem without priorities (or equivalently the priority set cover problem with uniform priorities), the problem is known to be solvable in polynomial time [21]. However, in the full version of the paper we show that the introduction of priorities renders the problem NP-hard.

Let  $H = \{h_1, \dots, h_n\}$  be a set of pseudo-halfspaces and let  $P$  be a set of points in  $\mathbb{R}^2$ . We denote the boundary of  $h_i$  by  $\ell_i$ . We assume that curves  $\ell_1, \dots, \ell_n$  lie in general position i.e., no more than two of them intersect at any point in the plane. Each pseudo-halfspace and each point also has an associated priority. We assume without loss of generality that the priorities of all the pseudo-halfspaces are distinct. For any point  $p$ , let  $H(p)$  denote the subset of pseudo-halfspaces covering  $p$ . We define  $depth(p)$  as  $|H(p)|$ .

► **Lemma 1.** *Let  $t$  be a positive integer. Let  $P'$  be any subset of the points in  $P$  s.t. for any point  $p \in P'$ ,  $depth(p) \leq t$  and for any two distinct points  $p, q \in P'$ ,  $H(p) \neq H(q)$ . Then,  $|P'|$  is  $O(nt^2)$ .*

Note that the above lemma implies that the shallow cell complexity of the set system  $(P, \{H(p) : p \in P\})$  is linear.

For any point  $p$ , we can assume without loss of generality that  $p$  is contained in a bounded cell in the arrangement of the boundaries of the pseudo-halfspaces in  $H(p)$ . This can be guaranteed by adding three *dummy* pseudo-halfspaces of priority larger than all the points in  $P$  so that  $P$  is contained in a bounded cell defined by them. The dummy halfspaces increase the depth of each point by 3 but this does not affect the upper bound claimed above.

In order to prove Lemma 1, we define a new set  $Q$  of points as follows. For every triple of pseudo-halfspaces  $h_i, h_j, h_k$  s.t.  $\pi(h_i) < \pi(h_j), \pi(h_k)$  and  $h_i$  contains  $\ell_j \cap \ell_k$ , we define a point  $q = q(i, j, k)$  located at  $\ell_j \cap \ell_k$  with priority  $\pi(h_i)$ . Note that  $Q$  may contain several points with the same location but with different priorities. However no two points in  $Q$  have the same location and priority.

We map each  $p \in P'$  to a point in  $Q$  as follows. We consider the arrangement of the boundaries of all the pseudo-halfspaces whose priority is at least that of  $p$ . As mentioned above, we can assume that  $p$  lies in a bounded cell  $C$  of this arrangement. Let  $h_i$  be the pseudo-halfspace with the lowest priority in  $H(p)$ . Note that the cell  $C$  must have at least three vertices since this is a pseudo-line arrangement. Thus, it must have a vertex defined by the boundaries of two pseudo-halfspaces  $h_j$  and  $h_k$  other than  $h_i$ . Note that  $\pi(h_i) < \pi(h_j), \pi(h_k)$  and  $h_i$  contains  $\ell_j \cap \ell_k$ . We map  $p$  to  $q(i, j, k)$ .

Since  $\ell_j \cap \ell_k$  is adjacent to at most four cells in the arrangement each of which can contain at most one point of  $P'$ , at most four points in  $P'$  are mapped to  $q(i, j, k)$ . Note also that if  $p$  is mapped to  $q(i, j, k)$  then the depth of  $p$  and  $q(i, j, k)$  differ by at most 2 depending on whether  $h_j$  and  $h_k$  contain  $p$ . Thus, in order to prove the upper bound in Lemma 1, it suffices to prove the upper bound on the number of points in  $Q$  of depth at most  $t$ .

▷ **Claim 2.** Let  $t$  be a positive integer. The number of points in  $Q$  of depth at most  $t$  is  $O(nt^2)$ .

*Proof.* First note that any point  $q = q(i, j, k) \in Q$  has depth at least three since it is contained in the three pseudo-halfspaces  $h_i, h_j$  and  $h_k$ . We first prove that the number of points  $q(i, j, k)$  of depth 3 is  $O(n)$ . We then use the Clarkson-Shor technique [16] to prove the lemma.

If a point  $q(i, j, k) \in Q$  has depth 3 then note that  $h_i$  is the pseudo-halfspace of the highest priority below  $\min\{\pi(h_j), \pi(h_k)\}$  containing  $\ell_j \cap \ell_k$ . This means that if we imagine inserting the pseudo-halfspaces into an initially empty arrangement in the decreasing order of their priorities then  $\ell_j \cap \ell_k$  is a vertex on the boundary of the arrangement until  $h_i$  is inserted at which point it is no longer a vertex on the boundary of the arrangement. Since inserting any pseudo-halfspace can create at most two new vertices on the boundary of the arrangement, the total number of vertices that appear on the boundary of the arrangement throughout the process is  $O(n)$  and since only one point of depth 3 in  $Q$  is located at any such point, the number of points in  $Q$  of depth 3 is also  $O(n)$ .

We now bound the number  $N_t$  of points in  $Q$  of depth  $\leq t$  as follows. Imagine picking a sample of the pseudo-halfspaces where each pseudo-halfspace is picked independently with probability  $\rho = 1/t$ . Let  $Q'$  be the subset of points in  $Q$  that are still present and have depth 3 in the sample. The probability that a point  $q(i, j, k)$  is still present in the sample and has depth 3 in the sample is  $\rho^3(1-\rho)^{t-3}$  since this happens iff  $h_i, h_j$  and  $h_k$  are in the sample but none of remaining  $t-3$  pseudo-halfspaces covering  $q(i, j, k)$  are. Thus,  $\mathbb{E}(Q') \geq N_t \cdot \rho^3(1-\rho)^{t-3}$ . On the other hand since the expected number of pseudo-halfspaces in the sample is  $\rho n$ , we also have  $\mathbb{E}(Q') = O(\rho n)$ . Thus,  $N_t = O\left(\frac{n}{\rho^2(1-\rho)^{t-3}}\right) = O(nt^2)$ . ◁

Lemma 1 now follows. The following theorem is an immediate consequence of Lemma 1 and the results in [13].

► **Theorem 3.** *There is a polynomial time  $O(1)$  LP-relative<sup>2</sup> approximation for the weighted priority set cover problem defined by a set of points and a set of pseudo-halfspaces in the plane.*

► **Theorem 4.** *The capacitated set cover problem defined by points and pseudo-halfspaces in the plane has a polynomial time  $O(1)$ -approximation algorithm.*

<sup>2</sup> with respect to the standard LP relaxation for set cover

**Proof.** Since the shallow-cell complexity of the set system defined by points and pseudo-halfspaces (without priorities) is linear, the result of Bansal and Pruhs [6] implies an  $O(1)$  LP-relative approximation for the multicover problem of this set system. Now, Lemma 1 implies that the set system defined by pseudo-halfspaces and points with priorities is linear, and therefore implies an  $O(1)$  LP-relative approximation for the priority problem via the result of Chan et al. [13]. Now, by the result of Chakrabarty et al. [11], the result follows. ◀

Next, we show that the set system  $(P, H, \pi)$  has a planar support. Our result only requires the boundaries of the pseudo-halfspaces and not the direction of the pseudo-halfspace. Therefore, we consider the following problem: the input is a set  $P$  of points and a set  $L$  of pseudolines with priorities. We construct a plane graph  $G$  with vertex set  $P$  such that the subgraphs induced on the points covered by  $h^+(\ell)$  and  $h^-(\ell)$  are connected. Here,  $h^+(\ell)$  and  $h^-(\ell)$  are two pseudo-halfspaces with priority  $\pi(\ell)$  and boundary  $\ell$ . Such a graph  $G$  is called a support on  $P$  with respect to  $L$ .

The proof is constructive: we process the points in increasing order of priority, and maintain a support graph on the processed points with respect to  $L$ . We additionally maintain that this support graph is a triangulation on the processed points and has the property that each edge in the graph is a simple curve that crosses any  $\ell \in L$  at most once. In order to construct the graph, we use the following well known result called the Levi extension lemma.

► **Lemma 5** (Levi extension Lemma [20]). *Given a pseudoline arrangement  $L$  and two points  $p$  and  $q$  not lying on the same pseudoline in  $L$ , there exists a simple curve  $\ell$  through  $p$  and  $q$  such that  $L \cup \ell$  is a pseudoline arrangement.*

Using the Lemma above, we now construct a planar support.

► **Theorem 6.** *Let  $P$  be a set of  $n$  points,  $L$  a family of pseudolines in  $\mathbb{R}^2$ , and  $\pi : P \cup L \rightarrow \mathbb{R}$  be priorities. Then, there exists a graph  $T$  which is a support on  $P$  w.r.t.  $L$  such that each edge in  $T$  crosses any pseudoline  $\ell \in L$  at most once. Further, for any  $n \geq 3$ ,  $T$  is a triangulation.*

**Proof.** Let  $p_1, \dots, p_n$  be an ordering of the points  $P$  in increasing order of priority. We process the points in this order, and for each  $i = 1, \dots, n$ , we maintain a graph  $T_i$  that is a support graph on the points  $p_1, \dots, p_i$  with respect to  $L$  and such that: any edge of  $T_i$  crosses the boundary of any pseudoline in  $L$  at most once. We call such a graph with its embedding a *nice graph*.

For  $i = 1$ , the graph  $T_1$  consisting of  $p_1$  and no edges clearly satisfies both the conditions, and is a nice graph. For  $i = 2$ , by Lemma 5, there is a curve  $\gamma_{12}$  between  $p_1$  and  $p_2$  such that  $L_2 = \gamma_{12} \cup L$  is a pseudoline arrangement. The edge  $e_{12}$  is defined as  $\gamma_{12}[p_1, p_2]$ , i.e., the segment on  $\gamma_{12}$  between  $p_1$  and  $p_2$ . It is clear that  $T_2$  is a nice graph.

For  $i = 3$ , we use Lemma 5 with the pseudoline arrangement  $L_2$  to construct a pseudoline  $\gamma_{13}$  through points  $p_1$  and  $p_3$ , such that  $L_2 \cup \gamma_{13}$  is a pseudoline arrangement. Invoking Lemma 5 again with  $L_2 \cup \gamma_{13}$ , we obtain a pseudoline  $\gamma_{23}$  between  $p_2$  and  $p_3$ , such that  $L_3 = L_2 \cup \gamma_{13} \cup \gamma_{23}$  is a pseudoline arrangement. The new pseudolines added define edges  $e_{13} = \gamma_{13}[p_1, p_3]$  and  $e_{23} = \gamma_{23}[p_2, p_3]$ . Set  $T_3 = T_2 \cup e_{13} \cup e_{23}$ . It is easy to see that  $T_3$  is a nice graph.

Let  $T_{i-1}$  be the graph constructed for the first  $i - 1$  points, and let  $L_{i-1}$  be the union of  $L$  and the additional pseudolines added in the first  $i - 1$  iterations. We construct  $T_i$  as follows: Let  $p_i$  lie in a triangle  $\Delta$  defined by points  $p_a, p_b$ , and  $p_c$  - note that  $p_i$  may lie in the external face. Suppose that  $p_i$  lies in an interior face of  $T_{i-1}$ . Using Lemma 5

with the arrangement  $L_{i-1}$  lets us construct a pseudoline  $\gamma_{ia}$  through  $p_i$  and  $p_a$ , such that  $L_{i-1} \cup \gamma_{ia}$  is a pseudoline arrangement. This gives us the edge  $e_{ia} = \gamma_{ia}[p_i, p_a]$ . Note that the interior of  $e_{ia}$  lies in  $\Delta$ . Otherwise, if  $e_{ia}$  crosses  $\Delta$ , it crosses one of the pseudolines defining the boundaries of  $\Delta$ , which it can not cross again without violating the assumption that  $L_{i-1} \cup \gamma_{ia}$  is a pseudoline arrangement. If  $p_i$  lies in an external face of  $T_{i-1}$ , we first pick a point  $o$  that lies in an internal face of  $T_{i-1}$ , does not lie on any pseudolines in  $L_{i-1}$  and is not one of the points in  $P$ . Let  $C$  be a unit circle centered at  $o$ . We temporarily apply an inversion<sup>3</sup> with respect to  $C$  to turn the external face in which  $p_i$  lies into an internal face. We then proceed as before and apply the inversion again to undo the first inversion.

By a similar argument, we construct edges  $e_{ib}$  and  $e_{ic}$ , and set  $L_i = L_{i-1} \cup \gamma_{ia} \cup \gamma_{ib} \cup \gamma_{ic}$ , and  $T_i = T_{i-1} \cup e_{ia} \cup e_{ib} \cup e_{ic}$ . Since  $p_i$  lies on  $\gamma_{ia}, \gamma_{ib}$  and  $\gamma_{ic}$  it follows that the interiors of  $e_{ia}, e_{ib}$  and  $e_{ic}$  are pairwise internally disjoint. By construction, the edges  $e_{ia}, e_{ib}$  and  $e_{ic}$  lie in the same face of  $T_{i-1}$ , and therefore do not intersect the interiors of edges in  $T_{i-1}$ .

To show that  $T_i$  is a nice graph, we still need to show that  $T_i$  is a support on  $P_i$  with respect to  $L$ . Consider a pseudo-halfspace  $h = h(\ell)$  defined by a pseudoline  $\ell \in L$ . First, we will show that  $h' = h \setminus \cup_{e \in T_i} \text{interior}(e)$  is path connected. If not, let  $p$  and  $q$  be two points in  $h'$  that cannot be connected by a continuous path in  $h'$ . Let  $H_p$  be the set of points reachable from  $p$  i.e.,  $H_p$  is the set of points in  $h$  to which there is path from  $p$  in  $h'$ , and let  $H_q$  be the set of points reachable from  $q$ . Since  $h$  is connected and we have removed only the interiors of the edges that are pairwise non-intersecting, there are no vertices on the boundary of  $H_p$  and  $H_q$ . Since each edge is a simple curve, it implies that there is an edge  $e$  that separates  $H_p$  and  $H_q$ , i.e., the boundary of  $e$  crosses  $\ell$  twice. This leads to a contradiction.

Now, let  $u$  and  $v$  be any two points in  $P_i$  that are contained in  $h$ . We show that there is a path between  $u$  and  $v$  in the subgraph of  $T_i$  induced by  $h \cap P_i$ . Since  $u$  and  $v$  do not lie in the interior of any of the edges in  $T_i$ ,  $u, v \in h'$ . By the previous argument, there is a simple curve  $\sigma$  in  $h'$  joining  $u$  and  $v$ . Observe that adjacent points of  $P_i$  on  $\sigma$  lie in the same face of  $T_i$ , and therefore are adjacent in  $T_i$ , since  $T_i$  is a triangulation. This implies that there is a path in the subgraph of  $T_i$  induced by  $h \cap P_i$  joining  $u$  and  $v$ .

Let  $p_k$  be the point of highest priority that is contained in  $h$ . Then, by the fact that  $T_k$  is a support on  $P_k$  with respect to  $L$  and  $h$  covers exactly the points in  $P_k \cap h$ , we conclude that the subgraph of  $T_k$  induced by the points covered by  $h$  is connected. Since  $T_n$  contains  $T_k$  as subgraph, the subgraph of  $T_n$  induced by the points in  $P = P_n$  covered by  $h$  is also connected. The theorem follows.  $\blacktriangleleft$

An immediate consequence of the above theorem is the following.

► **Corollary 7.** *The set system  $(P, H, \pi)$  admits a planar support.*

► **Corollary 8.** *The set system  $(H, P, \pi)$  admits a planar support if the union of the pseudo-halfspaces in  $H$  do not cover the entire plane.*

**Proof.** If there is a point  $o$  in the plane that is not covered by any of the pseudo-halfspaces in  $H$  then using the duality between points and pseudolines [4], we can map points to pseudo-halfspaces and pseudo-halfspaces to points while maintaining incidences. Then, Corollary 7 implies that the set system  $(H, P, \pi)$  admits a planar support.  $\blacktriangleleft$

<sup>3</sup> [https://en.wikipedia.org/wiki/Inversive\\_geometry](https://en.wikipedia.org/wiki/Inversive_geometry)

The next theorem follows directly from the above result and the results in [26] and [27] which show that the existence of a suitable planar support implies a PTAS.

► **Theorem 9.** *The Set Cover, Hitting Set, Point Packing, and Region Packing problems with priorities defined by a set of points and pseudo-halfspaces in the plane admit a PTAS. The set multi-cover problem defined by a set of points and pseudo-halfspaces with priorities admits a  $(2 + \epsilon)$ -approximation algorithm for any  $\epsilon > 0$ .*

The definitions of the problems mentioned in the theorem above without the priorities can be found in [26] and [27] and naturally extend to the version with priorities.

## 5 Disks

Chan et al. [12] showed that the set cover problem with horizontal and vertical strips in the plane is APX-hard. The input is a set  $P$  of  $n$  points in the plane and a set  $\mathcal{S}$  of vertical or horizontal strips of the form  $V(a, b) = \{(x, y) \in \mathbb{R}^2 : a \leq x \leq b\}$  or  $H(a, b) = \{(x, y) \in \mathbb{R}^2 : a \leq y \leq b\}$ . We show below that the set system defined by axis aligned strips and points in the plane can be implemented using disks and points with priorities. This implies that the set cover problem defined by points and axis aligned strips in the plane can be reduced to the priority set cover problem defined by a set of points and unit disks in the plane in polynomial time.

► **Theorem 10.** *Given a set  $\mathcal{S}$  of horizontal and vertical strips and a set of points  $P$  in the plane, we can map each strip  $S \in \mathcal{S}$  to a unit radius disk  $S'$  and each point  $p \in P$  to another point  $p'$  with appropriate priorities so that  $S'$  covers  $p'$  iff  $S$  contains  $p$ .*

**Proof.** Let  $n$  be the number of points in  $P$ . We can assume without loss of generality that the points in  $P$  lie on an  $n \times n$  grid  $G$  and have cartesian coordinates  $(i, j)$  where  $i, j \in [n]$ . For convenience, we will assume that  $P$  consists of all points in the grid since if the theorem holds for such a  $P$  then it certainly holds for any subset of it. We refer to the point with cartesian coordinates  $(i, j)$  in  $G$  as  $p_{ij}$ .

Let  $D$  be a unit radius disk centered at the origin  $o$ . We first define  $n + 2$  points  $z_0, z_1, \dots, z_{n+1}$  on the boundary of  $D$  as follows. The point  $z_i$  has polar coordinates  $(1, \theta_i)$  i.e., cartesian coordinates  $(\cos \theta_i, \sin \theta_i)$  where  $\theta_i = i \cdot \frac{\pi}{4(n+1)}$ . Let  $\hat{u}$  be a unit vector along the positive  $x$ -axis. We map the point  $p_{ij}$  on the grid to the point  $q_{ij} = z_i + j \cdot \frac{\epsilon}{n} \hat{u}$  where  $\epsilon$  is a sufficiently small constant. We assign priority  $n - j$  to the point  $q_{ij}$ . Note that the points on  $j^{\text{th}}$  column of the grid are mapped to the points  $q_{1j}, \dots, q_{nj}$  that lie on the boundary of a unit radius disk whose center is at  $o + j \cdot \frac{\epsilon}{n} \hat{u}$ . We will call this disk  $C_j$ . The points on the  $i^{\text{th}}$  row of the grid  $G$  are mapped to points on a horizontal segment of length  $\epsilon$  whose left end-point is  $z_i$ . We denote the set of points  $\{q_{ij} : i, j \in [n]\}$  by  $Q$ .

Consider any vertical strip  $S = V(a, b)$  which contains the points in columns  $a, a + 1, \dots, b$  of  $G$ . We map this strip to the disk  $S'$  which is identical to  $C_b$  but has priority  $n - a$ . It can be verified that a point  $q_{ij} \in Q$  is covered by  $S'$  iff the corresponding point  $p_{ij}$  is covered by  $S$ . Now consider any horizontal strip  $S = H(a, b)$  which contains the points in rows  $a, a + 1, \dots, b$  of  $G$ . We map this strip to a disk  $S'$  defined as follows. Let  $u$  be the mid-point on the arc on  $\partial D$  joining  $z_{a-1}$  and  $z_a$ . Similarly, let  $v$  be the mid-point on the arc on  $\partial D$  joining  $z_b$  and  $z_{b+1}$ .  $S'$  is the unique disk of unit radius whose center lies outside  $D$  and whose boundary intersects the boundary of  $D$  at the points  $u$  and  $v$ . For sufficiently small  $\epsilon$ ,  $S'$  contains exactly the subset of points in  $Q$  that correspond to the points in  $P$  contained in  $S$ . We assign a priority of  $n$  to  $S'$  so that it covers all the points it contains. The theorem follows. ◀



The following is an immediate consequence of Theorem 10 and the results in [12].

► **Corollary 11.** *The priority set cover problem defined by a set of points  $P$  and a set of unit radius disks  $\mathcal{D}$  in the plane is APX-hard.*

► **Corollary 12.** *The shallow cell complexity of the set system defined by unit radius disks and points with priority is quadratic.*

**Proof.** This follows from Theorem 10 and the fact that the shallow cell complexity of the set system defined by axis aligned strips and points in the plane is quadratic. To see the latter, consider  $n$  disjoint horizontal strips  $H_1, \dots, H_n$  and  $n$  disjoint vertical strips  $V_1, \dots, V_n$ . Then for every pair of indices  $i, j \in [n]$ ,  $H_i$  and  $V_j$  intersect at a point in the plane that is not contained in any other strip. ◀

► **Remark.** Since the shallow cell complexity is quadratic, the quasi-uniform sampling technique [28, 13] cannot be directly applied to obtain a constant factor approximation for the priority set cover problem defined by points and unit disks in the plane.

The next lemma shows that the standard local search algorithm does not work for the priority set cover problem defined by unit disks and points in the plane. For minimization problems, the standard local search algorithm is the following. It has a fixed parameter  $k$ . The algorithm starts with any feasible solution and tries to decrease the size of the solution by removing at most  $k$  elements from the current solution and adding fewer elements to the solution without violating feasibility. When such improvements are not possible it returns the current solution.

► **Lemma 13.** *For any positive integer  $k$ , there exist instances of the priority set cover problem defined by unit disks and points in the plane such that the standard local search algorithm with parameter  $k$  does not yield a solution with a bounded approximation ratio.*

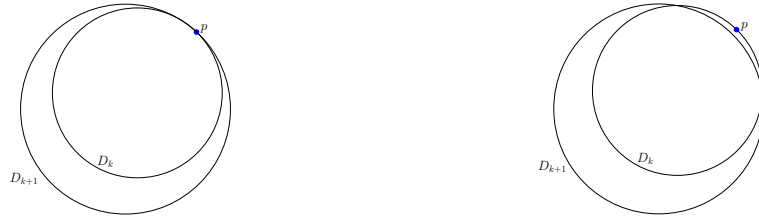
**Proof.** We will construct an instance of the set cover problem defined by points and axis aligned strips in the plane for which the standard local search algorithm with swap size  $k$  does not yield a solution with a bounded approximation ratio. This along with Theorem 10 implies the statement in the theorem.

Let  $H_1, \dots, H_m$  be  $m$  disjoint horizontal strips and let  $V_1, \dots, V_n$  be  $n$  disjoint vertical strips where  $m \gg n > k$ . For any  $i, j \in [n]$ , let  $p_{ij}$  be a point in  $H_i \cap V_j$ . Consider the set cover problem defined the all the horizontal and vertical strips and the points  $\{p_{ij} : i, j \in [n]\}$ . Then, the horizontal strips form a locally optimal solution i.e., the solution cannot be improved by swapping out at most  $k$  strips from this solution and swapping in fewer strips. This is because if any horizontal strip is dropped, we would need to add all the vertical strips, of which there are more than  $k$ , in order to obtain a feasible solution. Since  $m \gg n$  this solution is arbitrarily large compared to the optimal solution formed by the vertical strips. ◀

We now show that we can construct an arbitrarily set of disks such that every pair intersects at a depth 2. This implies that the shallow-cell complexity of this set system is quadratic.

► **Theorem 14.** *For any positive integer  $n$ , there exist a set of  $n$  disks  $\mathcal{D} = \{D_1, \dots, D_n\}$  whose radii are nearly equal (i.e., the ratio of any two of the radii can be made arbitrarily close to 1) and a set of points  $P$  with  $\binom{n}{2}$  points s.t. for any pair of disks  $i, j \in [n]$  s.t.  $i < j$ , there exists a point  $p_{ij} \in P$  which is covered by only the disks  $D_i$  and  $D_j$  among the disks in  $\mathcal{D}$ .*

## 12:10 On the Geometric Priority Set Cover Problem



■ **Figure 1** Constructing disk  $D_{k+1}$  from disk  $D_k$ . The radius of  $D_{k+1}$  is only slightly bigger than the radius of  $D_k$ , the difference being arbitrarily small. The centers of the two disks are also arbitrarily close to each other.

**Proof.** We show the existence of a family of disks  $\mathcal{D}$  so that for any  $j \in [n]$ , if we consider the arrangement of the disks in  $\mathcal{D}_j := \{D_1, \dots, D_j\}$ , then i) the boundary of every disk in  $\mathcal{D}_j$  contributes at least one arc to the boundary of the union of the disks in  $\mathcal{D}_j$  and ii) the boundary of  $D_j$  intersects the boundary of every other disk non-tangentially on the boundary of the union of the disks in  $\mathcal{D}_j$ .

We show the existence of such disks by induction on the number of disks. The base case is  $n = 1$  and is trivially true. Suppose that we have shown this for  $n = k$  for some  $k \geq 1$ . We now show that we can add a disk  $D_{k+1}$  to the existing collection so that the above properties i) and ii) hold for  $n = k + 1$ . By the inductive hypothesis,  $D_k$  contributes an arc to the boundary of the union of the disks in  $\mathcal{D}_k$ . Let  $p$  be the mid-point of such an arc. Without loss of generality assume that the radius of  $D_k$  is 1. We construct the disk  $D_{k+1}$  in two steps. See Figure 1.

First we tentatively set  $D_{k+1}$  to be a disk of radius  $1 + \delta_{k+1}$  for some  $\delta_{k+1} > 0$  so that  $D_{k+1}$  contains  $D_k$  and the boundaries of  $D_k$  and  $D_{k+1}$  intersect tangentially at  $p$ . We set  $\delta_{k+1}$  to be sufficiently small so that none of the vertices in the arrangement of disks in  $\mathcal{D}_k$  lie in the region  $D_{k+1} \setminus D_k$ . At this point  $D_{k+1}$  almost satisfies the required properties. Since  $\delta_{k+1}$  is very small  $D_{k+1}$  is almost the same as  $D_k$  and is obtained by “growing”  $D_k$  slightly. This means that for each  $j < k$ ,  $D_j$  still contributes an arc to the boundary of the union of disks in  $\mathcal{D}_k$  and the boundary of  $D_{k+1}$  intersects the boundary  $D_j$  on the boundary of the union of the disks in  $\mathcal{D}_{k+1}$ .  $D_{k+1}$  however intersects  $D_k$  tangentially at  $p$  which also means that the boundary of  $D_k$  does not contribute any arc to the boundary of the union of the disks in  $\mathcal{D}_{k+1}$ . To fix these problems, we move the center of  $D_{k+1}$  by a distance  $\epsilon_{k+1} > 0$  in the direction  $c - p$  where  $c$  is the center of  $D_k$ . We choose  $\epsilon_{k+1}$  to be sufficiently small so that during the movement, the boundary of  $D_k$  does not touch any of the vertices in the arrangement of the disks in  $\mathcal{D}_k$ . It can be checked that after this movement the set of disks  $\mathcal{D}_{k+1}$  satisfies the two properties. This concludes the inductive proof. By making  $\epsilon_{k+1}$  and  $\delta_{k+1}$  appropriately small for every  $k$ , we can ensure that the disks have nearly equal radii.

We assign the priority  $n - i$  to the disk  $D_i$ . For any  $i < j$ , we define the point  $p_{ij}$  to be the point located where the boundaries of  $D_i$  and  $D_j$  intersect on the boundary of the union of the disks in  $\mathcal{D}_j$  and having priority  $n - j$ . Note that the point  $p_{ij}$  is covered by both  $D_i$  and  $D_j$ . It is however not contained in any of other disks in  $\{D_1, \dots, D_j\}$  and it is not covered by any of the disks in  $\{D_{j+1}, \dots, D_n\}$  since those disks have a lower priority than that of  $p_{ij}$ . ◀

► **Remark.** It can be shown that the values of  $\epsilon_{k+1}$  and  $\delta_{k+1}$  in the above proof can be chosen so that they can be encoded using  $\text{poly}(n)$  bits. The formal proof of this statement will appear in the extended version of the paper.



► **Corollary 15.** *The priority set cover problem defined by a set of points and nearly equal size disks in the plane does not admit a strongly polynomial time approximation algorithm with approximation factor smaller than 1.36 unless  $P = NP$ . Under the unique games conjecture, this implies that the priority set cover problem does not admit a strongly polynomial time algorithm with approximation factor smaller than 2.*

**Proof.** We give an approximation preserving reduction from vertex cover to priority set cover problem defined by a set of points and a set of nearly equal sized disks in the plane. The corollary then follows from the results known for the vertex cover problem [18, 23]. Given a graph  $G$  with  $n$  vertices  $v_1, \dots, v_n$  and  $m$  edges, we use Theorem 14 to obtain a set  $\mathcal{D}$  of  $n$  disks and a set  $P$  with  $\binom{n}{2}$  points. The disk  $D_i$  corresponds to the vertex  $v_i$  of  $G$ . For each edge  $\{v_i, v_j\}$  in  $G$ , we retain the point  $p_{ij} \in P$ . We remove all other points. Let  $Q$  be the set of points retained. Then, the priority set cover problem defined by the points in  $Q$  and the disks in  $\mathcal{D}$  is equivalent to the vertex cover problem in  $G$ . ◀

We now show that there exists a polynomial time constant factor approximation algorithm for the weighted priority set cover problem defined by a set of points  $P$  and a set of unit disks  $\mathcal{D}$  in the plane. We first define an LP-relaxation for this problem:

$$\text{minimize } \sum_{D \in \mathcal{D}} w_D x_D \quad \text{s.t. for each } p \in P : \sum_{D \in \mathcal{D} : p \in D} x_D \geq 1.$$

We will show that there is a polynomial time algorithm that outputs a solution of size at most a constant times the value  $\text{OPT}_{LP}$  of an optimal solution to the above LP.

► **Theorem 16.** *There is a polynomial time LP-relative  $O(1)$ -approximation algorithm for the weighted priority set cover problem defined by a set of points  $P$  and a set of unit radius disks  $\mathcal{D}$  in the plane.*

**Proof.** We first prove the theorem for disks containing a common point which without loss of generality is assumed to be the origin  $o$ . Let  $x^*$  be an optimal solution to the LP-relaxation. Consider any one of the four quadrants formed by the axes and consider the priority set cover problem restricted to that quadrant. Note that  $x^*$  is also a feasible solution to this problem. Since the boundaries of any two disks containing  $o$  intersect at most once in the quadrant, they behave like pseudo-halfspaces with respect to the quadrant. By Lemma 1, the shallow cell complexity of the corresponding set system is linear and therefore quasi-uniform sampling [13] yields an LP-relative  $O(1)$ -approximation for this problem. Since there are four quadrants, by taking the union of the solutions for each of the quadrants, we obtain an LP-relative  $O(1)$  approximation for the priority set cover problem where all disks contain a common point  $o$ .

Now, we consider the case of a general set of unit disks in the plane. We partition the given set of disks into a constant number of families s.t. each family consists of disjoint groups of disks so that disks in each group intersect at a common point but disks from different groups do not intersect.

An  $O(1)$ -approximation for the priority set cover problem defined by such a family of disks follows from the fact that disks in different groups don't interact and for each group we have an  $O(1)$ -approximation.

The families of the required type are obtained as follows. We place a uniform grid over the plane having cells of size  $\sqrt{2} \times \sqrt{2}$  i.e., each cell is a square with diameter 2. Since there are only a finite number of disks, we can also choose the grid in such a way that the center of each disk lies in the interior of some cell. We associate each unit disk with the cell

## 12:12 On the Geometric Priority Set Cover Problem

containing its center. Note that the disks associated with a cell contain the center of the cell. Next we color the cells with a constant number of colors so that two cells whose centers have distance less than 4 get distinct colors. The disks associated with cells of a single color then is a family of the required type. Each distinct color defines a family. Let  $\mathcal{F}_1, \dots, \mathcal{F}_k$  be the families obtained. As argued above, there is an  $O(1)$ -approximation algorithm for the priority set cover problem defined by any particular family. However, a point can belong to disks of several different families. So, we need a way to assign each point to a particular family. To do this, we consider an optimal solution  $x^*$  to the priority cover problem defined by all disks and points. Since, for any point  $p$ , we have that  $\sum_{D \in \mathcal{D}: p \prec D} x_D \geq 1$ , there is some family  $\mathcal{F}_i$  s.t.  $\sum_{D \in \mathcal{F}_i: p \prec D} x_D \geq 1/k$ . We assign the point  $p$  to one such family.

Let  $P_i$  be the subset of points assigned to family  $\mathcal{F}_i$ . Then note that  $\hat{x}_D = k \cdot x_D^*$ ,  $D \in \mathcal{F}_i$  is a feasible solution to the LP-relaxation for the priority set cover problem defined by the points in  $P_i$  and the disks in  $\mathcal{F}_i$ . Let  $S_i$  be a solution to this problem using an LP-relative  $O(1)$ -approximation. This means that  $S_i$  has weight at most  $O(1) \cdot \sum_{D \in \mathcal{F}_i} w_D \hat{x}_D$ . Note that  $S = \bigcup_{i=1}^k S_i$  is a solution to the weighted priority set cover problem defined by all disks and points and has weight at most

$$O(1) \cdot \sum_{i=1}^k \sum_{D \in \mathcal{F}_i} w_D \hat{x}_D \leq O(1) \cdot \sum_{i=1}^k \sum_{D \in \mathcal{F}_i} k \cdot w_D x_D^* = O(1) \cdot k \cdot \sum_D w_D x_D^* = O(1) \cdot \sum_D w_D x_D^*$$

since  $k$  is a constant.  $S$  is therefore an LP-relative  $O(1)$ -approximation to the weighted priority set cover problem defined by all disks and points. ◀

► **Theorem 17.** *There is a polynomial time LP-relative  $O(1)$ -approximation algorithm for the weighted priority set cover problem defined by a set of points  $P$  and a set of unit squares  $\mathcal{S}$  in the plane.*

**Proof.** The proof is identical to the proof of Theorem 16, except that we have unit squares instead of unit disks. ◀

► **Theorem 18.** *The capacitated set cover problem with unit squares, or unit disks admits an  $O(1)$ -approximation.*

**Proof.** The result of Chakrabarty et al. [11] shows that there exists an  $O(1)$ -approximation for the capacitated set cover problem whenever we have an  $O(1)$ -LP-relative approximation for the multicover problem, and an  $O(1)$ -LP-relative approximation for the priority cover problem. The result of Bansal and Pruhs [6] implies an  $O(1)$ -LP-relative approximation for the multicover problem, and Theorem 16 implies an  $O(1)$ -LP-relative approximation for the priority problem. The result for unit disks follows. For unit squares, the result similarly follows from that of Bansal and Pruhs [6] and Theorem 17. ◀

## 6 Conclusion

We studied the priority set cover problem for several simple geometric set systems in the plane. For pseudo-halfspaces in the plane we were able to obtain a PTAS but for unit disks in the plane the problem is APX-hard and we obtained a constant factor approximation. Obtaining a relatively small approximation factor is an interesting open question. For unit squares in the plane we also obtain a constant factor approximation but it is not clear if the problem is APX-hard. In fact even for orthants in the plane (possibly containing orthants of opposite types), it is not clear if there is a PTAS. In particular, we do not know if the

standard local search yields a PTAS. There are instances showing that the corresponding set system does not admit a planar support. Another interesting open problem is to obtain a constant factor approximation algorithms for disks or square of arbitrary size in the plane.

---

## References

- 1 Anna Adamaszek and Andreas Wiese. Approximation schemes for maximum weight independent set of rectangles. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, FOCS '13, pages 400–409, Washington, DC, USA, 2013. IEEE Computer Society.
- 2 Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 645–656, 2014.
- 3 Pankaj K. Agarwal, Ravid Cohen, Dan Halperin, and Wolfgang Mulzer. Dynamic maintenance of the lower envelope of pseudo-lines. *CoRR*, abs/1902.09565, 2019. [arXiv:1902.09565](https://arxiv.org/abs/1902.09565).
- 4 Pankaj K. Agarwal and Micha Sharir. Pseudo-line arrangements: Duality, algorithms, and applications. *SIAM J. Comput.*, 34(3):526–552, 2005. doi:10.1137/S0097539703433900.
- 5 Rom Aschner, Matthew J. Katz, Gila Morgenstern, and Yelena Yuditsky. Approximation schemes for covering and packing. In *WALCOM: Algorithms and Computation*, volume 7748 of *Lecture Notes in Computer Science*, pages 89–100. Springer Berlin Heidelberg, 2013.
- 6 Nikhil Bansal and Kirk Pruhs. Weighted geometric set multi-cover via quasi-uniform sampling. *JoCG*, 7(1):221–236, 2016.
- 7 Aniket Basu Roy, Sathish Govindarajan, Rajiv Raman, and Saurabh Ray. Packing and covering with non-piercing regions. *Discrete & Computational Geometry*, 2018.
- 8 Karl Bringmann, Sándor Kisfaludi-Bak, Michal Pilipczuk, and Erik Jan van Leeuwen. On geometric set cover for orthants. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 26:1–26:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.26.
- 9 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(1):463–479, 1995.
- 10 Robert D. Carr, Lisa Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 106–115. ACM/SIAM, 2000. URL: <http://dl.acm.org/citation.cfm?id=338219.338241>.
- 11 Deeparnab Chakrabarty, Elyot Grant, and Jochen Könemann. On column-restricted and priority covering integer programs. In Friedrich Eisenbrand and F. Bruce Shepherd, editors, *Integer Programming and Combinatorial Optimization, 14th International Conference, IPCO 2010, Lausanne, Switzerland, June 9-11, 2010. Proceedings*, volume 6080 of *Lecture Notes in Computer Science*, pages 355–368. Springer, 2010. doi:10.1007/978-3-642-13036-6\_27.
- 12 Timothy M. Chan and Elyot Grant. Exact algorithms and apx-hardness results for geometric packing and covering problems. *Comput. Geom. Theory Appl.*, 47(2), February 2014.
- 13 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1576–1585, 2012.
- 14 Timothy M. Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012. doi:10.1007/s00454-012-9417-5.

## 12:14 On the Geometric Priority Set Cover Problem

- 15 Chandra Chekuri, Kenneth L. Clarkson, and Sarel Har-Peled. On the set multicover problem in geometric settings. *ACM Trans. Algorithms*, 9(1):9:1–9:17, 2012. doi:10.1145/2390176.2390185.
- 16 Kenneth L Clarkson and Peter W Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(5):387–421, 1989.
- 17 Gautam K. Das, Robert Fraser, Alejandro López-Ortiz, and Bradford G. Nickerson. On the discrete unit disk cover problem. In Naoki Kato and Amit Kumar, editors, *WALCOM: Algorithms and Computation - 5th International Workshop, WALCOM 2011, New Delhi, India, February 18-20, 2011. Proceedings*, volume 6552 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2011. doi:10.1007/978-3-642-19094-0\_16.
- 18 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.
- 19 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the  $\log n$  barrier. In *Algorithms – ESA 2010 - 18th Annual European Symposium, Liverpool, United Kingdom, September 6–8, 2010, Proceedings*, pages 243–254, 2010.
- 20 B. Grünbaum and Conference Board of the Mathematical Sciences. *Arrangements and Spreads*. Regional conference series in mathematics. Conference Board of the Mathematical Sciences, 1972. URL: <https://books.google.co.in/books?id=M2NjAQAACAAJ>.
- 21 Sarel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *J. Comput. Geom.*, 3(1):65–85, 2012. doi:10.20382/jocg.v3i1a4.
- 22 Dorit S. Hochbaum and Wolfgang Maass. Fast approximation algorithms for a nonconvex covering problem. *Journal of algorithms*, 8(3):305–323, 1987.
- 23 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.
- 24 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Quasi-polynomial time approximation scheme for weighted geometric set cover on pseudodisks and halfspaces. *SIAM Journal on Computing*, 44(6):1650–1669, 2015.
- 25 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.
- 26 Rajiv Raman and Saurabh Ray. Planar support for non-piercing regions and applications. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 69:1–69:14, 2018.
- 27 Rajiv Raman and Saurabh Ray. Improved approximation algorithm for set multicover with non-piercing regions. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPICs*, pages 78:1–78:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ESA.2020.78.
- 28 Kasturi Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 641–648, 2010.

# The Complexity of Sharing a Pizza

Patrick Schneider   

Department of Mathematical Sciences, University of Copenhagen, Denmark

---

## Abstract

Assume you have a 2-dimensional pizza with  $2n$  ingredients that you want to share with your friend. For this you are allowed to cut the pizza using several straight cuts, and then give every second piece to your friend. You want to do this fairly, that is, your friend and you should each get exactly half of each ingredient. How many cuts do you need?

It was recently shown using topological methods that  $n$  cuts always suffice. In this work, we study the computational complexity of finding such  $n$  cuts. Our main result is that this problem is PPA-complete when the ingredients are represented as point sets. For this, we give a new proof that for point sets  $n$  cuts suffice, which does not use any topological methods.

We further prove several hardness results as well as a higher-dimensional variant for the case where the ingredients are well-separated.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** pizza sharing, Ham-Sandwich theorem, PPA, computational geometry, computational complexity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.13

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.06752>

**Funding** The author has received funding from the European Research Council under the European Unions Seventh Framework Programme ERC Grant agreement ERC StG 716424 – CASe.

## 1 Introduction

### 1.1 Mass partitions

The study of *mass partitions* is a large and rapidly growing area of research in discrete and computational geometry. It has its origins in the classic *Ham-Sandwich theorem* [44]. This theorem states that any  $d$  mass distributions in  $\mathbb{R}^d$  can be simultaneously bisected by a hyperplane. A mass distribution  $\mu$  in  $\mathbb{R}^d$  is a measure on  $\mathbb{R}^d$  such that all open subsets of  $\mathbb{R}^d$  are measurable,  $0 < \mu(\mathbb{R}^d) < \infty$  and  $\mu(S) = 0$  for every lower-dimensional subset  $S$  of  $\mathbb{R}^d$ . A vivid example of this result is, that it is possible to share a 3-dimensional sandwich, consisting of bread, ham and cheese, with a friend by cutting it with one straight cut such that both will get exactly half of each ingredient. This works no matter how the ingredients lie. In fact, as Edelsbrunner puts it, this even works if the cheese is still in the fridge [19].

But what if there are more ingredients, for example on a pizza? One way to bisect more than  $d$  masses is to use more complicated cuts, such as algebraic surfaces of fixed degree [44] or piece-wise linear cuts with a fixed number of turns [28, 40]. Another option is to use several straight cuts, as introduced by Bereg et al. [8]: Consider some arrangement of  $n$  hyperplanes in  $\mathbb{R}^d$ . The cells of this arrangement allow a natural 2-coloring, where two cells get a different color whenever they share a  $(d - 1)$ -dimensional face. We say that an arrangement bisects a mass distribution  $\mu$  if the cells of each color contain exactly half of  $\mu$ . See Figure 1 for an illustration. It was conjectured by Langerman that any  $nd$  mass distributions in  $\mathbb{R}^d$  can be simultaneously bisected by an arrangement of  $n$  hyperplanes ([30], see also [5]). In a series of papers, this conjecture has been resolved for 4 masses in the



© Patrick Schneider;

licensed under Creative Commons License CC-BY 4.0

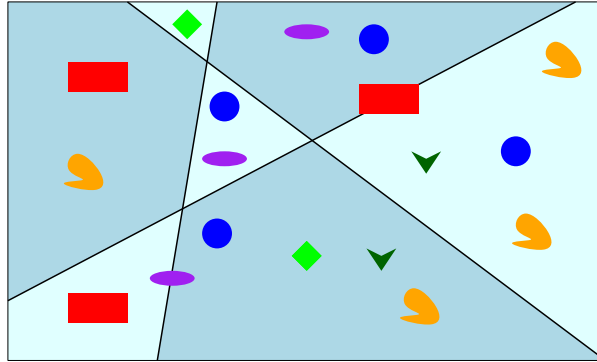
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 13; pp. 13:1–13:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A bisection of 6 masses with 3 cuts.

plane [5], for any number of masses in any dimension that is a power of 2 (and thus in particular also in the plane) [26] and in a relaxed setting for any number of masses in any dimension [41]. However, the general conjecture remains open.

There are many other variants of mass partitions that have been studied, see e.g. [27, 38] for recent surveys. In this work, we focus on the algorithmic aspects of the 2-dimensional variant of bisections with hyperplane arrangements, that is, bisections with line arrangements. For this, let us formally define the involved objects.

► **Definition 1** (Partition induced by an arrangement of oriented lines). *Let  $A = (\ell_1, \dots, \ell_n)$  be a set of oriented lines in the plane. For each  $\ell_i$ , define  $R_i^+$  and  $R_i^-$  the part of the plane on the positive and negative side of  $\ell_i$ , respectively. Define  $R^+(A)$  as the part of the plane lying in an even number of  $R_i^+$  and not on any of the  $\ell_i$ . Similarly, define  $R^-(A)$  as the part of the plane lying in an odd number of  $R_i^+$  and not on any of the  $\ell_i$ . Now,  $R^+(A)$  and  $R^-(A)$  are disjoint, and they partition  $\mathbb{R}^2 \setminus \{\ell_1, \dots, \ell_n\}$  into two parts.*

Note that reorienting one line just swaps  $R^+(A)$  and  $R^-(A)$ , so up to symmetry, the two sides are already determined by the underlying unoriented line arrangement. We will thus often forget about the orientations and just say that a mass is bisected by a line arrangement. Hubbard and Karasev [26] have shown the following:

► **Theorem 2** (Planar pizza cutting theorem [26]). *Any  $2n$  mass distributions in the plane can be simultaneously bisected by an arrangement of  $n$  lines.*

From an algorithmic point of view, we want to restrict our attention to efficiently computable mass distributions.

► **Definition 3** (Computable mass distribution). *A computable mass distribution is a continuous function  $\mu$  which assigns to each arrangement of  $n$  oriented lines two values  $\mu(R^+(A))$  and  $\mu(R^-(A))$ , such that  $\mu(R^+(A)) + \mu(R^-(A)) = \mu(R^+(A')) + \mu(R^-(A'))$  for any two arrangements  $A$  and  $A'$ . We further assume that  $\mu$  can be computed in time polynomial in the description of the input arrangement.*

We now have that the following problem always has a solution.

► **Definition 4** (PIZZACUTTING). *The problem PIZZACUTTING takes as input  $2n$  computable mass distributions  $\mu_1, \dots, \mu_{2n}$  and returns an arrangement  $A$  of oriented lines in the plane such that for each  $i$  we have  $\mu_i(R^+(A)) = \mu_i(R^-(A))$ .*



An important special case of masses are point sets with the counting measure. They do not quite fit the above framework of mass distributions, as the number of points on a line can be non-zero. This can however be resolved by rounding: we say that a line arrangement bisects a point set  $P$ , if there are at most  $\lfloor \frac{|P|}{2} \rfloor$  many points of  $P$  in both parts. Note that if the number of points in  $P$  is odd, this implies that at least one point needs to lie on some line. In the following, we will assume that all point sets are in general position, that is, no three points lie on a common line. With this, we can assume that for each point set at most one point lies on a line. Standard arguments (see e.g. [33]) show that the existence of partitions for mass distributions imply the analogous result for point sets with this definition of bisection. Alternatively, we give a direct proof of the following in Section 4.

► **Corollary 5** (Discrete planar pizza cutting theorem). *Any  $2n$  point sets in the plane can be simultaneously bisected by an arrangement of  $n$  lines.*

Thus, also the following discrete version always has a solution.

► **Definition 6** (DISCRETEPIZZACUTTING). *The problem DISCRETEPIZZACUTTING takes as input  $2n$  point sets  $P_1, \dots, P_{2n}$  in general position in the plane and returns an arrangement  $A$  of oriented lines in the plane such that for each  $i$  we have  $|P_i \cap R^+(A)| = |P_i \cap R^-(A)| = \lfloor \frac{|P_i|}{2} \rfloor$ .*

The pizza cutting problem can be viewed as a higher-dimensional generalization of the consensus halving and necklace splitting problems.

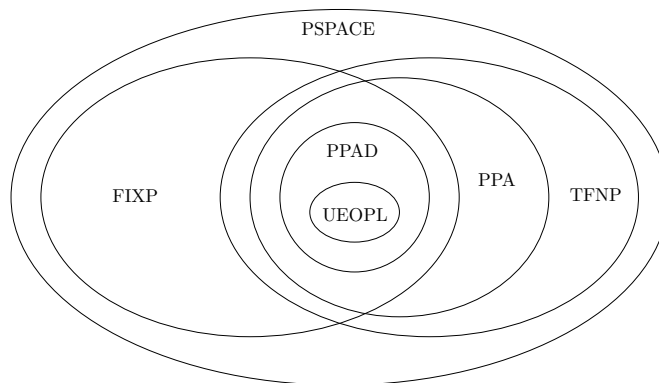
► **Definition 7** (CONSENSUSHALVING/NECKLACESPLITTING). *The problem CONSENSUSHALVING takes as input  $n$  valuation functions  $v_1, \dots, v_n$  on the interval  $[0, 1]$  and returns a partition of  $[0, 1]$  into  $n + 1$  intervals (that is, using  $n$  cuts), each labeled “+” or “−”, such that for each valuation function we have  $v_i(\mathcal{I}^+) = v_i(\mathcal{I}^-)$  (where  $\mathcal{I}^x$  denotes the union of intervals labeled “ $x$ ”). The problem NECKLACESPLITTING is the same, but taking as input  $n$  point sets, again using the above definition of bisections of point sets.*

Again, a solution to the problems is always guaranteed to exist. In the case of mass distributions, this result is known as the *Hobby-Rice theorem* [25]. For necklaces, the statement holds even for the generalized problem of sharing with more than two people [2, 3]. In this work, whenever we refer to the *Necklace splitting theorem*, we mean the version for two people.

Finally, for all above problems, we can also consider the decision version, where we are given one more measure or point set than the number that can always be bisected, and we need to decide whether there still is a bisection. We denote these problems by adding “DECISION” to their name.

## 1.2 Algorithms and complexity

Most proofs of existence of certain mass partitions use topological methods, which, by their nature, are not algorithmic. Thus, there has been quite some effort in developing algorithms that find these promised partitions, ideally efficiently. Arguably the most famous result in this direction are the algorithms for Ham-Sandwich cuts by Lo, Steiger and Matoušek [32, 31]. While in the plane, their algorithm runs in linear time, in general the runtime shows an exponential dependency on the dimension. This curse of dimensionality seems to be a common issue for many algorithmic version of mass partition problems, and most problems have only been studied from an algorithmic point of view in low dimensions, where the constructed algorithms either rely on a relatively small space of solutions or a simplified proof which allows for an algorithmic formulation, see e.g. [1, 6, 37].



■ **Figure 2** Containment relations of some complexity classes for total problems.

The curse of dimensionality was made explicit for the first time by Knauer, Tiwary and Werner, who showed that deciding whether there is a Ham-Sandwich cut through a given point in arbitrary dimensions is  $W[1]$ -hard (and thus also NP-hard) [29]. More recently, in several breakthrough papers, Filos-Ratsikas and Goldberg have shown that computing Ham-Sandwich cuts in arbitrary dimensions is PPA-complete, and so are NECKLACESPLITTING and CONSENSUSHALVING, the latter even in an approximation version [23, 24].

The class PPA was introduced in 1994 by Papadimitriou [36]. It captures search problems, where the existence of a solution is guaranteed by a parity argument in a graph. More specifically, the defining problem is the following search problem in a (potentially exponentially sized) graph  $G$ : given a vertex of odd degree in  $G$ , where  $G$  is represented via a polynomially-sized circuit which takes as input a vertex and outputs its neighbors, find another vertex of odd degree. The class PPA is a subclass of TFNP (Total Function NP), which are total search problems where solutions can be verified efficiently.

A subclass of PPA that is of importance in this work is UEOPPL [22], which is a subclass of PPAD [36]. PPAD is similar to PPA, but instead of an undirected graph, we are given a directed graph in which each vertex has at most one predecessor and at most one successor. We are given a vertex without predecessor, and our goal is to find another vertex without predecessor or successor. If we are further given a potential function which strictly increases on a directed path such that there is a unique vertex with maximal potential, finding this vertex is the defining problem for the class UEOPPL.

The class UEOPPL is related to mass partitions through the fact that finding the unique discrete Ham-Sandwich cut in the case that the point sets are *well-separated* is in UEOPPL [14]. We say that  $k$  point sets  $P_1, \dots, P_k$  in  $\mathbb{R}^d$  are well-separated, if for no  $d$ -tuple of them their convex hulls can be intersected with a  $(d-2)$ -dimensional affine subspace.<sup>1</sup> This definition extends to masses by forbidding intersections of affine subspaces with the convex hulls of their supports. In fact, for well-separated masses and point sets, the  $\alpha$ -Ham-Sandwich theorem states that it is always possible to simultaneously cut off an arbitrary given fraction from each mass or point set with a single hyperplane [4, 43].

The class PPAD has so far mostly been related to the computation of Nash and market equilibria [10, 11, 12, 13, 15, 16, 20, 34, 39, 42, 45].

<sup>1</sup> Admittedly, this would be a very weird pizza.



Finally, the two last classes that are relevant for this work are FIXP and  $\exists\mathbb{R}$ . The class FIXP is the class of problems that can be reduced to finding a Brouwer fixed point [21], whereas  $\exists\mathbb{R}$  is the class of decision problems which can be written in the existential theory of the reals.

In the context of mass partitions, apart from the above mentioned results on Ham-Sandwich cuts, Consensus halvings and Necklace splittings, some of the above classes also appear in the complexity of *Square-Cut Pizza sharing*. In this variant, introduced by Karasev, Roldán-Pensado and Soberón,  $n$  masses in the plane are bisected by a cut which is a union of at most  $n$  axis-parallel segments, or in other words, a piecewise linear cut with at most  $n - 1$   $90^\circ$ -turns [28]. It was recently shown by Deligkas, Fearnley and Melissourgos that finding such a cut even for restricted inputs is PPA-complete, finding a cut where a constant number of additional turns are allowed is PPAD-hard, and that the corresponding decision version is NP-hard [17]. For more general masses, they show the problem to be FIXP-hard and the decision version to be  $\exists\mathbb{R}$ -complete. This work was heavily influenced and motivated by their paper: apart from the different setting, their results are very similar from the results in this work, showing the relation between those two pizza cutting variants. Indeed, in a new version of their paper, the authors of [17] prove some of the same hardness results as this work, and also some stronger hardness results for approximate bisections.

### 1.3 Our contributions

Our main contribution is that DISCRETEPIZZACUTTING is PPA-complete. While the hardness is rather straight-forward, the containment requires some more work. We give a new proof for the existence of pizza cuttings for point sets which, while inspired by the ideas of Hubard and Karasev, uses only elementary geometric techniques which allow us to place the problem in PPA.

We further prove that PIZZACUTTING is FIXP-hard and that PIZZACUTTINGDECISION is  $\exists\mathbb{R}$ -hard and that finding the minimum number of cuts required to bisect an instance of DISCRETEPIZZACUTTING is NP-hard.

Finally, for well-separated masses, we show that the  $\alpha$ -Ham-Sandwich theorem generalizes to pizza cuttings. For point sets in fixed dimensions, we give a linear time algorithm to find such a cut, whereas for arbitrary dimensions we place the problem in UEOPL.

Many of our results are more or less direct applications of known results. In particular, all hardness results follow from a proof of the existence of a Consensus Halving from the existence of a pizza cut. We present this proof and the hardness results in Section 2. In Section 3, we consider the case of well-separated masses and point sets, as some of the ideas are needed for our containment proof. The part where the most new ideas are used is Section 4, where we show the containment of DISCRETEPIZZACUTTING in PPA.

## 2 Hardness results

In this section, we give a proof of existence of Consensus halvings and necklace splittings using the planar pizza cutting theorem. This proof gives a natural reduction to the corresponding algorithmic problems, and thus a variety of hardness results follow.

► **Lemma 8.** *The planar pizza cutting theorem implies the Hobby-Rice theorem.*

**Proof.** Consider the moment curve  $\gamma$  in  $\mathbb{R}^2$ , that is, the curve parametrized by  $(t, t^2)$ . Note that any line  $\ell$  intersects  $\gamma$  in at most two points, call them  $g_1$  and  $g_2$ , with  $g_1$  being to the left of  $g_2$  (in the case of a single or no intersection, we may consider  $g_1 = -\infty$  or  $g_2 = \infty$

## 13:6 The Complexity of Sharing a Pizza

or both). Let  $x_1$  and  $x_2$  be the projections of  $g_1$  and  $g_2$  to the  $x$ -axis under the projection  $\pi(x, y) := x$ . Consider now a half-plane  $h$  bounded by  $\ell$  and consider its intersection with  $\gamma$ . If  $h$  lies below  $\ell$ , then  $h \cap \gamma$  projects to the interval  $[x_1, x_2]$ . If  $h$  lies above  $\ell$ , then  $h \cap \gamma$  projects to  $(-\infty, x_1] \cup [x_2, \infty)$ . Similarly, if  $\ell$  is vertical,  $h \cap \gamma$  projects either to  $(-\infty, x_1]$  or  $[x_2, \infty)$ . Thus, in all cases  $h \cap \gamma$  projects to an interval or the complement of an interval, which, in a slight abuse of notation, we denote by  $\pi(h)$ .

Given a valuation function  $v$ , we now want to define a mass distribution  $\mu$  in the plane. For this, it is enough to just define  $\mu$  on all half-planes. This we can do using the above observations: for any half-plane  $h$ , we define  $\mu(h) := v(\pi(h))$ .

This way, we have defined  $n$  mass distributions. Now, we do the same thing, but shift the interval  $[0, 1]$ , which is the support of the valuation functions, to the interval  $[2, 3]$ . More formally, we consider the projection  $\varphi(x, y) := x - 2$  and, given a valuation function  $v$ , define a mass distribution  $\eta$  as  $\eta(h) := v(\varphi(h))$ .

We have now defined  $2n$  mass distributions. By the pizza cutting theorem, there exists an arrangement  $A = (\ell_1, \dots, \ell_n)$  of  $n$  lines which simultaneously bisect these mass distributions. Consider the intervals  $I_1$  and  $I_2$  on  $\gamma$  defined by  $t \in [0, 1]$  and  $t \in [2, 3]$ . As there are at most  $2n$  intersections of  $A$  and  $\gamma$ , by the pigeonhole principle there are at most  $n$  intersections in one of them, say  $I_1$ . Let  $i_1, \dots, i_n$  be the projections of these intersections under the projection  $\pi$ . We claim that  $i_1, \dots, i_n$  simultaneously bisect  $v_1, \dots, v_n$ .

To show this, consider some valuation function  $v_i$ . By construction, we now have that  $v_i(\mathcal{I}^+) = \mu_i(R^+(A)) = \mu_i(R^-(A)) = v_i(\mathcal{I}^+)$ , which proves the claim. In the case where  $I_2$  has at most  $n$  intersections, we can do the same argument, replacing  $\pi$  with  $\varphi$  and  $\mu$  with  $\eta$ . ◀

Note that all the steps in the proof can be computed in polynomial time. Thus, as CONSENSUSHALVING is FIXP-hard [18], we immediately get the following:

► **Corollary 9.** *PIZZACUTTING is FIXP-hard.*

In the discrete setting, the above proof can be phrased even simpler: for each point  $x$  in  $[0, 1]$ , we just define two points  $(x, x^2)$  and  $(x + 2, (x + 2)^2)$ . As NECKLACESPLITTING is PPA-hard [24], we get that

► **Corollary 10.** *DISCRETEPIZZACUTTING is PPA-hard.*

Clearly, the construction in the proof above also works for more than  $n$  valuation functions. In [18], it was shown that deciding whether  $n + 1$  valuation functions can be bisected with  $n$  cuts is  $\exists\mathbb{R}$ -hard. It thus follows that it is  $\exists\mathbb{R}$ -hard to decide whether  $2n + 2$  masses can be bisected by  $n$  lines. However, in the case where  $n$  is even, we can also only use  $\pi$  and map all valuation functions to a single interval on  $\gamma$ , which analogously proves the Hobby-Rice theorem for even  $n$ . From an asymptotic point of view, this restriction to even values does not matter, so using this reduction, we get the following slightly stronger statement.

► **Corollary 11.** *PIZZACUTTINGDECISION is  $\exists\mathbb{R}$ -hard.*

Finally, it was shown in [9, 35] that finding the minimal number of cuts required to split a necklace is NP-hard. We again get the analogous result for discrete pizza cuttings.

► **Corollary 12.** *Finding the minimal number of lines that simultaneously bisect a family of  $2n$  point sets is NP-hard.*

### 3 Well-separated point sets

In this section we consider well-separated mass distributions and point sets. We generalize the  $\alpha$ -Ham-Sandwich theorem to pizza cuttings.

► **Theorem 13.** *Let  $\mu_1, \dots, \mu_{nd}$  be  $nd$  well-separated mass distributions in  $\mathbb{R}^d$ . Given a vector  $\alpha = (\alpha_1, \dots, \alpha_{nd})$ , with each  $\alpha_i \in [0, 1]$ , there exists an arrangement  $A$  of  $n$  oriented hyperplanes such that for each  $i \in \{1, \dots, nd\}$  we have  $\mu_i(R^+(A)) = \alpha_i \mu_i(\mathbb{R}^d)$ .*

**Proof.** By the  $\alpha$ -Ham-Sandwich theorem [4], for any  $d$  well-separated mass distributions  $\mu_1, \dots, \mu_d$  and any vector  $(\alpha_1, \dots, \alpha_d)$ , there exists a unique single hyperplane cutting each mass distribution in the required ratio. By the definition of well-separatedness, this hyperplane does not intersect the support of any other mass distribution. Partition the point set into  $n$  parts of  $d$  point sets each. For each part, pick some oriented hyperplane which intersects the support of all masses in this part. This defines an arrangement  $B$  of oriented hyperplanes. For each mass  $\mu_i$ , consider the intersection of its support with the positive side of the hyperplane intersecting it, as well as with  $R^+(B)$ . If these two coincide, set  $\alpha'_i := \alpha_i$ , otherwise set  $\alpha'_i := 1 - \alpha_i$ . Now, taking the  $\alpha$ -Ham-Sandwich cut for the vectors  $\alpha'$  gives the required arrangement. ◀

We call such an arrangement an  $\alpha$ -Pizza cut. From the discrete  $\alpha$ -Ham-Sandwich theorem [43], we analogously get the discrete version of the above.

► **Corollary 14.** *Let  $P_1, \dots, P_{nd}$  be  $nd$  well-separated point sets in  $\mathbb{R}^d$ . Given a vector  $\alpha = (k_1, \dots, k_{nd})$ , where each  $k_i$  is an integer with  $0 \leq k_i \leq |P_i|$ , there exists an arrangement  $A$  of  $n$  oriented hyperplanes such that for each  $i \in \{1, \dots, nd\}$  we have  $|P_i \cap R^+(A)| = k_i$ .*

In [14], it was shown that the problem of computing an  $\alpha$ -Ham-Sandwich cut for point sets is in UEOPL. As our  $\alpha$ -Pizza cuts are just a union of  $\alpha$ -Ham-Sandwich cuts, their result generalizes to our setting.

► **Corollary 15.** *The problem of computing an  $\alpha$ -Pizza cut for point sets is in UEOPL.*

► **Remark 16.** The computation of the vector  $\alpha'$  does not directly translate into the setting of UEOPL. However, in [14], they start with an arbitrary hyperplane, and then rotate it in a well-defined fashion to the solution. This immediately translates to our setting: we just start with an arbitrary arrangement, and the choice of the direction of rotation works analogously to the choice in [14].

Further, Bereg [7] has shown that an  $\alpha$ -Ham-Sandwich cut for  $d$  point sets of  $m$  points total in  $\mathbb{R}^d$  can be computed in time  $m2^{O(d)}$ . In particular, if  $d$  is fixed, this algorithm runs in linear time. Again, we get the same result for  $\alpha$ -Pizza cuts.

► **Corollary 17.** *Let  $P_1, \dots, P_{nd}$  be well-separated points sets in  $\mathbb{R}^d$  with  $\sum_{i=1}^{nd} |P_i| = m$ . Then an  $\alpha$ -Pizza cut for  $P_1, \dots, P_{nd}$  can be computed in time  $m2^{O(d)}$ .*

**Proof.** Partition the point sets into parts  $P_{(i-1)d+1}, \dots, P_{id}$ , for  $i \in \{1, \dots, n\}$ . Compute the vector  $\alpha'$  in time  $O(nd)$ . As  $m \geq nd$ , the runtime of the second part dominates the total runtime. For each  $i \in \{1, \dots, n\}$ , use Bereg's algorithm to compute the  $\alpha$ -Ham-Sandwich cut for  $P_{(i-1)d+1}, \dots, P_{id}$ . It follows that the solution is an  $\alpha$ -Pizza cut. The runtime of the algorithm is

$$\sum_{i=1}^n (|P_{(i-1)d+1}| + \dots + |P_{id}|) 2^{O(d)} = \sum_{i=1}^{nd} |P_i| 2^{O(d)} = m 2^{O(d)}. \quad \blacktriangleleft$$

## 4 Containment results

In this section, we prove that DISCRETEPIZZACUTTING is in PPA. We do this by giving a new proof of the discrete planar pizza cutting theorem, which allows for an algorithm in PPA. Before we go into the details of the proof, we briefly sketch the main ideas. The main structure of our proof is similar to and inspired by the original proof of Hubard and Karasev [26], but we replace their topological arguments with easier ones that only use the combinatorics of point sets, but hence do not work for the more general case of mass distributions.

The main idea is that we continuously transform well-separated point sets into the point sets which we want to bisect. In the beginning of this process, we have several bisections, namely one for each partition of the labels of the point sets into pairs, and this number is odd. During the process, we pull these bisecting arrangements along. Every time the orientation of some triple of points changes, it can happen that one of the arrangements is not bisecting anymore. The main step in the proof is, that in these cases, we can always slightly change this arrangement so that it is again bisecting, or that there is a second arrangement that also is not bisecting anymore. In other words, some bisections might vanish during the process, but if they do, then they always vanish in pairs. This step is also where our proof differs from the one by Hubard and Karasev.

Once we have this, the remainder of the proof is rather simple: as we started with an odd number of bisections, and they always vanish in pairs, the number of bisections is always odd, and thus in particular at least 1. Further, we can build a graph where each vertex corresponds to a point set in our process with an arrangement, where the vertices are connected whenever one arrangement is the pulled along version of the other one, or when both point sets are the same and the arrangements are the two arrangements that vanish at this point of the process. (In the end, some of these connections will be paths instead of single edges.) Adding an additional vertex which we connect to all starting arrangements, we get a graph in which the only odd-degree vertices are this additional vertex and the final solutions.

### 4.1 A proof of the discrete planar pizza cutting theorem

We now proceed to give a detailed proof of the discrete planar pizza cutting theorem (Corollary 5). Let  $P_1, \dots, P_{2n}$  be the point sets we wish to bisect. Let further  $m := \sum_{i=1}^{2n} |P_i|$ .

► **Lemma 18.** *We may assume that each  $P_i$  contains an odd number of points.*

**Proof.** For each point set  $P_i$  with an even number of points, add some arbitrary new point  $q_i$  to  $P_i$ , such that the point sets are still in general position. Take some bisecting arrangement  $A$  of the resulting point sets. As all of these point sets consist of an odd number of points in general position, each of them must contain a point  $p_i$  which lies on a line of  $A$ . As each line in  $A$  can pass through at most two points by the general position assumption, there is exactly one such point in each point set. Now, remove  $q_i$  again. If  $p_i = q_i$ , the arrangement still bisects  $P_i$ . Otherwise, one side, without loss of generality  $R^+$ , contains one point too few. Rotate the line through  $p_i$  slightly so that  $p_i$  lies in  $R^+$ . Now the arrangement again bisects  $P_i$ . ◀

So, from now on we may assume that each  $P_i$  contains an odd number of points. Let  $Q_1, \dots, Q_{2n}$  be point sets of the same size, that is,  $|Q_i| = |P_i|$ , that are well-separated. Match each point  $q \in Q_i$  with a point  $p \in P_i$  and consider for each such pair the map  $\varphi(t) := tp + (1-t)q$ . These maps define a map  $\Phi(t)$ , which assigns to each  $t$  the point set defined by the  $\varphi(t)$ 's.

During this process, some orientations of triples of points change. We may assume, that no two triples change their orientation at the same time. Further, as each point crosses each line spanned by two other points at most once, the total number of orientation changes is in  $O(m^3)$ . Thus, the interval  $[0, 1]$  is partitioned into  $O(m^3)$  subintervals, in each of which the orientations of all triples stay the same, that is, the *order type* is invariant. At the boundaries of these subintervals, some three points are collinear.

Taking a representative of each subinterval and the point sets at their boundaries, we thus get a sequence of point sets

$$Q_i = P_i^{(0)}, P_i^{(0.5)}, P_i^{(1)}, \dots, P_i^{(k)}, P_i^{(k.5)}, P_i^{(k+1)}, \dots, P_i^{(C)} = P_i,$$

where  $C \in O(m^3)$  is the number of orientation changes,  $P^{(k)} := \bigcup_{i=1}^{2n} P_i^{(k)}$  is a point set in general position, and  $P^{(k.5)} := \bigcup_{i=1}^{2n} P_i^{(k.5)}$  is a point set with exactly one collinear triple. We will mainly work with the sets  $P^{(k)}$ , the sets  $P^{(k.5)}$  are just used to make some arguments easier to understand.

As argued before, for each  $P^{(k)}$ , any bisecting arrangement  $A$  contains exactly one point  $p_i^{(k)}(A)$  of each  $P_i^{(k)}$  on one of its lines. In particular,  $A$  is defined by a set of pairs of points  $(p_i^{(k)}(A), p_j^{(k)}(A))$ , where each pair defines one line of the arrangement.

► **Lemma 19.** *For  $P^{(0)}$ , the number of bisecting arrangements is odd.*

**Proof.** It is known that for 2 separated point sets, each of odd size, there is a unique Ham-Sandwich cut (see e.g. [43]). We have seen in the proof of Theorem 13, that for well-separated point sets, each bisecting arrangement corresponds to  $n$  Ham-Sandwich cuts, each for a pair of point sets. Thus, the number of bisecting arrangements is the same as the number of partitions of  $2n$  elements into pairs. This number is  $(2n - 1)!! = (2n - 1)(2n - 3) \cdots 3 \cdot 1$ , which is a product of odd numbers, and thus odd. ◀

Let us now follow some arrangement  $A$  through the process. More precisely, Let  $A^{(0)}$  be a bisecting arrangement for  $P^{(0)}$ , defined by pairs of points  $(p_i^{(0)}(A), p_j^{(0)}(A))$ . We now consider the sequence of arrangements  $A^{(k)}$ , where each  $A^{(k)}$  is defined by the corresponding pairs of points  $(p_i^{(k)}(A), p_j^{(k)}(A))$ . Clearly, if  $A^{(k)}$  is bisecting and the orientation change from  $P^{(k)}$  to  $P^{(k+1)}$  is not a point moving over a line in the arrangement  $A^{(k)}$ , then  $A^{(k+1)}$  is still bisecting. For the other changes, we need the following lemma. Here, we say that an arrangement  $A^{(k.5)}$  is *almost bisecting* for  $P^{(k.5)}$ , if it bisects each  $P_i^{(k.5)}$  except for one, for which two points are on a line of the arrangement, and for the remaining points one side contains exactly one point more.

► **Lemma 20.** *Let  $A^{(k)}$  and  $A^{(k+1)}$  be such that  $A^{(k)}$  is bisecting and  $A^{(k+1)}$  is not. Then there either exists a sequence of arrangements*

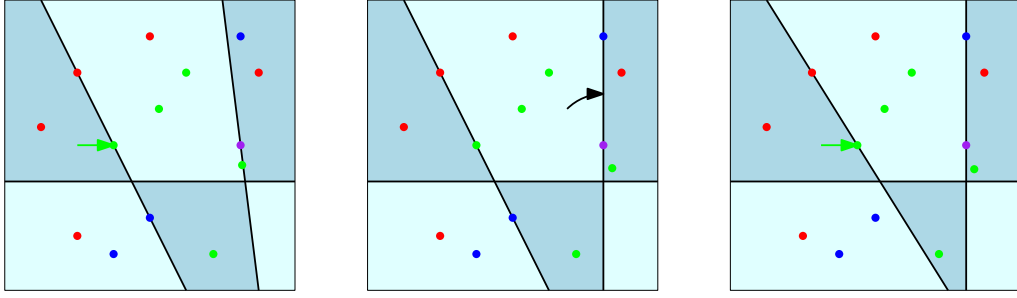
$$A^{(k)}, A^{(k.5)} = A_0^{(k.5)}, A_1^{(k.5)}, \dots, A_L^{(k.5)} = B^{(k.5)}, B^{(k+1)},$$

where each  $A_i^{(k.5)}$  is almost bisecting,  $B^{(k+1)}$  is bisecting and  $B^{(k)}$  is not bisecting, or there exists a sequence of arrangements

$$A^{(k)}, A^{(k.5)} = A_0^{(k.5)}, A_1^{(k.5)}, \dots, A_L^{(k.5)} = B^{(k.5)}, B^{(k)},$$

where each  $A_i^{(k.5)}$  is almost bisecting,  $B^{(k)}$  is bisecting and  $B^{(k+1)}$  is not bisecting. Further, in the second case, the sequence for  $B^{(k)}$  is the reverse of the sequence for  $A^{(k)}$ .

13:10 The Complexity of Sharing a Pizza



■ **Figure 3** Going from  $A^{(k.5)}$  (left) via  $B^{(k.5)}$  (middle) to  $B^{(k+1)}$  (right). In this example,  $B^{(k+1)}$  does not bisect the blue point set anymore.

In this lemma, the first case corresponds to a bisecting arrangement that can be changed to a new bisecting arrangement, whereas the second case corresponds to two bisecting arrangements disappearing at the same time.

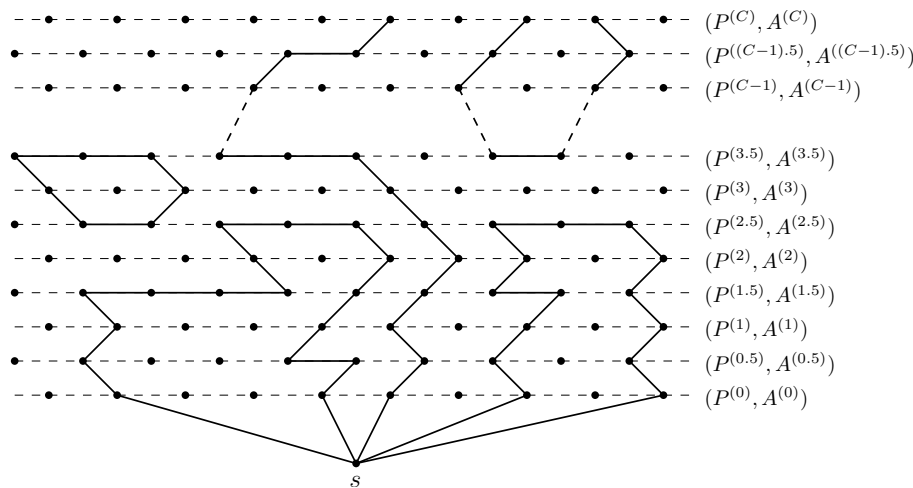
**Proof.** As mentioned above, the situation of the lemma can only occur if the orientation change from  $P^{(k)}$  to  $P^{(k+1)}$  corresponds to a point  $q^{(k)}$  moving over a line  $\ell$  of the arrangement  $P^{(k)}$ . Without loss of generality, let  $\ell$  be defined by the points  $p_1^{(k)} \in P_1^{(k)}$  and  $p_2^{(k)} \in P_2^{(k)}$ . There are two cases we consider: either  $q^{(k)}$  is in the same point set as one of the points on  $\ell$ , without loss of generality  $q^{(k)} \in P_1^{(k)}$ , or it is in a different point set, without loss of generality  $q^{(k)} \in P_3^{(k)}$ . We start with the second case. For an illustration of that case, see Figure 3.

**Case 1:**  $q^{(k)} \in P_3^{(k)}$ . Orient the arrangement in such a way that  $q^{(k)}$  is in  $R^+(A^{(k)})$ . In  $A^{(k.5)}$  the line  $\ell$  contains three points, namely  $p_1^{(k)}$ ,  $p_2^{(k)}$  and  $q^{(k)}$ . Note that  $A^{(k.5)}$  is almost bisecting, and  $R^+(A^{(k.5)})$  is the smaller side for  $P_3^{(k.5)}$ . There exists another line  $\ell'$  in  $A^{(k.5)}$  which contains a second point  $p_3^{(k.5)}$  of  $P_3^{(k.5)}$ . Rotate  $\ell'$  around the other point on it such that  $p_3^{(k.5)}$  is in  $R^+(A^{(k.5)})$ . Note that this direction of rotation is unique. Continue the rotation until  $\ell'$  hits another point  $q_j^{(k.5)} \in P_j^{(k.5)}$  which is not on some line of  $A^{(k.5)}$ . The resulting arrangement is now  $A_1^{(k.5)}$ .

Note that  $A_1^{(k.5)}$  is again almost bisecting, and that the point set which is not exactly bisected is  $P_j^{(k.5)}$ . While  $j \notin \{1, 2, 3\}$ , we can now find another point in  $P_j^{(k.5)}$  lying on a line of the arrangement  $A_l^{(k.5)}$ , rotate this line into the correct direction until a new point is hit, to get a new arrangement  $A_{l+1}^{(k.5)}$ . As all of these arrangements are different, and there are only finitely many possible arrangements, at some point we will have  $j \in \{1, 2, 3\}$ . This gives the arrangement  $B^{(k.5)}$ . There are now several options to consider: there are 3 different ways how  $p_1^{(k)}$ ,  $p_2^{(k)}$  and  $q^{(k)}$  can lie on  $\ell$ ,  $j$  can be 1, 2 or 3, and the smaller side of  $P_j^{(k.5)}$  can be  $R^+(B^{(k.5)})$  or  $R^-(B^{(k.5)})$ . In all of the cases, the only change from the arrangements  $B^{(k)}$  to  $B^{(k+1)}$  are the defining points of  $\ell$ . It follows that exactly one of  $B^{(k)}$  and  $B^{(k+1)}$  can be bisecting, depending on whether the point on  $\ell$  is in  $R^+(B^{(k+1)})$  or not.

**Case 2:**  $q^{(k)} \in P_1^{(k)}$ . In this case, we just have  $A^{(k.5)} = B^{(k.5)}$ .

Finally, the last claim follows from the fact that the directions of rotations from  $A_l^{(k.5)}$  to  $A_{l+1}^{(k.5)}$  are unique. ◀



■ **Figure 4** A schematic drawing of the graph whose leaves correspond to bisecting arrangements.

► **Remark 21.** During the process, it can also happen that two bisecting arrangement appear at the same time. In this case, we immediately get the analogous lemma, just reversing  $k$  and  $(k + 1)$ .

With all these lemmas at hand, we can now finally give a

**Proof of the discrete planar pizza cutting theorem.** Let  $P = (P_1, \dots, P_{2n})$  be in general position. By Lemma 18, we may assume that each  $P_i$  contains an odd number of points. Consider the sequence  $P^{(0)}, P^{(1)}, \dots, P^{(C)} = P$  of point sets. By Lemma 19,  $P^{(0)}$  has an odd number of bisecting arrangements. When going from  $P^{(k)}$  to  $P^{(k+1)}$ , either all bisecting arrangements stay bisecting or, by Lemma 20, one of them can be changed into a new bisecting arrangement or exactly two bisecting arrangements appear or disappear. It follows that for each  $k$ ,  $P^{(k)}$  has an odd number of bisecting arrangements. In particular, also  $P$  has an odd number of bisecting arrangements, and thus at least 1. ◀

## 4.2 Containment in PPA

In order to show that DISCRETEPIZZACUTTING is in PPA, we need to define a graph where the neighborhoods of each vertex can be efficiently computed and all odd-degree vertices, except the starting vertex, correspond to bisecting arrangements. In the following, we describe such a graph. For an illustration of the graph, see Figure 4.

**The vertex set.** Our vertex set consists of a starting vertex  $s$ , as well as vertices of the form  $(P^{(k)}, A^{(k)})$  and  $(P^{(k.5)}, A^{(k.5)})$ . For vertices  $(P^{(k)}, A^{(k)})$ ,  $A^{(k)}$  is an arrangement which is not necessarily bisecting, but whose lines contain exactly one point of each  $P_i^{(k)}$ . Similarly, for vertices  $(P^{(k.5)}, A^{(k.5)})$ , the arrangement  $A^{(k.5)}$  is not necessarily almost bisecting, but its lines contain at least one point of each  $P_i^{(k.5)}$  and exactly 2 for one of them. In particular, one of the lines of  $A^{(k.5)}$  is the line  $\ell$  through the unique three collinear points.

**The edge set.** The starting vertex  $s$  is connected to all vertices  $(P^{(0)}, A^{(0)})$ , for which  $A^{(0)}$  is bisecting. A vertex of the form  $(P^{(k)}, A^{(k)})$ ,  $k < L$  is connected to the vertices  $(P^{((k-1).5)}, A^{((k-1).5)})$  and  $(P^{(k.5)}, A^{(k.5)})$  if and only if  $A^{(k)}$  is bisecting. Otherwise, it is not connected to any other vertex. Similarly, the vertices  $(P^{(C)}, A^{(C)})$  are connected



## 13:12 The Complexity of Sharing a Pizza

to  $(P^{((C-1)\cdot 5)}, A^{((C-1)\cdot 5)})$  if and only if  $A^{(C)}$  is bisecting, and not connected to anything otherwise. Also a vertex of the form  $(P^{(k\cdot 5)}, A^{(k\cdot 5)})$  where  $A^{(k\cdot 5)}$  is not almost bisecting is not connected to any other vertex. For vertices  $(P^{(k\cdot 5)}, A^{(k\cdot 5)})$  with  $A^{(k\cdot 5)}$  almost bisecting we distinguish two cases. If  $\ell$  does not contain a point of the point set which is contained twice in the lines of  $A^{(k\cdot 5)}$ , we connect  $(P^{(k\cdot 5)}, A^{(k\cdot 5)})$  to the two vertices which correspond to the rotations defined in the proof of Lemma 20. Otherwise, we connect  $(P^{(k\cdot 5)}, A^{(k\cdot 5)})$  to the single vertex that we get by such a rotation, as well as to either  $(P^{(k)}, A^{(k)})$  or  $(P^{(k+1)}, A^{(k+1)})$ , depending on which one has the bisecting arrangement (note that we get from Lemma 20 that exactly one of the two arrangements is bisecting).

Note that all vertices have degree 0 or 2, except  $s$  and the vertices  $(P^{(C)}, A^{(C)})$  with  $A^{(C)}$  bisecting, that is, the vertices corresponding to solutions. All the vertices corresponding to solutions have degree 1. The starting vertex has degree  $(2n - 1)!!$ , which is exponential, so we cannot compute its neighborhood in polynomial time. This is however not an issue, as already noted in the paper where PPA is defined [36], provided that we can decide for any two vertices in polynomial time whether they are connected, and there is an efficiently computable *pairing function*, that is, a function  $\phi(v, v')$  which takes as input a vertex  $v$  and an outgoing edge  $\{v, v'\}$  and computes another vertex  $v''$  such that  $\{v, v''\}$  is also an edge. If  $v$  has even degree, then  $\phi(v, v') \neq v'$  for all  $v'$ . If  $v$  has odd degree, then there is exactly one  $v'$  for which  $\phi(v, v') = v'$ . The exact statement from [36] is as follows:

► **Lemma 22** ([36]). *Any problem defined in terms of an edge recognition algorithm and a pairing function is in PPA.*

It remains to show that we have both these ingredients.

► **Lemma 23.** *For any two vertices, we can decide in polynomial time whether they are connected.*

**Proof.** It follows from the construction that for any vertex except the starting vertex  $s$ , the neighborhood can be computed in polynomial time. In particular, it can also be checked whether two such vertices are connected. As for the starting vertex  $s$ , some other vertex can only be connected to it if its point set is  $P^{(0)}$  and its arrangement is bisecting. The first is encoded in the label of the vertex and the second can easily be checked in polynomial time. ◀

► **Lemma 24.** *There is a pairing function  $\phi$  which can be computed in polynomial time.*

**Proof.** As all vertices except  $s$  have degree 0, 1 or 2, and the neighborhoods can be computed in polynomial time, the pairing function follows trivially for these vertices. As for  $s$ , we note that its neighbors correspond to partitions of the  $2n$  point sets into pairs. In other words, its neighbors can be interpreted as perfect matchings in the complete graph  $K_{2n}$  with vertex set  $\{w_1, \dots, w_{2n}\}$ . It is thus enough to describe a pairing function for these perfect matchings. We do this in an algorithmic fashion.

Let  $M$  be some perfect matching. Consider the vertices  $w_1$  and  $w_2$ . Assume first that they are not connected to each other, that is, we have two distinct edges  $\{w_1, w_a\}$  and  $\{w_2, w_b\}$  in  $M$ . In that case, define the pairing  $M' := M \setminus \{\{w_1, w_a\}, \{w_2, w_b\}\} \cup \{\{w_1, w_b\}, \{w_2, w_a\}\}$ , that is, flip the edges incident to  $w_1$  and  $w_2$ . Note that the pairing of  $M'$  is again  $M$ , that is, it indeed is a pairing.

If  $w_1$  and  $w_2$  are connected to each other, repeat the same process with  $w_3$  and  $w_4$ , and so on. This process defines a pairing for each matching except  $\{\{w_1, w_2\}, \dots, \{w_{2n-1}, w_{2n}\}\}$ . Further, the algorithm clearly runs in polynomial time. We thus get a valid pairing function. ◀



We thus get from Lemma 22 that DISCRETEPIZZACUTTING is in PPA. Together with Corollary 10, we can thus conclude our main result.

► **Corollary 25.** *DISCRETEPIZZACUTTING is PPA-complete.*

## 5 Conclusion

We have shown several complexity results related to the pizza cutting problem. Our main result is that DISCRETEPIZZACUTTING is PPA-complete. While we have only considered the planar case, the problem as well as our arguments extend to higher dimensions. More precisely, the proof of the Hobby-Rice theorem using the pizza cutting theorem can be adapted to work for any dimension of the pizza cutting theorem. Further, the proof of the discrete pizza cutting theorem can be adapted to any dimension where the number of initial solutions is odd. This is the case if the dimension is a power of 2 [26]. Thus, in all these dimensions the analogous versions of DISCRETEPIZZACUTTING are also PPA-complete, assuming the dimension is fixed. On the other hand, it is an open problem whether the pizza cutting theorem also holds in other dimensions.

We have also shown that PIZZACUTTING is FIXP-hard. It is an interesting problem whether the problem is in FIXP or even harder. One way to show that the problem is in FIXP is to find a proof for the planar pizza cutting theorem using Brouwer’s fixpoint theorem. Such a proof would have the potential to generalize to any dimension, resolving the above question.

---

## References

- 1 Oswin Aichholzer, Nieves Atienza, Ruy Fabila-Monroy, Pablo Perez-Lantero, Jose M Diaz-Báñez, David Flores-Peñaloza, Birgit Vogtenhuber, and Jorge Urrutia. Balanced islands in two colored point sets in the plane. *arXiv preprint*, 2015. [arXiv:1510.01819](#).
- 2 Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987.
- 3 Noga Alon and Douglas B West. The Borsuk-Ulam theorem and bisection of necklaces. *Proceedings of the American Mathematical Society*, 98(4):623–628, 1986.
- 4 Imre Bárány, Alfredo Hubard, and Jesús Jerónimo. Slicing convex sets and measures by a hyperplane. *Discrete & Computational Geometry*, 39(1-3):67–75, 2008.
- 5 Luis Barba, Alexander Pilz, and Patrick Schnider. Sharing a pizza: bisecting masses with two cuts. *arXiv preprint*, 2019. [arXiv:1904.02502](#).
- 6 Sergey Bereg. Equipartitions of measures by 2-fans. In *International Symposium on Algorithms and Computation*, pages 149–158. Springer, 2004.
- 7 Sergey Bereg. Computing generalized ham-sandwich cuts. *Information Processing Letters*, 112(13):532–534, 2012.
- 8 Sergey Bereg, Ferran Hurtado, Mikio Kano, Matias Korman, Dolores Lara, Carlos Seara, Rodrigo I. Silveira, Jorge Urrutia, and Kevin Verbeek. Balanced partitions of 3-colored geometric sets in the plane. *Discrete Applied Mathematics*, 181:21–32, 2015.
- 9 P Bonsma, Th Epping, and Winfried Hochstättler. Complexity results on restricted instances of a paint shop problem for words. *Discrete Applied Mathematics*, 154(9):1335–1343, 2006.
- 10 Xi Chen, Decheng Dai, Ye Du, and Shang-Hua Teng. Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 273–282. IEEE, 2009.
- 11 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009.
- 12 Xi Chen, David Durfee, and Anthi Orfanou. On the complexity of nash equilibria in anonymous games. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 381–390, 2015.

## 13:14 The Complexity of Sharing a Pizza

- 13 Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 181–190, 2013.
- 14 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational Complexity of the  $\alpha$ -Ham-Sandwich Problem. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 15 Bruno Codenotti, Amin Saberi, Kasturi Varadarajan, and Yinyu Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theoretical Computer Science*, 408(2-3):188–198, 2008.
- 16 Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.
- 17 Argyrios Deligkas, John Fearnley, and Themistoklis Melissourgos. Pizza sharing is PPA-hard. *arXiv preprint*, 2020. [arXiv:2012.14236](https://arxiv.org/abs/2012.14236).
- 18 Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G Spirakis. Computing exact solutions of consensus halving and the Borsuk-Ulam theorem. *Journal of Computer and System Sciences*, 117:75–98, 2021.
- 19 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.
- 20 Edith Elkind, Leslie Ann Goldberg, and Paul Goldberg. Nash equilibria in graphical games on trees revisited. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 100–109, 2006.
- 21 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.
- 22 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, 2020.
- 23 Aris Filos-Ratsikas and Paul W Goldberg. Consensus halving is PPA-complete. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 51–64, 2018.
- 24 Aris Filos-Ratsikas and Paul W Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 638–649, 2019.
- 25 Charles R. Hobby and John R. Rice. A moment problem in  $L_1$  approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965. URL: <http://www.jstor.org/stable/2033900>.
- 26 Alfredo Hubard and Roman Karasev. Bisecting measures with hyperplane arrangements. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 169, pages 639–647. Cambridge University Press, 2020.
- 27 Mikio Kano and Jorge Urrutia. Discrete geometry on colored point sets in the plane—a survey. *Graphs and Combinatorics*, 37(1):1–53, 2021.
- 28 Roman N Karasev, Edgardo Roldán-Pensado, and Pablo Soberón. Measure partitions using hyperplanes with fixed directions. *Israel Journal of Mathematics*, 212(2):705–728, 2016.
- 29 Christian Knauer, Hans Raj Tiwary, and Daniel Werner. On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems. In *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, volume 9, pages 649–660, 2011.
- 30 Stefan Langerman. personal communication, 2017.
- 31 Chi-Yuan Lo, Jiří Matoušek, and William Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11(1):433–452, 1994.
- 32 Chi-Yuan Lo and William Steiger. An optimal time algorithm for ham-sandwich cuts in the plane. In *Proc. 2<sup>nd</sup> Canadian Conference on Computational Geometry (CCCG 1990)*, pages 5–9, 1990.
- 33 Jiri Matoušek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*. Springer Publishing Company, Inc., 2007.

- 34 Ruta Mehta. Constant rank bimatrix games are PPAD-hard. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 545–554, 2014.
- 35 Frédéric Meunier. Discrete splittings of the necklace. *Mathematics of Operations Research*, 33(3):678–688, 2008.
- 36 Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994.
- 37 Alexander Pilz and Patrick Schnider. Bisecting three classes of lines. *Computational Geometry*, 98:101775, 2021.
- 38 Edgardo Roldan-Pensado and Pablo Soberon. A survey of mass partitions. *arXiv preprint*, 2020. [arXiv:2010.00478](https://arxiv.org/abs/2010.00478).
- 39 Aviad Rubinfeld. Inapproximability of Nash equilibrium. *SIAM Journal on Computing*, 47(3):917–959, 2018.
- 40 Patrick Schnider. Equipartitions with Wedges and Cones. *arXiv preprint*, 2019. [arXiv:1910.13352](https://arxiv.org/abs/1910.13352).
- 41 Patrick Schnider. Ham-sandwich cuts and center transversals in subspaces. *Discrete & Computational Geometry*, 64(4):1192–1209, 2020.
- 42 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Finding clearing payments in financial networks with credit default swaps is PPAD-complete. *LIPICs: Leibniz International Proceedings in Informatics*, 67, 2017.
- 43 William Steiger and Jihui Zhao. Generalized ham-sandwich cuts. *Discrete & Computational Geometry*, 44(3):535–545, 2010.
- 44 A. H. Stone and J. W. Tukey. Generalized “sandwich” theorems. *Duke Math. J.*, 9(2):356–359, June 1942.
- 45 Vijay V Vazirani and Mihalis Yannakakis. Market equilibrium under separable, piecewise-linear, concave utilities. *Journal of the ACM (JACM)*, 58(3):1–25, 2011.



# Dynamic Data Structures for $k$ -Nearest Neighbor Queries

Sarita de Berg 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Frank Staals 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

---

## Abstract

Our aim is to develop dynamic data structures that support  $k$ -nearest neighbors ( $k$ -NN) queries for a set of  $n$  point sites in  $O(f(n) + k)$  time, where  $f(n)$  is some polylogarithmic function of  $n$ . The key component is a general query algorithm that allows us to find the  $k$ -NN spread over  $t$  substructures simultaneously, thus reducing a  $O(tk)$  term in the query time to  $O(k)$ . Combining this technique with the logarithmic method allows us to turn any static  $k$ -NN data structure into a data structure supporting both efficient insertions and queries. For the fully dynamic case, this technique allows us to recover the deterministic, worst-case,  $O(\log^2 n / \log \log n + k)$  query time for the Euclidean distance claimed before, while preserving the polylogarithmic update times. We adapt this data structure to also support fully dynamic *geodesic*  $k$ -NN queries among a set of sites in a simple polygon. For this purpose, we design a shallow cutting based, deletion-only  $k$ -NN data structure. More generally, we obtain a dynamic  $k$ -NN data structure for any type of distance functions for which we can build vertical shallow cuttings. We apply all of our methods in the plane for the Euclidean distance, the geodesic distance, and general, constant-complexity, algebraic distance functions.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases** data structure, simple polygon, geodesic distance, nearest neighbor searching

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.14

**Related Version** *Full Version:* <https://arxiv.org/abs/2109.11854>

## 1 Introduction

In the  $k$ -nearest neighbors ( $k$ -NN) problem we are given a set of  $n$  point sites  $S$  in some domain, and we wish to preprocess these points such that given a query point  $q$  and an integer  $k$ , we can find the  $k$  sites in  $S$  “closest” to  $q$  efficiently. This static problem has been studied in many different settings [4, 5, 10, 18, 19]. We study the dynamic version of the  $k$ -nearest neighbors problem, in which the set of sites  $S$  may be subject to updates; i.e. insertions and deletions. We are particularly interested in two settings: (i) a setting in which the domain containing the sites contains (polygonal) obstacles, and in which we measure the distance between two points by their geodesic distance: the length of the shortest obstacle avoiding path, and (ii) a setting in which only insertions into  $S$  are allowed (i.e. no deletions).

In many applications involving distances and shortest paths, the entities involved cannot travel towards their destination in a straight line. For example, a person in a city center may want to find the  $k$  closest restaurants that currently have seats available, but since he or she cannot walk through walls, this should be reflected by the distances. This introduces complications, as a shortest path in a polygon with  $m$  vertices may have complexity  $\Theta(m)$ . We wish to limit the resulting dependency on  $m$  in the space and time bounds as much as possible. In particular, we wish to avoid spending  $\Omega(m)$  time every time the availability of the seats in a restaurant changes (which may cause an insertion or deletion of a site in  $S$ ).



© Sarita de Berg and Frank Staals;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 14; pp. 14:1–14:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The second setting is motivated by classification problems. In  $k$ -nearest neighbor classifiers the sites in  $S$  all have a label, and the label of a query point  $q$  is predicted based on the labels of the  $k$  sites nearest to  $q$  [11]. When this label turns out to be sufficiently accurate, it is customary to then extend the data set by adding  $q$  to  $S$ . This naturally gives an interest in insertion-only data structures that can efficiently answer  $k$ -NN queries.

**The static problem.** If the set of sites is static, and  $k$  is known a-priori, one option is to build the (geodesic)  $k^{\text{th}}$ -order Voronoi diagram of  $S$  [20]. This yields very fast ( $O(\log(n+m) + k)$ ) query times. However it is costly in space, as even in a simple polygon the diagram has size  $O(k(n-k) + km)$ . For the Euclidean plane, much more space efficient solutions have been developed. There is an optimal linear space data structure achieving  $O(\log n + k)$  query time after  $O(n \log n)$  deterministic preprocessing time [1, 10]. Very recently, Liu [19] showed how to achieve the same query time for general constant-complexity distance functions for sites in  $\mathbb{R}^2$ , using  $O(n \log \log n)$  space and roughly  $O(n \log^4 n)$  expected preprocessing time (the exact bound depends on the algebraic degree of the functions). In case the domain is a simple polygon  $\mathcal{P}$ , the problem has not explicitly been studied. The only known solution using less space than just storing the  $k^{\text{th}}$ -order Voronoi diagram is the dynamic 1-NN structure of Agarwal et al. [2]. It uses  $O(n \log^3 n \log m + m)$  space, and answers queries in  $O(k \text{polylog}(n+m))$  time (by deleting and reinserting the  $k$ -closest sites to answer a query).

**Issues when inserting sites.** Since nearest neighbor searching is decomposable, we can apply the logarithmic method [23] to turn a static  $k$ -NN searching data structure into an insertion-only data structure. In the Euclidean plane this yields a linear space data structure with  $O(\log^2 n)$  insertion time. However, since this partitions the set of sites into  $O(\log n)$  subsets, and we do not know how many of the  $k$ -nearest sites appear in each subset, we may have to consider up to  $k$  sites from *each* of the subsets. This yields an  $O(k \log n)$  term in the query time. We will present a general technique that allows us to avoid this  $O(\log n)$  factor.

**Fully dynamic data structures.** In case we wish to support both insertions and deletions, the problem becomes more complicated, and the existing solutions much more involved. When we need to report only one nearest neighbor (i.e. 1-NN searching) in the plane, efficient fully dynamic data structures exist [6, 8, 16]. Actually, all these data structures are variants of the same data structure by Chan [6]. For the Euclidean distance, the current best result uses linear space, and achieves  $O(\log^2 n)$  worst-case query time,  $O(\log^2 n)$  insertion time, and  $O(\log^4 n)$  deletion time [8]. These results are deterministic, and the update times are amortized. The variant by Kaplan et al. [16] achieves similar results for general distance functions. These data structures can also be used to answer  $k$ -NN queries, but when used in this way essentially suffer from the same problem as in the insertion-only case. That is, we get a query time of  $O(\log^2 n + k \log n)$  time [6].

Chan argues that the above data structure for Euclidean 1-NN searching can be extended to answer  $k$ -NN queries in  $O(\log^2 n / \log \log n + k)$  time, while retaining polylogarithmic updates [7]. Chan's data structure essentially maintains a collection of  $k$ -NN data structures built on subsets of the sites. A careful analysis shows that some of these structures can be rebuilt during updates, and that the cost of these updates is not too large. Queries are then answered by performing  $k_i$ -NN queries on several disjoint subsets of sites  $S_1, \dots, S_t$  that together are guaranteed to contain the  $k$  nearest sites. However – perhaps because the details of the 1-NN searching data structure are already fairly involved – one aspect in the query

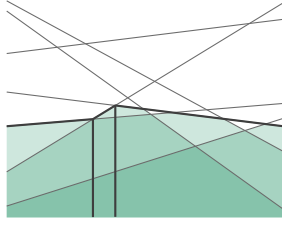
algorithm is missing: how to determine the value  $k_i$  to query subset  $S_i$  with. While it seems that this issue can be fixed using randomization [9]<sup>1</sup>, our general  $k$ -NN query technique (Section 3) allows us to recover deterministic, worst-case  $O(\log^2 n / \log \log n + k)$  query time.

Very recently, Liu [19] stated that one can obtain  $O(\log^2 n + k)$  query time while supporting  $O(\text{polylog } n)$  expected amortized updates also for general distance functions by using the data structure of Kaplan et al. [16]. However, it is unclear why that would be the case, as all details are missing. Using the Kaplan et al. data structure as is yields an  $O(k \log n)$  term in the query time as with Chan’s original version [6]. If the idea is to also apply the ideas from Chan’s later paper [7] the same issue of choosing the  $k_i$ ’s appears. Similarly, extending the geodesic 1-NN data structure [2] to  $k$ -NN queries yields  $O(k \text{ polylog}(n + m))$  query time.

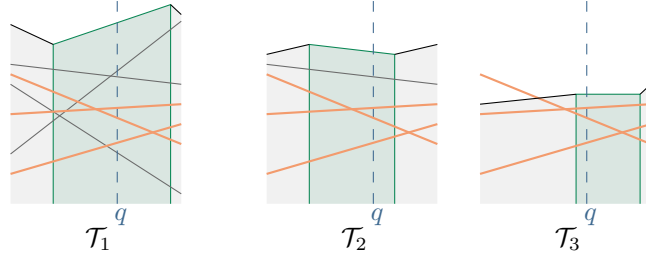
**Organization and Results.** We develop dynamic data structures for  $k$ -NN queries whose query time are of the form  $O(f(n) + k)$ , where  $f(n)$  is some function of  $n$ . In particular, we wish to avoid an  $O(k \log n)$  term in the query time. To this end, we present a general query technique that given  $t$  disjoint subsets of sites  $S_1, \dots, S_t$ , each stored in a static data structure that supports  $k'$ -NN queries in  $O(Q(n) + k')$  time, can report the  $k$  nearest neighbors among  $\bigcup_{i=1}^t S_i$  in  $O(Q(n)t + k)$  time. Our technique, presented in Section 3, is completely combinatorial, and is applicable to any type of sites. In Section 4, we then use this technique to obtain a  $k$ -NN data structure that supports queries in  $O(Q(n) \log n + k)$  time and insertions in  $O((P(n)/n) \log n)$  time, where  $P(n)$  is the time required to build the static data structure. In the specific case of the Euclidean plane, we obtain a linear space data structure with  $O(\log^2 n + k)$  query time and  $O(\log^2 n)$  insertion time. At a slight increase of insertion time we can also match the query time of Chan’s [7] fully dynamic data structure. For general, constant-complexity, algebraic distance functions, we obtain the same query and insertion times (albeit the insertion time holds in expectation). In the case where the sites  $S$  are points inside a simple polygon  $\mathcal{P}$  with  $m$  vertices, we use our technique to obtain the first *static*  $k$ -NN data structure that uses near-linear space, supports efficient (i.e. without the  $O(k \log n)$  term) queries, *and* can be constructed efficiently. We do get an  $O(\log m)$  factor in the query time, as computing the distance between two sites already takes  $O(\log m)$  time. Our data structure uses  $O(n \log n + m)$  space, can be constructed in  $O(n(\log n \log^2 m + \log^3 m) + m)$  time, and supports  $O((\log(n + m) + k) \log m)$  time queries. In turn, this then leads to a data structure supporting efficient insertions. In Section 5, we argue that our general query algorithm is the final piece of the puzzle for the fully dynamic case. For the Euclidean plane, this allows us to recover the deterministic, worst-case  $O(\log^2 n / \log \log n + k)$  query time claimed before [7, 19]. The amortized update times remain polylogarithmic. We obtain the same query time and similar update times for more general distance functions.

For the geodesic case there is one final hurdle. Chan’s algorithm uses partition-tree based “slow” dynamic  $k$ -NN data structure of linear size in its subroutines. Liu uses a similar trick after linearizing the distance functions into  $\mathbb{R}^c$ , for some constant  $c$  [19]. Unfortunately, these ideas are not applicable in the geodesic setting, as it is unknown if an appropriate partition tree can be built, and the dimension after linearization would depend on  $m$ . Instead, we design a simple, shallow-cutting based, alternative “slow” dynamic (geodesic)  $k$ -NN data structure. This way, we obtain an efficient (i.e.  $O(\text{polylog}(n + m))$  expected updates,  $O(\log^2 n \log^2 m + k \log m)$  queries) fully dynamic  $k$ -NN data structure. Omitted proofs are in the full version of this paper [12].

<sup>1</sup> The main idea is that the data structure as is *can* be used to efficiently report all sites within a fixed distance from the query point (reporting all planes below a query point in  $\mathbb{R}^3$ ). Combining this with an earlier random sampling idea [5] one can then also answer  $k$ -NN queries.



■ **Figure 1** A 2-shallow cutting of a set of lines  $F$  in  $\mathbb{R}^2$  consisting of 3 prisms. The at most  $k$ -level  $L_{\leq k}(F)$  is shown in green for  $k = 0, 1, 2$ .



■ **Figure 2** Example of the dynamic 1-NN data structure. Only one shallow cutting ( $\Lambda_{k_j}$ ) is shown for each tower. The orange planes in  $\mathcal{T}_1$  and  $\mathcal{T}_2$  are pruned when building  $\Lambda_{k_{j-1}}$ , but are not removed from the conflict lists in  $\Lambda_{k_j}$ . When querying for the  $k$ -NN, the green prisms in  $\Lambda_{k_j}$  of each tower are considered. Note that the three orange planes occur in each of the conflict lists.

## 2 Preliminaries

We can easily transform a  $k$ -nearest neighbors problem in  $\mathbb{R}^2$  to a  $k$ -lowest functions problem in  $\mathbb{R}^3$  by considering (the graphs of) the distance functions  $f_s(x)$  of the sites  $s \in S$ . We discuss these problems interchangeably, furthermore we identify a function with its graph.

### 2.1 Shallow cuttings

Let  $F$  be a set of bivariate functions. We consider the arrangement of  $F$  in  $\mathbb{R}^3$ . The *level* of a point  $q \in \mathbb{R}^3$  is defined as the number of functions in  $F$  that pass strictly below  $q$ . The *at most  $k$ -level*  $L_{\leq k}(F)$  is then the set of points in  $\mathbb{R}^3$  that have level at most  $k$ .

A  *$k$ -shallow cutting*  $\Lambda_k(F)$  of  $F$  is a set of disjoint cells covering  $L_{\leq k}(F)$ , such that each cell intersects at most  $O(k)$  functions [21]. When  $F$  is clear from the context we may write  $\Lambda_k$  rather than  $\Lambda_k(F)$ . We are interested only in the case where the cells are (*pseudo-*)*prisms*: constant-complexity regions that are bounded from above by a function, from the sides by vertical (with respect to the  $z$ -direction) planes, and unbounded from below. For example, if  $F$  is a set of planes, we can define the top of each prism to be a triangle. This allows us to find the prism containing a query point  $q$  by a point location query in the downward projection of the cutting. See Figure 1. The subset  $F_{\nabla} \subseteq F$  intersecting a prism  $\nabla$  is the *conflict list* of  $\nabla$ . When, for every subset  $F' \subseteq F$ , the lower envelope  $L_0(F')$  has linear complexity (for example, in the case of planes), a shallow cutting of *size* (the number of cells)  $O(n/k)$  can be computed efficiently [19]. In general, let  $T(n, k)$  be the time to construct a  $k$ -shallow cutting of size  $S(n, k)$  on  $n$  functions, and  $Q(n, k)$  be the time to locate the prism containing a query point. We assume these functions are non-decreasing in  $n$  and non-increasing in  $k$ , and that  $S(n, k) = \frac{n}{k} f(n)$ , for some function  $f(n)$ .

### 2.2 A dynamic nearest neighbor data structure

We briefly discuss the main ideas used in the existing dynamic 1-NN data structures [6, 8, 16], as these also form a key component in our fully dynamic  $k$ -NN data structures. For ease of exposition, we describe the data structure when  $F$  is a set of linear functions (planes). To ensure the analysis is correct for our definition of  $n$  (the current number of sites in  $S$ ), we rebuild the data structure from scratch whenever  $n$  has doubled or halved. The cost of this is subsumed in the cost of the other operations [6].



The data structure consists of  $t = O(\log_b n)$  “towers”  $\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(t)}$ . Each tower  $\mathcal{T}^{(i)}$  consists of a hierarchy of shallow cuttings that is built on a subset of planes  $F^{(i)} \subseteq F$ . For  $\mathcal{T}^{(1)}$  we have  $F^{(1)} = F$ , and a sequence of  $\ell = \lfloor \log(n/k_0) \rfloor$  shallow cuttings, for a fixed constant  $k_0$ . For  $j = 0, \dots, \ell$  we have a  $k_j$ -shallow cutting of a subset of the planes  $F_j \subseteq F^{(1)}$ , where  $k_j = 2^j k_0$ . We set  $F_\ell = F^{(1)}$  and construct these cuttings from  $j = \ell$  to 0. After computing  $\Lambda_{k_j}(F_j)$ , we find the set  $F_j^\times$  of “bad” planes that intersect more than  $c \log n$  prisms in all cuttings computed so far. We *prune* these planes by setting  $F_{j-1} = F_j \setminus F_j^\times$  and removing all planes in  $F_j^\times$  from the conflict lists of the prisms in  $\Lambda_{k_j}(F_j)$ . These bad planes are removed only from the conflict lists of the current cutting, and can still occur in conflict lists of higher level cuttings. In the final  $\Lambda_{k_0}(F_0)$  cutting, each conflict list has a constant size of  $O(k_0)$ . By  $F_{\text{live}}^{(1)}$  we denote the set of planes that have not been pruned during this process. We then set  $F^{(i+1)} = F^{(i)} \setminus F_{\text{live}}^{(i)}$ , and recursively build  $\mathcal{T}^{(i+1)}$  on the functions in  $F^{(i+1)}$ . This partitions  $F$  into sets  $F_{\text{live}}^{(1)}, \dots, F_{\text{live}}^{(t)}$ . Chan [8] recently achieved an overall construction time of  $O(n \log n)$ , by using information of previously computed cuttings to efficiently build the cuttings later in the sequence. Kaplan et al. [16, Lemma 7.1] prove that for any  $\zeta \in (0, 1)$  choosing  $c \geq \frac{2}{\zeta}$ , for a sufficiently large (but constant)  $\gamma$ , ensures that  $|F_{\text{live}}^{(1)}| \geq (1 - \zeta)n$  after building  $\mathcal{T}^{(1)}$ . When  $\zeta = 1/b$ , we get  $O(\log_b n)$  towers, for some fixed  $b \geq 2$  as desired. A plane then occurs  $O(b \log n)$  times in a tower.

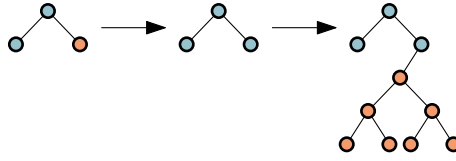
**Updates.** When updates take place, planes can move from a set  $F_{\text{live}}^{(i)}$  to some  $F_{\text{live}}^{(i')}$ , but the live sets remain a partition of  $F$ . To insert a plane  $f$  into  $F$ , we create a new tower containing only  $f$ . When  $|F^{(i+1)} \cup \dots \cup F^{(t)}|$  reaches  $3/4 \cdot |F^{(i)}|$  we rebuild the towers  $\mathcal{T}^{(i)}, \dots, \mathcal{T}^{(t)}$ . Such a rebuild occurs only after  $\Omega(|F^{(i)}|)$  insertions, so the amortized insertion time is  $O(\log^2 n)$ .

Deletions are not performed explicitly on the conflict lists. Instead, for each prism  $\nabla$  we keep track of the number of planes in  $F_\nabla$  that have been deleted so far, denoted by  $d_\nabla$ . When deleting a plane  $f$ , we increase  $d_\nabla$  for all prisms with  $f \in F_\nabla$ , and remove  $f$  from the set  $F_{\text{live}}^{(i)}$  that includes  $f$ . When too many planes in a conflict list have been deleted (i.e.  $d_\nabla$  becomes too large), we *purge* the prism. When a prism in  $\mathcal{T}^{(i)}$  is purged, we mark it as such, and we reinsert all planes  $f' \in F_\nabla \cap F_{\text{live}}^{(i)}$ . These planes are effectively moved from  $F_{\text{live}}^{(i)}$  to some other  $F_{\text{live}}^{(i')}$ . Chan [8] shows that each increment of  $d_\nabla$  causes amortized  $O(1)$  reinsertions. This gives an amortized deletion time of  $O(\log^4 n)$ .

**Queries.** We can answer  $k$ -NN queries in  $O(\log^2 n + k \log_b n)$  time, and thus 1-NN queries in  $O(\log^2 n)$  time, as follows. For each tower we consider the prism containing  $q$  in the shallow cutting at level  $j_k := \lceil \log(Ck/n) \rceil$ , for some large enough constant  $C$ . Each such prism has a conflict list of size  $O(k)$ , and thus we can find the  $k$ -lowest *live* planes in each conflict list in  $O(k)$  time. Chan [6] proves that considering only these planes is sufficient.

Liu [19] recently claimed the data structure, in particular the version of Kaplan et al. [16], supports  $k$ -NN queries in  $O(\log^2 n + k)$  time. However, we see an issue with this approach. When a plane is pruned during the preprocessing, or when a prism is purged, the plane is only removed from the conflict lists of the current shallow cutting. It can thus still occur in other shallow cuttings in the hierarchy. This means that we can encounter the same plane multiple times when querying each tower for the  $k$ -lowest planes. See Figure 2 for an illustration. As there are  $O(\log_b n)$  towers, this yields an  $O(k \log_b n)$  term in the query time.

**General distance functions.** Kaplan et al. [16] showed that this data structure is applicable for any set of functions  $F$  for which we can compute small  $k$ -shallow cuttings. The following lemma summarizes the properties of the data structure in this setting.



■ **Figure 3** Example of expansion. Blue elements are included in a clan, orange elements are not. The expansion (building the next subheap) occurs when all elements have been included in a clan.

► **Lemma 1.** Let  $b \geq 2$  be any fixed value, and  $S(n, k)$  be the size of a  $k$ -shallow cutting of  $F$ . There is a dynamic nearest neighbor data structure that has the following properties.

1. The data structure consists of  $O(\log_b n)$  towers.
2. A function occurs  $O(b \log n \cdot S(n, 1)/n)$  times in a conflict list in a single tower.
3. The insertion time is  $O(b \log_b n \cdot P(n)/n)$ , where  $P(n)$  is the preprocessing time.
4. A deletion causes amortized  $O(b \log_b n \log n \cdot S(n, 1)/n)$  reinsertions.
5. To find the  $k$ -NN of a query point  $q$  it is sufficient to consider the prisms containing  $q$  of the shallow cuttings at level  $j_k := \lceil \log(Ck/n) \rceil$ , for some large enough constant  $C$ .

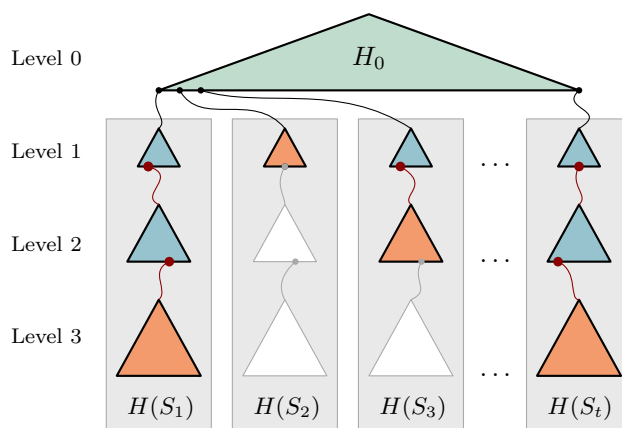
### 3 Querying multiple $k$ -NN data structures simultaneously

We introduce a method to find the  $k$ -nearest neighbors of a query point  $q$  among  $t$  (disjoint)  $k'$ -NN data structures together storing a set of sites  $S$ . Suppose the query time of such a  $k'$ -NN data structure is  $O(Q(n) + k')$ , for a non-decreasing function  $Q$ . Naively, querying each data structure for the  $k$  closest sites would take  $O(Q(n)t + kt)$  time. Our method allows us to find the  $k$ -NN over all these data structures in  $O(Q(n)t + k)$  time instead.

**Query algorithm.** We use the heap selection algorithm of Frederickson [13] to answer  $k$ -NN queries efficiently. This algorithm finds the  $k$  smallest elements of a binary min-heap of size  $N \gg k$  in  $O(k)$  time by forming groups of elements, called *clans*, in the original heap. Representatives of these clans are then added to another heap, and smaller clans are created from larger clans and organised in heaps recursively. For our purposes, we need to consider (only) how clans are formed in the original heap, because we do not construct the entire heap beforehand. Instead, the heap is expanded during the query only when necessary. See Figure 3 for an example. Note that any (non-root) element of the heap will only be included in a clan by the Frederickson algorithm after its parent has been included in a clan.

The heap  $H$ , on which we call the heap selection algorithm, contains all sites  $s \in S$  exactly once, with the distance  $d(s, q)$  as key for each site. Let  $S_1, \dots, S_t$  be the partition of  $S$  into  $t$  disjoint sets, where  $S_i$  is the set of sites stored in the  $i$ -th  $k'$ -NN data structure. For each set of sites  $S_j$ ,  $j \in 1, \dots, t$ , we define a heap  $H(S_j)$  containing all sites in  $S_j$ . We then “connect” these  $t$  heaps by building a dummy heap  $H_0$  of size  $O(t)$  that has the roots of all  $H(S_j)$  as leaves. We set the keys of the elements of  $H_0$  to  $-\infty$ . Let  $H$  be the complete data structure that we obtain this way, see Figure 4. We can now compute the  $k$  sites closest to  $q$  by finding the  $|H_0| + k$  smallest elements in  $H$  and reporting the non-dummy sites.

What remains is how to (incrementally) build the heaps  $H(S_j)$  while running the heap selection algorithm. Each such heap consists of a hierarchy of *subheaps*  $H_1(S_j), \dots, H_{O(\log n)}(S_j)$ , such that every element of  $S_j$  appears in exactly one  $H_i(S_j)$ . Moreover, since the sets  $S_1, \dots, S_j$  are pairwise disjoint, any site  $s \in S$  will appear in exactly one  $H_i(S_j)$ . The *level 1* heaps,  $H_1(S_j)$ , consist of the  $k_1 = Q(n)$  sites in  $S_j$  closest to  $q$ , which we find by querying the static  $k'$ -NN data structure on  $S_j$ . The subheap  $H_i(S_j)$  at level  $i > 1$  is built only



■ **Figure 4** The heap constructed for a  $k$ -NN query. Subheaps of which all elements have been included in a clan are *blue*. Subheaps of which not all elements have been included are *orange*. The *white* subheaps have not been built so far, as not all elements of their predecessor are in a clan yet.

after the last element  $e$  of  $H_{i-1}(S_j)$  is included in a clan. We then add a pointer from  $e$  to the root of  $H_i(S_j)$ , such that the root of  $H_i(S_j)$  becomes a child of  $e$ , as in Figure 3. To construct a subheap  $H_i(S_j)$  at level  $i > 1$ , we query the static data structure of  $S_j$  using  $k_i = k_1 2^{i-1}$ . The new subheap is built using all sites returned by the query that have not been encountered earlier. This ensures that the heap property is preserved.

**Analysis of the query time.** As stated before, finding the  $k$ -smallest non-dummy elements of  $H$  takes  $O(k + |H_0|)$  time [13]. Here, we analyse the time used to construct  $H$ .

First, the level 0 and level 1 heaps are built. Building  $H_0$  takes only  $O(t)$  time. To build the level 1 heaps, we query each of the substructures using  $k_1 = Q(n)$ . In total these queries take  $O((Q(n) + k_1)t) = O(Q(n)t)$  time. Retrieving the next  $k_i$  elements to build  $H_i(S_j)$  for  $i > 1$  requires a single query and thus takes  $O(Q(n) + k_i)$  time. To bound the time used to build all heaps at level greater than 1, we use the following lemma.

► **Lemma 2.** *The size of a subheap  $H_i(S_j)$ ,  $j \in \{1, \dots, t\}$ , at level  $i > 1$  is exactly  $k_1 2^{i-2}$ .*

To pay for building  $H_i(S_j)$ , we charge  $O(1)$  to each element of  $H_{i-1}(S_j)$ . Because we choose  $k_1 = Q(n)$ , Lemma 2 implies that  $|H_{i-1}(S_j)| = \Omega(Q(n))$ , and that  $k_i = k_1 2^{i-1} = 2^2 k_1 2^{i-3} = O(|H_{i-1}(S_j)|)$ . Note that the heap  $H_i(S_j)$ ,  $i > 1$ , is only built when all elements of  $H_{i-1}(S_j)$  have been included in a clan. Thus, we only charge elements of heaps of which all elements have been included in a clan (shown blue in Figure 4). In total,  $O(k)$  elements (not in  $H_0$ ) are included in a clan, so the total size of these subheaps is  $O(k)$ . From this, and the fact that all subheaps are disjoint, it follows that we charge  $O(1)$  to only  $O(k)$  sites.

► **Lemma 3.** *Let  $S_1, \dots, S_t$  be disjoint sets of point sites of sizes  $n_1, \dots, n_t$ , each stored in a data structure that supports  $k'$ -NN queries in  $O(Q(n_i) + k')$  time. There is a  $k$ -NN data structure on  $\bigcup_i S_i$  that supports queries in  $O(Q(n)t + k)$  time. The data structure uses  $O(\sum_i C(n_i))$  space, where  $C(n_i)$  is the space required by the  $k$ -NN structure on  $S_i$ .*

Throughout this section, we used the standard assumption that for any two points  $p, q$  their distance  $d(p, q)$  can be computed in constant time. When evaluating  $d(p, q)$  takes  $T$  time, our technique achieves a query time of  $O(Q(n)t + kT)$  by setting  $k_1 = Q(n)/T$  and charging  $O(T)$  to each site of  $H_{i-1}(S_j)$  to pay for building  $H_i(S_j)$ .

#### 4 An insertion-only data structure

We describe a method that transforms a static  $k$ -NN data structure with query time  $O(Q(n) + k)$  into an insertion-only  $k$ -NN data structure with query time  $O(Q(n) \log n + k)$ . Insertions take  $O((P(n)/n) \log n)$  time, where  $P(n)$  is the preprocessing time of the static data structure, and  $C(n)$  is its space usage. We assume  $Q(n)$ ,  $P(n)$ , and  $C(n)$  are non-decreasing.

To support insertions, we use the logarithmic method [23]. We partition the sites into  $O(\log n)$  groups  $S_1, \dots, S_{O(\log n)}$  with  $|S_i| = 2^i$  for  $i \in \{1, \dots, O(\log n)\}$ . To insert a site  $s$ , a new group containing only  $s$  is created. When there are two groups of size  $2^i$ , these are removed and a new group of size  $2^{i+1}$  is created. For each group we store the sites in the static  $k$ -NN data structure. This results in an amortized insertion time of  $O((P(n)/n) \log n)$ . This bound can also be made worst-case [23]. The main remaining issue is then how to support queries in  $O(Q(n) \log n + k)$  time, thus avoiding an  $O(k \log n)$  term in the query time. Applying Lemma 3 directly solves this problem, and we thus obtain the following result.

► **Theorem 4.** *Let  $S$  be a set of  $n$  point sites, and let  $\mathcal{D}$  be a static  $k$ -NN data structure of size  $O(C(n))$ , that can be built in  $O(P(n))$  time, and answer queries in  $O(Q(n) + k)$  time. There is a  $k$ -NN data structure on  $S$  of size  $O(C(n))$  that supports queries in  $O(Q(n) \log n + k)$  time, and insertions in  $O((P(n)/n) \log n)$  time.*

##### 4.1 Points in the plane

In the Euclidean metric,  $k$ -nearest neighbors queries in the plane can be answered in  $O(\log n + k)$  time, using  $O(n)$  space and  $O(n \log n)$  preprocessing time [1, 10]. Hence:

► **Corollary 5.** *There is an insertion-only data structure of size  $O(n)$  that stores a set of  $n$  sites in  $\mathbb{R}^2$ , allows for  $k$ -NNs queries in  $O(\log^2 n + k)$  time, and insertions in  $O(\log^2 n)$  time.*

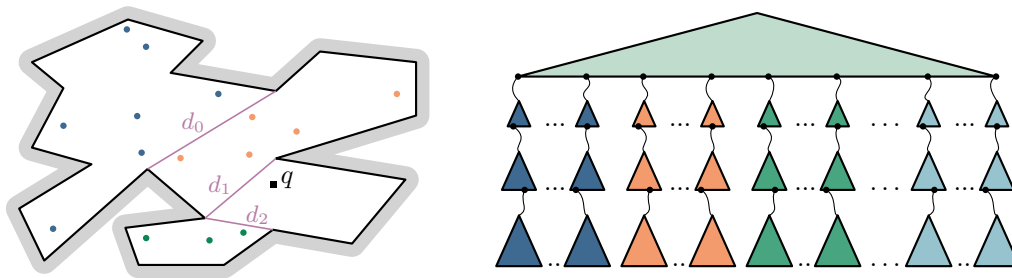
If we increase the size of each group in the logarithmic method to  $b^i$ , with  $b = \log^\varepsilon n$  and  $\varepsilon > 0$ , we get only  $O(\log_b n)$  groups instead of  $O(\log n)$ . This reduces the query time to  $O(\log^2 n / \log \log n + k)$ , matching the fully dynamic data structure. However, this also increases the insertion time to  $O(\log^{2+\varepsilon} n / \log \log n)$ . For general constant-complexity distance functions, we achieve the same query time using Liu's data structure [19], using  $O(n \log \log n)$  space and expected  $O(\text{polylog } n)$  insertion time.

##### 4.2 Points in a simple polygon

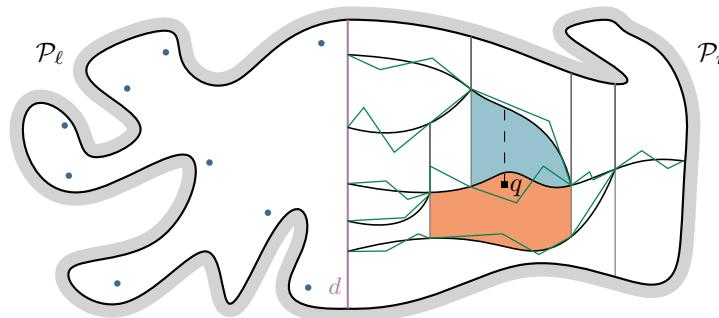
Next, we consider  $k$ -NN queries on a set  $S$  of  $n$  point sites inside a simple polygon  $\mathcal{P}$  with  $m$  vertices. For any two points  $p$  and  $q$  the (geodesic) distance  $d(p, q)$  is defined as the length of the shortest path  $\pi(p, q)$  between  $p$  and  $q$  fully contained within  $\mathcal{P}$ . Using  $O(m)$  space and preprocessing time, we can store  $\mathcal{P}$  so that  $d(p, q)$  can be computed in  $O(\log m)$  time [15].

To apply Theorem 4, we need a static data structure for geodesic  $k$ -NN queries. As we sketch below, we can obtain such a data structure by combining the approach of Chan [5] and Agarwal et al. [2]. However, building this data structure takes  $O(nm)$  time. We show that using more ideas from Agarwal et al. [2], together with our algorithm from Section 3, we can obtain a static  $k$ -NN data structure that can also be built efficiently. This in turn leads to an efficient insertion-only data structure.

**A static data structure.** The initial data structure consists of a hierarchy of lower envelopes of random samples  $R_0 \subset R_1 \subset \dots \subset R_{\log n}$ . For each sample, we store a (topological) vertical decomposition of the downward projection of the lower envelope and the conflict lists of the



■ **Figure 5** A partial decomposition of  $\mathcal{P}$  and the corresponding heap used in a  $k$ -NN query for  $q$ .



■ **Figure 6** Approximation (in green) of the vertical decomposition of the Voronoi diagram of  $S_\ell$  in  $\mathcal{P}_r$ . To find the trapezoid containing  $q$ , we consider both colored trapezoids.

corresponding (pseudo-)prisms. We can then find a prism in one of the vertical decompositions that contains the query point and whose conflict list has size  $O(k)$  in  $O(\log(n+m) + k \log m)$  time [5, 22]. This allows us to answer  $k$ -NN queries in the same time. The crux in this approach is in how to compute the conflict lists. We can naively compute these in  $O(mn)$  time by explicitly constructing the geodesic distance function for each site [14], and intersecting it with each of the  $O(n)$  pseudo-prisms. It is unclear how to improve on this bound.

► **Theorem 6.** *Let  $S$  be a set of  $n$  sites in a simple polygon  $\mathcal{P}$  with  $m$  vertices. In  $O(n(\log n \log^2 m + \log^3 m))$  time we can build a data structure of size  $O(n \log n \log m)$ , excluding the size of the polygon, that can answer  $k$ -NN queries with respect to  $S$  in  $O(\log(n+m) \log m + k \log m)$  time.*

**Proof Sketch.** To circumvent the issue above, we recursively partition the polygon  $\mathcal{P}$  into two subpolygons  $\mathcal{P}_r$  and  $\mathcal{P}_\ell$  of roughly the same size by a diagonal  $d$  [2]. We denote by  $S_r$  and  $S_\ell$  the sites in  $\mathcal{P}_r$  and  $\mathcal{P}_\ell$ , respectively. Theorem 22 of [2] provides us with a data structure that can find the  $k$ -NN among sites in  $S_\ell$  for a query point in  $\mathcal{P}_r$ . This is essentially the data structure that was described above. However, because the Voronoi diagram of sites in  $S_\ell$  restricted to  $\mathcal{P}_r$  is a Hamiltonian abstract Voronoi diagram [17], we can efficiently compute the conflict lists by only considering the functions intersecting the corners of each prism. We improve the query time of this data structure to  $O(\log(n+m) + k \log m)$  by incorporating the idea of Oh and Ahn [22] to approximate a geodesic Voronoi diagram by a polygonal subdivision, see Figure 6. Storing the points of both  $S_\ell$  and  $S_r$  at each of the  $O(\log m)$  levels of the decomposition in this data structure, and using our technique from Section 3 to query the levels simultaneously, see Figure 5, results in the stated query time. ◀

► **Corollary 7.** *Let  $\mathcal{P}$  be a simple polygon with  $m$  vertices. There is a data structure of size  $O(n \log n \log m + m)$  that stores a set of  $n$  point sites in  $\mathcal{P}$ , allows for geodesic  $k$ -NN queries in  $O(\log(n + m) \log n \log m + k \log m)$  expected time, and inserting a site in  $O(\log^2 n \log^2 m + \log n \log^3 m)$  time.*

## 5 A fully dynamic data structure

In this section, we consider  $k$ -NN queries while supporting both insertions and deletions, building on the results of Chan [7]. We first fill in the part missing from Chan [7]’s query algorithm. We then discuss a simple deletion-only  $k$ -NN structure. This allows us to adapt Chan’s  $k$ -NN data structure to more general distance functions like the geodesic distance.

### 5.1 A dynamic data structure for planes

Chan [7] describes how to adjust his 1-NN data structure to efficiently perform  $k$ -NN queries. There are two main changes: the conflict lists are stored in  $k$ -NN data structures  $\mathcal{D}_0$ , and the number of towers is reduced by using  $b = \log^\varepsilon n$ . Only the *live* planes of the conflict list of each prism  $\nabla$  are stored in the  $\mathcal{D}_0$  data structures. Each such structure uses linear space, can perform  $k'$ -NN queries in  $O(Q_0(|F_\nabla|) + k')$  time and deletions in  $D_0(|F_\nabla|)$  time. A different data structure is used to store small and large conflict lists. After building  $\mathcal{T}^{(i)}$ , each data structure  $\mathcal{D}_0$  of a prism  $\nabla$  is built on  $F_\nabla \cap F_{\text{live}}^{(i)}$ . The total space usage is  $O(n \log n)$ .

The insertions remain unchanged, but deleting a plane  $h$  requires extra work. In addition to increasing  $d_\nabla$  for each prism containing  $h$ ,  $h$  is explicitly removed from the  $\mathcal{D}_0$  data structures. Note that  $\mathcal{T}^{(i)}$  for which  $h \in F_{\text{live}}^{(i)}$  is the only tower whose  $\mathcal{D}_0$  data structures contain  $h$ . When a prism in tower  $\mathcal{T}^{(i)}$  is purged, we also delete its planes from the other  $\mathcal{D}_0$  data structures in  $\mathcal{T}^{(i)}$  to retain this property. This gives an amortized expected update time of  $U(n) = O(\log^{6+\varepsilon} n)$  [7]. The improvement of Kaplan et al. [16] reduces this to  $O(\log^{5+\varepsilon} n)$ . It follows from Lemma 1 and the above modifications that:

► **Lemma 8** (Chan [7]). *Let  $q$  be a query point. In  $O(t \log n)$  time, we can find  $t = O(\log_b n)$  prisms  $\nabla_1, \dots, \nabla_t$ , such that: (i) all prisms contain  $q$ , (ii) the conflict list of each prism has size  $O(k)$ , (iii) the conflict lists are pairwise disjoint and stored in a  $\mathcal{D}_0$  data structure, and (iv) the  $k$  sites in  $S$  closest to  $q$  appear in the union of the conflict lists of those prisms.*

So, to answer  $k$ -NN queries we can use a  $k_i$ -NN query on each  $\mathcal{D}_0$  data structure of the prisms  $\nabla_1, \dots, \nabla_t$ , where  $k_i$  is the number of sites from the  $k$ -nearest neighbours of  $q$  that appear in the conflict list of  $\nabla_i$ . This takes  $O(\sum_{i=1}^t Q_0(k) + k_i) = O(\sum_{i=1}^t Q_0(k) + k)$  time. However, it is unclear how to compute those  $k_i$  values. Fortunately, we can use Lemma 3 to find the  $k$ -nearest neighbors over all of the substructures in  $O(Q_0(k) \log_b n + k)$  time. Plugging in the appropriate query time  $Q_0(k)$  (see Chan [7] and Section 5.3), this achieves a total query time of  $O(\log^2 n / \log \log n + k)$  time as claimed.

### 5.2 A simple deletion-only data structure

Let  $H$  be a set of  $n$  planes, and let  $r \in \mathbb{N}$  be a parameter. We develop a data structure that supports reporting the  $t$  lowest planes above a query point  $q \in \mathbb{R}^2$  in  $O(n/r + \log r + t)$  time, and deletions in  $O(r \log n)$  time. Our entire data structure consists of just  $\ell = O(\log r)$   $k_i$ -shallow cuttings  $\Lambda_{k_0}, \dots, \Lambda_{k_\ell}$  of the planes, where  $k_i = \lfloor 2^i(n/r) \rfloor$ . Hence, this uses  $O(n \log r)$  space. We can compute the shallow cuttings along with their conflict lists in  $O(n \log n)$  time [10]. Note that when  $r > n$ , it can be that  $k_i = 0$  for some  $i$ . In this case, we simply do



not build any of the cuttings that have  $k_i = 0$ . For our application, we are mostly interested in the deletion time of the data structure, and less in the query time. By picking  $r$  to be small, we can make deletions efficient at the cost of making the query time fairly terrible.

**Deletions.** If we delete a plane, we remove it from all conflict lists in all cuttings. Since cutting  $\Lambda_{k_i}$  has size  $O(r/2^i)$ , each plane occurs at most  $O(r/2^i)$  times in this cutting. Hence, the total time to go through all of these prisms is  $\sum_{i=0}^{O(\log r)} r/2^i = O(r)$  time. When more than half of the planes from any conflict list are removed, we rebuild the entire data structure. Because every conflict list contains at least  $n/r$  planes, at least  $\frac{n}{2r}$  deletions take place before a global rebuild. We charge the  $O(n \log n)$  cost of rebuilding to these planes, so we charge  $O(r \log n)$  to each deletion. Deletions thus take amortized  $O(r \log n)$  time.

**Queries.** We report the  $t$ -lowest planes at a query point  $q$  as follows. We consider the cutting for which  $k_i = 2^i(n/r) = O(t)$ , so at level  $i = \lceil \log(ctr/n) \rceil$ , for some large enough constant  $C$ . When  $t < n/r$  there is no such cutting, so we query the lowest level cutting instead. We find the prism containing  $q$  by a point location query. As the largest cutting has size  $O(r)$ , this takes  $O(\log r)$  time. We then simply report the  $t$  lowest planes at  $q$  by going through the entire conflict list. This results in a query time of  $O(\log r + n/r + t)$ .

**Reducing space usage.** When  $n$  is large w.r.t.  $r$ , we can use a similar approach to Chan [6, 7] to achieve linear space usage, by storing only the prisms of the shallow cuttings, and storing the planes in an auxiliary data structure [3]. We then obtain the following result.

► **Lemma 9.** *For any fixed  $r$ , we can construct a data structure of size  $O(n \log r)$ , or  $O(n)$  when  $n \geq r^{1/\varepsilon}$ , in  $O(n \log n)$  time that stores a set of  $n$  planes, allows for  $t$ -lowest planes queries in  $O(\log r + n/r + t)$  time and deletions in  $O(r \log n)$  time.*

**General data structure.** The general idea in the above data structure can be applied to any type of functions for which we have an algorithm to compute  $k$ -shallow cuttings. Note that the “lowest” cutting we use is an  $n/r$ -shallow cutting. It follows that constructing all shallow cuttings takes  $O(T(n, n/r) \log r)$  time. To delete a function, we remove it from the conflict lists in  $\sum_{i=0}^{O(\log r)} S(n, k_i) = O((r/n)S(n, 1))$  time, and we charge  $O((r/n)T(n, n/r) \log r)$  to the deletion to pay for the global rebuild. To answer a query, we simply find the prism containing  $q$  in one cutting, so the query time is  $O(Q(n, n/r) + n/r + t)$ . We thus have:

► **Lemma 10.** *For any fixed  $r$ , we can construct a data structure of size  $O(S(n, 1) \log r)$  in  $O(T(n, n/r) \log r)$  time that stores a set of  $n$  functions, allows for  $t$ -lowest functions queries in  $O(Q(n, n/r) + n/r + t)$  time and deletions in  $O((r/n)(S(n, 1) + T(n, n/r) \log r))$  time.*

### 5.3 A general dynamic data structure

To generalize the dynamic  $k$ -NN data structure from Section 5.1 to other types of distance functions, we replace the  $\mathcal{D}_0$  data structures by the data structure of Section 5.2. Queries and updates are performed as before (see Sections 2.2 and 5.1). This results in a dynamic  $k$ -NN data structure that can be used for any type of distance functions for which we can construct  $k$ -shallow cuttings. Next, we analyze the space usage and the running time in case of the Euclidean distance, as this is somewhat easier to follow, and then generalize to arbitrary distance functions.

## 14:12 Dynamic Data Structures for $k$ -Nearest Neighbor Queries

**Query time.** Our  $\mathcal{D}_0$  data structure has query time  $Q_0(n') = O(\log r + n'/r)$  and deletion time  $D_0(n') = O(r \log n')$ . Because we query the cutting at level  $j_k$ , the size of each conflict list we query is  $O(k)$ . By using our scheme to find the  $k$ -nearest neighbors over the substructures simultaneously, the query time becomes:

$$Q(n) = O([\log n + Q_0(O(k))] \log_b n + k) = O((\log n + (\log r + k/r)) \log_b n + k).$$

If we set  $r = \log n$ , and  $b = \log^\varepsilon n$ , we get  $Q(n) = O(\log^2 n / \log \log n + k)$ , matching the query time of Chan's approach.

**Update time.** Lemma 1 states that insertion time is given by  $I(n) = O(b \log_b n \cdot (P(n)/n))$ , where  $P(n)$  is the preprocessing time of  $\mathcal{D}$ . Our preprocessing time increases w.r.t. to the original data structure, since after building the hierarchy of shallow cuttings for a tower, we additionally need to build the structures  $\mathcal{D}_0$  on each of the conflict lists. As before, building the shallow cuttings takes  $O(n \log n)$  time [8]. Next, we analyse the time to build all data structures  $\mathcal{D}_0$ . Note that the cutting at level  $j$  in the hierarchy consists of  $O(n/k_j)$  prisms, and the size each conflict list in the cutting is  $O(k_j)$ . Let  $\alpha$  be the constant bounding the size of the conflict lists. Using that  $P_0(n') = O(n' \log n')$ , we find the following running time:

$$\begin{aligned} \sum_{j=0}^{\log \frac{n}{k_0}} O\left(\frac{n}{k_j}\right) \cdot P_0(\alpha k_j) &= \sum_{j=0}^{\log \frac{n}{k_0}} O\left(\frac{n}{k_j}\right) \cdot O(\alpha k_j \log(\alpha k_j)) \\ &= \sum_{j=0}^{\log \frac{n}{k_0}} O(n \log(\alpha k_j)) = O(n \log^2 n). \end{aligned}$$

The preprocessing time thus adheres to the recurrence relation  $P(n) \leq P(n/b) + O(n \log^2 n)$ , which solves to  $P(n) = O(n \log^2 n)$ . It follows that  $I(n) = O(b \log_b n \cdot (P(n)/n)) = O(b \log^2 n \log_b n) = O(\log^{3+\varepsilon} n / \log \log n)$ . Note that the improvement of building all shallow cuttings in a tower in  $O(n \log n)$  time does not improve the insertion time to  $O(\log^2 n)$  as in the 1-NN data structure, because building the  $\mathcal{D}_0$  data structures is the dominant term.

When deleting a plane  $h$ , with  $h \in F_{\text{live}}^{(i)}$ , we remove  $h$  from all  $\mathcal{D}_0$  of  $\mathcal{T}^{(i)}$  with  $h \in \mathcal{D}_0$ . There are at most  $O(b \log n)$  such data structures  $\mathcal{D}_0$ . By Lemma 1, deleting a plane causes amortized  $O(b \log n \log_b n)$  reinsertions. Each reinserted plane is also removed from the structures  $\mathcal{D}_0$  of a single tower. We can thus formulate the deletion time as  $D(n) = O(b \log n \log_b n \cdot (b \log n \cdot D_0(n) + I(n)))$ . Plugging in  $D_0(n) = O(\log^2 n)$  and  $I(n) = O(b \log^2 n \log_b n)$ , we find  $D(n) = O(b^2 \log^4 n \log_b n + b^2 \log^3 n \log_b^2 n) = O(\log^{5+\varepsilon} n / \log \log n)$ .

**Space usage.** The space usage of a  $\mathcal{D}_0$  data structure storing  $n'$  planes is  $O(n' \log r)$ . The space usage is thus  $S(n) = \sum_{j=0}^{\log \frac{n}{k_0}} \frac{n}{k_j} \cdot O(k_j \log r) = O(n \log n \log \log n)$ . Note that this can be reduced to  $O(n \log n)$  by using the space reduction idea mentioned in Section 5.2.

We can use the same scheme for any distance measure that allows for constructing a  $k$ -shallow cutting. In the full version of this paper [12], we prove the following lemma (Lemma 11), which we then apply to constant description complexity distance functions and geodesic distance functions (refer to full version [12] for details) to obtain Theorem 12.

► **Lemma 11.** *Given an algorithm to construct a  $k$ -shallow cutting of size  $S(n, k)$  on  $n$  functions in  $T(n, k)$  time, such that locating the prism containing a query point takes  $Q(n, k)$  time, we can construct a data structure of size  $O(S(n, 1)^2/n \cdot \log n \log \log n)$  that maintains a set of  $n$  functions and allows for  $k$ -lowest functions queries in  $O(Q(n, 1) \log n / \log \log n + k)$  time. Inserting a function takes  $O((S(n, 1)T(n, 1)/n^2) \log^{2+\varepsilon} n)$  amortized time, and deleting a function takes  $O((T(n, 1)/\log \log n + T(n, n/\log n)) \cdot S(n, 1)^2 \log^{4+\varepsilon} n/n^3)$  amortized time.*



► **Theorem 12.** *There is a fully dynamic data structure of size  $S(n)$  that stores a set of  $n$  sites and allows for  $k$ -nearest neighbors queries in  $Q(n)$  time, insertions in  $I(n)$  expected amortized time, and deletions in  $D(n)$  expected amortized time.  $\mathcal{P}$  is a simple polygon with  $m$  vertices, and  $\lambda_s(t)$  denotes the maximum length of a Davenport-Schinzel sequence of order  $s$  on  $t$  symbols and  $s$  a constant depending on the functions.*

	Euclidean in $\mathbb{R}^2$	General in $\mathbb{R}^2$	Geodesic in $\mathcal{P}$
$Q(n)$	$O\left(\frac{\log^2 n}{\log \log n} + k\right)$	$O\left(\frac{\log^2 n}{\log \log n} + k\right)$	$O\left(\frac{\log^2 n \log^2 m}{\log \log n} + k \log m\right)$
$I(n)$	$O\left(\frac{\log^{3+\varepsilon} n}{\log \log n}\right)$	$O(\log^{5+\varepsilon} n \lambda_{s+2}(\log n))$	$O(\log^{8+\varepsilon} n \log m + \log^{7+\varepsilon} n \log^3 m)$
$D(n)$	$O\left(\frac{\log^{5+\varepsilon} n}{\log \log n}\right)$	$O(\log^{7+\varepsilon} n \lambda_{s+2}(\log n))$	$O\left(\frac{\log^{12+\varepsilon} n \log m + \log^{11+\varepsilon} n \log^3 m}{\log \log n}\right)$
$S(n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log^5 n \log m \log \log n + m)$

## References

- 1 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 180–186. SIAM, 2009. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496791>.
- 2 Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. In *34th International Symposium on Computational Geometry, SoCG*, volume 99 of *LIPICs*, pages 4:1–4:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.SoCG.2018.4.
- 3 Pankaj K. Agarwal and Jirí Matousek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995. doi:10.1007/BF01293483.
- 4 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008. doi:10.1145/1327452.1327494.
- 5 Timothy M. Chan. Random sampling, halfspace range reporting, and construction of  $\leq k$ -levels in three dimensions. *SIAM J. Comput.*, 30(2):561–575, 2000. doi:10.1137/S0097539798349188.
- 6 Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.
- 7 Timothy M. Chan. Three problems about dynamic convex hulls. *Int. J. Comput. Geom. Appl.*, 22(4):341–364, 2012. doi:10.1142/S0218195912600096.
- 8 Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. In *35th International Symposium on Computational Geometry, SoCG*, volume 129 of *LIPICs*, pages 24:1–24:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.SoCG.2019.24.
- 9 Timothy M. Chan. personal communication, 2021.
- 10 Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discret. Comput. Geom.*, 56(4):866–881, 2016. doi:10.1007/s00454-016-9784-4.
- 11 Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.
- 12 Sarita de Berg and Frank Staals. Dynamic data structures for  $k$ -nearest neighbor queries, 2021. arXiv:2109.11854.
- 13 Greg N. Frederickson. An optimal algorithm for selection in a min-heap. *Inf. Comput.*, 104(2):197–214, 1993. doi:10.1006/inco.1993.1030.
- 14 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.
- 15 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.

- 16 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/s00454-020-00243-7.
- 17 Rolf Klein and Andrzej Lingas. Hamiltonian abstract voronoi diagrams in linear time. In *Algorithms and Computation, 5th International Symposium, ISAAC, Proceedings*, volume 834 of *Lecture Notes in Computer Science*, pages 11–19. Springer, 1994. doi:10.1007/3-540-58325-4\_161.
- 18 Der-Tsai Lee. On  $k$ -nearest neighbor voronoi diagrams in the plane. *IEEE Transactions on Computers*, C-31(6):478–487, 1982.
- 19 Chih-Hung Liu. Nearly optimal planar  $k$  nearest neighbors queries under general distance functions. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2842–2859. SIAM, 2020. doi:10.1137/1.9781611975994.173.
- 20 Chih-Hung Liu and D. T. Lee. Higher-order geodesic voronoi diagrams in a polygonal domain with holes. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1633–1645. SIAM, 2013. doi:10.1137/1.9781611973105.117.
- 21 Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry Theory and Applications*, 2(3):169–186, 1992.
- 22 Eunjin Oh and Hee-Kap Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discret. Comput. Geom.*, 63(2):418–454, 2020. doi:10.1007/s00454-019-00063-4.
- 23 Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. doi:10.1007/BFb0014927.

# Preference-Based Trajectory Clustering – An Application of Geometric Hitting Sets

Florian Barth ✉

Universität Stuttgart, Germany

Stefan Funke ✉

Universität Stuttgart, Germany

Claudius Proissl ✉

Universität Stuttgart, Germany

---

## Abstract

In a road network with multicriteria edge costs we consider the problem of computing a minimum number of driving preferences such that a given set of paths/trajectories is optimal under at least one of these preferences. While the exact formulation and solution of this problem appears theoretically hard, we show that in practice one can solve the problem exactly even for non-homeopathic instance sizes of several thousand trajectories in a road network of several million nodes. We also present a parameterized guaranteed-polynomial-time scheme with very good practical performance.

**2012 ACM Subject Classification** Theory of computation → Discrete optimization; Theory of computation → Computational geometry

**Keywords and phrases** Route planning, personalization, computational geometry

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.15

**Supplementary Material** *Software (Source Code and Data)*: <https://doi.org/10.17605/osf.io/4qkuv>

**Funding** This work was in part supported by the Deutsche Forschungsgemeinschaft (DFG) within the priority program 1894: Volunteered Geographic Information: Interpretation, Visualization and Social Computing.

## 1 Introduction

It is well observable in practice that drivers' preferences are not homogeneous. If we have two alternative paths  $\pi_1, \pi_2$  between a given source-target pair, characterized by 3 costs/metrics (travel time, distance, and ascent along the route) each, e.g.,  $c(\pi_1) = (27min, 12km, 150m)$ , and  $c(\pi_2) = (19min, 18km, 50m)$ , there are most likely people who prefer  $\pi_1$  over  $\pi_2$  and vice versa. In previous works, people have tried to formalize these preferences. The most common model here assumes a linear dependency on the metrics. While typical real-world trajectories are not necessarily optimal in such a model, they are usually decomposable into very few optimal subtrajectories, see, e.g., [4]. This preference model allows for *personalized route planning*, where a routing query not only consists of source and destination but also a weighting of the metrics in the network.

Since this weighting of the metrics – often called *preference* – is hard to specify as a user of such a personalized route planning system, methods have been developed which infer the preferences from paths that the user has traveled before. The larger a set of paths is, though, the less likely it is that a single preference/weighting exists which explains all paths, i.e., for which all paths are optimal. One might, for example, think of different driving styles/preferences when commuting versus leisure trips through the country side. So a natural question to ask is, what is the minimum number of preferences necessary to explain a set of given paths in a road network with multiple metrics on the edges. This can also be



© Florian Barth, Stefan Funke, and Claudius Proissl;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 15; pp. 15:1–15:14

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

interpreted as a trajectory clustering task where routes are to be classified according to their purpose. In our example, one might be able to differentiate between commute and leisure. Or in another setting, where routes of different drivers are analyzed, one might be able to cluster them into speeders and cruisers depending on the routes they prefer.

The goal of this paper is the formulation of this natural optimization problem in the context of multicriteria routing and investigate the theoretical and practical challenges of solving this problem.

## Related Work

The linear preference model in the navigation context has been used in numerous works, e.g., [13, 11, 12, 9], to allow for personalized route planning services. Fewer papers deal with the inference of personal preferences like [17, 8]. The latter paper is also most related to this work as it introduced preference inference based on a linear programming formulation, which we will also instrument in our approach. Note, though, that while [8] already talked about the problem of minimizing the number of preferences to explain a set of trajectories, no serious attempt at getting optimal or close-to-optimal solutions has been made.

## Our Contribution

In this paper we show how to mathematically formulate the problem of finding the minimum number of preferences to explain a set of given paths in a road network with multiple edge metrics. We investigate both theoretical challenges as well as practical solvability and provide a guaranteed polynomial-time heuristic as well as an exact (but potentially superpolynomial-time) solution. Our experimental results show that for problem instances based on data from the OpenStreetMap project, we can compute optimal preference sets for thousands of paths within few minutes on road networks with several million nodes and edges.

## 2 Preliminaries

Our work is based on a *linear* preference model, i.e., for a given directed graph  $G(V, E)$  we have for every edge  $e \in E$  a  $d$ -dimensional cost vector  $c(e) \in \mathbb{R}^d$ , where  $c_1(e), c_2(e), \dots, c_d(e) \geq 0$  correspond to non-negative quantities like travel time, distance, non-negative ascent,  $\dots$ , which are to be minimized. A path  $\pi = e_1 e_2 \dots e_k$  in the network then has an associated cost vector  $c(\pi) := \sum_{i=1}^k c(e_i)$ .

A preference to distinguish between different alternative paths is specified by a vector  $\alpha \in [0, 1]^d$ ,  $\sum \alpha_i = 1$ . For example,  $\alpha^T = (0.4, 0.5, 0.1)$  might express that the respective driver does not care much about ascents along the route, but considers travel time and distance similarly important. Alternative paths  $\pi_1$  and  $\pi_2$  are compared by evaluating the respective scalar products of the cost vectors of the path and the preference, i.e.,  $c(\pi_1)^T \cdot \alpha$  and  $c(\pi_2)^T \cdot \alpha$ . Smaller scalar values in our linear model correspond to a preferred alternative. An  $st$ -path  $\pi$  (which is a path with source  $s$  and target  $t$ ) is optimal for a fixed preference  $\alpha$  if no other  $st$ -path  $\pi'$  exists with  $c(\pi')^T \cdot \alpha < c(\pi)^T \cdot \alpha$ .

Of course, this linear model is a bold simplification of actual drivers' preferences, yet it has been observed that real-world trajectories can typically be decomposed into very few optimal subtrajectories [4] (significantly less than for a single metric), so this linear model appears to be a reasonable approximation of reality.

## 2.1 Personalized Route Planning

Using the linear model it is easy to allow for *personalized route planning*, i.e., a query does not only consist of a source  $s$  and a target  $t$ , but also specifies a preference  $\alpha \in [0, 1]^d$ . The expected answer to such a query  $(s, t, \alpha)$  is a path  $\pi$  that is optimal for preference  $\alpha$ . Dijkstra’s algorithm can be instrumented to answer such queries by evaluating the scalar product of the edge cost vector and  $\alpha$  when relaxing an edge in the course of the algorithm.

Since Dijkstra’s algorithm is not really useful for continent-sized road networks people have come up with speed-up schemes, see [13, 12, 11], yet it appears considerably more difficult to achieve the speed-up factors of respective schemes for the single metric case.

## 2.2 Preference Inference

From a practical point of view, it is very unintuitive (or rather: almost impossible) for a user to actually express his driving preferences as such a vector  $\alpha$ , even if he is aware of the units of the cost vectors on the edges. Hence it would be very desirable to be able to *infer* his preferences from paths he likes or which he has traveled before. [8] proposed a technique for preference inference, which essentially instruments linear programming to determine an  $\alpha$  for which a given path  $\pi$  is optimal or certify that none exists. Given an  $st$ -path  $\pi$  in a road network, in principle, their proposed LP has non-negative variables  $\alpha_1, \dots, \alpha_d$  and one constraint for each  $st$ -path  $\pi'$  which states, that “for the  $\alpha$  we are after,  $\pi'$  should not be preferred over  $\pi$ ”. So the LP looks as follows:

$$\begin{array}{llll}
 \max & & \alpha_1 & \\
 \forall st\text{-paths } \pi' : & \alpha^T(c(\pi) - c(\pi')) & \leq 0 & \textit{optimality constraints} \\
 & \alpha_i & \geq 0 & \textit{non-negativity constraints} \\
 & \sum \alpha_i & = 1 & \textit{scaling constraint}
 \end{array}$$

Note, that an objective function is not really necessary, as we only care about feasibility, that is, existence of an  $\alpha$  satisfying all constraints. As such, this linear program is of little use, since typically there is an exponential number of  $st$ -paths, so just writing down the complete LP seems infeasible. Fortunately, due to the equivalence of optimization and separation, it suffices to have an algorithm at hand which – for a given  $\alpha$  – decides in polynomial time whether all constraints are fulfilled or if not, provides a violated constraint (such an algorithm is called a *separation oracle*). In our case, this is very straightforward: we simply compute (using, e.g., Dijkstra’s algorithm) the optimum  $st$ -path for a given  $\alpha$ . If the respective path has better cost (wrt  $\alpha$ ) than  $\pi$ , we add the respective constraint and resolve the augmented LP for a new  $\alpha$ , otherwise we have found the desired  $\alpha$ . Via the Ellipsoid method [14] this approach has polynomial running time, in practice the dual Simplex algorithm has proven to be very efficient.

### 3 Driving Preferences and Geometric Hitting Sets

The approach from Section 2.2 can easily be extended to decide for a *set* of paths (with different source-target pairs) whether there exists a single preference  $\alpha$  for which they are optimal (i.e., which explains this route choice). It does not work, though, if different routes for the same source-target pair are part of the input or simply no single preference can

explain all chosen routes. The latter seems quite plausible when considering that one would probably prefer other road types on a leisure trip on the weekend versus the regular commute trip during the week. So the following optimization problem is quite natural to consider:

Given a set of trajectories  $T$  in a multiweighted graph, determine a set  $A$  of preferences of minimal cardinality, such that each  $\pi \in T$  is optimal with respect to at least one  $\alpha \in A$ .

We call this problem *preference-based trajectory clustering* (PTC).

For a concrete problem instance from the real world, one might hope that each preference in the set  $A$  then corresponds to a driving style like speeder or cruiser. Also note, that a real-world trajectory often is not optimal for a single  $\alpha$  (prime example for that would be a round trip), yet, studies like in [4] show that it can typically be decomposed into very few optimal subtrajectories if multiple metrics are available.

In [8], a sweep algorithm is introduced that computes an approximate solution of PTC. It is, however, relatively easy to come up with examples where the result of this sweep algorithm is by a factor of  $\Omega(|T|)$  worse than the optimal solution. We aim to improve this result by finding practical ways to solve PTC optimally as well as approximately with better quality guarantees. Our (surprisingly efficient) strategy is to explicitly compute for each trajectory  $\pi$  in  $T$  the polyhedron of preferences for which  $\pi$  is optimal and to translate PTC into a geometric hitting set problem.

Fortunately, the formulation as a linear program as described in 2.2 already provides a way to compute these polyhedra. The constraints in the LP from 2.2 exactly characterize the possible values of  $\alpha$  for which one path  $\pi$  is optimal. These values are the intersection of half-spaces described by the optimality constraints and the non-negativity constraints of the LP. We call this (convex) intersection *preference polyhedron*. A preference polyhedron  $P$  of path  $\pi$  is  $d - 1$  dimensional, where  $d$  is the number of metrics. This is because the LP's scaling constraint reduces the dimension by one and we can substitute  $\alpha_d$  by  $1 - \sum_{i < d} \alpha_i$ . In the remainder of this Section, we discuss how to construct preference polyhedra from given paths and reformulate PTC as a minimum geometric hitting set problem.

### 3.1 Exact Polyhedron Construction

A straightforward, yet quite inefficient way of constructing the preference polyhedron for a given path  $\pi$  is to actually determine *all* simple  $st$ -paths and perform the respective half-space intersection. Even when restricting to pareto-optimal paths and real-world networks, their number is typically huge. So we need more efficient ways to construct the preference polyhedron.

#### 3.1.1 Boundary Exploration

With the tool of linear programming at hand, one possible way of exploring the preference polyhedron is by repeated invocation of the LP as in Section 2.2 but with varying objective functions. For sake of simplicity let us assume that the preference polyhedron is full (that is,  $d - 1$ <sup>1</sup>) dimensional. We first determine  $d - 1$  distinct extreme points of the polyhedron to obtain a  $d - 1$ -simplex as first (inner) approximation of the preference polyhedron. We then repeatedly invoke the LP with an objective function corresponding to the normal vectors of

---

<sup>1</sup> after elimination of  $\alpha_d$

the facets of the current approximation of the polyhedron. The outcome is either that the respective facet is part of the final preference polyhedron, or a new extreme point is found which destroys this facet and induces new facets to be investigated later on. If the final preference polyhedron has  $f$  facets, this approach clearly requires  $O(f)$  invocations of the LP solver (and some effort to maintain the current approximation of the preference polyhedron as the convex hull of the extreme points found so far). While theoretically appealing, in practice this approach is not very competitive due to the overhead of repeated LP solving.

### 3.1.2 Corner Cutting Approach

We developed the following *Corner Cutting Approach (CCA)* which in practice turns out to be extremely efficient to compute for a given path  $\pi$  its preference polyhedron, even though we cannot bound its running time in terms of the complexity of the produced polyhedron.

CCA is an iterative algorithm, which computes the feasibility polyhedron via a sequence of half-space intersections. Let  $P_\pi$  be the preference polyhedron of path  $\pi$ . In the  $i$ -th iteration CCA computes a polyhedron  $P_i$  with  $P_\pi \subseteq P_i \subseteq P_{i-1}$ .  $P_0$  is the entire preference space with a number of corners equal to the number of metrics. Each of these corners is initially marked as unchecked.

In iteration  $i$  CCA takes one corner  $\alpha^{(i)}$  of  $P_{i-1}$  that has not been checked before. If no such corner exists, CCA terminates and returns  $P_{CCA} := P_{i-1}$ . Otherwise, it marks the corner  $\alpha^{(i)}$  as checked and computes the optimal path  $\pi(\alpha^{(i)})$ . If  $(\alpha^{(i)})^T (c(\pi) - c(\pi(\alpha^{(i)}))) = 0$ , the corner  $\alpha^{(i)}$  belongs to  $P_\pi$  and there is nothing to do. Otherwise, the constraint  $\alpha^T (c(\pi) - c(\pi(\alpha^{(i)}))) \leq 0$  is violated by a part of  $P_{i-1}$ . We call this part  $P'_i$ . It is clear that  $\alpha^{(i)} \in P'_i$ . Finally, the polyhedron  $P_i := P_{i-1} \setminus P'_i$  is computed by intersecting  $P_{i-1}$  with the half-space  $\alpha^T (c(\pi) - c(\pi(\alpha^{(i)}))) \leq 0$ . This intersection may introduce new corners, which are marked as unchecked. Afterwards, the next iteration starts.

We prove that CCA indeed computes  $P_\pi$ . Let  $P_{CCA}$  be the output of CCA. We first show that  $P_\pi \subseteq P_{CCA}$ .  $P_\pi \subseteq P_0$  is trivially true. Furthermore, in each iteration  $i$  it is clear that  $P'_i \cap P_\pi = \emptyset$ . Hence, for each iteration  $i$ , we have  $P_\pi \subseteq P_i$  and therefore  $P_\pi \subseteq P_{CCA}$ . The other direction  $P_{CCA} \subseteq P_\pi$  follows from the fact that  $P_\pi$  is convex and that all corners of  $P_{CCA}$  belong to  $P_\pi$  as they are marked as checked.

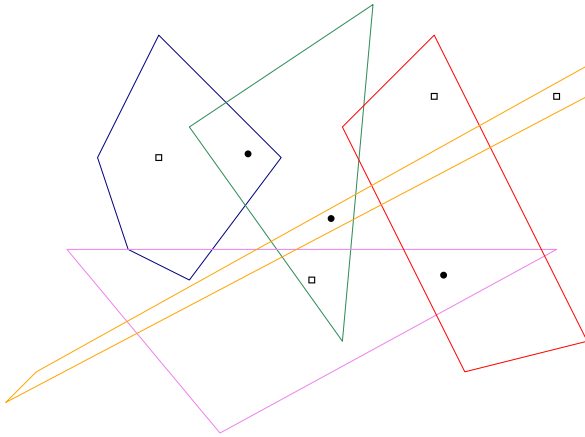
CCA also terminates for finite graphs. Each optimal path  $\pi(\alpha^{(i)})$  can add finitely many corners to  $P_i$  only once. Since the number of optimal paths is finite the number of corners to be considered is finite as well.

The great advantage of CCA compared to the boundary exploration approach is the avoidance of the linear programming solver.

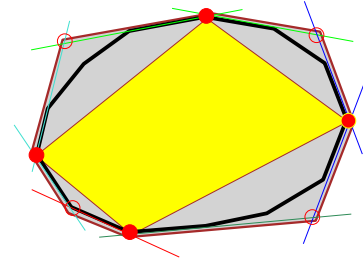
## 3.2 Minimum Geometric Hitting Set

Using the preference polyhedra we are armed to rephrase our original problem as a *geometric hitting set (GHS)* problem. In an instance of GHS we typically have geometric objects (possibly overlapping) in space and the goal is to find a set of points (a *hitting set*) of minimal cardinality, such that each of the objects contains at least one point of the hitting set. Figure 1 shows an example of how preference polyhedra of different optimal paths could look like in case of three metrics. In terms of GHS, our PTC problem is equivalent to finding a hitting set for the preference polyhedra of minimum cardinality, and the “hitters” correspond to respective preferences. In Figure 1 we have depicted two feasible hitting sets (white squares and black circles) for this instance. Both solutions are minimal in that no hitter can be removed without breaking feasibility. However, the white squares (in contrast to the black circles) do not describe a minimum solution as one can hit all polyhedra with less points.





■ **Figure 1** Example of a geometric hitting set problem as it may occur in the context of PTC. Two feasible hitting sets are shown (white squares and black circles).



■ **Figure 2** Inner (yellow) and outer approximation (grey) of the preference polyhedron (black).

While the GHS problem allows to pick arbitrary points as hitters, it is not hard to see that it suffices to restrict to vertices of the polyhedra and intersection points between the polyhedra boundaries, or more precisely vertices in the arrangement of feasibility polyhedra. We describe the computation of these candidates for the hitting sets in Section 3.3.

The GHS instance is then formed in a straightforward manner by having all the hitting set candidates as ground set, and subsets according to containment in respective preference polyhedra. For an exact solution we can formulate the problem as an integer linear program (ILP). Let  $\alpha^{(1)}, \alpha^{(2)}, \dots, \alpha^{(l)}$  be the hitting set candidates and  $\mathcal{U} := \{P_1, P_2, \dots, P_k\}$  be the set of preference polyhedra. We create a variable  $X_i \in \{0, 1\}$  indicating whether  $\alpha^{(i)}$  is picked as a hitter and use the following ILP formulation:

$$\begin{aligned} & \min \sum_i X_i \\ \forall P \in \mathcal{U} : & \sum_{\alpha^{(i)} \in P} X_i \geq 1 \\ \forall i : & X_i \in \{0, 1\} \end{aligned}$$

While solving ILPs is known to be NP-hard, it is often feasible to solve ILPs derived from real-world problem instances even of non-homeopathic size.

### 3.3 Hitting Set Instance Construction via Arrangements of Hyperplanes

To obtain the actual hitting set instance, we overlay the individual preference polyhedra. This can be done via construction of the arrangement of the hyperplanes bounding the preference polyhedra. Each vertex in this arrangement then corresponds to a candidate for the hitting set. If  $N$  is the total number of hyperplanes bounding all preference polyhedra in  $D$ -dimensional space, then this arrangement has complexity  $O(N^D)$  and can be computed by a topological sweep within the same time bound [6]. For  $K$  polyhedra with overall  $N$  bounding hyperplanes we obtain a hitting set instance with  $K$  sets and  $O(N^D)$  potential hitter candidates.





(a) Before inserting new optimal path.

(b) After inserting new optimal path.

■ **Figure 3** Example of preference polyhedra of optimal paths with the same source and target and with equal cost in the third metric.

### 3.4 Challenges

While our proposed approach to determine the minimum number of preferences to explain a set of given paths is sound, it raises two major issues. First, the complexity of a single preference polyhedron might be exponential (or just too large for actual computation), so just writing down the geometric hitting set instance becomes infeasible in practice. Second, solving a geometric hitting set instance to optimality is far from trivial. In the following we will briefly discuss these two issues and then in the next section come up with remedies.

#### 3.4.1 Preference Polyhedron Complexity

In the following, we show that the complexity of preference polyhedra can be arbitrarily high. We proceed in two steps. In the first step we show that if we find  $k$  optimal  $st$ -paths with two metrics we can construct a preference polyhedron with more than  $k$  facets by adding a third metric. In the second step we show that for a single  $st$ -pair there can be arbitrarily many preference polyhedra with non-zero volume with two metrics.

Let us first assume we have two metrics and  $k > 1$  optimal  $st$ -paths  $\pi_1$  to  $\pi_k$ . Furthermore, we assume that each of the  $k$  optimal paths has a preference polyhedron with non-zero volume (a polyhedron is a line in the two-metrics case). We refer to the  $j$ -th entry of cost vector  $c(\pi_i)$  with  $c(\pi_i)_j$ . Let  $x := \max_{1 \leq i \leq k} c(\pi_i)_1 + c(\pi_i)_2$  be the maximum of the sums of the cost vectors.

We now introduce a third metric with a constant cost of  $3x$  for each path. It is clear that each of the  $k$  optimal paths is optimal for the preference  $\alpha_3 = [0, 0, 1]$ . Hence, each preference polyhedron is now a triangle as shown in Figure 3a.

Finally, we create a new optimal path  $\pi_{k+1}$  with the cost vector  $c(\pi_{k+1}) := [2x, 2x, 2x]$ . This path is clearly optimal for the preference  $\alpha_3$ . Since  $\alpha_3^T (c(\pi_{k+1}) - c(\pi_i))$  is strictly less than zero for each  $1 \leq i \leq k$  the volume of the preference polyhedron of  $\pi_{k+1}$  is non-zero. Furthermore, restricted to the first two metrics  $\pi_{k+1}$  is not optimal. This directly follows from the definition of  $x$ . Hence, the preference polyhedron of  $\pi_{k+1}$  shares a constraint with each of the  $k$  other optimal paths as shown in Figure 3b.

We now discuss that with two metrics there can be arbitrarily many shortest paths with the same source and target that have a non-zero volume preference polyhedron.

Let us assume that there are  $k + 1$   $st$ -paths  $\pi_0, \pi_1, \dots, \pi_k$  with cost vectors  $c(\pi_i) := [i^2, (k - i)^2]$ . This could be easily realized with one-edge paths. A preference is in this case a tuple  $[1 - \alpha, \alpha]$  with  $0 \leq \alpha \leq 1$ .

For any  $0 \leq i < k$  we have  $c(\pi_i) - c(\pi_{i+1}) = [-2i - 1, 2(k - i) - 1]$ . With  $\alpha = \frac{2i+1}{2k}$  we get

$$[1 - \alpha, \alpha]^T (c(\pi_i) - c(\pi_{i+1})) = \left[ \frac{2(k-i)-1}{2k}, \frac{2i+1}{2k} \right]^T [-2i-1, 2(k-i)-1] = 0$$

Hence, for  $0 < i < k$  the path  $\pi_i$  is optimal for the range  $\alpha \in \left[ \frac{2i-1}{2k}, \frac{2i+1}{2k} \right]$ . Path  $\pi_0$  is optimal for the range  $\alpha \in \left[ 0, \frac{1}{2k} \right]$  and path  $\pi_k$  is optimal for the range  $\alpha \in \left[ \frac{2k-1}{2k}, 1 \right]$ .

The question remains whether the number of optimal  $st$ -paths can be exponential in the graph size. If this is the case, then it follows with the construction above that there are preference polyhedra with exponential complexity.

### 3.4.2 MGHS Hardness

The minimum hitting set problem (or equivalently the set cover problem) in its general form is known to be NP-hard and even hard to approximate substantially better than a  $\ln n$  factor, see [2]. A simple greedy algorithm yields a  $O(\log n)$  approximation guarantee, which in the general case is about the best one can hope for. For special instances, e.g., when the instance is derived from a geometric setting (as ours), better approximation guarantees can sometimes be achieved. While the exact solution remains still NP-hard even for seemingly simple geometric instances [7], quite strong guarantees can be shown depending on the characteristics of the objects to be hit. Sometimes even PTAS are possible, see, e.g. [16]. Unfortunately, apart from convexity, none of the favourable characterizations seem to be applicable in our case. We cannot even exclude infinite VC dimension in hope for a  $O(\log OPT)$  approximation [5], as the preference polyhedra might have almost arbitrarily many corners.

## 4 Polynomial-Time Heuristics with Instance-based Lower Bounds

The previous section suggests that if we require worst-case polynomial running time, we have to resort to approximation of some kind. Both, generation of the geometric hitting set instance as well as solving of the instance might not be possible in polynomial time. We address both issues in this section.

### 4.1 Approximate Instance Generation

It appears difficult to show polynomial bounds on the size of a single preference polyhedron, so approximation with enforced bounded complexity seems a natural approach. There are well-known techniques like coresets [1] that allow arbitrarily (specified by some  $\epsilon$ ) accurate approximation of convex polyhedra in space polynomial in  $1/\epsilon$  (which is independent of the complexity of the original polyhedron). We follow the coreset approach and also make use of the special provenance of the polyhedron to be approximated.

For  $d$  metrics, our polyhedron lives in  $d - 1$  dimensions, so we uniformly  $\epsilon$ -sample the unit  $(d - 2)$ -sphere using  $O((1/\epsilon)^{d-2})$  samples. Each of the samples gives rise to an objective function vector for our linear program, we solve each such LP instance to optimality. This determines  $O((1/\epsilon)^{d-2})$  extreme points of the polyhedron in equally distributed directions. Obviously, the convex hull of these extreme points is *contained within* and with decreasing  $\epsilon$  converges towards the preference polyhedron. Guarantees for the convergence in terms of  $\epsilon$  have been proven before, but are not necessary for our (practical) purposes. We call the convex hull of these extreme points the *inner approximation* of the preference polyhedron.

What is interesting in our context is the fact that each extreme point is defined by  $d - 1$  half-spaces. So we can also consider the set of half-spaces that define the computed extreme points and compute their intersection. Clearly, this half-space intersection *contains* the preference polyhedron. We call this the *outer approximation* of the preference polyhedron.

Let us illustrate our approach for a graph with  $d = 3$  metrics, so the preference polyhedron lives in the 2-dimensional plane, see the black polygon/polyhedron in Figure 2. Note that we do not have an explicit representation of this polyhedron but can only probe it via LP optimization calls. To obtain inner and outer approximation we determine the extreme points of this implicitly (via the LP) given polyhedron, by using objective functions  $\max \alpha_1, \max \alpha_2, \min \alpha_1, \min \alpha_2$ . We obtain the four solid red extreme points. Their convex hull (in yellow) constitutes the inner approximation of the preference polyhedron. Each of the extreme points is defined by 2 constraints (halfplanes supporting the two adjacent edges of the extreme points of the preference polyhedron). In Figure 2, these are the light green, blue, dark green, and cyan pairs of constraints. The half-space intersection of these constraints form the *outer approximation* in gray.

### 4.1.1 Sandwiching the Optimum Hitting Set Size

If – by whatever means – we are able to solve geometric hitting set instances optimally, our inner and outer approximations of the preference polyhedra yield upper and lower bounds to the solution size for the actual preference polyhedra. That is, if for example the optimum hitting set of the instance derived from the *inner* approximations has size 23 and the optimum hitting set of the instance derived from the *outer* approximations has size 17, we know that the optimum hitting set size of the actual exact instance lies between 17 and 23. Furthermore, the solution for the instance based on the inner approximations is also feasible for the actual exact instance. So in this case we would have an instance-based (i.e., not apriori, but only aposteriori) approximation guarantee of  $23/17 \approx 1.35$ . If for the application at hand this approximation guarantee is not sufficient, we can try improving by refining the inner and outer approximations. In the limit, inner and outer approximations coincide with the exact preference polyhedra.

## 4.2 Approximate Instance Solving

In this section, we discuss approximation algorithms to replace the ILP shown in Section 3.2, which is not guaranteed to run in polynomial time.

The geometric objects to be hit are the preference polyhedra of the given set of paths, the hitter candidates and which polyhedra they hit are computed via the geometric arrangement as described in Section 3.3.

### 4.2.1 Naive Greedy Approach

The standard greedy approach for hitting set iteratively picks the hitter that hits most objects, which have not been hit before. We call this algorithm *Naive Greedy* or short NG. It terminates as soon as all objects have been hit. The approximation factor of this algorithm is  $O(\log n)$ , where  $n$  is the number of objects to be hit, see [15]. The information which preference hits which polyhedron comes from the arrangement described in Section 3.3.

After computing the cover, NG iterates over the picked hitters and removes them if feasibility is not violated. In this way the computed hitting set is guaranteed to be minimal (but not necessarily minimum).

### 4.2.2 LP-guided Greedy Approach

While naive greedy performs quite well in practice, making use of a precomputed optimal solution to the LP relaxation of the ILP formulation from Section 3.2 can improve the quality of the solution. We call this algorithm *LP Greedy* or LPG. It starts with an empty set  $S^*$  and iterates over a random permutation of the cover constraints of the LP. Note that each of these constraints  $C_i$  corresponds to a preference polyhedron  $P_i$ .

At each constraint  $C_i$  LPG checks if  $P_i$  is hit by at least one preference in  $S^*$ . If not, one of its hitters is randomly picked based on the weights of the LP solution and added to  $S^*$ . To be more precise, the probability of a hitter  $\alpha$  to be drawn is equal to its weight divided by the sum of the weights of all hitters of  $P_i$ .

After iterating over all constraints it is clear that  $S^*$  is a feasible solution. Finally, LPG iterates over  $S^*$  in random order removing elements if feasibility is not violated. This ensures that  $S^*$  is minimal. This process is repeated several times and the best solution is returned.

## 5 Experimental Results

In this section, we assess how real-world relevant the theoretical challenges of polyhedron complexity and NP-hardness of the GHS problem are, and compare exact solutions to our approximation approaches.

### 5.1 Experimental Setup

We run our experiments on a server with two intel Xeon E5-2630 v2 running Ubuntu Linux 20.04 with 378GB of RAM. The running times reported are wall clock times in seconds. Some parts of our implementation make use of all 24 CPU threads. We cover two different scenarios by extracting different graphs from OpenStreetMap of the German state of Baden-Württemberg. This first graph is a road network with the cost types *distance*, *travel time for cars* and *travel time for trucks*. It contains about 4M nodes and 9M edges. The second graph represents a network for cyclists with the cost types *distance*, *height ascent*, and *unsuitability for biking*. The latter metric was created based on the road type (big road  $\Rightarrow$  very unsuitable) and bicycle path tagging. The cycling graph has a size of more than double of the road graph with about 11M nodes and 23M edges. The Dijkstra separation oracle was accelerated using a precomputed multi-criteria contraction hierarchy from [9]. For the Sections 5.3 to 5.5, we used a set of 50 preferences chosen u.a.r. per instance and created different quantities of paths with those preferences. We therefore know an apriori upper bound for the size of the optimal hitting set for our instances. In Section 5.6, we show that the number of preferences used does not change the characteristics of our approaches.

### 5.2 Implementation Details

Our implementation consists of multiple parts. The routing and the computation of the (approximate) preference polyhedra of paths is implemented in the rust programming language (compiled with rustc version 1.51) and uses GLPK [3] (version 4.65) as a library for solving LPs. We intentionally refrained from using non-opensource solutions like CPLEX or GUROBI, as they might not be accessible to everyone. The preference polyhedra are processed in a C++ implementation (compiled with g++ version 10.2) which uses the CGAL library [18, 19] (version 5.0.3) to compute the arrangements with exact arithmetic and output the hitting set instances. We transform the hitting set instances into the ILP formulation from Section 3.2

and solve this ILP formulation and its LP relaxation with GLPK. Finally, we implemented the two greedy algorithms described in Section 4.2.1 and 4.2.2 which solve the hitting set instances in C++. Source code and data under <https://doi.org/10.17605/osf.io/4qkvu>.

### 5.3 Geometric Hitting Set Instance Generation

First, we assess the generation of the GHS instances via preference polyhedra construction and computation of the geometric arrangement. In our tables, we refer to the corner cutting approach as “exact” and the approximate approach as “inner- $k$ ”/ “outer- $k$ ” where  $k$  is the number of directions that were approximated. Since the hitter candidates from the geometric arrangement contain a lot of redundancies (which slows down in particular the (I)LP solving), e.g., some hitters being dominated by others, we prune the resulting candidate set in a straightforward set minimization routine. The preference polyhedra construction as well as the set minimization routine are the multithreaded parts of our implementation. Table 1 shows run times and the average polyhedron complexity for a problem instance with 10,000 paths. In this instance, approximating in twelve directions takes more time than the exact calculation with CCA. This is due the polyhedron complexity being very low in practice. It shows that CCA is a valid approach in practice. Set minimizing is particularly expensive for the inner approximations since considerably more candidates are not dominated.

■ **Table 1** Statistics about instance generation: average number of polyhedron corners, polyhedra construction time (multithreaded), construction time for arrangement, time for set minimization (multithreaded). Car graph with 10,000 paths. Time in seconds.

Algo.	Polyh. Compl	Polyh. Time	Arr. Time	SetMin Time
Inner-12	3.8	70.6	352.3	1450.0
Exact	4.7	54.9	356.7	414.8
Outer-12	4.5	70.6	361.4	473.0

■ **Table 2** Instance generation and solving for various polyhedra approximations. Car graph with 1,000 paths. Time in seconds.

Algo.	Polyh. Time	Arr. Time	ILP Sol.	ILP Time
Inner-4	6.7	4.4	110	>3600
Inner-8	8.4	5.5	50	>3600
Inner-16	11.3	4.8	45	15.5
Inner-32	16.5	5.0	42	3.7
Inner-64	26.9	4.9	39	1.8
Inner-128	48.8	5.0	36	2.3
Exact	6.5	5.2	36	0.7
Outer-128	48.8	5.1	36	0.8
Outer-64	26.9	5.1	36	0.8
Outer-32	16.5	5.0	36	0.6
Outer-16	11.3	5.2	36	1.8
Outer-8	8.4	5.0	36	1.6
Outer-4	6.7	5.9	35	11.5

### 5.4 Geometric Hitting Set Solving

We now compare the two greedy approaches from Section 4.2 with the ILP formulation of Section 3.2. For the ILP solver, we set a time limit of 1 hour after which the computation was aborted and the best found solution (if any) was reported. The results for two instances on the bicycle graph are shown in Table 3. The naive greedy approach has by far the smallest run time but it also reports worse results than the LP-based greedy which was able to find the optimal solution for exact and outer approximations of the two instances. The ILP solver always reports the optimal solution but it might take a very long time to do so. Especially, the inner approximations seem to yield hard GHS instances before they converge to the exact problem instance. A state of art commercial ILP solver might improve run time drastically.

■ **Table 3** Comparison of GHS solving algorithms on two instances on the bicycle graph. The times (given in seconds) for the LP-Greedy and ILP algorithm include solving the LP-relaxation first.

Algorithm	Paths	Greedy Solution	Greedy Time	LP-Greedy Solution	LP-Greedy Time	ILP Solution	ILP Time
Inner-12	5000	210	4.2	181	40.8	–	>3600.0
Exact	5000	54	2.9	48	24.8	48	68.4
Outer-12	5000	57	3.1	48	30.5	48	95.5
Inner-12	10000	421	10.9	399	144.6	–	>3600.0
Exact	10000	58	7.6	50	73.5	50	188.9
Outer-12	10000	59	8.1	50	81.1	50	114.6

## 5.5 Varying Polyhedron Approximation

In Table 2 we show that increasing the number of directions in the approximation approach makes its results converge to the exact approach. Interestingly, the outer approximation converges faster than the inner approximation. It typically yields good lower bounds already for four approximation directions and higher. In Table 2 it yields a tight lower bound for eight directions while the inner approximation only achieves a tight upper bound with 128 directions.

## 5.6 Dependence on the number of preferences

So far, we have only considered PTC problem instances that were derived from 50 initial preferences. To ensure that the fixed number of preferences does not bias our results, we also ran instances generated with 10, 20, 100 and 1000 preferences. Tables 4 and 5 hold the results for instances with 1,000 paths on both graphs. As expected, we found larger optimal solutions as the number of preferences increased. Although, the solution size did increase only to 53 and 177 for the car and bicycle graph, respectively. We attribute this to the probably bounded inherent complexity of the graphs and cost types used. We also note a slight increase in ILP run times with spikes when computing the solution to the inner approximation as discussed in Section 5.4. Otherwise, there is no performance difference.

## 6 Conclusions

We have exhibited an example for a real-world application where theoretical complexities and hardness do not prevent the computation of optimal results even for non-toy problem instances. The presented method allows the analysis of large trajectory sets as they are, for example, collected within the OpenStreetMap project. Our results suggest that exact solutions are computable in practice, even more so if commercial ILP solvers like CPLEX or GUROBI are employed. From a theoretical point of view, it would be interesting to prove or disprove our guess that exponentially (in the network size) many optimal paths between one source-target pair exist. Another open problem is to find an explanation for the observed average preference polyhedra complexity, which is surprisingly low. On a more abstract level, our approach of approximating the hitting set problem with increasing precision by refining the inner and outer approximations until an optimal solution can be guaranteed could also be viewed as an extension of the framework of *structural filtering* ([10]), where the high-level idea is to certify exactness of possibly error-prone calculations. It could be interesting to apply this not only to the instance generation step but simultaneously also to the hitting set solution process.

■ **Table 4** Run times and optimal solution of instances generated with varying amount of preferences. The instances each consist of 1.000 paths on the car graph.

Algorithm	$\alpha$	Polygon Time	Arrangement Time	Set Minimization Time	ILP Solution	ILP Time
Inner-8	10	7.9	4.8	1.2	23	6.5
Exact	10	5.9	4.5	0.4	10	0.3
Outer-8	10	7.9	4.4	0.4	10	0.2
Inner-8	20	7.8	7.7	2.8	28	96.6
Exact	20	5.2	6.0	0.6	17	0.8
Outer-8	20	7.8	6.1	0.6	17	0.8
Inner-8	100	8.3	5.5	1.7	57	3601.0
Exact	100	6.3	5.8	0.6	49	23.1
Outer-8	100	8.3	5.6	0.6	49	21.1
Inner-8	1000	8.0	5.7	2.1	61	1166.5
Exact	1000	5.9	5.3	0.5	53	2.1
Outer-8	1000	8.0	5.2	0.5	53	21.9

■ **Table 5** Run times and optimal solution of instances generated with varying amount of preferences. The instances each consist of 1.000 paths on the bicycle graph.

Algorithm	$\alpha$	Polygon Time	Arrangement Time	Set Minimization Time	ILP Solution	ILP Time
Inner-8	10	47.9	3.0	0.6	45	0.9
Exact	10	57.9	2.9	0.4	10	0.1
Outer-8	10	47.9	3.0	0.5	10	0.1
Inner-8	20	42.4	2.9	0.7	69	3600.0
Exact	20	47.7	3.1	0.5	20	0.3
Outer-8	20	42.4	3.1	0.6	20	0.3
Inner-8	100	43.8	2.0	0.3	125	398.3
Exact	100	50.1	2.1	0.2	85	0.2
Outer-8	100	43.8	2.1	0.3	83	0.3
Inner-8	1000	43.9	2.0	0.4	193	1.2
Exact	1000	49.0	2.2	0.3	177	1.6
Outer-8	1000	43.9	2.2	0.3	177	3.7



---

**References**

---

- 1 Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.
- 2 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for  $k$ -restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.
- 3 Andrew Makhorin. GLPK – GNU Project – Free Software Foundation (FSF), 2012. URL: <https://www.gnu.org/software/glpk/glpk.html>.
- 4 Florian Barth, Stefan Funke, Tobias Skovgaard Jepsen, and Claudius Proissl. Scalable unsupervised multi-criteria trajectory segmentation and driving preference mining. In *BigSpatial@SIGSPATIAL*, pages 6:1–6:10. ACM, 2020.
- 5 Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
- 6 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1987.
- 7 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981.
- 8 Stefan Funke, Sören Laue, and Sabine Storandt. Deducing individual driving preferences for user-aware navigation. In *SIGSPATIAL/GIS*, pages 14:1–14:9. ACM, 2016.
- 9 Stefan Funke, Sören Laue, and Sabine Storandt. Personal routes with high-dimensional costs and dynamic approximation guarantees. In *SEA*, volume 75 of *LIPICs*, pages 18:1–18:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.
- 10 Stefan Funke, Kurt Mehlhorn, and Stefan Näher. Structural filtering: a paradigm for efficient and exact geometric programs. *Comput. Geom.*, 31(3):179–194, 2005.
- 11 Stefan Funke, André Nusser, and Sabine Storandt. On  $k$ -path covers and their applications. *VLDB J.*, 25(1):103–123, 2016.
- 12 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *SIGSPATIAL/GIS*, pages 45:1–45:10. ACM, 2015.
- 13 Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 124–137. SIAM, 2010.
- 14 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method. In *Geometric Algorithms and Combinatorial Optimization*, pages 64–101. Springer, 1993.
- 15 David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.
- 16 Nabil H. Mustafa and Saurabh Ray. PTAS for geometric hitting set problems via local search. In *Symposium on Computational Geometry*, pages 17–22. ACM, 2009.
- 17 Johannes Oehrlin, Benjamin Niedermann, and Jan-Henrik Haunert. Inferring the parametric weight of a bicriteria routing model from trajectories. In *SIGSPATIAL/GIS*, pages 59:1–59:4. ACM, 2017.
- 18 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.3 edition, 2020. URL: <https://doc.cgal.org/5.0.3/Manual/packages.html>.
- 19 Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.3 edition, 2020. URL: <https://doc.cgal.org/5.0.3/Manual/packages.html#PkgArrangementOnSurface2>.

# Efficiently Partitioning the Edges of a 1-Planar Graph into a Planar Graph and a Forest

Sam Barr ✉

University of Waterloo, Canada

Therese Biedl ✉

University of Waterloo, Canada

---

## Abstract

---

1-planar graphs are graphs that can be drawn in the plane such that any edge intersects with at most one other edge. Ackerman showed that the edges of a 1-planar graph can be partitioned into a planar graph and a forest, and claims that the proof leads to a linear time algorithm. However, it is not clear how one would obtain such an algorithm from his proof. In this paper, we first reprove Ackerman's result (in fact, we prove a slightly more general statement) and then show that the split can be found in linear time by using an edge-contraction data structure by Holm, Italiano, Karczmarz, Łącki, Rotenberg and Sankowski.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Theory of computation → Dynamic graph algorithms

**Keywords and phrases** 1-planar graphs, edge partitions, algorithms, data structures

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.16

**Funding** *Therese Biedl*: Research supported by NSERC; FRN RGPIN-2020-03958.

## 1 Introduction

In this paper, we study the class of 1-planar graphs: graphs that can be drawn in the plane such that every edge crosses at most one other edge. 1-planar graphs were introduced by Ringel [21], motivated by the problem of coloring the vertices and faces of a planar graph. Since then, there have been many publications concerning 1-planar graphs, both for theoretical results such as coloring, as well as algorithmic results such as solving drawing and optimization problems. The reader may refer to [18] for an annotated bibliography from 2017 and [15] for a more recent book that includes developments since then.

Many of the results for 1-planar graphs are obtained by first converting the 1-planar graph  $G$  into a planar graph  $G'$ , applying results for planar graphs, and then expanding the result from  $G'$  back to  $G$ . (We will give specific examples below.) There are several ways of how to create  $G'$ , e.g., by deleting edges or by replacing crossings with dummy-vertices. Of particular interest to us here is to make a 1-planar graph planar by deleting edges. Put differently, we want an *edge partition*, i.e., write  $E(G) = E' \cup E''$  such that  $E'$  forms a planar graph while  $E''$  has some special structure that makes it possible to expand a solution for  $G'$  to one for  $G$ .

**Previous Work.** The main focus of this paper is a result by Ackerman [1]. He established that the edges of a 1-planar graph can be partitioned such that one partition induces a planar graph and the other induces a forest. This was an extension of an earlier result from Czap and Hudák [8], who proved it for optimal 1-planar graphs (simple 1-planar graphs with the maximum  $4n - 8$  edges). Other partitions of near-planar graphs have also been studied; we list a few here. Lenhart et al. [19] show that optimal 1-planar graphs can be partitioned into a maximal planar graph and a planar graph of maximum degree four (the bound of four



© Sam Barr and Therese Biedl;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 16; pp. 16:1–16:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is shown to be optimal). Bekos et al. [4] provide edge partition results for some  $k$ -planar graphs (graphs that can be drawn in the plane such that any edge crosses at most  $k$  other edges). Di Giacomo et al. [9] prove edge partition results for so-called NIC-graphs, a subclass of 1-planar graphs.

For algorithmic purposes, we need to find such edge partitions in linear time. With one exception, the above papers either explicitly come with a linear-time algorithm to find the edge-partition, or such an algorithm can easily be derived from the proof. The one exception is the paper by Ackerman [1]. He claims that the partition of a 1-planar graph into a planar graph and a forest can be found in linear time, but provides no details. Moreover, while his proof clearly gives rise to a polynomial-time algorithm, it is not clear how one would achieve linear time (or even  $O(n \log n)$  time), since he relies on contracting edges while repeatedly testing whether two vertices share a face and occasionally splitting the graph into subgraphs, and neither of these operations can trivially be done in constant time in planar graphs. (We confirmed this in private communication with Ackerman.)

**Our Results.** In this paper, we show that a partition of a 1-planar graph into a planar graph and a forest can be found in linear time. We were not able to use Ackerman's proof for this directly, so as a first step we re-prove the result in a slightly different way to avoid some problematic situations and so that then a linear-time algorithm can be established. A crucial ingredient for this is a data structure by Holm, Italiano, Karczmarz, Łącki, Rotenberg and Sankowski [14] which allows for efficiently contracting edges of planar graphs. To our knowledge, this data structure has not been implemented. Because of this, we also show that the partition can be computed in  $O(n \log n)$  time using a simpler data structure based on incidence lists.

- As a consequence of our result, a number of related problems can be solved in linear time:
- Angelini et al. [3] studied the problem of finding simultaneous quasi-planar drawings of graphs where some edges are fixed. They used Ackerman's partition result in order to find such a simultaneous drawing of a 1-planar graph and a planar graph and cited its claimed linear runtime; with our result the linear runtime of [3] is established.
  - It is known that every 1-planar graph has *arboricity* 4, i.e., its edges can be partitioned into 4 forests. (This follows from Nash-Williams formula for arboricity [20] since 1-planar graphs have at most  $4n - 8$  edges [6].) The arboricity (and the corresponding edge-partition) of a graph can be computed in polynomial time [11], but to our knowledge not in linear time. For *planar* graphs, a split into 3 forests can be found in linear time [24]. So with our result, the partition of a 1-planar graph into 4 forests can be done in linear time as well, by first partitioning into a forest and a planar graph and then applying [24] onto the planar graph.
  - If  $G$  is a bipartite 1-planar graph, then it has at most  $3n - 8$  edges [16] and hence arboricity 3. With our result we can partition it into a forest and a planar bipartite graph in linear time. The planar bipartite graph can be partitioned into two forests in linear time [22]. In consequence, every 1-planar bipartite graph can be partitioned into three forests in linear time.
  - From a partition into  $d$  forests one easily obtains an edge orientation with in-degree at most  $d$ . For bipartite graphs, such an edge-orientation can be used to prove  $(d+1)$ -list-colorability [13], and this list-coloring can be found in  $O(dn)$  time [5]. Putting everything together, therefore our paper fills the one missing gap to show that 1-planar bipartite graphs can be 4-list-colored in linear time.

Our paper is structured as follows: In Section 2 we go over necessary terminology and present Ackerman’s proof in order to demonstrate that it does not immediately lead to a linear time algorithm. In Section 3 we present our new proof. In Section 4 we use our alternative proof to design an efficient algorithm for finding the partition, before concluding in Section 5.

## 2 Background

We assume basic familiarity with graph theory (see e.g. [10]). All graphs in this paper are finite and connected, but not necessarily simple.

For a (multi)graph  $G$  and a vertex  $x$  of  $G$ ,  $d(x)$  is the number of edges incident to  $x$ .

We recall that a (multi)graph  $G$  is called *planar* if it can be drawn in the plane without edges crossing. Let  $G$  be a planar (multi)graph given with a drawing  $\Gamma$ . The maximal regions of  $\mathbb{R}^2 \setminus \Gamma$  are the *faces* of  $G$ . We add a *chord* to a face  $f$  by adding an edge between two non-adjacent vertices on the boundary of  $f$ , and drawing the edge through the region of  $f$ . For each vertex  $x$  with incident edges  $e_1, \dots, e_k$ , the drawing  $\Gamma$  places these edges in some rotational clockwise order around  $x$ . The space between two edges which are adjacent in this rotational order form an *angle*. The *degree* of a face  $f$  is the number of angles contained in the face. We say that a face  $f$  is a *quadrangle* if it has degree 4. Note that this includes both faces with 4 vertices on their boundary, and some faces with fewer than 4 vertices on their boundary (see Figure 1(a)). If  $f$  is a quadrangle with exactly 4 vertices on its boundary, then we call  $f$  a *simple quadrangle* (the quadrangles of a simple planar graph will all be simple quadrangles). A face of degree 3 is a *triangle*, and a face of degree 2 is a *bigon*.

The *facial cycle* of a quadrangle  $f$  is a 4-tuple  $\langle z_0, z_1, z_2, z_3 \rangle$  such that each  $z_i$  is on the boundary of  $f$  and there are edges  $z_i z_{i+1}$  and  $z_i z_{i-1}$  (arithmetic modulo 4) which form an angle in  $f$ . Note that  $z_0, z_1, z_2, z_3$  need not be distinct if  $f$  has loops or parallel edges, or if it is incident to a bridge. We say that  $z_0$  and  $z_2$  are *opposing vertices in  $f$*  (we will often omit mentioning the face when it is clear from context). Likewise  $z_1$  and  $z_3$  are opposing vertices in  $f$ . Note that it is possible for a vertex to oppose itself in a quadrangle.

*Stellating* a face  $f$  of a planar graph is the process of adding a new vertex  $s$  inside  $f$ , and adding an edge from  $s$  to every vertex on the boundary of  $f$ . Given two vertices  $a$  and  $b$ , we *contract  $a$  and  $b$*  by creating a new vertex  $c$ , adding an edge  $vc$  for each edge  $va$  and  $vb$ , and deleting  $a$  and  $b$  and all their incident edges. If  $a$  and  $b$  were adjacent, then  $c$  has a loop, and if  $a$  and  $b$  were both adjacent to a vertex  $v$ , then  $c$  will have parallel edges to  $v$ . If  $a$  and  $b$  were both on the boundary of some face  $f$  in a planar graph, then we can *contract  $a$  and  $b$  through  $f$*  by placing this new vertex  $c$  inside  $f$ . This preserves planarity, but destroys the face  $f$ .

A *1-planar* graph is a graph  $G$  that can be drawn in the plane such that any edge intersects with at most one other edge. Here “drawing” always means a *good drawing* (see e.g. [23]), in particular this means that no three edges cross in a point, that no edge intersects itself, and that incident edges do not cross. From now on, whenever we speak of a 1-planar graph, we assume that one particular 1-planar drawing has been fixed. A pair of edges that intersect each other are a *crossing pair*, and the point where they intersect is known as the *crossing point*. The *planarization* is the graph  $G^\times$  obtained by replacing each crossing point with a new vertex. A 1-planar graph  $G$  is *planar-maximal* if no *uncrossed edge* (i.e., an edge which does not intersect any other edge) can be added to the fixed drawing of  $G$  without adding a loop or a bigon.

## 16:4 Partitioning the Edges of a 1-Planar Graph

For algorithmic purposes, a drawing of a planar graph can be specified by giving the rotational clockwise order of edges at every vertex; this specifies the circuits bounding the faces uniquely. A drawing of a 1-planar graph can be specified by giving a drawing of its planarization, with the vertices resulting from a crossing point marked as such. Since testing 1-planarity is NP-hard [12], we assume that any 1-planar graph  $G$  is given with such a drawing. We also assume that  $G$  is planar-maximal, because any 1-planar graph can be made planar-maximal in linear time by adding edges [2], and having more edges can only make partitioning more difficult.

For ease of notation, we define a shortcut for our partition problem.

► **Definition 1.** *A graph  $G$  has a PGF-partition if its edge-set  $E(G)$  can be partitioned into two sets  $A$  and  $B$  such that  $G[A]$  is a planar graph and  $G[B]$  is a forest.*

### 2.1 Ackerman's Proof

To establish the difficulties of achieving a linear time algorithm, we briefly review here Ackerman's proof for the existence of a PGF-partition.

Let  $G$  be a (planar-maximal) 1-planar graph without loops drawn in the plane. Remove all crossing pairs of  $G$ . Call the resulting graph  $H$  the (*planar*) *skeleton* of  $G$  [2]. Observe that the faces of  $H$  are either bigons, triangles, or quadrangles, and that there is a 1-1 mapping between the quadrangles of  $H$  and the crossing pairs of  $G$ . Moreover, by this 1-1 mapping and the fact that the two edges forming a crossing pair do not share an endpoint, one can see that the quadrangles of  $H$  are in fact simple quadrangles. Ackerman, similarly to Czap and Hudák [8], establishes the following.

► **Lemma 2** (Ackerman [1]). *Let  $G$  be a 1-planar graph, and let  $H$  be the skeleton of  $G$ . If we can add a chord to every quadrangle of  $H$  such that the chords induce a forest, then  $G$  has a PGF-partition.*

**Proof.** Let  $C$  be the set of chords added to  $H$ . By the 1-1 mapping between quadrangles of  $H$  and crossing pairs of  $G$ , we know that exactly one edge from each crossing pair of  $G$  is contained in  $C$ . In particular, each edge  $e \in C$  forms a crossing pair with some edge  $e'$  of  $G$ . Let  $C'$  be the set of these edges  $e'$ . By assumption  $G[C]$  is a forest. Moreover, the graph  $H \cup C'$  is the graph  $H$  plus a chord added to each quadrangle of  $H$ , and so is a planar graph. As  $H \cup C'$  is also the graph induced by the edge-set  $E(G) \setminus C$ , this gives us the desired partition. ◀

Thus, in order to prove the existence of a PGF-partition, it suffices to show (typically by induction on the number of quadrangles) that such a set of chords can be found. For the induction to go through, Ackerman additionally forbids the chords from containing a path between two adjacent pre-specified vertices  $x, y$ .

► **Theorem 3** (Ackerman [1]). *Let  $H$  be a planar multigraph without loops such that every face has degree at most four and all quadrangles are simple. Let  $x, y$  be a pair of adjacent vertices of  $H$ . Then we can add a chord to every simple quadrangle of  $H$  such that the subgraph induced by the chords is a forest and does not contain a path between  $x$  and  $y$ .*

**Proof.** Proceed by induction on the number of quadrangles in  $H$ . If  $H$  has no quadrangles, then the statement is trivial. Otherwise, let  $f$  be a quadrangle with facial cycle  $\langle z_0, z_1, z_2, z_3 \rangle$ ; by assumption  $f$  is simple.

**Case 1.** The only face containing  $z_0$  and  $z_2$  is  $f$ ; in particular  $z_0$  and  $z_2$  are not adjacent. Contract  $z_0$  and  $z_2$  through  $f$ . Let  $H'$  be the graph resulting from this contraction. Observe that  $H'$  has one fewer quadrangle than  $H$ . All other quadrangles remain simple since  $f$  was the only face containing  $z_0$  and  $z_2$ . Apply induction on  $H'$  with the same pair of adjacent vertices  $x, y$  to receive a set of chords  $C'$ , and then further add the chord  $z_0z_2$ . Chords have now been added to every quadrangle of  $H$ , and it is easy to see that we have not added a cycle, or a path from  $x$  to  $y$ , in the chords.

**Case 2.** There is some face  $f' \neq f$  containing  $z_0$  and  $z_2$ , but the only face containing  $z_1$  and  $z_3$  is  $f$ . Proceed as in Case 1, except contract  $z_1$  and  $z_3$ .

**Case 3.** None of the above. Then there is a face  $f' \neq f$  containing  $z_0$  and  $z_2$ , and there is a face  $f'' \neq f$  containing  $z_1$  and  $z_3$ . Ackerman argues that  $f' = f''$  (see also Lemma 5), and therefore  $f'$  is also a simple quadrangle. Observe that  $G$  can be split into four connected subgraphs  $H_0, H_1, H_2, H_3$ , where  $H_i$  contains  $z_i$  and  $z_{i+1}$  (addition modulo 4) on the boundary (see Figure 1(e)). One of these subgraphs, say  $H_0$ , will contain the adjacent pair  $x, y$ . Apply induction on  $H_0$  with  $x, y$ , and apply induction on the other  $H_i$  with the pair  $z_i, z_{i+1}$ . After induction, add the chords  $z_0z_2$  in  $f$ , and  $z_1z_3$  in  $f'$ . One verifies that the added chords do not add a path between  $x$  and  $y$  and that the chords do not create a cycle. ◀

### 3 An Alternative Existence Proof

While Ackerman's proof clearly leads to a polynomial time algorithm for finding the partition, it is not obviously linear since distinguishing between the cases and contracting are not obviously doable in constant time:

1. We need to test whether a given pair of vertices share more than one face.
2. The graph changes via contractions, and it is not obvious whether the existing data structures for efficiently contracting edges in planar graphs (e.g. [14]) would support (1) in constant time.
3. In Case 3 of Ackerman's proof, we need to identify the four subgraphs  $H_0, H_1, H_2, H_3$ . Furthermore, we need to determine which of these subgraphs contains the pair  $x, y$ . Neither operation is obviously doable in constant time.

We now give a different proof of the existence of a PGF-partition that either avoids these issues or addresses explicitly how to resolve them. The biggest change is how we handle Case 3 of Ackerman's proof. Ackerman used here a split into four graphs, which is necessary in order to maintain that all quadrangles are simple. We prove a more general statement that permits non-simple quadrangles and hence avoids having to split the graph. Moreover, we generalize Ackerman's "forbidden pair"  $x$  and  $y$  by choosing chords in such a way that the chords do not induce a path between *any* pair of vertices that were adjacent in the original graph  $H$ . Doing so simplifies the induction since we no longer need to keep track of where the vertices  $x$  and  $y$  are. Before we state this result we need a few helper-results that hold for all quadrangles (simple or not).

► **Lemma 4.** *Let  $H$  be a plane multigraph without loops, let  $f$  be a quadrangle of  $H$ , and let  $\langle z_0, z_1, z_2, z_3 \rangle$  be the facial cycle of  $f$ . If  $z_i = z_{i+2}$  (addition modulo 4) for some  $i$ , then  $z_{i+1} \neq z_{i+3}$ , and there is no face  $f' \neq f$  that contains  $z_{i+1}$  and  $z_{i+3}$ .*

16:6 Partitioning the Edges of a 1-Planar Graph

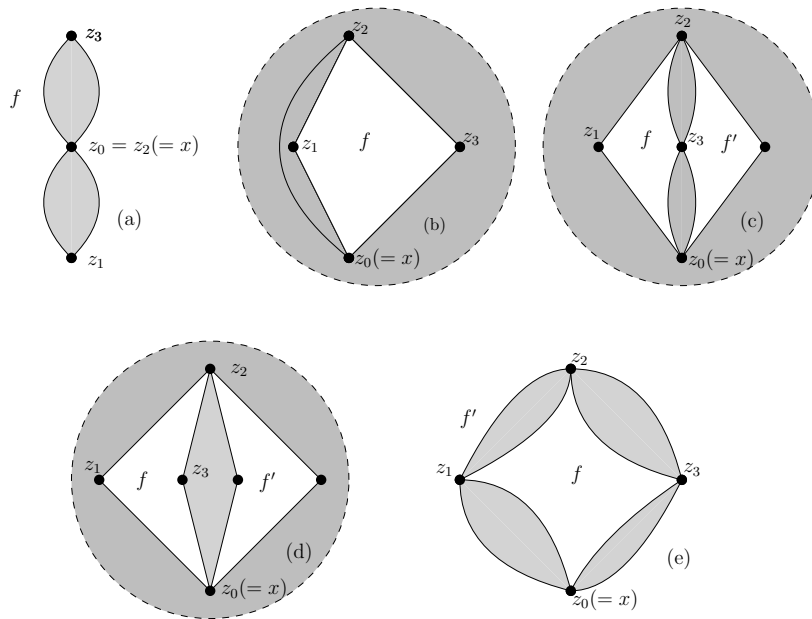
**Proof.** Up to renaming, we may assume that  $i = 0$ , so  $z_0 = z_2$  (see Figure 1(a)). Assume for contradiction that  $z_1 = z_3$ . Then  $f$  consists of several parallel edges between  $z_0 = z_2$  and  $z_1 = z_3$ , and thus  $f$  is not a quadrangle.

Assume by way of contradiction that there is some face  $f' \neq f$  which contains  $z_1$  and  $z_3$ . Subdivide one of the  $z_0z_3$  edges to obtain a new vertex  $y$  adjacent to  $z_0$  and  $z_3$ , and further add an edge  $yz_1$  within  $f$ . We have split  $f$  and attained a simple quadrangle  $f''$  with facial cycle  $\langle z_0, z_1, y, z_3 \rangle$ . Stellate  $f''$  with a new vertex  $c$ , and add an edge  $z_1z_3$  through the face  $f'$ . All these steps maintain the planarity of  $H$ . Moreover, the five vertices  $z_0, z_1, y, z_3, c$  are pairwise adjacent. But this forms a  $K_5$  which is not planar, a contradiction. ◀

The following lemma was shown (without being stated explicitly) in Case 3 of Ackerman’s proof.

► **Lemma 5.** *Let  $H$  be a plane multigraph without loops, let  $f$  be a quadrangle of  $H$ , and let  $\langle z_0, z_1, z_2, z_3 \rangle$  be the facial cycle of  $f$ . If  $z_i$  and  $z_{i+2}$  (addition modulo 4) are both on some face  $f' \neq f$  for some  $i$ , then no face  $f'' \neq f, f'$  contains both  $z_{i+1}$  and  $z_{i+3}$ .*

**Proof.** Up to renaming we may assume that  $i = 0$ , so  $z_0$  and  $z_2$  are on  $f$  and  $f'$  (see Figure 1(b-e)). Suppose for contradiction that such a face  $f''$  exists. By Lemma 4,  $z_1 \neq z_3$  and  $z_0 \neq z_2$ . Stellate  $f$  with a new vertex  $c$ , add an edge  $z_0z_2$  through  $f'$ , and add an edge  $z_1z_3$  through  $f''$ . The original multigraph  $H$  was planar, and all of these operations preserve planarity. However, the five vertices  $z_0, z_1, z_2, z_3$ , and  $c$  are pairwise adjacent and form a  $K_5$ , which is not planar, a contradiction. ◀



■ **Figure 1** Some configurations where we contract  $z_1$  and  $z_3$ .

Now we reprove the existence of quadrangle-chords that form a forest.

► **Theorem 6.** *Let  $H$  be a plane multigraph without loops such that every face has degree at most 4. Then it is possible to add a chord to every quadrangle of  $H$  such that the graph induced by the chords is a forest. Moreover, for all pairs of adjacent vertices  $a$  and  $b$  of  $H$ , there is no path from  $a$  to  $b$  in the chords.*



**Proof.** As in Ackerman’s proof, we prove the claim by induction on the number of quadrangles in  $H$  and remove each quadrangle by contracting an opposing pair of vertices in the quadrangle. If  $H$  has no quadrangles, the claim is trivial. Otherwise, there are quadrangles left. Pick an arbitrary vertex  $x$  that is still incident to some quadrangles (later in our algorithm we will deal with all its incident quadrangles). Let  $f$  be one of its incident quadrangles with facial cycle  $\langle x = z_0, z_1, z_2, z_3 \rangle$ . We first pick two opposing vertices of  $f$  to contract.

**Case 1.** This case covers when we choose to contract  $z_1$  and  $z_3$ , and has three sub-cases. We contract  $z_1$  and  $z_3$  whenever

**Case 1.a**  $x = z_2$  are the same vertex, or

**Case 1.b**  $x$  and  $z_2$  are adjacent, or

**Case 1.c**  $x$  and  $z_2$  are opposing vertices of some quadrangle  $f' \neq f$ .<sup>1</sup>

Figure 1 illustrates possible configurations of face  $f$  where Case 1 applies: Case 1.a applies to (a), Case 1.b applies to (b), and Case 1.c applies to (c,d,e). Note that Case 1.c covers Case 3 of Ackerman’s proof, where  $x, z_1, z_2, z_3$  all belong to two simple quadrangles  $f, f'$  (see also Figure 1(e)). Our contraction turns  $f'$  into a non-simple quadrangle, but our proof can handle this.

**Case 2.** Otherwise, we contract  $x$  and  $z_2$ .

Table 1 demonstrates when we pick Case 1 and when we pick Case 2, and crucially shows cases which are impossible by Lemmas 4 and 5. To see that these lemmas apply in the second row and column, observe that adjacent vertices always share at least one face, and in particular if two opposing vertices are adjacent then they must share two faces other than the quadrangle they are opposing in.

■ **Table 1** All possible cases for the quadrangle  $f$  with facial cycle  $\langle x, z_1, z_2, z_3 \rangle$ . We either indicate which case in the proof of Theorem 6 would be chosen, or indicate the lemma that demonstrates that this case is impossible.

	$x = z_2$	$x \neq z_2;$ $(x, z_2) \in E(H)$	$x \neq z_2;$ $x, z_2$ are opposing in $f'$	Otherwise
$z_1 = z_3$	Impossible (Lemma 4)	Impossible (Lemma 4)	Impossible (Lemma 4)	Case 2
$z_1 \neq z_3;$ $(z_1, z_3) \in E(H)$	Impossible (Lemma 4)	Impossible (Lemma 5)	Impossible (Lemma 5)	Case 2
$z_1 \neq z_3;$ $z_1, z_3$ are opposing in $f''$	Impossible (Lemma 4)	Impossible (Lemma 5)	Impossible if $f' \neq f''$ (Lemma 5), Case 1.c otherwise	Case 2
Otherwise	Case 1.a	Case 1.b	Case 1.c	Case 2

Let  $z_i, z_{i+2}$  be two vertices chosen for contraction and let  $H'$  be the graph resulting from contracting  $z_i$  and  $z_{i+2}$ . By Table 1,  $z_i$  and  $z_{i+2}$  are not adjacent, and they are distinct. Therefore our contraction has destroyed the quadrangle  $f$  and not added any loops, so we can apply induction on  $H'$ . Let  $C'$  be the set of chords added to  $H'$ . By the inductive

<sup>1</sup> In fact, our proof does not require Case 1.c to be separated out; we could equally have contracted  $x$  and  $z_2$  in this case.

## 16:8 Partitioning the Edges of a 1-Planar Graph

hypothesis,  $C'$  induces a forest and for any edge  $ab \in H'$ , there is no path from  $a$  to  $b$  in  $C'$ . Uncontract  $z_i$  and  $z_{i+2}$ , and add a chord  $e := z_i z_{i+2}$  between them. Define  $C := C' \cup \{e\}$ . We now verify that  $C$  satisfies all conditions.

Let  $a, b$  be a pair of adjacent vertices of  $H$ . Assume by way of contradiction that there is a path from  $a$  to  $b$  in  $C$ . By the inductive hypothesis, the path must use  $e$ . Furthermore,  $e$  cannot be the edge  $ab$  since  $z_i$  and  $z_{i+2}$  are not adjacent, so the path must use some edges from  $C'$ . Let  $c_1, \dots, c_{k_1}, e, c_{k_1+1}, \dots, c_{k_2}$  be the edges on this path,  $k_2 \geq 1$ . But then  $c_1, \dots, c_{k_2}$  would be a path from  $a$  to  $b$  within  $C'$  in  $H' = H/e$ , a contradiction.

Assume by way of contradiction that  $C$  induces a cycle in  $H$ . Since  $e$  is not a loop in  $H$ , the cycle must use edges from  $C'$ . Let  $e, c_1, \dots, c_k$  be the cycle,  $k \geq 2$ . Then  $c_1, \dots, c_k$  would induce a cycle within  $C'$  in  $H' = H/e$ , a contradiction. ◀

### 4 Efficient Implementation

It is still not immediately clear how one would implement the above theorem in order to achieve linear runtime, since as in Ackerman's proof we need to repeatedly test how many quadrangles two vertices share. However, if we are more careful about the order in which we contract each quadrangle, an efficient implementation can be achieved. The crucial idea will be to pick some vertex  $x$  and contract all quadrangles incident to  $x$ . This will allow us to store additional information relative to  $x$  and hence speed up testing which case applies. We note that this idea alone would not suffice to make Ackerman's proof run in linear time, as one would still need to find a way to implement Case 3 (where he splits the graph into four subgraphs and determines which subgraph contains a given pair of vertices) of Ackerman's proof efficiently.

#### 4.1 Data Structure Interface

As mentioned earlier, one of the major ingredients to achieve fast run-time is to use the data structure by Holm et al. [14] for contraction in planar graphs, but we will also provide a (simpler but slower) alternative. We will discuss these later (in Subsection 4.4) when we analyze the run-time, but note here the two operations provided by [14] that will be needed:

- $x = \text{contract}(e)$  takes a reference to an edge  $e$ , contracts  $e$ , and returns the vertex resulting from the contraction.

Note that contracting  $e$  creates a loop in the graph, especially if there are multiple copies of this edge, while the proof of Theorem 6 assumed that the graph has no loops. We could remove loops (the data structure by Holm et al. can report newly created loops after `contract`), but this turns out to be unnecessary: We will only contract edges at artificial gadgets inserted into the graph, and the created loops are at quadrangles that are destroyed afterwards and those will not pose problems.

- $\text{neighbors}(x)$  returns an iterator over  $\{\langle xv, v \rangle : xv \in E\}$  where  $E$  is the edge set of the graph. In other words, it returns an iterator to tuples containing each edge incident to  $x$  and the endpoint of this edge. No guarantee is given as to the order of the neighbors.

We assume that  $\text{neighbors}(x)$  has  $O(1)$  runtime and that the returned list can be iterated over in  $O(d(x))$  time (recall that  $d(x)$  denotes the number of edges incident to  $x$ ). Since edge-contraction can create parallel edges, it is possible that  $\text{neighbors}(x)$  contains parallel edges, and hence the second element of the tuple need not be unique. Again this will not pose problems later.

In Subsection 4.2, we will add labels and other meta-data to vertices of our graph. We make no assumptions as to how the meta-data are updated when two vertices are contracted, and so we will maintain those manually.

## 4.2 Preprocessing

We take as input a 1-planar graph  $G$ , given by specifying its planarization via the rotational clockwise order of edges at the vertices, and assuming that vertices of the planarization resulting from crossing points are marked as such.  $G$  need not be simple, but we assume that it has no loops (they can always be added to the planar part) and for ease of stating bounds we assume that it has  $O(n)$  edges. From  $G$ , we can construct a planar-maximal supergraph  $G^+$  in linear time [2], and along the way construct the planarization  $(G^+)^{\times}$  of  $G^+$ . As before we use  $H$  to denote the skeleton of  $G^+$ , but we do not construct it explicitly. Instead, notice that the vertices of  $(G^+)^{\times}$  marked as crossings correspond uniquely to the quadrangles of  $H$ . For this reason, we assume that these vertices are marked with a label *quad* and we call such a vertex a *quadrangle-vertex* (whereas the corresponding face of  $H$  is called a *quadrangle-face*).

Our proof of Theorem 6 relies heavily on having faces, while the data structure of Holm et al. makes no provisions for accessing faces. For this reason, we keep the quadrangle-vertices in the graph as representatives of the quadrangle-faces. This will make it possible to implement the operation of “contract  $z_i$  and  $z_{i+2}$  within quadrangle  $f$ ” used in Theorem 6 via edge-contractions at the corresponding quadrangle-vertex  $f$ . (See Procedure 1 for details.) We also assume that any quadrangle-vertex  $f$  has references to the two original edges in  $G$  that crossed; when doing such a contraction within  $f$  we can hence also record the corresponding edge  $z_i z_{i+2}$  for inclusion in the forest-part of the partition.

### ■ Procedure 1 ContractThrough( $u, v, f$ ).

---

**Result:** Contract two vertices  $u$  and  $v$  in  $H$  through a quadrangle-face  $f$ , and return the resulting vertex  $y$ .

// pre:  $f$  is labelled *quad*

// pre:  $u$  and  $v$  are opposing on face of  $H$  corresponding to  $f$

Find the original edge  $uv$  of  $G$  that is stored with  $f$ .

Record edge  $uv$  as belonging to the forest of the partition.

**for**  $\langle e, w \rangle$  **in** neighbors( $f$ ) **do**

**if**  $v$  equals  $w$  **or**  $v$  equals  $u$  **or**  $w$  has label  $quad_i$  for some  $0 \leq i \leq 3$  **then**  
    |  $y := \text{contract}(e)$

Remove labels  $quad, quad_i$  from  $y$

**return**  $y$

---

Our proof of Theorem 6 also requires knowing the order of vertices along a quadrangle, and the data structures do not support this directly. Therefore at any quadrangle-vertex  $f$  we stellate each of the four incident triangular faces; see Figure 2. Since we have not yet contracted any edges, we have access to the rotational clockwise order of edges at  $f$ ; we can hence label the added vertices with  $quad_i$  (for  $0 \leq i \leq 3$ ) in clockwise order around  $f$ . With this, we can retrieve the clockwise order  $\langle z_0, \dots, z_3 \rangle$  of vertices on the quadrangle-face corresponding to  $f$  in constant time. (See Procedure 2 for details.)

We use  $H^{\diamond}$  for the graph that results after all these modifications (it can be viewed as the planar skeleton  $H$  with a “diamond”-gadget inserted into each quadrangle-face). We also add the following meta-data to each vertex of  $H^{\diamond}$ :

- A boolean `adj`, initialized to **false**.
- A boolean `in_worklist`, initialized to **false**.
- An integer `opposing`, initialized to 0.

## 16:10 Partitioning the Edges of a 1-Planar Graph

### ■ Procedure 2 FacialCycle( $f$ ).

---

```
Result: Reconstruct the facial circuit of a quadrangle-face, given the corresponding
quadrangle-vertex.
// pre:  $f$  is a vertex of  $H^\diamond$  with label  $quad$ 
for vertex  $v$  in neighbors( $f$ ) do
  | for  $i = 0, \dots, 3$  do
  | | if  $v$  has label  $quad_i$  then  $f_i := v$ 
// By construction  $f_i$  has neighbours  $\{z_{i-1}, z_i, f\}$  (indices are mod 4)
for  $i = 0, \dots, 3$  do
  |  $N_i :=$  neighbors( $f_i$ )  $\cap$  neighbors( $f_{i+1}$ )
  | if  $|N_i|$  equals 2 then  $z_i := N_i \setminus \{f\}$ 
// If  $f$  is not simple then  $|N_i| = 3$  for two values of  $i$ , see Fig.2.
  | But then  $z_i$  is determined since we know  $z_{i-1} = z_{i+1}$  already
for  $i = 0, \dots, 3$  do
  | if  $|N_i|$  equals 3 then  $z_i := N_i \setminus N_{i+2}$ 
return  $\langle z_0, z_1, z_2, z_3 \rangle$ 
```

---

The main idea of our algorithm is to iteratively contract all the quadrangles incident to some vertex  $x$ . As we do this, we will use `adj` to mark vertices that are adjacent to  $x$ , `in_worklist` to mark unprocessed quadrangles incident to  $x$ , and `opposing` to keep track of the number of quadrangle-faces where vertex  $y$  is the opposing vertex of  $x$ .

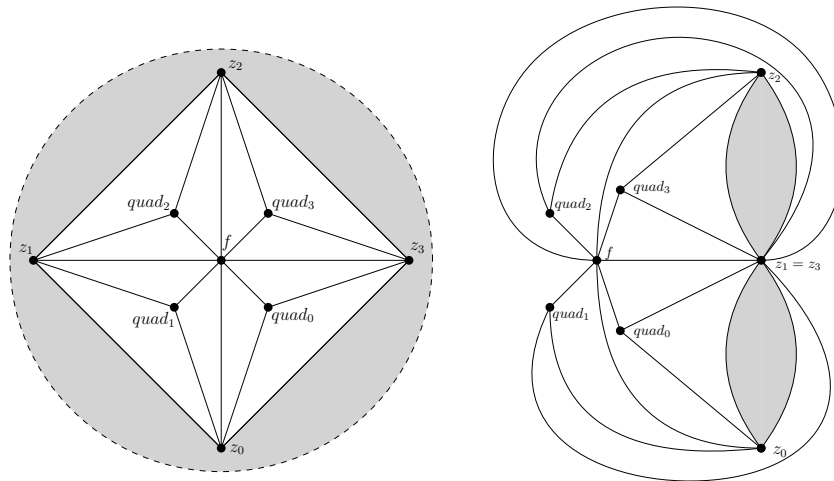
Since  $G$  has  $O(n)$  edges,  $H$  has  $O(n)$  faces, so  $H^\diamond$  has  $O(n)$  edges. All steps in this preprocessing can hence be done in linear time.

### 4.3 Handling the Quadrangles around a Vertex

The main subroutine of our algorithm handles all quadrangles incident to some vertex  $x$  by contracting each of them using the criteria laid out in Theorem 6 to decide which vertices to contract. To do so, we will initialize and maintain a work-list of faces incident to  $x$  that we need to contract (using `in_worklist` to avoid putting duplicate quadrangles into the worklist). We also mark vertices in  $H^\diamond$  as `opposing` and `adj` to  $x$  as needed; this can be done in  $O(d(x))$  time by retrieving all neighbours of  $x$  via `neighbours`. (See Procedure 3 for details.)

Now we iteratively contract all the faces in the worklist according to Theorem 6; with `adj` and `opposing` we can determine the correct case in constant time. (See Procedure 4 for details.) In Case 1, we need to update some values of `opposing`: the vertex  $z_2$  which was opposing  $x$  now no longer opposes  $x$  in  $f$  (since  $f$  was destroyed), so we decrement  $z_2$ .`opposing`. Likewise the new vertex  $v$  resulting from the contraction will be opposing  $x$  in all those quadrangles in which  $z_1$  and  $z_3$  previously opposed  $x$ , so we set  $v$ .`opposing` correspondingly. In Case 2, when we contract some vertex  $z_2$  into  $x$ , we need to add the quadrangles incident to  $z_2$  to our worklist, for which we can re-use Procedure 3.

Lastly, once our worklist is empty (and hence there are no more quadrangles incident to  $x$ ), we reset the meta-data of  $x$  and its neighbors, so that when repeating the procedure with a different vertex as  $x$  there are no stray vertices with meta-data set to erroneous values. (See Procedure 5.)



■ **Figure 2** The gadget added to every quadrangle of  $H$ , shown for both a quadrangle with four vertices on its boundary (left) and one with three vertices on its boundary (right).

■ **Procedure 3** `InitializeAtOneVertex( $y$ ,  $worklist$ )`.

---

**Result:** Adds all quadrangle-vertices incident to a vertex  $y$  to a worklist, taking care not to put duplicates in the worklist.

// pre:  $y$  is a vertex of  $H$

// pre:  $y$  equals  $x$  (the vertex we currently work on), or  $y$  will be contracted into  $x$

**for** vertex  $v \in \text{neighbors}(y)$  **do**

$v.\text{adj} := \text{true}$

**if**  $v$  has label *quad* **and not**  $v.\text{in\_worklist}$  **then**

$v.\text{in\_worklist} := \text{true}$

$\text{worklist.push}(v)$

$\langle z_0, z_1, z_2, z_3 \rangle := \text{FacialCycle}(v)$

relabel  $z_i$  such that  $y$  equals  $z_0$

$z_2.\text{opposing} += 1$

## 4.4 Putting it All Together

The following summarizes our algorithm: after preprocessing, and for as long as there is a quadrangle-face  $f$  left, process all quadrangles at a vertex  $x$  on  $f$  and record all edges that belong to the forest along the way. See Procedure 6 for a detailed description.

It remains to analyze the run-time. For now, we ignore the time required to perform the contractions and analyze the time for handling all quadrangles at one vertex  $x$ . Initialization takes  $O(d(x))$  time, and most other steps take constant time per handled quadrangle, with one notable exception: When we contract some vertex  $z_2$  into  $x$ , we must update the worklist, which takes time  $O(d(z_2))$ . Complicating matters further,  $z_2$  may actually be the result of prior contractions, so its degree may be more than what it was in  $H^\diamond$ , and we must ensure that degrees of vertices are not counted repeatedly.

To handle this, let  $H_f^\diamond$  be the graph that results from  $H^\diamond$  after *all* quadrangle-vertices have been contracted, and let  $s(x)$  be the set of vertices that were contracted into  $x$ , either directly (when handling the quadrangles at  $x$ ) or indirectly (i.e., if they had been contracted

## 16:12 Partitioning the Edges of a 1-Planar Graph

### ■ Procedure 4 HandleQuadsAtOneVertex( $x$ ).

---

**Result:** Contract all the quadrangle-faces incident to a vertex  $x$

```

worklist := []
InitializeAtOneVertex(x, worklist) // see Proc. 3
for quadrangle-vertex f in worklist do
    ⟨z0, z1, z2, z3⟩ := FacialCycle(f) // see Proc. 2
    relabel zi such that x equals z0
    if z2 equals x or z2.adj is true or z2.opposing ≥ 2 then // Case 1
        opposing1 := z1.opposing
        opposing3 := z3.opposing
        v := ContractThrough(z1, z3, f) // see Proc. 1
        v.adj := true
        v.opposing := opposing1 + opposing3
        z2.opposing -= 1
    else // Case 2
        InitializeAtOneVertex(z2, worklist)
        x := ContractThrough(x, z2, f)
CleanupAtOneVertex(x) // see Proc. 5

```

---

### ■ Procedure 5 CleanupAtOneVertex( $y$ ).

---

**Result:** Cleanup the metadata at a vertex  $y$  and its neighbors.

// pre:  $y$  is a vertex of  $H$

```

for vertex v ∈ neighbors(y) ∪ {y} do
    v.adj := false
    v.in_worklist := false
    v.opposing := 0

```

---

### ■ Procedure 6 FindPGFPartition( $G$ ).

---

**Result:** Find a PGF-partition of a graph  $G$ .

Add edges to make  $G$  planar maximal 1-planar

Compute planarization  $G^\times$ , mark vertices of crossings with *quad*

```

foreach vertex f marked quad do
    Insert four vertices in four incident faces of f
    Mark these vertices with quad0, ..., quad3 according to embedding
while there remains a vertex f labeled quad do
    x := some neighbor of f not labeled quadi
    HandleQuadsAtOneVertex(x)

```

Return all edges that were recorded as forest  $F$  and  $G \setminus F$  as planar graph

---

into one of the vertices  $z_2$  that later get contracted into  $x$ ). Crucially, note that if  $x, x'$  are two vertices that are parameters during a call to Procedure 4 (i.e., we contract all quadrangles incident to the vertex), then  $s(x)$  and  $s(x')$  are disjoint. This holds because once we are done with the first of them (say  $x$ ), all vertices in  $s(x)$  have been combined with  $x$  and no longer have any incident quadrangles. Since we only contract vertices into  $x'$  that are incident to quadrangles, none of the vertices in  $s(x)$  becomes part of  $s(x')$ .

Hence for each vertex  $x$  of  $H_f^\diamond$ , the amount of work done in Procedure 4 is proportional to the sum of the degrees  $d_{H^\diamond}(y)$  for each  $y \in s(x)$ . Since  $H^\diamond$  has  $O(n)$  edges, and all other parts of the algorithm take constant time per quadrangle-vertex, the total amount of work done is at most

$$O\left(\sum_{x \in V(H_f^\diamond)} \sum_{y \in s(x)} d_{H^\diamond}(y)\right) = O\left(\sum_{x \in V(H^\diamond)} d_{H^\diamond}(x)\right) = O(|V(H^\diamond)|) = O(|V(G)|)$$

hence the algorithm is linear (ignoring the time for contractions).

Now we consider the run-time of possible data structures for contractions. Our first approach is to represent the graph with incidence lists, where every vertex has a list of incident edges, each edge knows both of its endpoints, and every list knows its length. We can implement `neighbors(x)` in constant time by simply returning an iterator to the incidence list at  $x$ . Contracting two vertices  $u$  and  $v$  can be done in  $O(\min\{d(u), d(v)\})$  time by re-attaching the edges of the vertex with smaller degree to the vertex with larger degree. As with UNION-FIND data structures implemented with linked lists (see e.g. Section 4.6 of [17]), one shows that the amortized time for this is  $O(\log n)$  per contraction. In particular, for graphs with linearly many edges, a set of  $\Theta(n)$  contractions can be done in  $O(n \log n)$  time. With this we have our first result.

► **Theorem 7.** *Let  $G$  be a 1-planar graph implemented with incidence lists and given with a 1-planar embedding. It is possible to find a PGF-partition of  $G$  in  $O(n \log n)$  time.*

To improve this runtime, we appeal to the following result by Holm et al.

► **Theorem 8** (Holm et al. [14]). *Let  $G$  be a planar graph with  $n$  vertices and  $m$  edges. Then there exists a data structure that supports `contract` and `neighbors` and that can be initialized in  $O(n + m)$  time. Any calls to `neighbors` can be processed in worst case constant time, and any sequence of calls to `contract` can be performed in time  $O(n + m)$ .*

Since our graph has  $O(n)$  edges, we have the main result of this paper.

► **Theorem 9.** *Let  $G$  be a 1-planar graph with a given 1-planar embedding. Then in  $O(n)$  time we can find an edge-partition of  $G$  into a forest and a planar graph.*

## 5 Conclusion

In this paper, we reproved a result from Ackerman that all 1-planar graphs admit a partition into a planar graph and a forest. Our proof is more general than Ackerman's: the forest we find is guaranteed to not contain a path between adjacent vertices of the input graph. Using this proof and a data structure from Holm et al. for efficiently contracting the edges of a planar graph, we were able to find this partition in linear time. In consequence, a number of results for 1-planar graphs (such as splitting into 4 forests or 4-list-coloring if the graph is bipartite) can now be achieved in linear time. We also showed that the same algorithm can be implemented in  $O(n \log n)$  time with a simpler data structure that uses only incidence lists.



As for open problems, the most interesting one is whether the partition could be found even *without* being given the 1-planar drawing. (Recall that it is NP-hard to find such a drawing [12], though it is polynomial for optimal 1-planar graphs [7].) All papers listed in the introduction for finding various edge partitions of 1-planar graphs require such an embedding. If this is difficult, could we at least do some of the implications (such as splitting into 4 forests or orienting such that all in-degrees are at most 4) in linear time without a given 1-planar drawing?

---

## References

- 1 Eyal Ackerman. A note on 1-planar graphs. *Discrete Appl. Math.*, 175:104–108, 2014. doi:10.1016/j.dam.2014.05.025.
- 2 Md. Jawaherul Alam, Franz J. Brandenburg, and Stephen G. Kobourov. Straight-line grid drawings of 3-connected 1-planar graphs. In Stephen Wismath and Alexander Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2013. doi:10.1007/978-3-319-03841-4\_8.
- 3 Patrizio Angelini, Henry Förster, Michael Hoffmann, Michael Kaufmann, Stephen Kobourov, Giuseppe Liotta, and Maurizio Patrignani. The QuaSEFE problem. In *International Symposium on Graph Drawing and Network Visualization*, volume 11904 of *Lecture Notes in Computer Science*, pages 268–275. Springer, 2019. doi:10.1007/978-3-030-35802-0\_21.
- 4 Michael A. Bekos, Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, Fabrizio Montecchiani, and Chrysanthi Raftopoulou. Edge partitions of optimal 2-plane and 3-plane graphs. *Discrete Mathematics*, 342(4):1038–1047, 2019. doi:10.1016/j.disc.2018.12.002.
- 5 Therese Biedl, Anna Lubiw, and Owen Merkel. List coloring bipartite graphs embedded on a surface. Unpublished Manuscript, 2019.
- 6 R. Bodendiek, H. Schumacher, and K. Wagner. Bemerkungen zu einem sechsfarbenproblem von g. ringel. *Abh. Math. Semin. Univ. Hambg.*, pages 41–52, 1983. doi:10.1007/BF02941309.
- 7 Franz J. Brandenburg. Recognizing optimal 1-planar graphs in linear time. *Algorithmica*, 80(1):1–28, 2018. doi:10.1007/s00453-016-0226-8.
- 8 Július Czap and Dávid Hudák. On drawings and decompositions of 1-planar graphs. *Electron. J. Combin.*, 20(2):Paper 54, 8, 2013. doi:10.37236/2392.
- 9 Emilio Di Giacomo, Walter Didimo, William S. Evans, Giuseppe Liotta, Henk Meijer, Fabrizio Montecchiani, and Stephen K. Wismath. New results on edge partitions of 1-plane graphs. *Theoretical Computer Science*, 713:78–84, 2018. doi:10.1016/j.tcs.2017.12.024.
- 10 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fifth edition, 2018. doi:10.1007/978-3-662-53622-3.
- 11 Harold N. Gabow and Herbert H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7(5-6):465–497, 1992. doi:10.1007/BF01758774.
- 12 Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. doi:10.1007/s00453-007-0010-x.
- 13 Shai Gutner and Michael Tarsi. Some results on (a:b)-choosability. *Discrete Mathematics*, 309(8):2260–2270, 2009. doi:10.1016/j.disc.2008.04.061.
- 14 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki Łącki, Eva Rotenberg, and Piotr Sankowski. Contracting a planar graph efficiently. In *25th European Symposium on Algorithms*, volume 87 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 50, 15. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2017.
- 15 Seok-Hee Hong and Takeshi Tokuyama. *Beyond Planar Graphs*. Springer Nature, Singapore, 2020.
- 16 Dmitri V. Karpov. An upper bound on the number of edges in an almost planar bipartite graph. *J. Math Sci.*, 196:737–746, 2014. doi:10.1007/s10958-014-1690-9.
- 17 Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005.

- 18 Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Comput. Sci. Rev.*, 25:49–67, 2017. doi:10.1016/j.cosrev.2017.06.002.
- 19 William J. Lenhart, Giuseppe Liotta, and Fabrizio Montecchiani. On partitioning the edges of 1-plane graphs. *Theor. Comput. Sci.*, 662:59–65, 2017. doi:10.1016/j.tcs.2016.12.004.
- 20 C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *J. London Math. Soc.*, 39:12, 1964. doi:10.1112/jlms/s1-39.1.12.
- 21 Gerhard Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abh. Math. Sem. Univ. Hamburg*, 29:107–117, 1965. doi:10.1007/BF02996313.
- 22 Gerhard Ringel. Two trees in maximal planar bipartite graphs. *J. Graph Theory*, 17(6):755–758, 1993. doi:10.1002/jgt.3190170610.
- 23 Marcus Schaefer. The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics [electronic only]*, 20, April 2013. doi:10.37236/2713.
- 24 Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '90*, pages 138–148, USA, 1990. Society for Industrial and Applied Mathematics.



# On the Kernel and Related Problems in Interval Digraphs

Mathew C. Francis ✉

Indian Statistical Institute, Chennai Centre, India

Pavol Hell ✉

School of Computing Science, Simon Fraser University, Burnaby, Canada

Dalu Jacob ✉

Indian Statistical Institute, Chennai Centre, India

---

## Abstract

Given a digraph  $G$ , a set  $X \subseteq V(G)$  is said to be an *absorbing set* (resp. *dominating set*) if every vertex in the graph is either in  $X$  or is an in-neighbour (resp. out-neighbour) of a vertex in  $X$ . A set  $S \subseteq V(G)$  is said to be an *independent set* if no two vertices in  $S$  are adjacent in  $G$ . A kernel (resp. solution) of  $G$  is an independent and absorbing (resp. dominating) set in  $G$ . The problem of deciding if there is a kernel (or solution) in an input digraph is known to be NP-complete. Similarly, the problems of computing a minimum cardinality kernel, absorbing set (or dominating set) and the problems of computing a maximum cardinality kernel, independent set are all known to be NP-hard for general digraphs. We explore the algorithmic complexity of these problems in the well known class of *interval digraphs*. A digraph  $G$  is an interval digraph if a pair of intervals  $(S_u, T_u)$  can be assigned to each vertex  $u$  of  $G$  such that  $(u, v) \in E(G)$  if and only if  $S_u \cap T_v \neq \emptyset$ . Many different subclasses of interval digraphs have been defined and studied in the literature by restricting the kinds of pairs of intervals that can be assigned to the vertices. We observe that several of these classes, like interval catch digraphs, interval nest digraphs, adjusted interval digraphs and chronological interval digraphs, are subclasses of the more general class of *reflexive interval digraphs* – which arise when we require that the two intervals assigned to a vertex have to intersect. We see as our main contribution the identification of the class of reflexive interval digraphs as an important class of digraphs. We show that all the problems mentioned above are efficiently solvable, in most of the cases even linear-time solvable, in the class of reflexive interval digraphs, but are APX-hard on even the very restricted class of interval digraphs called *point-point digraphs*, where the two intervals assigned to each vertex are required to be degenerate, i.e. they consist of a single point each. The results we obtain improve and generalize several existing algorithms and structural results for reflexive interval digraphs. We also obtain some new results for undirected graphs along the way: (a) We get an  $O(n(n+m))$  time algorithm for computing a minimum cardinality (undirected) independent dominating set in cocomparability graphs, which slightly improves the existing  $O(n^3)$  time algorithm for the same problem by Kratsch and Stewart; and (b) We show that the RED BLUE DOMINATING SET problem, which is NP-complete even for planar bipartite graphs, is linear-time solvable on *interval bigraphs*, which is a class of bipartite (undirected) graphs closely related to interval digraphs.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms

**Keywords and phrases** Interval digraphs, kernel, absorbing set, dominating set, independent set, algorithms, approximation hardness

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.17

**Related Version** *Full Version*: <https://arxiv.org/abs/2107.08278> [11]

**Funding** *Pavol Hell*: Support from an NSERC Discovery Grant is gratefully acknowledged. Part of this work was also supported by the Smt Rukmini Gopalakrishnchar Chair Professorship at IISc (November-December, 2019).



© Mathew C. Francis, Pavol Hell, and Dalu Jacob;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 17; pp. 17:1–17:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Let  $H = (V, E)$  be an undirected graph. A set  $S \subseteq V(H)$  is said to be an *independent set* in  $H$  if for any two vertices  $u, v \in S$ ,  $uv \notin E(H)$ . A set  $S \subseteq V(H)$  is said to be a *dominating set* in  $H$  if for any  $v \in V(H) \setminus S$ , there exists  $u \in S$  such that  $uv \in E(H)$ . A set  $S \subseteq V(H)$  is said to be an *independent dominating set* in  $H$  if  $S$  is dominating as well as independent. Note that any maximal independent set in  $H$  is an independent dominating set in  $H$ , and therefore every undirected graph contains an independent dominating set, which implies that the problem of deciding whether an input undirected graph contains an independent dominating set is trivial. On the other hand, finding an independent dominating set of *maximum cardinality* is NP-complete for general graphs, since independent dominating sets of maximum cardinality are exactly the independent sets of maximum cardinality in the graph. The problem of finding a minimum cardinality independent dominating set is also NP-complete for general graphs [12] and also in many special graph classes (refer [18] for a survey). We study the directed analogues of these problems, which are also well-studied in the literature.

Let  $G = (V, E)$  be a directed graph. A set  $S \subseteq V(G)$  is said to be an *independent set* in  $G$ , if for any two vertices  $u, v \in S$ ,  $(u, v), (v, u) \notin E(G)$ . A set  $S \subseteq V(G)$  is said to be an *absorbing (resp. dominating) set* in  $G$ , if for any  $v \in V(G) \setminus S$ , there exists  $u \in S$  such that  $(v, u) \in E(G)$  (resp.  $(u, v) \in E(G)$ ). As any set of vertices that consists of a single vertex is independent and the whole set  $V(G)$  is absorbing as well as dominating, the interesting computational problems that arise here are that of finding a maximum independent set, called INDEPENDENT-SET, and that of finding a minimum absorbing (resp. dominating) set in  $G$ , called ABSORBING-SET (resp. DOMINATING-SET). A set  $S \subseteq V(G)$  is said to be an *independent dominating (resp. absorbing) set* if  $S$  is both independent and dominating (resp. absorbing). Note that unlike undirected graphs, the problem of finding a maximum cardinality independent dominating (resp. absorbing) set is different from the problem of finding a maximum cardinality independent set for directed graphs.

Given a digraph  $G$ , a collection  $\{(S_u, T_u)\}_{u \in V(G)}$  of pairs of intervals is said to be an *interval representation* of  $G$  if  $(u, v) \in E(G)$  if and only if  $S_u \cap T_v \neq \emptyset$ . A digraph  $G$  that has an interval representation is called an *interval digraph* [6]. We consider a loop to be present on a vertex  $u$  of an interval digraph if and only if  $S_u \cap T_u \neq \emptyset$ . An interval digraph is a *reflexive interval digraph* if there is a loop on every vertex. Let  $G$  be a digraph. If there exists an interval representation of  $G$  such that  $T_u \subseteq S_u$  for each vertex  $u \in V(G)$  then  $G$  is called an *interval nest digraph* [26]. If  $G$  has an interval representation in which intervals  $S_u$  and  $T_u$  for each vertex  $u \in V(G)$  are required to have a common left end-point, the interval digraphs that arise are called *adjusted interval digraphs* [9]. Note that the class of reflexive interval digraphs is a superclass of both interval nest digraphs and adjusted interval digraphs. Another class of interval digraphs, called *interval-point digraphs* arises when the interval  $T_u$  for each vertex  $u$  is required to be degenerate (it is a point) [6]. Note that interval-point digraphs may not be reflexive. We call a digraph  $G$  a *point-point digraph* if there is an interval representation of  $G$  in which both  $S_u$  and  $T_u$  are degenerate intervals for each vertex  $u$ . Clearly, point-point digraphs form a subclass of interval-point digraphs and they are also not necessarily reflexive.

In this paper, we show that the reflexivity of an interval digraph has a huge impact on the algorithmic complexity of several problems related to domination and independent sets in digraphs. In particular, we show that all the problems we study are efficiently solvable on reflexive interval digraphs, but are NP-complete and/or APX-hard even on point-point digraphs. Along the way we obtain new characterizations of both these graph classes, which reveal some of the properties of these digraphs.

An undirected graph is a *comparability* graph if its edges can be oriented in such a way that it becomes a partial order. The complements of comparability graphs are called *cocomparability graphs*.

**Our results.** We provide a vertex-ordering characterization for reflexive interval digraphs and two simple characterizations for point-point digraphs including a forbidden structure characterization. Our characterization of point-point digraphs directly yields a linear time recognition algorithm for that class of digraphs (note that Müller's [22] recognition algorithm for interval digraphs directly gives a polynomial-time recognition algorithm for reflexive interval digraphs). From our vertex-ordering characterization of reflexive interval digraphs, it follows that the underlying undirected graphs of every reflexive interval digraph is a cocomparability graph. Also a natural question that arises here is whether the underlying graphs of reflexive interval digraphs is the same as the class of cocomparability graphs. We show that this is not the case by demonstrating that the underlying graphs of reflexive interval digraphs cannot contain an induced  $K_{3,3}$ . This can be used to strengthen a result of Prisner [26] about interval nest digraphs: our results imply that the underlying undirected graphs of interval nest digraphs and their reversals are  $K_{3,3}$ -free weakly triangulated cocomparability graphs. Also, as the INDEPENDENT SET problem is linear time solvable on cocomparability graphs [19], the problem is also linear time solvable on reflexive interval digraphs. This improves and generalizes the  $O(nm)$ -time algorithm for the same problem on interval nest digraphs. In contrast, we prove that the INDEPENDENT SET problem is APX-hard for point-point digraphs.

Domination in digraphs is a topic that has been explored less when compared to its undirected counterpart. Even though bounds on the minimum dominating sets in digraphs have been obtained by several authors (see the book [13] for a survey), not much is known about the computational complexity of finding a minimum cardinality absorbing set (or dominating set) in directed graphs. Even for tournaments, the best known algorithm for DOMINATING-SET does not run in polynomial-time [20, 27]. In [20], the authors give an  $n^{O(\log n)}$  time algorithm for the DOMINATING-SET problem in tournaments and they also note that SAT can be solved in  $2^{O(\sqrt{v})}n^K$  time (where  $v$  is the number of variables,  $n$  is the length of the formula and  $K$  is a constant) if and only if the DOMINATING-SET in a tournament can be solved in polynomial time. Thus, determining the algorithmic complexity of the DOMINATING-SET problem even in special classes of digraphs seems to be much more challenging than the algorithmic question of finding a minimum cardinality dominating set in undirected graphs.

For a bipartite graph having two specified partite sets  $A$  and  $B$ , a set  $S \subseteq B$  such that  $\bigcup_{u \in B} N(u) = A$  is called an *A-dominating set*. Note that the graph does not contain an  $A$ -dominating set if and only if there are isolated vertices in  $A$ . The problem of finding an  $A$ -dominating set of minimum cardinality in a bipartite graph with partite sets  $A$  and  $B$  is more well-known as the RED-BLUE DOMINATING SET problem, which was introduced for the first time in the context of the European railroad network [30] and plays an important role in the theory of fixed parameter tractable algorithms [7]. This problem is equivalent to the well known SET COVER and HITTING SET problems [12] and therefore, it is NP-complete for general bipartite graphs. The problem remains NP-complete even for planar bipartite graphs [1]. The class of *interval bigraphs* are closely related to the class of interval digraphs. These are undirected bipartite graphs with partite sets  $A$  and  $B$  such that there exists a collection of intervals  $\{S_u\}_{u \in V(G)}$  such that  $uv \in E(G)$  if and only if  $u \in A$ ,  $v \in B$ , and  $S_u \cap S_v \neq \emptyset$ .

**Our results.** We observe that the problem of solving ABSORBING-SET on a reflexive interval digraph  $G$  can be reduced to the problem of solving RED-BLUE DOMINATING SET on an interval bigraph whose interval representation can be constructed from an interval representation of  $G$  in linear time. Further, we show that RED-BLUE DOMINATING SET is linear time solvable on interval bigraphs (given an interval representation). Thus the problem ABSORBING-SET (resp. DOMINATING-SET<sup>1</sup>) is linear-time solvable on reflexive interval digraphs, given an interval representation of the digraph as input. If no interval representation is given, Müller’s algorithm [22] can be used to construct one in polynomial time, and therefore these problems are polynomial time solvable on reflexive interval digraphs even when no interval representation of the input graph is known. In contrast, we prove that the ABSORBING-SET (resp. DOMINATING-SET) problem remains APX-hard even for point-point digraphs.

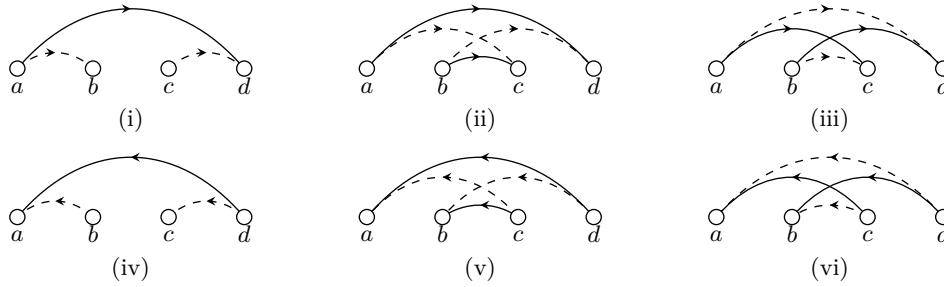
An independent absorbing set in a directed graph is more well-known as a *kernel* of the graph, a term introduced by Von Neumann and Morgenstern [21] in the context of game theory. They showed that for digraphs associated with certain combinatorial games, the existence of a kernel implies the existence of a winning strategy. Most of the work related to domination in digraphs has been mainly focused on kernels. We follow the terminology in [26] and call an independent dominating set in a directed graph a *solution* of the graph. It is easy to see that a kernel in a directed graph  $G$  is a solution in the directed graph obtained by reversing every arc of  $G$  and vice versa. Note that unlike in the case of undirected graphs, a kernel need not always exist in a directed graph. Therefore, besides the computational problems of finding a minimum or maximum sized kernel, called MIN-KERNEL and MAX-KERNEL respectively, the comparatively easier problem of determining whether a given directed graph has a kernel in the first place, called KERNEL, is itself a non-trivial one. In fact, the KERNEL problem was shown to be NP-complete in general digraphs by Chvátal [4]. Later, Fraenkel [10] proved that the KERNEL problem remains NP-complete even for planar digraphs of degree at most 3 having in- and out-degrees at most 2. It can be easily seen that the MIN-KERNEL and MAX-KERNEL problems are NP-complete for those classes of graphs for which the KERNEL problem is NP-complete. A digraph is said to be *kernel-perfect* if every induced subgraph of it has a kernel. Several sufficient conditions for digraphs to be kernel-perfect has been explored [28, 8, 21]. The KERNEL problem is trivially solvable in polynomial-time on any kernel-perfect family of digraphs. But the algorithmic complexity status of the problem of computing a kernel in a kernel-perfect digraph also seems to be unknown [24]. Prisner [26] proved that interval nest digraphs and their reversals are kernel-perfect, and a kernel can be found in these graphs in time  $O(n^2)$  if a representation of the graph is given. Note that the MIN-KERNEL problem can be shown to be NP-complete even in some kernel-perfect families of digraphs that have polynomial-time computable kernels (see Remark 13).

**Our results.** We show that reflexive interval digraphs are kernel-perfect and hence the KERNEL problem is trivial on this class of digraphs. We construct a linear-time algorithm that computes a kernel in a reflexive interval digraph, given an interval representation of

---

<sup>1</sup> Note that the reversal of a reflexive interval digraphs is also a reflexive interval digraph. It is easy to see that the reversal of any digraph can be constructed in linear time. Moreover, an interval representation of the reversal of a reflexive interval digraph  $G$  can be obtained in linear time given an interval representation of  $G$  (if  $\{(S_u, T_u)\}_{u \in V(G)}$  is the interval representation of  $G$ , then  $\{(T_u, S_u)\}_{u \in V(G)}$  is an interval representation of the reversal of  $G$ ). Therefore, all our results about the KERNEL, MIN-KERNEL, MAX-KERNEL, and ABSORBING-SET problems can be adapted to obtain similar results about the SOLUTION, MIN-SOLUTION, MAX-SOLUTION, and DOMINATING-SET problems.





**Figure 1** Forbidden structures for reflexive interval digraphs (possibly  $b = c$  in (i), (ii), (iv) and (v)). A dashed arc from  $u$  to  $v$  indicates the absence of the edge  $(u, v)$  in the graph.

digraph as an input. This improves and generalizes Prisner’s similar results about interval nest digraphs mentioned above. Moreover, we give an  $O((n + m)n)$  time algorithm for the MIN-KERNEL and MAX-KERNEL problems for a superclass of reflexive interval digraphs. As a consequence, we obtain an improvement over the  $O(n^3)$  time algorithm for finding a minimum independent dominating set in comparability graphs that was given by Kratsch and Stewart [17]. Our algorithms for MIN-KERNEL and MAX-KERNEL problems have a better running time of  $O(n^2)$  for adjusted interval digraphs. On the other hand, we show that even the problem KERNEL is NP-complete for point-point digraphs. Moreover, the MIN-KERNEL and MAX-KERNEL problems are APX-hard on point-point digraphs.

### 1.1 Notation

For a closed interval  $I = [x, y]$  of the real line (here  $x, y \in \mathbb{R}$  and  $x \leq y$ ), we denote by  $l(I)$  the left end-point  $x$  of  $I$  and by  $r(I)$  the right end-point  $y$  of  $I$ . We use the following observation throughout the paper: if  $I$  and  $J$  are two intervals, then  $I \cap J = \emptyset \Leftrightarrow (r(I) < l(J)) \vee (r(J) < l(I))$ .

Let  $G = (V, E)$  be a directed graph. For  $u, v \in V(G)$ , we say that  $u$  is an *in-neighbour* (resp. *out-neighbour*) of  $v$  if  $(u, v) \in E(G)$  (resp.  $(v, u) \in E(G)$ ). For a vertex  $v$  in  $G$ , we denote by  $N_G^+(v)$  and  $N_G^-(v)$  the set of out-neighbours and the set of in-neighbours of the vertex  $v$  in  $G$  respectively. When the graph  $G$  under consideration is clear from the context, we abbreviate  $N_G^+(v)$  and  $N_G^-(v)$  to just  $N^+(v)$  and  $N^-(v)$  respectively.

For  $i, j \in \mathbb{N}$  such that  $i \leq j$ , let  $[i, j]$  denote the set  $\{i, i + 1, \dots, j\}$ . Let  $G$  be a digraph with vertex set  $[1, n]$ . Then for  $i, j \in [1, n]$ , we define  $N_{>j}^+(i) = N^+(i) \cap [j + 1, n]$ ,  $N_{>j}^-(i) = N^-(i) \cap [j + 1, n]$ ,  $N_{<j}^+(i) = N^+(i) \cap [1, j - 1]$ , and  $N_{<j}^-(i) = N^-(i) \cap [1, j - 1]$ . We shorten  $N_{>i}^+(i)$  and  $N_{>i}^-(i)$  to  $N_{>}^+(i)$  and  $N_{>}^-(i)$  respectively. Let  $N_{>}(i) = N_{>}^+(i) \cup N_{>}^-(i)$  and  $\overline{N_{>}}(i) = [i + 1, n] \setminus N_{>}(i)$ .

## 2 Ordering characterization

We first show that a digraph is a reflexive interval digraph if and only if there is a linear ordering of its vertex set such that none of the structures shown in Figure 1 are present.

► **Theorem 1** ( $\star$ ).<sup>2</sup> A digraph  $G$  is a reflexive interval digraph if and only if  $V(G)$  has an ordering  $<$  in which for any  $a, b, c, d \in V(G)$  such that  $a < b < c < d$ , none of the structures in Figure 1 occur ( $b$  and  $c$  can be the same vertex in (i), (ii), (iv), (v) of Figure 1).

Now we define the following.

► **Definition 2** (DUF-ordering). A directed umbrella-free ordering (or in short a DUF-ordering) of a digraph  $G$  is an ordering  $<$  of  $V(G)$  satisfying the following properties for any three distinct vertices  $i < j < k$ :

1. if  $(i, k) \in E(G)$ , then either  $(i, j) \in E(G)$  or  $(j, k) \in E(G)$ , and
2. if  $(k, i) \in E(G)$ , then either  $(k, j) \in E(G)$  or  $(j, i) \in E(G)$ .

► **Definition 3** (DUF-digraph). A digraph  $G$  is a directed umbrella-free digraph (or in short a DUF-digraph) if it has a DUF-ordering.

Then the following corollary is an immediate consequence of Theorem 1.

► **Corollary 4.** Every reflexive interval digraph is a DUF-digraph.

Let  $G$  be an undirected graph. We define the *symmetric digraph* of  $G$  to be the digraph obtained by replacing each edge of  $G$  by symmetric arcs.

The following is a characterization of cocomparability graphs due to Kratsch and Stewart [17].

► **Theorem 5** ([17]). An undirected graph  $G$  is a cocomparability graph if and only if there is an ordering  $<$  of  $V(G)$  such that for any three vertices  $i < j < k$ , if  $ik \in E(G)$ , then either  $ij \in E(G)$  or  $jk \in E(G)$ .

Let  $G$  be a DUF-digraph with a DUF-ordering  $<$ . Let  $H$  be the underlying undirected graph of  $G$ . Clearly,  $<$  is an ordering of  $V(H)$  that satisfies the property given in Theorem 5, implying that  $H$  is a cocomparability graph. Thus we have the following corollary.

► **Corollary 6.** The underlying undirected graph of every DUF-digraph is a cocomparability graph.

Note that there exist digraphs which are not DUF-digraphs but their underlying undirected graphs are cocomparability (for example, a directed triangle with edges  $(a, b)$ ,  $(b, c)$  and  $(c, a)$ ). But we can observe that the class of underlying undirected graphs of DUF-digraphs is precisely the class of cocomparability graphs, since the symmetric digraphs of cocomparability graphs are all DUF-digraphs (for any cocomparability graph  $H$ , a vertex ordering of  $H$  that satisfies the property given in Theorem 5 is also a DUF-ordering of the symmetric digraph of  $H$ ). In contrast, the class of underlying undirected graphs of reflexive interval digraphs forms a strict subclass of cocomparability graphs. We prove this by showing that no directed graph that has  $K_{3,3}$  as its underlying undirected graph can be a reflexive interval digraph ( $K_{3,3}$  can easily be seen to be a cocomparability graph). This would also imply by Corollary 4 that the class of reflexive interval digraphs forms a strict subclass of DUF-digraphs.

► **Theorem 7** ( $\star$ ). The underlying undirected graph of a reflexive interval digraph cannot contain  $K_{3,3}$  as an induced subgraph.

---

<sup>2</sup> The proofs of the statements marked with a  $\star$  are omitted due to space constraints. Refer [11] for the omitted proofs.

Prisner [26] proved that the underlying undirected graphs of interval nest digraphs are weakly triangulated graphs. By Corollaries 4, 6 and Theorem 7, we can conclude that the underlying undirected graphs of reflexive interval digraphs are  $K_{3,3}$ -free cocomparability graphs. This strengthens the result of Prisner, since now we have that the underlying undirected graphs of interval nest digraphs are  $K_{3,3}$ -free weakly triangulated cocomparability graphs.

### 3 Algorithms for reflexive interval digraphs

In this section, we present polynomial-time algorithms for the KERNEL, MIN-KERNEL, MAX-KERNEL, ABSORBING-SET, and INDEPENDENT-SET problems on reflexive interval digraphs.

#### 3.1 Kernel

We use the following result of Prisner that is implied by Theorem 4.2 of [26].

► **Theorem 8** ([26]). *Let  $\mathcal{C}$  be a class of digraphs that is closed under taking induced subgraphs. If in every graph  $G \in \mathcal{C}$ , there exists a vertex  $z$  such that for every  $y \in N^-(z)$ ,  $N^+(z) \setminus N^-(z) \subseteq N^+(y)$ , then the class  $\mathcal{C}$  is kernel-perfect.*

► **Lemma 9** (\*). *Let  $G$  be a reflexive interval digraph  $G$  with interval representation  $\{(S_u, T_u)\}_{u \in V(G)}$ . Let  $z$  be the vertex such that  $r(S_z) = \min\{r(S_v) : v \in V(G)\}$ . Then for every  $y \in N^-(z)$ ,  $N^+(z) \setminus N^-(z) \subseteq N^+(y)$ .*

Since reflexive interval digraphs are closed under taking induced subgraphs, by Theorem 8 and Lemma 9, we have the following.

► **Theorem 10.** *Reflexive interval digraphs are kernel-perfect.*

It follows from the above theorem that the decision problem KERNEL is trivial on reflexive interval digraphs. As explained below, we can also compute a kernel in a reflexive interval digraph efficiently, if an interval representation of the digraph is known.

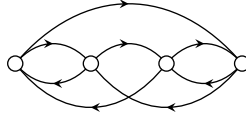
Let  $G$  be a reflexive interval digraph with an interval representation  $\{(S_u, T_u)\}_{u \in V(G)}$ . Let  $G_0 = G$  and  $z_0$  be the vertex in  $G$  such that  $r(S_{z_0}) = \min\{r(S_v) : v \in V(G)\}$ . For  $i \geq 1$ , recursively define  $G_i$  to be the induced subdigraph of  $G$  with  $V(G_i) = V(G_{i-1}) \setminus (\{z_{i-1}\} \cup N^-(z_{i-1}))$  and if  $V(G_i) \neq \emptyset$ , define  $z_i$  to be the vertex such that  $r(S_{z_i}) = \min\{r(S_v) : v \in V(G_i)\}$ . Let  $t$  be smallest integer such that  $V(G_{t+1}) = \emptyset$ . Note that this implies that  $V(G_t) = \{z_t\} \cup N_{G_t}^-(z_t)$ . Clearly  $t \leq n$  and  $r(S_{z_0}) < r(S_{z_1}) < \dots < r(S_{z_t})$ . By Lemma 9, we have that for each  $i \in \{1, 2, \dots, t\}$ ,  $z_i$  has the following property: for any  $y \in N_{G_i}^-(z_i)$  we have  $N_{G_i}^+(z_i) \setminus N_{G_i}^-(z_i) \subseteq N_{G_i}^+(y)$ .

We now recursively define a set  $K_i \subseteq V(G_i)$  as follows: Define  $K_t = \{z_t\}$ . For each  $i \in \{t-1, t-2, \dots, 0\}$ ,

$$K_i = \begin{cases} \{z_i\} \cup K_{i+1} & \text{if } (z_i, z_j) \notin E(G), \text{ where } j = \min\{l : z_l \in K_{i+1}\} \\ K_{i+1} & \text{otherwise.} \end{cases}$$

► **Lemma 11** (\*). *For each  $i \in \{1, 2, \dots, t\}$ ,  $K_i$  is a kernel of  $G_i$ .*

By the above lemma, we have that  $K_0$  is a kernel of  $G$ . We can now construct an algorithm that computes a kernel in a reflexive interval digraph  $G$ , given an interval representation of it. We assume that the interval representation of  $G$  is given in the form of a list of left and



■ **Figure 2** Example of a DUF-digraph that has no kernel.

right endpoints of intervals corresponding to the vertices. We can process this list from left to right in a single pass to compute the list of vertices  $z_0, z_1, \dots, z_t$  in  $O(n + m)$  time. We then process this new list from right to left in a single pass to generate a set  $K$  as follows: initialize  $K = \{z_t\}$  and for each  $i \in \{t-1, t-2, \dots, 0\}$ , add  $z_i$  to  $K$  if it is not an in-neighbor of the last vertex that was added to  $K$ . Clearly, the set  $K$  can be generated in  $O(n + m)$  time. It is easy to see that  $K = K_0$  and therefore by Lemma 11,  $K$  is a kernel of  $G$ . Thus, we have the following theorem.

► **Theorem 12.** *A kernel of a reflexive interval digraph can be computed in linear-time, given an interval representation of the digraph as input.*

The linear-time algorithm described above is an improvement and generalization of a result of Prisner [26], who showed that interval nest digraphs are kernel-perfect, and a kernel can be found in these graphs in time  $O(n^2)$  if an interval representation of the graph is given.

Now it is interesting to note that even for some kernel perfect digraphs with a polynomial-time computable kernel, the problems MIN-KERNEL and MAX-KERNEL turn out to be NP-complete. The following remark provides an example of such a class of digraphs.

► **Remark 13.** Let  $\mathcal{C}$  be the class of symmetric digraphs of undirected graphs. Note that the class  $\mathcal{C}$  is kernel-perfect, as for any  $G \in \mathcal{C}$  the kernels of the digraph  $G$  are exactly the independent dominating sets of its underlying undirected graph. Note that any maximal independent set of an undirected graph is also an independent dominating set of it. Therefore, as a maximal independent set of any undirected graph can be found in linear-time, the problem KERNEL is linear-time solvable for the class  $\mathcal{C}$ . On the other hand, note that the problems MIN-KERNEL and MAX-KERNEL for the class  $\mathcal{C}$  is equivalent to the problems of finding a minimum cardinality independent dominating set and a maximum cardinality independent set for the class of undirected graphs, respectively. Since the latter problems are NP-complete for the class of undirected graphs, we have that the problems MIN-KERNEL and MAX-KERNEL are NP-complete in  $\mathcal{C}$ .

Note that unlike the class of reflexive interval digraphs, the class of DUF-digraphs are not kernel-perfect. Figure 2 provides an example for a DUF-digraph that has no kernel. Since that graph is a semi-complete digraph (i.e. each pair of vertices is adjacent), and every vertex has an out-neighbor which is not its in-neighbor, it cannot have a kernel. The ordering of the vertices of the graph that is shown in the figure can easily be verified to be a DUF-ordering. In contrast to Remark 13, even though DUF-digraphs may not have kernels, we show in the next section that the problems KERNEL, MIN-KERNEL, and MAX-KERNEL can be solved in polynomial time in the class of DUF-digraphs. In fact we give a polynomial time algorithm that, given a DUF-digraph  $G$  with a DUF-ordering as input, either finds a minimum (or maximum) sized kernel in  $G$  or correctly concludes that  $G$  does not have a kernel.

### 3.2 Minimum sized kernel

Let  $G$  be a DUF-digraph with vertex set  $[1, n]$ . We assume without loss of generality that  $\leq = (1, 2, \dots, n)$  is a DUF-ordering of  $G$ .

For any vertex  $i \in \{1, 2, \dots, n\}$ , let  $P_i = \{j : j \in \overline{N_{>}(i)} \text{ such that } [i+1, j-1] \subseteq N^-(i) \cup N^-(j)\}$  and let  $G[i, n]$  denote the subgraph induced in  $G$  by the set  $[i, n]$ . Note that we consider  $[i+1, j-1] = \emptyset$ , if  $j = i+1$ . For a collection of sets  $\mathcal{S}$ , we denote by  $\text{Min}(\mathcal{S})$  an arbitrarily chosen set in  $\mathcal{S}$  of the smallest cardinality. For each  $i \in \{1, 2, \dots, n\}$ , we define a set  $K(i)$  as follows. Here, when we write  $K(i) = \infty$ , we mean that the set  $K(i)$  is undefined.

$$K(i) = \begin{cases} \{i\}, & \text{if } N_{>}^-(i) = \{i+1, \dots, n\} \\ \{i\} \cup \text{Min}\{K(j) \neq \infty : j \in P_i\}, & \text{if } P_i \neq \emptyset \text{ and } \exists j \in P_i \text{ such that } K(j) \neq \infty \\ \infty, & \text{otherwise} \end{cases}$$

Note that it follows from the above definition that  $K(n) = \{n\}$ . For each  $i \in \{1, 2, \dots, n\}$ , let  $OPT(i)$  denote a minimum sized kernel of  $G[i, n]$  that also contains  $i$ . If  $G[i, n]$  has no kernel that contains  $i$ , then we say that  $OPT(i) = \infty$ . We then have the following lemma.

► **Lemma 14** ( $\star$ ). *The following hold.*

1. *If  $K(i) \neq \infty$ , then  $K(i)$  is a kernel of  $G[i, n]$  that contains  $i$ , and*
2. *if  $OPT(i) \neq \infty$ , then  $K(i) \neq \infty$  and  $|K(i)| = |OPT(i)|$ .*

Suppose that  $G$  has a kernel. Now let  $OPT$  denote a minimum sized kernel in  $G$ . Let  $\mathcal{K} = \{K(j) \neq \infty : [1, j-1] \subseteq N^-(j)\}$ . Note that we consider  $[1, j-1] = \emptyset$  if  $j = 1$ . By Lemma 14(1), it follows that every member of  $\mathcal{K}$  is a kernel of  $G$ . So if  $G$  does not have a kernel, then  $\mathcal{K} = \emptyset$ . The following lemma shows that the converse is also true.

► **Lemma 15** ( $\star$ ). *If  $G$  has a kernel, then  $\mathcal{K} \neq \emptyset$  and  $|OPT| = |\text{Min}(\mathcal{K})|$ .*

We thus have the following theorem.

► **Theorem 16**. *The DUF-digraph  $G$  has a kernel if and only if  $K(j) \neq \infty$  for some  $j$  such that  $[1, j-1] \subseteq N^-(j)$ . Further, if  $G$  has a kernel, then the set  $\{K(j) \neq \infty : [1, j-1] \subseteq N^-(j)\}$  contains a kernel of  $G$  of minimum possible size.*

► **Theorem 17** ( $\star$ ). *The MIN-KERNEL problem can be solved for DUF-digraphs in  $O((n+m)n)$  time if the DUF-ordering is known. Consequently, for a reflexive interval digraph, the MIN-KERNEL problem can be solved in  $O((n+m)n)$  time if the interval representation is given as input.*

► **Corollary 18** ( $\star$ ). *An independent dominating set of minimum possible size can be found in  $O((n+m)n)$  time in cocomparability graphs.*

The above corollary is an improvement over the results of Kratsch and Stewart [17], who proved that an independent dominating set of minimum possible size can be computed in  $O(n^3)$  time in cocomparability graphs.

We now show that a minimum sized kernel of an adjusted interval digraph, whose interval representation is known, can be computed more efficiently than in the case of DUF-digraphs.

► **Corollary 19** ( $\star$ ). *The MIN-KERNEL problem can be solved in adjusted interval digraphs in  $O(n^2)$  time, given an interval representation of the digraph.*

► **Remark 20**. Note that the MAX-KERNEL problem can also be solved in  $O((n+m)n)$  time for the class of DUF-digraphs, by a minor modification of our algorithm that solves MIN-KERNEL problem (replace  $\text{Min}\{K(j) \neq \infty : j \in P_i\}$  in the recursive definition of  $K(i)$  by  $\text{Max}\{K(j) \neq \infty : j \in P_i\}$  and follow the same procedure. Then we have that if a kernel exists, then a maximum sized kernel is given by  $\text{Max}(\mathcal{K})$ ). Further, the recursive definition can also be easily adapted to the weighted versions of the problems MIN-KERNEL and MAX-KERNEL to obtain  $O((n+m)n)$  time algorithms for those problems too.

### 3.3 Minimum absorbing set

Given any digraph  $G$ , the splitting bigraph  $B_G$  is defined as follows:  $V(B_G)$  is partitioned into two sets  $V' = \{u' : u \in V(G)\}$  and  $V'' = \{u'' : u \in V(G)\}$ , and  $E(B_G) = \{u'v'' : (u, v) \in E(G)\}$ . Müller [22] observed that  $G$  is an interval digraph if and only if  $B_G$  is an interval bigraph (since if  $\{(S_u, T_u)\}_{u \in V(G)}$  is an interval representation of a digraph  $G$ , then  $\{\{S_u\}_{u' \in V'}, \{T_u\}_{u'' \in V''}\}$  is an interval bigraph representation of the bipartite graph  $B_G$ ).

Recall that for a bipartite graph having two specified partite sets  $A$  and  $B$ , a set  $S \subseteq B$  such that  $\bigcup_{u \in B} N(u) = A$  is called an  $A$ -dominating set (or a red-blue dominating set). If  $G$  is a reflexive interval digraph, then every  $V'$ -dominating set of  $B_G$  corresponds to an absorbing set of  $G$  and vice versa. To be precise, if  $S \subseteq V''$  is a  $V'$ -dominating set of  $B_G$ , then  $\{u \in V(G) : u' \in S\}$  is an absorbing set of  $G$  and if  $S \subseteq V(G)$  is an absorbing set of  $G$ , then  $\{u'' \in V'' : u \in S\}$  is a  $V'$ -dominating set of  $B_G$  (note that this is not true for general interval digraphs). Thus finding a minimum cardinality absorbing set in  $G$  is equivalent to finding a minimum cardinality  $V'$ -dominating set in the bipartite graph  $B_G$ . We show in this section that the problem of computing a minimum cardinality  $A$ -dominating set is linear time solvable for interval bigraphs. This implies that the ABSORBING-SET problem can be solved in linear time on reflexive interval digraphs.

Consider an interval bigraph  $H$  with partite sets  $A$  and  $B$ . Let  $\{I_u\}_{u \in V(H)}$  be an interval representation for  $H$ ; i.e.  $uv \in E(H)$  if and only if  $u \in A$ ,  $v \in B$  and  $I_u \cap I_v \neq \emptyset$ . Let  $|A| = t$ . We assume without loss of generality that  $A = \{1, 2, \dots, t\}$ , where  $r(I_i) < r(I_j) \Leftrightarrow i < j$ . We also assume that there are no isolated vertices in  $A$ , as otherwise  $H$  does not have any  $A$ -dominating set. For each  $i \in \{1, 2, \dots, t\}$ , we compute a minimum cardinality subset  $DS(i)$  of  $B$  that dominates  $\{i, i+1, \dots, t\}$ , i.e.  $\{i, i+1, \dots, t\} \subseteq \bigcup_{u \in DS(i)} N(u)$ . Then  $DS(1)$  will be a minimum cardinality  $A$ -dominating set of  $H$ . We first define some parameters that will be used to define  $DS(i)$ .

Let  $i \in \{1, 2, \dots, t\}$ . We define  $\rho(i) = \max_{u \in N(i)} r(I_u)$  and let  $R(i)$  be a vertex in  $N(i)$  such that  $r(I_{R(i)}) = \rho(i)$ . Since  $A$  does not contain any isolated vertices,  $\rho(i)$  and  $R(i)$  exist for each  $i \in \{1, 2, \dots, t\}$ . Let  $\lambda(i) = \min\{j : \rho(i) < l(I_j)\}$ . Note that  $\lambda(i)$  may not exist.

► **Lemma 21** ( $\star$ ). *Let  $i \in \{1, 2, \dots, t\}$ . If  $\lambda(i)$  exists, then  $R(i)$  dominates every vertex in  $\{i, i+1, \dots, \lambda(i) - 1\}$  and otherwise,  $R(i)$  dominates every vertex in  $\{i, i+1, \dots, t\}$ .*

We now explain how to compute  $DS(i)$  for each  $i \in \{1, 2, \dots, t\}$ . We recursively define  $DS(i)$  as follows:

$$DS(i) = \begin{cases} \{R(i)\} \cup DS(\lambda(i)) & \text{if } \lambda(i) \text{ exists} \\ \{R(i)\} & \text{otherwise.} \end{cases}$$

► **Lemma 22** ( $\star$ ). *For each  $i \in \{1, 2, \dots, t\}$ , the set  $DS(i)$  as defined above is a minimum cardinality subset of  $B$  that dominates  $\{i, i+1, \dots, t\}$ .*

It is not difficult to verify that given an interval representation of the interval bigraph  $H$  with partite sets  $A$  and  $B$ , the parameters  $R(i)$  and  $\lambda(i)$  can be computed for each  $i \in A$  in  $O(n+m)$  time. Also, given a reflexive interval digraph  $G$ , the interval bigraph  $B_G$  can be constructed in linear time. Thus we have the following theorem.

► **Theorem 23**. *The RED-BLUE DOMINATING SET problem can be solved in interval bigraphs in linear time, given an interval representation of the bigraph as input. Consequently, the ABSORBING-SET problem can be solved in linear time in reflexive interval digraphs, given an interval representation of the input digraph.*

Note that even if an interval representation of an interval bigraph is not known, it can be computed in polynomial time using Müller's algorithm [22]. Thus given just the adjacency list of the graph as input, the RED-BLUE DOMINATING SET problem is polynomial-time solvable on interval bigraphs and the ABSORBING-SET problem is polynomial-time solvable on reflexive interval digraphs.

### 3.4 Maximum independent set

We have the following theorem due to McConnell and Spinrad [19].

► **Theorem 24** ([19]). *An independent set of maximum possible size can be computed for cocomparability graphs in  $O(n + m)$  time.*

Let  $G$  be a DUF-digraph. Let  $H$  be the underlying undirected graph of  $G$ . Then by Corollary 6, we have that  $H$  is a cocomparability graph. Note that the independent sets of  $G$  and  $H$  are exactly the same. Therefore any algorithm that finds a maximum cardinality independent set in cocomparability graphs can be used to solve the INDEPENDENT-SET problem in DUF-digraphs. Thus by the above theorem, we have the following corollary.

► **Corollary 25.** *The INDEPENDENT-SET problem can be solved for DUF-digraphs in  $O(n + m)$  time. Consequently, the INDEPENDENT-SET problem can be solved for reflexive interval digraphs in  $O(n + m)$  time.*

The above corollary generalizes and improves the  $O(mn)$  time algorithm due to Prisner's [26] observation that underlying undirected graph of interval nest digraphs are weakly triangulated and the fact that maximum cardinality independent set problem can be solved for weakly triangulated graphs in  $O(mn)$  time [14]. Note that the weighted INDEPENDENT-SET problem can also be solved in DUF-digraphs in  $O(n + m)$  time, as the problem of finding a maximum weighted independent set in a cocomparability graph can be solved in linear time [16].

## 4 Hardness results for point-point digraphs

### 4.1 Characterizations for point-point digraphs

In this section we give a characterization for point-point digraphs which will be further useful for proving our NP-completeness results for this class. Let  $G = (V, E)$  be a digraph. We say that  $a, b, c, d$  is an *anti-directed walk* of length 3 if  $a, b, c, d \in V(G)$ ,  $(a, b), (c, b), (c, d) \in E(G)$  and  $(a, d) \notin E(G)$  (the vertices  $a, b, c, d$  need not be pairwise distinct, but it follows from the definition that  $a \neq c$  and  $b \neq d$ ). Recall that  $B_G = (X, Y, E)$  is a splitting bigraph of  $G$ , where  $X = \{x_u : u \in V(G)\}$  and  $Y = \{y_u : u \in V(G)\}$  and  $x_u y_v \in E(B_G)$  if and only if  $(u, v) \in E(G)$ . We then have the following theorem.

► **Theorem 26.** *Let  $G$  be a digraph. Then the following conditions are equivalent:*

1.  $G$  is a point-point digraph.
2.  $G$  does not contain any anti-directed walk of length 3.
3. The splitting bigraph of  $G$  is a disjoint union of complete bipartite graphs.

**Proof.** (1  $\Rightarrow$  2): Let  $G$  be a point-point digraph with a point-point representation  $\{(S_u, T_u)\}_{u \in V(G)}$ . Suppose that there exist vertices  $a, b, c, d$  in  $G$  such that  $(a, b), (c, b), (c, d) \in E(G)$ . By the definition of point-point representation, we then have  $S_a = T_b = S_c = T_d$ . This implies that  $(a, d) \in E(G)$ . Therefore we can conclude that  $G$  does not contain any anti-directed walk of length 3.



(2  $\Rightarrow$  3): Suppose that  $G$  does not contain any anti-directed walk of length 3. Let  $B_G = (X, Y, E)$  be the splitting bigraph of  $G$ . Let  $x_u y_v$  be any edge in  $B_G$ , where  $u, v \in V(G)$ . Clearly, by the definition of  $B_G$ ,  $(u, v) \in E(G)$ . We claim that the graph induced in  $B_G$  by the vertices  $N(x_u) \cup N(y_v)$  is a complete bipartite graph. Suppose not. Then it should be the case that there exist two vertices  $x_a \in N(y_v)$  and  $y_b \in N(x_u)$  such that  $x_a y_b \notin E(B_G)$ , where  $a, b \in V(G)$ . By the definition of  $B_G$ , we then have that  $(a, v), (u, v), (u, b) \in E(G)$  and  $(a, b) \notin E(G)$ . So  $a, v, u, b$  is an anti-directed walk of length 3 in  $G$ , which is a contradiction to 2. This proves that for every  $p \in X$  and  $q \in Y$  such that  $pq \in E(B_G)$ , the set  $N(p) \cup N(q)$  induces a complete bipartite subgraph in  $B_G$ . Therefore, each connected component of  $B_G$  is a complete bipartite graph. (This can be seen as follows: Suppose that there is a connected component  $C$  of  $B_G$  that is not complete bipartite. Choose  $p \in X \cap C$  and  $q \in Y \cap C$  such that  $pq \notin E(B_G)$  and the distance between  $p$  and  $q$  in  $B_G$  is as small as possible. Let  $t$  be the distance between  $p$  and  $q$  in  $B_G$ . Clearly,  $t$  is odd and  $t \geq 3$ . Consider a shortest path  $p = z_0, z_1, z_2, \dots, z_t = q$  from  $p$  to  $q$  in  $B_G$ . By our choice of  $p$  and  $q$ , we have that  $z_1 z_{t-1} \in E(B_G)$ . But then  $p \in N(z_1)$ ,  $q \in N(z_{t-1})$  and  $pq \notin E(B_G)$ , contradicting our observation that  $N(z_1) \cup N(z_{t-1})$  induces a complete bipartite graph in  $B_G$ .)

(3  $\Rightarrow$  1): Suppose that  $G$  is a digraph such that the splitting bigraph  $B_G$  is a disjoint union of complete bipartite graphs, say  $H_1, H_2, \dots, H_k$ . Now we can obtain a point-point representation  $\{(S_u, T_u)\}_{u \in V(G)}$  of the digraph  $G$  as follows: For each  $i \in \{1, 2, \dots, k\}$ , define  $S_u = i$  if  $x_u \in V(H_i)$  and  $T_v = i$  if  $y_v \in V(H_i)$ . Note that  $(u, v) \in E(G)$  if and only if  $x_u y_v \in E(B_G)$  if and only if  $x_u, y_v \in V(H_i)$  for some  $i \in \{1, 2, \dots, k\}$ . Therefore we can conclude that  $(u, v) \in E(G)$  if and only if  $S_u = T_v = i$  for some  $i \in \{1, 2, \dots, k\}$ . Thus the digraph  $G$  is a point-point digraph.  $\blacktriangleleft$

► **Corollary 27.** *Point-point digraphs can be recognized in linear time.*

## 4.2 Subdivision of an irreflexive digraph

For an undirected graph  $G$ , the  $k$ -subdivision of  $G$ , where  $k \geq 1$ , is defined as the graph  $H$  having vertex set  $V(H) = V(G) \cup \bigcup_{ij \in E(G)} \{u_{ij}^1, u_{ij}^2, \dots, u_{ij}^k\}$ , obtained from  $G$  by replacing each edge  $ij \in E(G)$  by a path  $i, u_{ij}^1, u_{ij}^2, \dots, u_{ij}^k, j$ . The 0-subdivision of an undirected graph  $G$  is defined to be  $G$  itself.

The following theorem is adapted from Theorem 5 of Chlebík and Chlebíková [3].

► **Theorem 28** ([3]). *Let  $G$  be an undirected graph having  $m$  edges.*

1. *The problem of computing a maximum cardinality independent set is APX-complete when restricted to  $2k$ -subdivisions of 3-regular graphs for any fixed integer  $k \geq 0$ .*
2. *The problem of finding a minimum cardinality dominating set (resp. independent dominating set) is APX-complete when restricted to  $3k$ -subdivisions of graphs having degree at most 3 for any fixed integer  $k \geq 0$ .*

Note that the independent sets, dominating sets and independent dominating sets of an undirected graph  $G$  are exactly the independent sets, dominating sets (which are also the absorbing sets), and solutions (which are also the kernels) of the symmetric digraph of  $G$ . Clearly the symmetric digraph of  $G$  is irreflexive. Since the MAX-KERNEL problem is equivalent to the INDEPENDENT-SET problem in symmetric digraphs, we then have the following corollary of Theorem 28.

► **Corollary 29** ( $\star$ ). *The problems INDEPENDENT-SET, ABSORBING-SET, MIN-KERNEL and MAX-KERNEL are APX-complete on irreflexive symmetric digraphs of in- and out-degree at most 3.*

Note that for  $k \geq 1$ , the symmetric digraph of the  $2k$ -subdivision or  $3k$ -subdivision of an undirected graph contains an anti-directed walk of length 3 (unless the graph contains no edges), and therefore by Theorem 26, is not a point-point digraph. Thus we cannot directly deduce the APX-hardness of the problems under consideration for point-point digraphs from Theorem 28.

We define the subdivision of an irreflexive digraph, so that the techniques of Chlebík and Chlebíková can be adapted for proving hardness results on point-point digraphs.

► **Definition 30.** *Let  $G$  be an irreflexive digraph (i.e.  $G$  contains no loops). For  $k \geq 1$ , define the  $k$ -subdivision of  $G$  to be the digraph  $H$  having vertex set  $V(H) = V(G) \cup \bigcup_{(i,j) \in E(G)} \{u_{ij}^1, u_{ij}^2, \dots, u_{ij}^k\}$ , obtained from  $G$  by replacing each edge  $(i, j) \in E(G)$  by a directed path  $i, u_{ij}^1, u_{ij}^2, \dots, u_{ij}^k, j$ .*

Note that the  $k$ -subdivision of any irreflexive digraph is also an irreflexive digraph. We then have the following lemma.

► **Lemma 31.** *For any  $k \geq 1$ , the  $k$ -subdivision of any irreflexive digraph is a point-point digraph.*

**Proof.** Let  $k \geq 1$  and let  $G$  be any irreflexive digraph. By Theorem 26, it is enough to show that the  $k$ -subdivision  $H$  of  $G$  does not contain any anti-directed walk of length 3. Note that by the definition of  $k$ -subdivision, all the vertices in  $V(H) \setminus V(G)$  have both in-degree and out-degree exactly equal to one. Further, for every vertex  $v$  in  $H$  such that  $v \in V(G)$ , we have that  $N^+(v), N^-(v) \subseteq V(H) \setminus V(G)$ . Suppose for the sake of contradiction that  $u, v, w, x$  is an anti-directed walk of length 3 in  $H$ . Recall that we then have  $(u, v), (w, v), (w, x) \in E(H)$ ,  $u \neq w$  and  $v \neq x$ . By the above observations, we can then conclude that  $v \in V(G)$  and further that  $u, w \in V(H) \setminus V(G)$ . Then since  $(w, x) \in E(H)$  and  $v \neq x$ , we have that  $w$  has out-degree at least 2, which contradicts our earlier observation that every vertex in  $V(H) \setminus V(G)$  has out-degree exactly one. This proves the lemma. ◀

Now we have the following theorem.

► **Theorem 32** (★). *The problem INDEPENDENT-SET is APX-hard for point-point digraphs.*

### 4.3 Kernels

As a first step towards determining the complexity of the problems KERNEL, MIN-KERNEL and MAX-KERNEL for point-point digraphs, we show the following.

► **Lemma 33** (★). *Let  $G$  be an irreflexive digraph and let  $k \geq 1$ . Then  $G$  has a kernel if and only if the  $2k$ -subdivision of  $G$  has a kernel. Moreover,  $G$  has a kernel of size  $q$  if and only if the  $2k$ -subdivision of  $G$  has a kernel of size  $q + km$ . Further, given a kernel of size  $q + km$  of the  $2k$ -subdivision of  $G$ , we can construct a kernel of size  $q$  of  $G$  in polynomial time.*

The above lemma can be used to show the existence of a polynomial-time reduction from the KERNEL problem in general digraphs to the KERNEL problem in point-point digraphs. Thus we have the following theorem.

► **Theorem 34** (★). *The problem KERNEL is NP-complete for point-point digraphs.*

Note that KERNEL is known to be NP-complete even on planar digraphs having degree at most 3 and in- and out-degrees at most 2 [10]. The above reduction transforms the input digraph in such a way that every newly introduced vertex has in- and out-degree exactly

## 17:14 Kernels in Interval Digraphs

1 and the in- and out-degrees of the original vertices remain the same. Moreover, if the input digraph is planar, the digraph produced by the reduction is also planar. Thus we can conclude that the problem KERNEL remains NP-complete even for planar point-point digraphs having degree at most 3 and in- and out-degrees at most 2.

An *L-reduction* as defined below is an approximation-preserving reduction for optimization problems.

► **Definition 35** ([23]). *Let  $A$  and  $B$  be two optimization problems with cost functions  $c_A$  and  $c_B$  respectively. Let  $f$  be a polynomially computable function that maps the instances of problem  $A$  to the instances of problem  $B$ . Then  $f$  is said to be an *L-reduction* from  $A$  to  $B$  if there exist a polynomially computable function  $g$  and constants  $\alpha, \beta \in (0, \infty)$  such that the following conditions hold:*

1. *If  $y'$  is a solution to  $f(x)$  then  $g(y')$  is a solution to  $x$ , where  $x$  is an instance of the problem  $A$ .*
2.  *$OPT_B(f(x)) \leq \alpha OPT_A(x)$ , where  $OPT_B(f(x))$  and  $OPT_A(x)$  denote the optimum value of respective instances for the problems  $B$  and  $A$  respectively.*
3.  *$|OPT_A(x) - c_A(g(y'))| \leq \beta |OPT_B(f(x)) - c_B(y')|$ .*

In order to prove that a problem  $P$  is APX-hard, it is enough to show that the problem  $P$  has an L-reduction from an APX-hard problem. Using Lemma 33, one can construct an L-reduction from the MIN-KERNEL and MAX-KERNEL problems for irreflexive symmetric digraphs having in- and out-degree at most 3 to the MIN-KERNEL and MAX-KERNEL problems for  $2k$ -subdivisions of irreflexive symmetric digraphs having in- and out-degree at most 3. Thus, by Corollary 29, we have the following theorem.

► **Theorem 36** ( $\star$ ). *For  $k \geq 1$ , the problems MIN-KERNEL and MAX-KERNEL are APX-hard for  $2k$ -subdivisions of irreflexive symmetric digraphs having in- and out-degree at most 3. Consequently, the problems MIN-KERNEL and MAX-KERNEL are APX-hard for point-point digraphs.*

### 4.4 Minimum Absorbing set

In order to show the approximation hardness of the ABSORBING-SET problem for point-point digraphs, first we prove the following lemma.

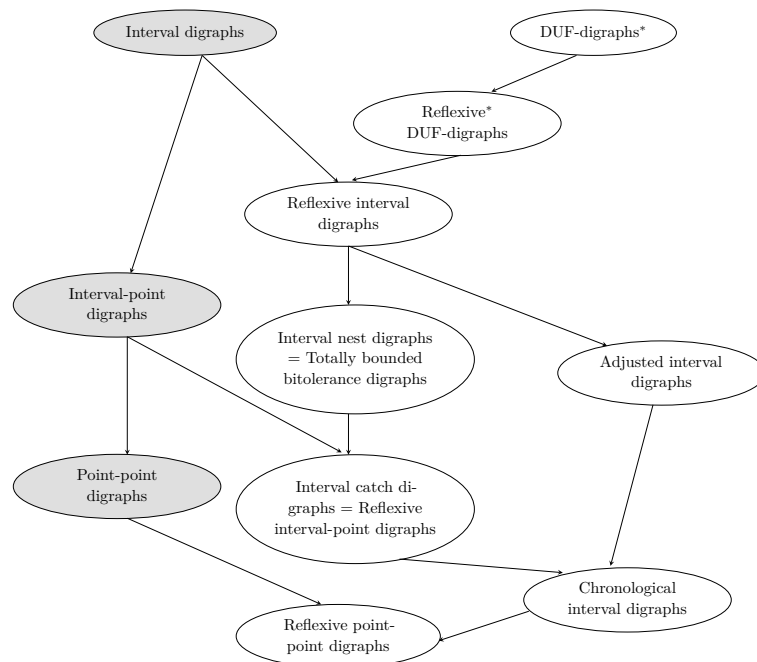
► **Lemma 37** ( $\star$ ). *Let  $G$  be an irreflexive digraph and let  $k \geq 1$ . Then  $G$  has an absorbing set of size at most  $q$  if and only if the  $2k$ -subdivision of  $G$  has an absorbing set of size at most  $q + km$ . Further, given an absorbing set of size at most  $q + km$  in the  $2k$ -subdivision of  $G$ , we can construct in polynomial time an absorbing set of size at most  $q$  in  $G$ .*

The above lemma can be used to construct an L-reduction from the ABSORBING-SET problem for irreflexive symmetric digraphs having in- and out-degree at most 3 to the ABSORBING-SET problem for  $2k$ -subdivisions of irreflexive symmetric digraphs having in- and out-degree at most 3. By Corollary 29, we then have the following theorem.

► **Theorem 38** ( $\star$ ). *For  $k \geq 1$ , the problem ABSORBING-SET is APX-hard for  $2k$ -subdivisions of irreflexive symmetric digraphs having in- and out-degree at most 3. Consequently, the problem ABSORBING-SET is APX-hard for point-point digraphs.*

## 5 Conclusion

The class of interval nest digraphs coincides with the class of totally bounded bitolerance digraphs which was introduced by Bogart and Trenk [2]. Thus totally bounded bitolerance digraphs are a subclass of reflexive interval digraphs, and all the results that we obtain for reflexive interval digraphs hold also for this class of digraphs. Figure 3 shows the inclusion relations between the classes of digraphs that were studied in this paper ( $\star$ ). After work on this paper had been completed, we have been made aware of a recent manuscript of Jaffke, Kwon and Telle [15], in which unified polynomial time algorithms have been obtained for the problems considered in this paper for some classes reflexive intersection digraphs including reflexive interval digraphs. Since their algorithms are more general in nature, their time complexities are much larger than the ones for the algorithms presented in this paper.



■ **Figure 3** Inclusion relations between graph classes. In the diagram, there is an arrow from  $\mathcal{A}$  to  $\mathcal{B}$  if and only if the class  $\mathcal{B}$  is contained in the class  $\mathcal{A}$ . Moreover, each inclusion is strict. The problems studied are efficiently solvable in the classes shown in white, while they are NP-hard and/or APX-hard in the classes shown in gray (\* the complexity of the ABSORBING-SET problem on DUF-digraphs and reflexive DUF-digraphs remain open).

Müller [22] showed the close connection between interval digraphs and interval bigraphs and used this to construct the only known polynomial time recognition algorithm for both these classes. Since this algorithm takes  $O(nm^6(n+m)\log n)$  time, the problem of finding a forbidden structure characterization for either of these classes or a faster recognition algorithm are long standing open questions in this field. But many of the subclasses of interval digraphs, like adjusted interval digraphs [29], chronological interval digraphs [5], interval catch digraphs [25], and interval point digraphs [26] have simpler and much more efficient recognition algorithms. It is quite possible that simpler and efficient algorithms for recognition exist also for reflexive interval digraphs. As for the case of interval nest digraphs, no polynomial time recognition algorithm is known. The complexities of the recognition problem and ABSORBING-SET problem for DUF-digraphs also remain as open problems.

## References


- 1 Jochen Alber, Hans L Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.
- 2 Kenneth P Bogart and Ann N Trenk. Bounded bitolerance digraphs. *Discrete Mathematics*, 215(1-3):13–20, 2000.
- 3 Miroslav Chlebík and Janka Chlebíková. The complexity of combinatorial optimization problems on d-dimensional boxes. *SIAM Journal on Discrete Mathematics*, 21(1):158–169, 2007.
- 4 Vašek Chvátal. On the computational complexity of finding a kernel. *Report CRM-300, Centre de Recherches Mathématiques, Université de Montréal*, 592, 1973.
- 5 Sandip Das, Mathew Francis, Pavol Hell, and Jing Huang. Recognition and characterization of chronological interval digraphs. *the electronic journal of combinatorics*, pages P5–P5, 2013.
- 6 Sandip Das, M Sen, AB Roy, and Douglas B West. Interval digraphs: An analogue of interval graphs. *Journal of Graph Theory*, 13(2):189–202, 1989.
- 7 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In *International Colloquium on Automata, Languages, and Programming*, pages 378–389. Springer, 2009.
- 8 Pierre Duchet. A sufficient condition for a digraph to be kernel-perfect. *Journal of graph theory*, 11(1):81–85, 1987.
- 9 Tomás Feder, Pavol Hell, Jing Huang, and Arash Rafiey. Adjusted interval digraphs. *Electronic Notes in Discrete Mathematics*, 32:83–91, 2009.
- 10 Aviezri S Fraenkel. Planar kernel and grundy with  $d \leq 3$ ,  $d_{out} \leq 2$ ,  $d_{in} \leq 2$  are np-complete. *Discrete Applied Mathematics*, 3(4):257–262, 1981.
- 11 Mathew C Francis, Pavol Hell, and Dalu Jacob. On the kernel and related problems in interval digraphs. *arXiv preprint*, 2021. [arXiv:2107.08278](https://arxiv.org/abs/2107.08278).
- 12 Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the*, 1979.
- 13 TeresaW Haynes. *Domination in graphs: volume 2: advanced topics*. Routledge, 2017.
- 14 Ryan B. Hayward, Jeremy Spinrad, and R. Sritharan. Weakly chordal graph algorithms via handles. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '00*, pages 42–49, USA, 2000. Society for Industrial and Applied Mathematics.
- 15 Lars Jaffke, O-joung Kwon, and Jan Arne Telle. Classes of intersection digraphs with good algorithmic properties. *arXiv preprint*, 2021. [arXiv:2105.01413](https://arxiv.org/abs/2105.01413).
- 16 Ekkehard Köhler and Lalla Mouatadid. A linear time algorithm to compute a maximum weighted independent set on cocomparability graphs. *Information Processing Letters*, 116(6):391–395, 2016.
- 17 Dieter Kratsch and Lorna Stewart. Domination on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 6(3):400–417, 1993.
- 18 Ching-Hao Liu, Sheung-Hung Poon, and Jin-Yong Lin. Independent dominating set problem revisited. *Theoretical Computer Science*, 562:1–22, 2015.
- 19 Ross M McConnell and Jeremy P Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.
- 20 Nimrod Megiddo and Uzi Vishkin. On finding a minimum dominating set in a tournament. *Theoretical Computer Science*, 61(2-3):307–316, 1988.
- 21 Oskar Morgenstern and John Von Neumann. *Theory of games and economic behavior*. Princeton university press, 1953.
- 22 Haiko Müller. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, 78(1-3):189–205, 1997.
- 23 Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.
- 24 Adèle Pass-Lanneau, Ayumi Igarashi, and Frédéric Meunier. Perfect graphs with polynomially computable kernels. *Discrete Applied Mathematics*, 272:69–74, 2020.

- 25 Erich Prisner. A characterization of interval catch digraphs. *Discrete mathematics*, 73(3):285–289, 1989.
- 26 Erich Prisner. Algorithms for interval catch digraphs. *Discrete Applied Mathematics*, 51(1-2):147–157, 1994.
- 27 K Brooks Reid, Alice A McRae, Sandra Mitchell Hedetniemi, and Stephen T Hedetniemi. Domination and irredundance in tournaments. *Australasian journal of combinatorics*, 29:157–172, 2004.
- 28 Moses Richardson. Solutions of irreflexive relations. *Annals of Mathematics*, pages 573–590, 1953.
- 29 Asahi Takaoka. A recognition algorithm for adjusted interval digraphs. *Discrete Applied Mathematics*, 294:253–256, 2021.
- 30 Karsten Weihe. Covering trains by stations or the power of data reduction. *Proceedings of Algorithms and Experiments, ALEX*, pages 1–8, 1998.





# Interval Query Problem on Cube-Free Median Graphs

Soh Kumabe 

The University of Tokyo, Japan

---

## Abstract

In this paper, we introduce the *interval query problem* on cube-free median graphs. Let  $G$  be a cube-free median graph and  $\mathcal{S}$  be a commutative semigroup. For each vertex  $v$  in  $G$ , we are given an element  $p(v)$  in  $\mathcal{S}$ . For each query, we are given two vertices  $u, v$  in  $G$  and asked to calculate the sum of  $p(z)$  over all vertices  $z$  belonging to a  $u - v$  shortest path. This is a common generalization of range query problems on trees and grids. In this paper, we provide an algorithm to answer each interval query in  $O(\log^2 n)$  time. The required data structure is constructed in  $O(n \log^3 n)$  time and  $O(n \log^2 n)$  space. To obtain our algorithm, we introduce a new technique, named the *staircases decomposition*, to decompose an interval of cube-free median graphs into simpler substructures.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial algorithms

**Keywords and phrases** Data Structures, Range Query Problems, Median Graphs

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.18

**Related Version** *Full Version*: <https://arxiv.org/abs/2010.05652>

**Acknowledgements** We are grateful to our supervisor Prof. Hiroshi Hirai for supporting our work. He gave us a lot of ideas to improve our paper. In particular, he simplified the proofs and helped us improve the introduction and the overall structure of this paper.

## 1 Introduction

The *range query problem* [18] is one of the most fundamental problems in the literature on data structures, particularly for string algorithms [19]. Let  $f$  be a function defined on arrays. In the range query problem, we are given an array  $P = (p(1), \dots, p(n))$  of  $n$  elements and a *range query* defined by two integers  $i, j$  with  $1 \leq i \leq j \leq n$ . For each query  $(i, j)$ , we are asked to return the value  $f((p(i), \dots, p(j)))$ . The main interest of this problem is the case where  $f$  is defined via a *semigroup operator* [27]. Let  $\mathcal{S}$  be a semigroup with operator  $\oplus$ , and let  $P$  consist of elements in  $\mathcal{S}$ . Then, the function  $f$  is defined as  $f((p(i), \dots, p(j))) = p(i) \oplus \dots \oplus p(j)$ . Typical examples of semigroup operators are sum, max, and min. The fundamental result [27, 28] is that for any constant integer  $k$ , a range query can be answered in  $O(\alpha_k(n))$  time, where  $\alpha_k$  is a slow-growing function related to the inverse of the Ackermann function. The required data structure is constructed in linear time and space. *Range minimum query problem*, i.e.,  $\oplus = \min$ , is one of the well-studied problems in the literature, and it admits a constant-time algorithm with a data structure constructed in linear time and space [1, 4, 5, 18, 20].

This problem is generalized into trees and grids. In these settings, we are given a tree/grid  $G$  and an element  $p(v)$  for each vertex of  $G$ . As a query, given two vertices  $u, v$  in  $G$ , we are asked to calculate the sum<sup>1</sup> of the elements assigned at the vertices on a  $u - v$  shortest path. In particular, we are asked to calculate the sum of the elements on the unique  $u - v$  path for trees and the axis-parallel rectangle with corners  $(u, v)$  on its diagonal for grids. For constant

---

<sup>1</sup> In this paper, for simplicity, we represent the semigroup operation by the terms of summation; that is, we denote  $a \oplus a'$  by the word *sum* of  $a$  and  $a'$  for  $a, a' \in \mathcal{S}$ .



© Soh Kumabe;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiro Sadakane; Article No. 18; pp. 18:1–18:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

dimensional grids, an almost-constant time algorithm [11] with linear space on semigroup operators and a constant-time algorithm for range minimum query is known [29]. For range query problem on trees, an almost-constant time algorithm [9] with linear space is known on semigroup operators; see [8] for further survey on the problem on trees, particularly for dynamic version.

In this paper, we introduce a common generalization of the two above mentioned cases, named *interval query problem on median graphs*. Let  $G = (V(G), E(G))$  be a connected graph with  $n$  vertices. For two vertices  $u, v \in V(G)$ , let the *interval*  $I[u, v]$  be the set of vertices belonging to a  $u - v$  shortest path, where the length of a path is defined by the number of its edges. The graph  $G$  is called a *median graph* if for all  $u, v, w \in V(G)$ ,  $I[u, v] \cap I[v, w] \cap I[w, u]$  is a singleton [2, 7, 23]. The median graph  $G$  is said to be *cube-free* if  $G$  does not contain a cube as an induced subgraph. Trees and grids are examples of cube-free median graphs. In our problem, we are given a median graph  $G$  and an element  $p(v)$  of a commutative semigroup  $\mathcal{S}$  for each vertex  $v$  of  $G$ . As a query, given two vertices  $u, v$  in  $G$ , we are asked to calculate  $p(I[u, v])$ <sup>2</sup>. The interval query problem on cube-free median graphs is a common generalization of the range query problems on trees and grids.

In this paper, we provide an algorithm to the interval query problem on cube-free median graphs. The main result here is presented as follows:

► **Theorem 1.** *There is an algorithm to answer interval queries on cube-free median graphs in  $O(\log^2 n)$  time. The required data structure is constructed in  $O(n \log^3 n)$  time and  $O(n \log^2 n)$  space, where  $n$  is the number of vertices in a given cube-free median graph.*

The time complexity of answering a query matches the complexity for the two-dimensional *range tree* [21] in the *orthogonal range query problem*, without acceleration via *fractional cascading* [10].

To obtain the algorithm, we introduce a new technique, named the *staircases decomposition*. This technique provides a new method to decompose an interval of cube-free median graphs into a constant number of smaller intervals. Most of the candidates of the smaller intervals, which we refer to as *staircases*, are well-structured, and an efficient algorithm to answer the interval queries can be constructed. The rest are not necessarily staircases; however, each of them are one of the  $O(n \log n)$  candidates, and we can precalculate all the answers of the interval queries on these intervals.

Designing fast algorithms for median graphs is a recently emerging topic. The *distance labeling scheme* [24] is a type of data structure that is defined by the encoder and decoder pair. The encoder receives a graph and assigns a label for each vertex, whereas the decoder receives two labels and computes the distance of the two vertices with these labels. For cube-free median graphs, there is a distance labeling scheme that assigns labels with  $O(\log^3 n)$  bits for each vertex [13]. Very recently, a linear-time algorithm to find the *median* of median graphs was built [6]. This paper continues with this line of research and utilizes some of the techniques presented in these previous studies.

Various applications can be considered in the interval query problem on median graphs. The solution space of a 2-SAT formula forms a median graph, where two solutions are adjacent if one of them can be obtained by negating a set of pairwise dependent variables of the other [3, 22, 26]. For two solutions  $u$  and  $v$ , the interval  $I[u, v]$  corresponds to the set of the solutions  $x$ , such that for each truth variable, if the same truth value is assigned in  $u$  and  $v$ , so does  $x$ . Suppose we can answer the interval queries to calculate sum (resp.

<sup>2</sup> For a vertex subset  $X$ , we denote the sum of  $p(z)$  over all  $z \in X$  by  $p(X)$ .

min) in polylogarithmic time with a data structure of subquadratic time and space. Then, if we have the list of all feasible solutions of the given 2-SAT formula, we can calculate the number (resp. minimum weight) of these solutions in polynomial time of the number of variables for each query, without precalculating the answers for all possible queries. Note that, there is a polynomial-delay algorithm to enumerate all solutions to the given 2-SAT formula [16]. Therefore, if the number of the feasible solutions (and thus the number of vertices in the corresponding median graph) is small, we can efficiently list them. In social choice theory, the structure of median graphs naturally arises as a generalization of *single-crossing preferences* [15, 17] and every closed Condorcet domains admits the structure of a median graph [25]. For two preferences  $u$  and  $v$ , the voters with their preferences in interval  $I[u, v]$  prefer candidate  $x$  to candidate  $y$  whenever both  $u$  and  $v$  prefer  $x$  to  $y$ . Therefore, using interval query, we can count the number of voters  $w$  such that for all pairs of candidates, at least one of  $u$  and  $v$  has the same preference order as  $w$  between these candidates. Although these structures are not necessarily cube-free, we hope that our result will be the first and important step toward obtaining fast algorithms for these problems.

## 1.1 Algorithm Overview

Here we give high-level intuition to our algorithm. More detailed outline is given in Section 3.

Let  $G$  be a cube-free median graph. The first idea for our algorithm is to decompose  $G$  recursively. We recursively divide  $V(G)$  into some parts, called *fibers*. Roughly speaking, a fiber is a set of the vertices located on the similar direction from the special vertex  $m$  (see Figure 1). Each fiber induces a cube-free median graph and, if we take  $m$  properly, has at most  $|V(G)|/2$  vertices; there are at most  $O(\log n)$  recursion steps.

Let  $u, v$  be vertices of  $G$ . Consider calculating  $p(I[u, v])$ . If  $u$  and  $v$  are in the same fiber, we calculate it recursively. Otherwise, we can show that  $I[u, v]$  intersects with only a constant number of fibers and the intersections are intervals with one end on the boundary of the fiber (Section 6, see Figure 8). Thus, it is sufficient to construct an algorithm on such intervals.

To do this, we further decompose such an interval into more well-structured intervals, using our main technique named *staircases decomposition* (Section 4, see Figure 4, 5, and 6). Roughly speaking, we decompose the interval into at most two structured substructures names *staircases* (Figure 2) and a special interval of  $O(n)$  candidates. For special intervals  $I$ , we just use the precalculated  $p(V(I))$ . For staircases  $L$ , we construct an algorithm to calculate  $p(V(L))$  in  $O(\log^2 n)$  time (Section 5), using the fact that the boundary of the fiber is actually a tree [13]. We decompose this tree into paths by heavy-light decomposition and build segment trees to answer the queries.

## 2 Basic Tools for Cube-Free Median Graphs and Trees

In this section, we introduce basic facts about cube-free median graphs and trees.

Let  $G$  be a connected, undirected, finite graph. We denote the vertex set of  $G$  by  $V(G)$ . For two vertices  $u$  and  $v$  in  $G$ , we write  $u \sim v$  if  $u$  and  $v$  are adjacent. For two vertices  $u$  and  $v$  of  $G$ , the *distance*  $d(u, v)$  between them is the minimum number of edges on a path connecting  $u$  and  $v$ , and the *interval*  $I[u, v]$  is the set of vertices  $w$  which satisfies  $d(u, v) = d(u, w) + d(w, v)$ . The graph  $G$  is a *median graph* if for any three vertices  $u, v, w$ ,  $I[u, v] \cap I[v, w] \cap I[w, u]$  contains exactly one vertex, called *median* of  $u, v$  and  $w$ . Median graphs are bipartite and do not contain  $K_{2,3}$  as a subgraph. A median graph is *cube-free* if it does not contain a (three-dimensional) cube graph as an induced subgraph. The followings hold.

## 18:4 Interval Query Problem on Cube-Free Median Graphs

► **Lemma 2** ([14]). *Any interval in a cube-free median graph induces an isometric subgraph of a two-dimensional grid.*

► **Lemma 3** ([13]). *Let  $u, v, w_1, w_2$  be four pairwise distinct vertices of a median graph such that  $v \sim w_1, v \sim w_2$  and  $d(u, v) - 1 = d(u, w_1) = d(u, w_2)$ . Then, there is unique vertex  $z$  with  $w_1 \sim z, w_2 \sim z$  and  $d(u, z) = d(u, v) - 2$ .*

From now on, let  $G$  be a cube-free median graph with  $n$  vertices. Let  $X$  be a subset of  $V(G)$ . For vertex  $z \in V(G)$  and  $x \in X$ ,  $x$  is the *gate* of  $z$  in  $X$  if for all  $w \in X$ ,  $x \in I[z, w]$ . The gate of  $z$  in  $X$  is unique (if it exists) because it is the unique vertex in  $X$  that minimizes the distance from  $z$ .  $X$  is *gated* if all vertices  $z \in V(G)$  have a gate in  $X$ . The following equivalence result is known.

► **Lemma 4** ([12, 13]). *Let  $X$  be a vertex subset of the median graph  $G$ . Then, following three conditions are equivalent.*

- (a)  $X$  is gated.
- (b)  $X$  is convex, i.e.,  $I[u, v] \subseteq X$  for all  $u, v \in X$ .
- (c)  $X$  induces a connected subgraph and  $X$  is locally convex, i.e.,  $I[u, v] \subseteq X$  for all  $u, v \in X$  with  $d(u, v) = 2$ .

An induced subgraph of  $G$  is *gated* (resp. *convex*, *locally convex*) if its vertex set is gated (resp. convex, locally convex). The intersection of two convex subsets is convex. Any interval of median graphs are convex.

For a convex subset  $X$  and a vertex  $x \in X$ , the *fiber*  $F_X(x)$  of  $x$  with respect to  $X$  is the set of vertices in  $G$  whose gate in  $X$  is  $x$ . Two fibers  $F_X(x), F_X(y)$  are *neighboring* if there are vertices  $x' \in F_X(x)$  and  $y' \in F_X(y)$  such that  $x' \sim y'$ , which is equivalent to  $x \sim y$  [13]. Fibers for all  $x \in X$  define a partition of  $V(G)$ . For two adjacent vertices  $x, y \in X$ , the *boundary*  $T_X(x, y)$  of  $F_X(x)$  relative to  $F_X(y)$  is the set of the vertices which have a neighbor in  $F_X(y)$ .  $T_X(x, y)$  and  $T_X(y, x)$  are isomorphic. A vertex in  $T_X(x, y)$  has a unique neighbor in  $T_X(y, x)$ , which is the corresponding vertex under that isomorphism. For vertex  $x \in X$ , a *total boundary*  $T_X(x)$  of  $F_X(x)$  is the union of all  $T_X(x, y)$  for  $y \in X$  with  $x \sim y$ . The subgraph  $H$  is *isometric* in  $G$  if for all  $u, v \in V(H)$ , there is a path in  $H$  with length  $d(u, v)$ . A rooted tree has *gated branches* if any of its root-leaf path is convex. The next lemma exploits the structures of the boundaries of fibers of cube-free median graphs.

► **Lemma 5.** *Let  $X$  be a convex vertex subset of cube-free median graph  $G$ . Let  $x, y \in X$  and assume  $x \sim y$ . Then, the followings hold.*

- (i) ([13])  $T_X(x, y)$  induces a tree, which is convex.
- (ii) ([13])  $T_X(x)$  induces a tree with gated branches, which is isometric in  $G$ .

The following is folklore in a literature of median graphs. A proof is in full version.

► **Lemma 6** (folklore). *Let  $X$  be a convex vertex set of a median graph and let  $Y$  be a convex subset of  $X$ . For  $x \in X$ , let  $F(x)$  be the fiber of  $x$  with respect to  $X$ . Then,  $\bigcup_{y \in Y} F(y)$  is convex.*

Let  $T$  be a tree with gated branches. For a vertex  $v \in V(G)$  and  $w \in T$ ,  $w$  is an *imprint* of  $v$  if  $I[v, w] \cap T = \{w\}$ . If  $T$  is convex, the imprint is equal to the gate and therefore unique. Even if it is not the case, we can state following.

► **Lemma 7.** *Let  $T$  be a tree with gated branches rooted at  $r$ . Let  $u \in V(G)$ . Then, the following statements hold.*

- (i) ([13]) There are at most two imprints of  $u$  in  $T$ .
- (ii) Assume  $u$  has two distinct imprints  $w^1, w^2$  in  $T$ . Then,  $w^1, w^2 \in I[r, u]$ .

**Proof.** We prove (ii). From symmetry, we only prove  $w^1 \in I[r, u]$ . Let  $P_1$  be the root-leaf path of  $T$  that contains  $w^1$ . Then,  $P_1$  is convex and therefore  $d(r, u) = d(r, w^1) + d(w^1, u)$ . ◀

► **Lemma 8.** *Let  $T$  be a tree with gated branches and  $w \in V(T)$ . Then, the set of vertices with an imprint  $w$  in  $T$  is convex.*

**Proof.** Assume the contrary. Then, there are distinct vertices  $z_1, z_2, z_3$  with  $z_1 \sim z_2 \sim z_3$ , such that  $z_1$  and  $z_3$  have an imprint  $w$  but  $z_2$  doesn't. We have  $d(w, z_2) = d(w, z_1) + 1$ ; otherwise,  $d(w, z_2) = d(w, z_1) - 1$  because of bipartiteness of  $G$  holds and in this case,  $I[w, z_2] \subseteq I[w, z_1]$  holds and  $z_2$  has an imprint  $w$ . By the same reason we have  $d(w, z_2) = d(w, z_3) + 1$ . From definition of the imprint, there is a  $z_2 - w$  shortest path that contains a vertex of  $T$  other than  $w$ . Let  $z_4$  be the neighbor of  $z_2$  in this shortest path. Then,  $z_4$  does not have an imprint  $w$  and especially,  $z_1 \neq z_4 \neq z_3$ . Now we have  $d(w, z_4) + 1 = d(w, z_1) + 1 = d(w, z_3) + 1 = d(w, z_2)$  and obtain three squares that all two intersect at an edge from Lemma 3, which contradicts Lemma 2. ◀

For a vertex  $m \in V(G)$ , the *star*  $\text{St}(m)$  of  $m$  is the set of vertices  $x \in V(G)$  such that there is an edge or a square that contains both  $m$  and  $x$ .  $\text{St}(m)$  is convex. The vertex  $m \in V(G)$  is a *median* of  $G$  if it minimizes the sum of distances to all vertices in  $G$ . The following holds.

► **Lemma 9** ([13]). *All the fibers of  $\text{St}(m)$  of a median graph contains at most  $\frac{n}{2}$  vertices.*

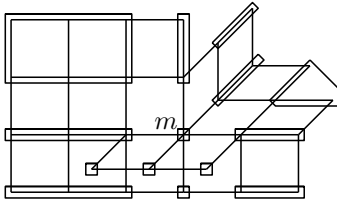
For a rooted tree  $T$  that is rooted at  $r$ , a vertex  $u \in V(T)$  is an *ancestor* of  $v$  and  $v$  is a *descendant* of  $u$  if there is a path from  $u$  to  $v$ , only going toward the leaves. The vertex subset  $X$  is a *column* of  $T$  if for any two vertices  $x, y$  in  $X$ ,  $x$  is either an ancestor or a descendant of  $y$ . The vertex  $t$  is the *lowest common ancestor* [20] of  $u$  and  $v$  if  $t$  is an ancestor of both  $u$  and  $v$  that minimizes the distance between  $u$  and  $t$  (or equivalently,  $v$  and  $t$ ) in  $T$ . There is a data structure that is constructed in linear time and space such that, given two vertices on  $T$ , it returns the lowest common ancestor of them in constant time [5].  $u$  is a *parent* of  $v$  and  $v$  is a *child* of  $u$  if  $u$  is an ancestor of  $v$  and  $u \sim v$ . Let  $X \subseteq V(T)$  and  $u \in V(T)$ . The *nearest ancestor* of  $u$  in  $X$  on  $T$  is the vertex  $v \in X$  such that  $v$  is an ancestor of  $u$  and minimizes  $d(u, v)$ .

Let  $T$  be a rooted tree rooted at  $r$ . For a vertex  $v \in V(T)$ , let  $T_v$  be the subtree of  $T$  rooted at  $v$ . An edge  $(u, v)$  in  $G$  such that  $u$  is the parent of  $v$  is a *heavy-edge* if  $|V(T_u)| \leq 2|V(T_v)|$  and a *light-edge* otherwise. Each vertex has at most one child such that the edge between them is a heavy-edge. The *heavy-path* is a maximal path that only contains heavy-edges. The *heavy-light decomposition* is the decomposition of  $T$  into heavy-paths. Note that, there is at most  $O(\log n)$  light-edges on any root-leaf path on  $T$ .

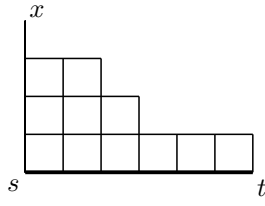
### 3 Outline and Organization

Here we roughly describe our algorithm using the notions in Section 2. Let  $G$  be a cube-free median graph. Let  $m$  be a median of  $G$ ,  $\text{St}(m)$  be the star of  $m$ , and for each  $x \in \text{St}(m)$ , let  $F(x)$  be the fiber of  $x$  in  $\text{St}(m)$  (see Figure 1). Let  $u, v$  be vertices of  $G$ .

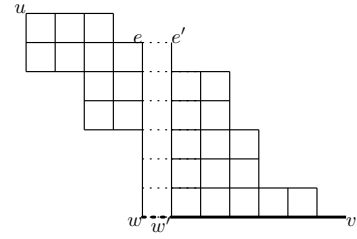
Consider calculating  $p(I[u, v])$ . If  $u$  and  $v$  are in the same fiber  $F(x)$  of  $\text{St}(m)$ , we calculate the answer by using the algorithm on  $F(x)$ , which is recursively defined. Lemma 9 ensures that the recursion depth is at most  $O(\log n)$ . Otherwise, we can show that  $I[u, v]$  intersects with only a constant number of fibers, and for each fiber  $F(x)$  that intersects  $I[u, v]$ ,  $I[u, v] \cap F(x)$  can be represented as  $I[u_x, v_x]$  for some vertices  $u_x, v_x \in F(x)$  such that  $v_x$  is on the total boundary of  $F(x)$ . Thus, it is sufficient to construct an algorithm to answer the query on the interval, such that one of the ends is on the total boundary of  $F(x)$ .



■ **Figure 1** A median graph and its decomposition into fibers of  $\text{St}(m)$ .



■ **Figure 2** Staircases with top  $x$  with base starts at  $s$  and ends at  $t$ .



■ **Figure 3** Decomposition of  $I[u, v]$  into an interval  $I[u, w]$  and staircases  $I[e', v]$ . The bold line represents  $P$ .

To do this, we introduce a technique to decompose intervals, which we name the *staircases decomposition*. Let  $T$  be a tree with gated branches and assume  $u \in V(G)$  and  $v \in V(T)$ . We partition an interval  $I[u, v]$  into an interval  $I$  and at most two special structures, which we name a *staircases* (Figure 2), which we describe in Section 4. Such a decomposition can be calculated in  $O(\log n)$  time with appropriate preprocessing. Here, we can take  $I$  as one of the  $O(n)$  candidates of intervals. We just precalculate and store the value  $p(I)$  for each candidate, and recall it when we answer the queries.

Now we just need an algorithm to calculate the value  $p(V(L))$  quickly for a staircases  $L$ . Let  $P$  be a root-leaf path of  $T$ . The segment trees can answer the staircases queries whose base is a subpath of  $P$  in  $O(\log n)$  time. To answer the general queries, we use a heavy-light decomposition of  $T$ .

The rest of the paper is organized as follows. In Section 4, we introduce the staircases decomposition of the intervals with one end on the tree with gated branches. In Section 5, we construct an algorithm and a data structure for the interval queries for the same cases. In Section 6, we prove that we can decompose a given interval into constant number of intervals with one of the ends on the total boundaries of the fibers of  $\text{St}(m)$ . This technique can also be applied to the query that asks the median of given three vertices. Some detailed parts in these sections are found in full version. An algorithm to construct our data structure efficiently is given in full version.

## 4 The Staircases Decomposition of the Intervals with One End on the Boundary

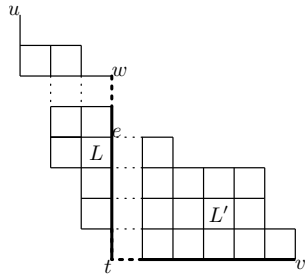
Let  $T$  be a tree with gated branches. In this section, we introduce a technique, *staircases decomposition*, to decompose an interval  $I[u, v]$  such that  $v$  is on  $T$ .

Let  $P = (s = w_0, \dots, w_k = t)$  be a convex path. For a vertex  $x$  with gate  $s$  in  $P$ , the interval  $I[x, t]$  induces *staircases* if for all  $i = 0, \dots, k$ , the set of vertices in  $I[x, t]$  with gate  $w_i$  in  $P$  induces a path.  $P$  is the *base* of  $L$  and the vertex  $x$  is the *top* of  $L$ . The base *starts* at  $s$  and *ends* at  $t$  (see Figure 2). Our staircases decomposition decomposes  $I[u, v]$  into an interval and at most two staircases such that their bases are columns of  $T$ .

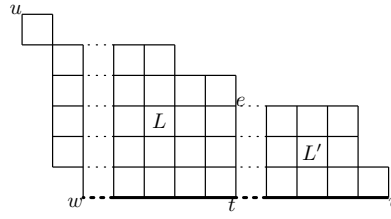
### 4.1 The case with One End on a Convex Path

Here we investigate the structure of an interval such that one of the endpoints is on a convex path  $P$ . Consider an interval  $I[u, v]$  such that  $v$  is on  $P$ . Let  $w$  be the gate of  $u$  in  $P$ . The purpose here is to prove that  $I[u, v]$  can be decomposed into the disjoint union of an interval  $I[u, w]$  and a staircases (see Figure 3), if  $w \neq v$ . We assume  $w \neq v$  because otherwise we

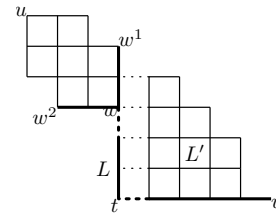




■ **Figure 4** Staircases decomposition of  $I[u, v]$  (single imprint, first case).



■ **Figure 5** Staircases decomposition of  $I[u, v]$  (single imprint, second case).



■ **Figure 6** Staircases decomposition of  $I[u, v]$  (double imprints).

have no need of decomposition. Let  $w'$  be the neighbor of  $w$  in  $P$  between  $w$  and  $v$ . We take the isometric embedding of  $I[u, v]$  into a two-dimensional grid (see Lemma 2). We naively introduce a  $xy$ -coordinate system with  $w = (0, 0)$ ,  $w' = (1, 0)$ ,  $u = (x_u, y_u)$  with  $y_u \geq 0$ , and  $v = (x_v, y_v)$  with  $y_v \leq 0$ . Now, we can state the following.

► **Lemma 10.** *If a vertex  $z = (x_z, y_z)$  on  $I[u, v] \cap V(P)$  is not on the  $x$ -axis, there is no vertex other than  $z$  in  $I[u, v]$  with gate  $z$  in  $P$ .*

**Proof.** Assume the contrary and let  $z' = (x_{z'}, y_{z'})$  be a vertex in  $I[u, v]$  with gate  $z$  in  $P$ . Because of the isometricity,  $x_z > 0$  and  $y_z < 0$  holds. Since  $z \in I[w, z']$ , we have  $x_{z'} \geq x_z$ . Since  $z' \in I[u, v]$ ,  $x_v \geq x_{z'}$  holds. Therefore, we can take a vertex  $z''$  in  $P$  with  $x$ -coordinate  $x_{z'}$ , but it means  $z' \in I[w, z'']$  and contradicts to the convexity of  $P$ . ◀

Since such  $z$  does not affect the possibility of decomposition (we can just add such vertices at the end of the staircases), we can assume that  $v = (x_v, 0)$  for  $x_v > 0$ . Moreover, from convexity, we have that all vertices in  $I[u, v]$  have non-negative  $y$ -coordinate. Thus,  $I[u, v] \setminus I[u, w]$  is the set of vertices with positive  $x$ -coordinate and forms staircases (see Figure 3), which is the desired result.

To build an algorithm to calculate  $p(I[u, v])$  as the sum of  $p(I[u, w])$  and  $p(I[u, v] \setminus I[u, w])$ , we should identify the top  $e'$  of the staircases. Instead of direct identification, we rather identify the unique neighbor of it in  $I[u, w]$ , named the *entrance*  $e$  of the staircases: The top  $e'$  can be determined as the neighbor of  $e$  with gate  $w'$  on  $P$ . Here, we have that  $e$  is the gate of  $u$  in the boundary of  $F(w)$  with respect to  $F(w')$ , where  $F(w)$  (resp.  $F(w')$ ) is the fiber of  $w$  (resp.  $w'$ ) with respect to  $P$ . Indeed, this gate should be in  $I[u, w]$  from the definition of the gate and  $e$  is the only candidate for it. We can calculate  $e$  in  $O(\log n)$  time by working on the appropriate data structure on total boundary of the fiber of  $w$  with respect to  $P$ . We discuss this algorithm in full version.

## 4.2 Single Imprint

Here we give the staircases decomposition of the interval  $I[u, v]$ , where  $v$  is on a tree  $T$  with gated branches, rooted at  $r$ . First, we treat the case that there is exactly one imprint  $w$  of  $u$  in  $T$  in  $I[u, v]$ . Let  $t$  be a lowest common ancestor of  $w$  and  $v$  in  $T$ . Note that,  $t$  might coincide with  $w$  or  $v$ . Let  $P$  (resp.  $P'$ ) be the root-leaf path of  $T$  that contains  $w$  (resp.  $v$ ).

Since  $P'$  is convex, we can decompose  $I[u, v]$  into a staircases  $L'$  with base on  $P'$  and an interval  $I[u, t]$ . Since  $P$  is convex, we can further decompose the interval  $I[u, t]$  into a staircases  $L$  with base on  $P$  and an interval  $I[u, w]$ . Now, for fixed  $T$ ,  $I[u, w]$  is one of the  $O(n)$  candidates of the intervals, because it is specified only by a vertex  $u$  and one of at most two imprints of  $u$  on  $T$ . This is the staircases decomposition we obtain here.



To bound the size of the data structure we construct in Section 5, we should ensure that staircases  $L$  and  $L'$  contains only vertices with an imprint on  $P$  and  $P'$ , respectively. Let  $B_L$  (resp.  $B_{L'}$ ) be the base of  $L$  (resp.  $L'$ ). We prove the following.

► **Lemma 11.** *The following statements hold.*

- (i)  $I[u, v]$  contains no vertices in  $T$  other than the vertices on the  $w - v$  path on  $T$ .
- (ii) For a vertex  $z$  in  $L'$ , the gate of  $z$  in  $P'$  is an imprint of  $z$  in  $T$ .
- (iii) For a vertex  $z$  in  $L$ , the gate of  $z$  in  $P$  is an imprint of  $z$  in  $T$ .

**Proof.** (i) Let  $z \in I[u, v] \cap V(T)$ . Then,  $d(u, v) = d(u, z) + d(z, v)$  holds. Since  $w$  is the unique imprint of  $u$  in  $T$ ,  $d(u, z) = d(u, w) + d(w, z)$  and  $d(u, v) = d(u, w) + d(w, v)$  holds. Therefore  $d(w, v) = d(w, z) + d(z, v)$  and it means  $z$  is on the unique path between  $w$  and  $v$  on  $T$ . (ii) Let  $w'_z$  be the gate of  $z$  in  $P'$ . We prove  $I[z, w'_z] \cap V(T) = \{w'_z\}$ . Assume  $x \in (I[z, w'_z] \cap V(T)) \setminus \{w'_z\}$ . From (i),  $x$  is on  $w - t$  path. From isometricity of  $T$ ,  $t \in I[x, w'_z] \subseteq I[z, w'_z]$  holds and it contradicts the definition of  $w'_z$ . (iii) Similar to (ii). ◀

We should also make algorithms to identify the top of the staircases  $L$  and  $L'$ . The top of  $L$  can be found by applying the discussion in previous subsection by precalculating the entrances for all possible patterns of  $u$  and  $w$ , because the start of the base of  $L$  is uniquely determined as a parent of  $w$ , independent of  $v$ . However, we cannot apply it to find the top of  $L'$ , because the start of the base of  $L'$  is a child of  $t$ , not a parent. Instead, we calculate the top of  $L'$  by case-analysis of the positional relation of the staircases. Intuitively, we divide cases by the angle formed by  $B_L$  and  $B_{L'}$ . We have essentially two cases<sup>3</sup> to tract, which this angle is  $\pi/2$  (Figure 4) or  $\pi$  (Figure 5) (we formally define these cases and prove that they cover all cases in full version). In the case in Figure 4, the entrance  $e$  of  $L'$  can be found on  $B_L$ . In the case in Figure 5,  $e$  can be found on the total boundary of the vertex set with imprint  $t$ . In both case, by appropriate data structure given in full version, we can find the entrance in  $O(\log n)$  time.

### 4.3 Double Imprints

Here we consider the staircases decomposition for the case that there are two imprints  $w^1, w^2$  of  $u$  in  $T$  in  $I[u, v]$ . Let  $w$  be the lowest common ancestor of  $w^1$  and  $w^2$  in  $T$ . From (ii) of Lemma 7 and isometricity of  $T$ ,  $d(u, w^1) + d(w^1, w) = d(u, r) - d(w, r) = d(u, w^2) + d(w^2, w)$  holds and particularly we have  $w^1, w^2 \in I[u, w]$ . From isometricity of  $T$ , we have  $w \in I[w_1, w_2] \subseteq I[u, v]$ . Let  $t$  be the lowest common ancestor of  $w$  and  $v$ . Then, from isometricity of  $T$ , we have  $t \in I[w, v] \subseteq I[u, v]$ . Note that, the lowest common ancestor of  $v$  and  $w^1$  (resp.  $w^2$ ) is also  $t$ , because otherwise we have  $w \notin I[u, v]$ . Let  $P$  (resp.  $P'$ ) be any root-leaf path of  $T$  that contains  $w$  (resp.  $v$ ).

Since  $P'$  is convex, we can decompose  $I[u, v]$  into a staircases  $L'$  with base on  $P'$  and an interval  $I[u, t]$ . Since the subpath of  $P$  between  $r$  and  $w$  is convex, we can further decompose the interval  $I[u, t]$  into a staircases  $L$  with base on  $P$  and an interval  $I[u, w]$  (actually, we can prove that  $L$  is a line). Now, for fixed  $T$ ,  $I[u, w]$  is one of the  $O(n)$  candidates of the intervals, because  $w$  is specified only by a vertex  $u$ , as the lowest common ancestor of two imprints of  $u$  in  $T$ . This is the staircases decomposition we obtain here.

Let  $B_L$  (resp.  $B_{L'}$ ) be the base of  $L$  (resp.  $L'$ ). From the same reason as the case with a single imprint, we prove the following lemma. The proof is similar to the proof of Lemma 11.

<sup>3</sup> To explain all cases by these two, we take  $T$  as the maximal tree with gated branches that contains the fiber we consider, rather than the fiber itself.

► **Lemma 12.** *The following statements hold.*

- (i)  $I[u, v]$  contains no vertices in  $T$  other than vertices in  $w^1 - v$  and  $w^2 - v$  path on  $T$ .
- (ii) For a vertex  $z$  in  $L'$ , the gate of  $z$  in  $P'$  is an imprint of  $z$  in  $T$ .
- (iii) For a vertex  $z$  in  $L$ , the gate of  $z$  in  $P$  is an imprint of  $z$  in  $T$ .

**Proof.** (i) Let  $z \in I[u, v] \cap V(T)$ . Then,  $d(u, v) = d(u, z) + d(z, v)$  holds. Let  $w^i$  be the imprint of  $u$  in  $T$  with  $d(u, z) = d(u, w^i) + d(w^i, z)$ . Then,  $d(u, v) = d(u, w^i) + d(w^i, v)$  holds. Therefore  $d(w^i, v) = d(w^i, z) + d(z, v)$  and it means  $z$  is on the unique path between  $w^i$  and  $v$  on  $T$ . (ii) Let  $w'_z$  be the gate of  $z$  in  $P'$ . We prove  $I[z, w'_z] \cap V(T) = \{w'_z\}$ . Assume  $x \in (I[z, w'_z] \cap V(T)) \setminus \{w'_z\}$ . From (i),  $x$  is on  $t - w^1$  or  $t - w^2$  path. From isometricity of  $T$ ,  $t \in I[x, w'_z] \subseteq I[z, w'_z]$  holds and it contradicts the definition of  $w'_z$ . (iii) Similar to (ii). ◀

We should also provide a way to identify the top of the staircases  $L'$ . We have only one case to tract, shown in Figure 6, which we can find the entrance on  $w^1 - t$  or  $w^2 - t$  path on  $T$  (we formally define the case in full version). We can find it in  $O(\log n)$  time in the algorithm in full version.

## 5 Query Processing of the Case with One End on the Tree with Gated Branches

In this section, we construct an algorithm and a data structure that answers the queries with one of the endpoints on the tree with gated branches. That part is the core of our algorithm.

### 5.1 Query Processing for Maximal Staircases with Base on Convex Path

Here we construct an algorithm and a data structure for the staircases whose base is contained in a convex path  $P$ . For simplicity, we assume that  $P$  contains  $2^q$  vertices for some integer  $q$ . We do not lose generality by this restriction because we can safely attach dummy vertices at the end of  $P$ . Let  $P = (w_0, \dots, w_{2^q-1})$ . Our data structure uses a segment tree defined on  $P$ . The information of the vertices with base  $w_i$  in  $P$  are stored by linking to  $w_i$ .

It is convenient to consider the *direction* of  $P$ , as if  $P$  is directed from  $w_0$  to  $w_{2^q-1}$ . The *reverse*  $\bar{P}$  of  $P$  is the same path as  $P$  as an undirected path but has different direction, i.e.,  $\bar{P} = (w_{2^q-1}, \dots, w_0)$ . We represent the path between  $w_x$  and  $w_y$  on  $P$  by  $P[x, y]$ .

Let us formally define the queries to answer here. A query is represented by three vertices  $x, w_a, w_b$  such that the gate of  $x$  on  $P$  is  $w_a$ , and asks to answer the value  $p(L(x, w_a, w_b))$ , where  $L(x, w_a, w_b)$  represents the staircases with top  $x$  and base starts at  $w_a$  and ends at  $w_b$ . We construct two data structures, the first one treats the case  $a \leq b$  and the second one treats the case  $a > b$ . The second data structure is just obtained by building the first data structure on the reverse of  $P$ , therefore we can assume that for all queries,  $w_a \leq w_b$  holds.

For  $i = 0, \dots, 2^q - 1$ , let  $F_i$  be the fiber of  $w_i$  with respect to  $P$ . For  $i = 0, \dots, 2^q - 2$  and  $z \in F_i$ , the *successor*  $\text{succ}_P(z)$  of  $z$  is the gate of  $z$  in  $F_{i+1}$  (see Figure 7). Intuitively,  $\text{succ}_P(z)$  represents the next step of  $z$  in the staircases with base in  $P$ ; more precisely, for  $a < i < b$ , if  $F_i \cap V(L(x, w_a, w_b))$  induces  $z - w_i$  path,  $F_{i+1} \cap V(L(x, w_a, w_b))$  induces  $\text{succ}_P(z) - w_{i+1}$  path.

Here we construct a complete binary tree, which is referred to as *segment tree*, to answer the queries. For each  $d = 0, \dots, q$  and for each  $i = 0, 1, \dots, 2^{q-d} - 1$ , we prepare a node that corresponds to  $P[i \times 2^d, (i+1) \times 2^d - 1]$ . For each node  $v$  that corresponds to  $P[l, r]$  and for each  $z \in F_l$ , we store the vertex  $s(z, l, r) = \text{succ}_P^{r-l}(z)$  and the value  $S(z, l, r) = p(L(z, w_l, w_r)) = p(I[\text{succ}_P^0(z), w_l]) \oplus \dots \oplus p(I[\text{succ}_P^{r-l}(z), w_r])$ , where the  $\text{succ}_P^k(z)$  is recursively defined by  $\text{succ}_P^0(z) = z$  and  $\text{succ}_P^{k+1}(z) = \text{succ}_P(\text{succ}_P^k(z))$  for all  $0 \leq k$ .

## 18:10 Interval Query Problem on Cube-Free Median Graphs

The Algorithm 1 calculates  $p(L(x, w_a, w_b))$ . We call the procedure  $\text{StaircasesQuery}_P(0, 2^q - 1, a, b, x)$  to calculate it, and the algorithm returns the pair of the vertex  $\text{succ}_P^{b-a+1}(x)$  and the value  $p(L(x, w_a, w_b))$ . The time complexity is  $O(q) = O(\log n)$ .

■ **Algorithm 1**  $\text{StaircasesQuery}_P(l, r, a, b, x)$ .

---

```

1: if  $[l, r] \subseteq [a, b]$  then
2:   return  $(s(x, l, r), S(x, l, r))$ 
3:  $med \leftarrow \lfloor \frac{l+r}{2} \rfloor$ 
4: if  $b \leq med$  then
5:   return  $\text{StaircasesQuery}_P(l, med, a, b, x)$ 
6: if  $med < a$  then
7:   return  $\text{StaircasesQuery}_P(med + 1, r, a, b, x)$ 
8:  $(x', S_1) \leftarrow \text{StaircasesQuery}_P(l, med, a, b, x)$ 
9:  $(x'', S_2) \leftarrow \text{StaircasesQuery}_P(med + 1, r, a, b, \text{succ}_P(x'))$ 
10: return  $(x'', S_1 \oplus S_2)$ 

```

---

This data structure is constructed as in Algorithm 2. The correctness is clear and the time complexity is  $O(nq) \leq O(n \log n)$ , assuming that we know the vertex  $\text{succ}_P(x)$  and the value  $p(I[x, w_i])$  for all  $i = 0, \dots, 2^q - 1$  and  $x \in F_i$ . The size of the data structure is clearly  $O(nq) \leq O(n \log n)$ . We give algorithms to calculate  $\text{succ}_P(x)$  and  $p(I[x, w_i])$  in full version.

■ **Algorithm 2** Construction of the Data Structure for Staircases with Base on Convex Path.

**Input:** A cube-free median graph  $G$ , a convex path  $P = (w_0, \dots, w_{2^q-1})$

---

```

1: for  $i = 0, \dots, 2^q - 1$  do
2:   for all  $x \in F_i$  do
3:      $s(x, i, i) \leftarrow x$ 
4:      $S(x, i, i) \leftarrow p(L(x, w_i, w_i)) = p(I[x, w_i])$ 
5:   for  $d = q - 1, \dots, 0$  do
6:     for  $i = 0, \dots, 2^{q-d} - 1$  do
7:        $a \leftarrow i \times 2^d, b \leftarrow (i + \frac{1}{2}) \times 2^d, c \leftarrow (i + 1) \times 2^d$ 
8:       for all  $x \in F_i$  do
9:          $s(x, a, c - 1) \leftarrow s(\text{succ}_P(s, a, b - 1), b, c - 1)$ 
10:         $S(x, a, c - 1) \leftarrow S(x, a, b - 1) \oplus S(\text{succ}_P(s(x, a, b - 1)), b, c - 1)$ 

```

---

## 5.2 Query Processing for Staircases with Base on the Tree with Gated Branches

Let  $T$  be a tree with gated branches. Here we construct an algorithm and a data structure for the staircases whose base is a column of  $T$ . The simplest idea is to prepare the data structure discussed in the previous subsection for all root-leaf paths on  $T$ , but in this case the total size of the data structure can be as bad as  $O(n^2 \log n)$ . To reduce the size, we instead prepare the above data structure on every heavy-path of heavy-light decomposition of  $T$ .

For a vertex  $w \in V(T)$ , let  $F(w)$  be the set of vertices with an imprint  $w$ . For an edge  $(w, w')$  of  $T$  and a vertex  $z \in F(w)$ , we denote  $\text{succ}_{w, w'}(z)$  by the gate of  $z$  in  $F(w')$ . For the staircases  $L$  whose base starts at  $w_1$  and ends at  $w_2$  such that  $w_1, w, w', w_2$  are located on some column of  $T$  in this order, if  $F(w) \cap V(L)$  induces  $z - w$  path,  $F(w') \cap V(L)$  is  $\text{succ}_{w, w'}(z) - w'$  path.



## 18:12 Interval Query Problem on Cube-Free Median Graphs

Now we consider calculating the answer as  $p(L) + p(L') + p(I)$ .  $p(L)$  and  $p(L')$  can be calculated in  $O(\log^2 n)$  time by above algorithm. Furthermore,  $p(I)$  is precalculated in the construction of our data structure and we can take this value in constant time. Therefore we can answer the interval query in the case with one end on the tree with gated branches in  $O(\log^2 n)$  time. We summarize our algorithm in full version.

Here we describe how  $p(I)$  can be precalculated. Recall that, we construct our data structure recursively on each fibers. Therefore, after constructing the smaller data structure on each fiber, we can calculate the value  $p(I)$  in  $O(\log^2 n)$  time by using an interval query on them to complete construction. This is the bottleneck part of our construction algorithm, along with  $O(\log n)$  recursion steps. Note that, this procedure can be implemented during preprocessing because there are only  $O(n)$  candidates of  $I$ . When answering to the queries, we do not need to use the smaller data structure; we have only to refer these precalculated values.

### 6 Decomposing Intervals into intervals with One End on the Boundary

In this section, we consider decomposing an interval with both ends in different fibers into smaller intervals with one end on boundaries (see Figure 8). Specifically, we bound the number of such fibers by 9. Let  $m$  be the median of  $G$ . For  $x \in \text{St}(m)$ , let  $F(x)$  be the fiber of  $x$  with respect to  $\text{St}(m)$ . For  $v \in V(G)$ , let  $r(v)$  be the vertex in  $\text{St}(m)$  that is nearest from  $v$ . From definition of fibers,  $v \in F(r(v))$  holds.

First, we prove that the intersection of an interval and a fiber is indeed an interval. The following lemma holds.

► **Lemma 13.** *Let  $u, v$  be vertices and let  $x \in \text{St}(m)$ . Let  $g_u, g_v$  be the gate of  $u, v$  in  $F(x)$ , respectively. Then,  $I[u, v] \cap F(x)$  coincides with  $I[g_u, g_v]$  if it is nonempty.*

**Proof.** Assume  $z \in I[u, v] \cap F(x)$ . From the definition of the gate, there is a  $u - z$  (resp.  $v - z$ ) shortest path that passes through  $g_u$  (resp.  $g_v$ ). Therefore there is a  $u - v$  shortest path that passes through  $u, g_u, z, g_v, v$  in this order, which means  $z \in I[g_u, g_v]$ . Converse direction is clear from  $I[g_u, g_v] \subseteq I[u, v]$ , which is from the definition of the gate. ◀

Note that, unless  $r(u) = r(v)$ , one of the gates of  $u$  or  $v$  in  $F(x)$  is on the total boundary of  $F(x)$ . Therefore, to obtain the desired structural result, we just need to bound the number of fibers with non-empty intersection with  $I[u, v]$ . We use the following lemma from [13].

► **Lemma 14 ([13]).** *Let  $u, v$  be vertices with  $r(u) \neq r(v)$ . Then, one of the  $m \in I[u, v]$ ,  $r(u) \sim r(v)$ , or  $d(m, r(u)) = d(m, r(v)) = d(r(u), r(v)) = 2$  holds.*

Assume  $m \in I[u, v]$ . Then,  $I[u, v] \cap F(x) \neq \emptyset$  means  $x \in I[u, v]$ . Therefore the number of such fibers  $F(x)$  is same as the number of vertices in  $I[u, v] \cap \text{St}(m)$ . Now, from the fact that  $I[u, v]$  has a grid structure (see Lemma 2) and  $\text{St}(m)$  consists of the vertices in an edge or a square that contains  $m$ , we have that  $|I[u, v] \cap \text{St}(m)| \leq 9$ .

If  $r(u) \sim r(v)$ , from Lemma 6, we have  $I[u, v] \subseteq F(r(u)) \cup F(r(v))$ . If  $d(r(u), r(v)) = 2$ , let  $w$  be the unique common neighbor of  $r(u)$  and  $r(v)$ . Then, from Lemma 6, we have  $I[u, v] \subseteq F(r(u)) \cup F(w) \cup F(r(v))$ . Therefore, in all cases, the number of fibers with nonempty intersection with  $I[u, v]$  is bounded by 9.

In all of these cases, we can list the fibers  $F(x)$  with nonempty intersection with the given interval  $I[u, v]$ ; it is the set of the fibers of the vertices in  $I[r(u), r(v)]$  because  $\bigcup_{x \in I[r(u), r(v)]} F(x)$  is convex, and, we can list them efficiently by using the list of all squares in  $G$ . Now it is sufficient to give a way to calculate the gate of  $u$  and  $v$  in each of these fibers for our algorithm. We give the algorithm in full version.

Above technique can also be applied for the following query. We are given three vertices  $v_1, v_2, v_3$  in a cube-free median graph  $G$  and asked to answer the median  $v$  of these three vertices. Let  $x$  be the median of  $r(v_1)$ ,  $r(v_2)$  and  $r(v_3)$ .  $x$  can be calculated in  $O(\log n)$  time because each of  $I[r(v_1), r(v_2)]$ ,  $I[r(v_2), r(v_3)]$  and  $I[r(v_3), r(v_1)]$  contains at most 9 vertices and  $x$  is the unique vertex in the intersection of these intervals. Now, we can state that  $v \in F(x)$ , because  $F(x)$  is the only fiber that can intersect all of  $I[v_1, v_2]$ ,  $I[v_2, v_3]$  and  $I[v_3, v_1]$ .

Let  $g_{v_1}$  (resp.  $g_{v_2}, g_{v_3}$ ) be the gate of  $v_1$  (resp.  $v_2, v_3$ ) in  $F(x)$ , which can be calculated in  $O(\log n)$  time. Then, from Lemma 13,  $v$  coincides with the median of  $g_{v_1}$ ,  $g_{v_2}$  and  $g_{v_3}$ . Therefore we can reduce the median query on the original graph into the median query on the fiber  $F(x)$  in  $O(\log n)$  time. By recursively working on the fiber, we can calculate  $v$  after  $O(\log n)$  recursion steps. Therefore the query can be answered in  $O(\log^2 n)$  time in total. The data structure required here is constructed in  $O(n \log^2 n)$  time just by taking the necessary parts of the algorithm in full version.

---

## References

- 1 Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: a survey and a new distributed algorithm. In *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 258–264, 2002.
- 2 S P Avann. Metric ternary distributive semi-lattices. *Proceedings of the American Mathematical Society*, 12(3):407–414, 1961.
- 3 Hans-Jurgen Bandelt and Victor Chepoi. Metric graph theory and geometry: a survey. *Contemporary Mathematics*, 453:49–86, 2008.
- 4 Michael A Bender and Martin Farach-Colton. The LCA problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94, 2000.
- 5 Michael A Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.
- 6 Laurine Bénêteau, Jérémie Chalopin, Victor Chepoi, and Yann Vaxès. Medians in median graphs and their cube complexes in linear time. In *Proceedings of the Forty-Seventh International Colloquium on Automata, Languages, and Programming*, page to appear, 2020.
- 7 Garrett Birkhoff and Stephen A Kiss. A ternary operation in distributive lattices. *Bulletin of the American Mathematical Society*, 53(8):749–752, 1947.
- 8 Gerth Stølting Brodal, Pooya Davoodi, and S Srinivasa Rao. Path minima queries in dynamic weighted trees. In *Workshop on Algorithms and Data Structures*, pages 290–301. Springer, 2011.
- 9 Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1-4):337–361, 1987.
- 10 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.
- 11 Bernard Chazelle and Burton Rosenberg. Computing partial sums in multidimensional arrays. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 131–139, 1989.
- 12 Victor Chepoi. Classification of graphs by means of metric triangles. *Metody Diskret. Analiz*, 49:75–93, 1989.
- 13 Victor Chepoi, Arnaud Labourel, and Sébastien Ratel. Distance labeling schemes for cube-free median graphs. In *44th International Symposium on Mathematical Foundations of Computer Science*, pages 15:1–15:14, 2019.
- 14 Victor Chepoi and Daniela Maftuleac. Shortest path problem in rectangular complexes of global nonpositive curvature. *Computational Geometry*, 46(1):51–64, 2013.


- 15 Adam Clearwater, Clemens Puppe, and Arkadii Slinko. Generalizing the single-crossing property on lines and trees to intermediate preferences on median graphs. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- 16 Nadia Creignou and J-J Hébrard. On generating all solutions of generalized satisfiability problems. *RAIRO-Theoretical Informatics and Applications*, 31(6):499–511, 1997.
- 17 Gabrielle Demange. Majority relation and median representative ordering. *SERIEs: Journal of the Spanish Economic Association*, 3(1):95–109, 2012.
- 18 Harold N Gabow, Jon Louis Bentley, and Robert E Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 135–143, 1984.
- 19 Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.
- 20 Dov Harel and Robert E Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
- 21 George S Lueker. A data structure for orthogonal range queries. In *Proceedings of the nineteenth Annual Symposium on Foundations of Computer Science*, pages 28–34, 1978.
- 22 Henry Martyn Mulder and Alexander Schrijver. Median graphs and helly hypergraphs. *Discrete Mathematics*, 25(1):41–50, 1979.
- 23 Ladislav Nebeský. Median graphs. *Commentationes Mathematicae Universitatis Carolinae*, 12(2):317–325, 1971.
- 24 David Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000.
- 25 Clemens Puppe and Arkadii Slinko. Condorcet domains, median graphs and the single-crossing property. *Economic Theory*, 67(1):285–318, 2019.
- 26 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.
- 27 Andrew C Yao. Space-time tradeoff for answering range queries. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 128–136, 1982.
- 28 Andrew C Yao. On the complexity of maintaining partial sums. *SIAM Journal on Computing*, 14(2):277–288, 1985.
- 29 Hao Yuan and Mikhail J Atallah. Data structures for range minimum queries in multidimensional arrays. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 150–160, 2010.



# Untangling Circular Drawings: Algorithms and Complexity

Sujoy Bhore ✉ 

Indian Institute of Science Education and Research, Bhopal, India

Guangping Li ✉ 

Algorithms and Complexity Group, TU Wien, Austria

Martin Nöllenburg ✉ 

Algorithms and Complexity Group, TU Wien, Austria

Ignaz Rutter ✉ 

Universität Passau, Germany

Hsiang-Yun Wu ✉ 

Research Unit of Computer Graphics, TU Wien, Austria

---

## Abstract

We consider the problem of untangling a given (non-planar) straight-line circular drawing  $\delta_G$  of an outerplanar graph  $G = (V, E)$  into a planar straight-line circular drawing by shifting a minimum number of vertices to a new position on the circle. For an outerplanar graph  $G$ , it is clear that such a crossing-free circular drawing always exists and we define the *circular shifting number*  $\text{shift}^\circ(\delta_G)$  as the minimum number of vertices that need to be shifted to resolve all crossings of  $\delta_G$ . We show that the problem CIRCULAR UNTANGLING, asking whether  $\text{shift}^\circ(\delta_G) \leq K$  for a given integer  $K$ , is NP-complete. Based on this result we study CIRCULAR UNTANGLING for *almost-planar* circular drawings, in which a single edge is involved in all the crossings. In this case we provide a tight upper bound  $\text{shift}^\circ(\delta_G) \leq \lfloor \frac{n}{2} \rfloor - 1$ , where  $n$  is the number of vertices in  $G$ , and present a polynomial-time algorithm to compute the circular shifting number of almost-planar drawings.

**2012 ACM Subject Classification** Human-centered computing → Graph drawings; Mathematics of computing → Permutations and combinations; Theory of computation → Problems, reductions and completeness

**Keywords and phrases** graph drawing, straight-line drawing, outerplanarity, NP-hardness, untangling

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.19

**Related Version** *Full Version:* <http://arxiv.org/abs/2111.09766>

**Funding** *Guangping Li:* Austrian Science Fund (FWF) under grant P 31119

*Martin Nöllenburg:* Austrian Science Fund (FWF) under grant P 31119

*Ignaz Rutter:* Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant Ru 1903/3-1

## 1 Introduction

The family of outerplanar graphs, i.e., the graphs that admit a planar drawing where all vertices are incident to the outer face, is an important subclass of planar graphs and exhibits interesting properties in algorithm design, e.g., they have treewidth at most 2. Being defined by the existence of a certain type of drawing, outerplanar graphs are a fundamental topic in the field of graph drawing and information visualization; they are relevant to circular graph drawing [28] and book embedding [3, 5]. Several aspects of outerplanar graphs have been studied over the years, e.g., characterization [9, 14, 29], recognition [1, 31], and drawing [15, 21, 27]. Moreover, outerplanar graphs and their drawings have been applied to various scientific fields, e.g., network routing [16], VLSI design [10], and biological data modeling and visualization [20, 32].



© Sujoy Bhore, Guangping Li, Martin Nöllenburg, Ignaz Rutter, and Hsiang-Yun Wu; licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 19; pp. 19:1–19:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we study the untangling problem for non-planar circular drawings of outerplanar graphs, i.e., we are interested in restoring the planarity property of a straight-line circular drawing with a minimum number of vertex shifts. Similar untangling concepts have been used previously for eliminating edge crossings in non-planar drawings of planar graphs [18]. More precisely, let  $G = (V, E)$  be an  $n$ -vertex outerplanar graph and let  $\delta_G$  be an outerplanar drawing of  $G$ , which can be described combinatorially as the (cyclic) order  $\sigma = (v_1, v_2, \dots, v_n)$  of  $V$  when traversing vertices on the boundary of the outer face counterclockwise. This order  $\sigma$  corresponds to a circular drawing by mapping each vertex  $v_i \in V$  to the point  $p_i$  on the unit circle  $\mathcal{O}$  with polar coordinate  $p_i = (1, 2\pi i/n)$  and drawing each edge  $(v_i, v_j) \in E$  as the straight-line segment between its endpoints  $p_i$  and  $p_j$ . Two edges  $e, e'$  cross in  $\delta_G$  if and only if their endpoints alternate in the order  $\sigma$ . We note that it is sufficient to consider circular drawings since any outerplanar drawing can be transformed into an equivalent circular drawing by morphing the boundary of the outer face to  $\mathcal{O}$ .

Our untangling problem is motivated by the problem of maintaining an outerplanar drawing of a *dynamic* outerplanar graph, which is subject to edge or vertex insertions and deletions, while maximizing the visual *stability* of the drawing [22, 23], i.e., the number of vertices that can remain in their current position. Such problems of maintaining drawings with specific properties for dynamic graphs have been studied before [2, 4, 12, 13], but not for the outerplanarity property.

The notion of untangling is often used in the literature for a crossing elimination procedure that makes a non-planar drawing of a planar graph crossing-free; see [11, 19, 25, 26]. Given a straight-line drawing  $\delta_G$  of a planar graph  $G$ , the problem to decide if one can untangle  $\delta_G$  by moving at most  $k$  vertices, is proved to be NP-hard [18, 30]. Lower bounds on the number of vertices that can remain fixed in an untangling process have also been studied [7, 8, 18]. Bose et al. [7] proved that  $\Omega(n^{1/4})$  vertices can remain fixed when untangling a drawing. Cano et al. [8] on the other hand provide a family of drawings, where at most  $O(n^{0.4948})$  vertices can remain fixed during untangling. Goaoc et al. [18] proposed an algorithm, which allows at least  $\sqrt{(\log n) - 1}/\log \log n$  vertices to be fixed when untangling a drawing. If the graph is outerplanar, the algorithm proposed by Goaoc et al. could eliminate all edge crossings while keeping at least  $\sqrt{n/2}$  vertices fixed. Notice that the drawing obtained by this algorithm is planar but not necessarily outerplanar. In this paper, we study untangling procedures to obtain an outerplanar drawing from a non-outerplanar drawing. To the best of our knowledge, there are no previous studies about untangling circular drawings.

**Preliminaries and Problem Definition.** Given a graph  $G = (V, E)$ , we say two vertices are *2-connected* if they are connected by two internally vertex-disjoint paths. A 2-connected component of  $G$  is a maximal set of pairwise 2-connected vertices. Two subsets  $A, B \subseteq V$  are *adjacent* if there is an edge  $ab \in E$  with  $a \in A$  and  $b \in B$ . A *bridge* (resp. *cut-vertex*) of  $G$  is an edge (resp. vertex) whose deletion increases the number of connected components of  $G$ .

A drawing of a graph is *planar* if it has no crossings, it is *almost-planar* if there is a single edge that is involved in all crossings, and it is outerplanar if it is planar and all vertices are incident to the outer face. A graph  $G = (V, E)$  is *outerplanar* if it admits an outerplanar drawing. In addition, a drawing where the vertices lie on a circle and the edges are drawn as straight-line segments is called a *circular drawing*. Every outerplanar graph  $G$  admits a planar circular drawing, as one can start with an arbitrary outerplanar drawing  $\delta_G$  of  $G$  and transform the outer face of  $\delta_G$  to a circle [28]. In this paper, we exclusively work with circular drawings of outerplanar graphs; we thus simply refer to them as drawings.

Given a non-planar circular drawing  $\delta_G$  of an  $n$ -vertex outerplanar graph  $G$  where vertices lie on the unit circle  $\mathcal{O}$ , we can transform the drawing  $\delta_G$  to an outerplanar drawing by moving the vertices on the circle  $\mathcal{O}$ . We call a sequence of moving operations that results in

an outerplanar drawing an *untangling* of  $\delta_G$ . Formally, given a circular drawing  $\delta_G$ , a vertex move operation (or shift) changes the position of one vertex in  $\delta_G$  to another position on the circle  $\mathcal{O}$  [18]. We define the *circular shifting number*  $\text{shift}^\circ(\delta_G)$  of an outerplanar drawing  $\delta_G$  to be the minimum number of vertices that are required to shift in order to untangle  $\delta_G$ . We say an untangling is *optimal* if the number of vertex moves of this untangling is the minimum over all valid untanglings of  $\delta_G$ . We study the following problems.

► **Problem 1.1** (MINIMUM CIRCULAR UNTANGLING (MINCU)). *Given a circular drawing  $\delta_G$  of an outerplanar graph  $G$ , find a sequence of  $\text{shift}^\circ(\delta_G)$  vertex moves that untangles  $\delta_G$ .*

► **Problem 1.2** (CIRCULAR UNTANGLING (CU)). *Given a circular drawing  $\delta_G$  of an outerplanar graph  $G$  and an integer  $K$ , decide if  $\text{shift}^\circ(\delta_G) \leq K$ .*

**Contributions.** In Section 2, we show that the problem CIRCULAR UNTANGLING is NP-complete. We then consider almost-planar drawings. In this case, we provide a tight upper bound on the circular shifting number in Section 3 and design a quadratic algorithm to compute a circular untangling with the minimum number of vertex moves in Section 4. Details of the omitted/sketched proofs (marked with  $\star$ ) are available in the full version [6] of the paper.

## 2 Complexity of Circular Untangling

The goal of this section is to prove the following theorem.

► **Theorem 2.1.** CIRCULAR UNTANGLING is NP-complete.

Ultimately, the NP-completeness follows by a reduction from the well-known NP-complete problem 3-PARTITION. However, we do not give a direct reduction but rather work via an intermediate problem, called DISTINCT INCREASING CHUNK ORDERING WITH REVERSALS that concerns increasing subsequences. A *chunk* is a sequence  $S = (s_i)_{i=1,\dots,n}$  of positive integers. For a chunk  $C$ , we denote  $C^{-1}$  as its reversal. In the following, we introduce two longest increasing subsequence problems.

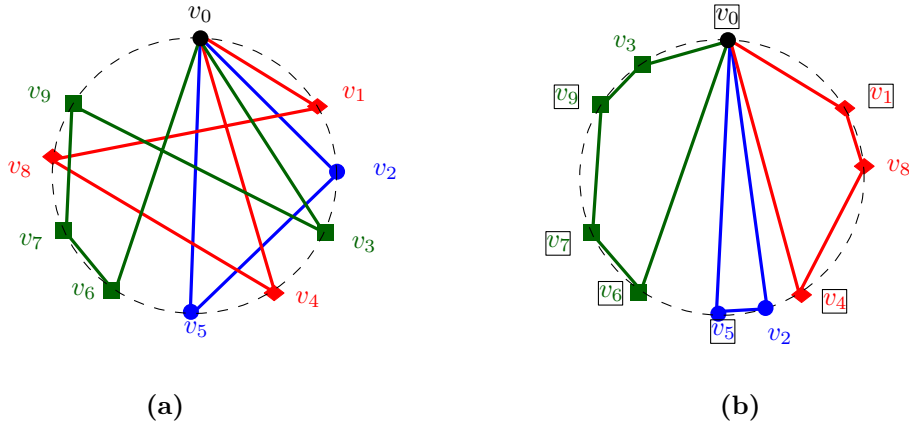
► **Problem 2.2** (INCREASING CHUNK ORDERING (ICO)). *Given  $\ell$  chunks  $C_1, \dots, C_\ell$  and a positive number  $M$ , the question is if there exists a permutation  $\pi$  of  $\{1, \dots, \ell\}$  such that the concatenation  $C_{\pi(1)}C_{\pi(2)} \cdots C_{\pi(\ell)}$  contains a strictly increasing subsequence (SISS) of length  $M$ .*

► **Problem 2.3** (INCREASING CHUNK ORDERING WITH REVERSALS (ICOREV)). *Given  $\ell$  chunks  $C_1, \dots, C_\ell$  and a positive integer  $M$ , the question is to determine whether a permutation  $\pi$  of  $\{1, \dots, \ell\}$  and a function  $\varepsilon: \{1, \dots, \ell\} \rightarrow \{-1, 1\}$  exist such that the concatenation  $C_{\pi(1)}^{\varepsilon(1)}C_{\pi(2)}^{\varepsilon(2)} \cdots C_{\pi(\ell)}^{\varepsilon(\ell)}$  contains a SISS of length  $M$ .*

These two problems also come in *distinct* variants, denoted by DISTINCT-ICO and DISTINCT-ICOREV, respectively, where all numbers in all input chunks need to be distinct. In the following, for two problem  $A$  and  $B$ , we write  $A \leq_p B$  if there is a polynomial-time reduction from  $A$  to  $B$ . It is readily seen that CIRCULAR UNTANGLING lies in NP. Therefore, Theorem 2.1 follows immediately from the following two reduction lemmas, whose proofs are given in the next two subsections.

► **Lemma 2.4.**  $\text{DISTINCT-ICOREV} \leq_p \text{CIRCULAR UNTANGLING}$

► **Lemma 2.5.**  $(\star) \text{ 3-PARTITION} \leq_p \text{DISTINCT-ICOREV}$



■ **Figure 1** The reduction from DISTINCT-ICOREV to CIRCULAR UNTANGLING. (a) The circular drawing  $\delta_G$  constructed from a DISTINCT-ICOREV instance with chunk set  $C = \{C_1 = (1, 8, 4), C_2 = (2, 5), C_3 = (6, 7, 9, 3)\}$ . (b) An example drawing obtained by applying an optimum untangling on  $\delta_G$ . Fixed vertices are marked in  $\square$ .

## 2.1 Proof of Lemma 2.4

Let  $I = (C, M)$  be an instance of DISTINCT-ICOREV with chunks  $C_1, \dots, C_\ell$ . By replacing each number with its rank among all occurring numbers, we may assume without loss of generality, that the numbers in the sequence are  $1, \dots, \sum_{i=1}^{\ell} |C_i| =: L$ .

We construct an instance  $I' = (\delta_G, K)$  of CIRCULAR UNTANGLING as follows; see Figure 1a. We create vertices  $v_1, \dots, v_L$  and an additional vertex  $v_0$ . For each chunk  $C_i$ , we create a cycle  $K_i$  that starts at  $v_0$ , visits the vertices that correspond to the elements of  $C_i$  in the given order, and then returns to  $v_0$ . That is,  $G$  consists of  $\ell$  cycles that are joined by the cut-vertex  $v_0$ . The drawing  $\delta_G$  is obtained by placing the vertices in the order  $\sigma_G = v_0, v_1, v_2, \dots, v_L$  clockwise. Finally, we set  $K := L - M$ . Clearly,  $I'$  can be constructed from  $I$  in polynomial time. It remains to prove the following.

► **Lemma 2.6.**  *$I$  is a yes-instance of DISTINCT-ICOREV if and only if  $I'$  is a yes-instance of CIRCULAR UNTANGLING.*

**Proof.** Observe that, since in  $\delta_G$  the vertices are ordered clockwise according to their numbering, the problem of untangling with at most  $L - M$  vertex moves is equivalent to finding a planar circular drawing of  $G$  whose clockwise ordering contains an increasing subsequence of at least  $M$  vertices, which can then be kept fixed; see Figure 1b.

The key observation is that, in every planar circular drawing of  $G$ , the vertices of each cycle  $K_i$  are consecutive, and the order of its vertices is the order along  $K_i$ , i.e., it is fixed up to reversal. Hence the choice of a circular drawing whose clockwise ordering contains an increasing subsequence of at least  $M$  vertices directly corresponds to a permutation and reversal of the chunks  $C_i$ . ◀

## 2.2 Proof of Lemma 2.5

Let  $I = (A, K)$  be an instance of 3-PARTITION. The input to the 3-PARTITION problem consists of a multiset  $A = \{a_1, \dots, a_{3m}\}$  of  $3m$  positive integers and a positive integer  $K$  such that  $\frac{K}{4} < a_i < \frac{K}{2}$ , for  $i = 1, \dots, 3m$ . The question is whether  $A$  can be partitioned into  $m$  disjoint triplets  $T_1, \dots, T_m$  such that  $\sum_{a \in T_j} a = K$ , for all  $j = 1, \dots, m$ . It is well-known

that 3-PARTITION is strongly NP-complete, i.e., the problem is NP-complete even if the integers in  $A$  and  $K$  are polynomially bounded in  $m$ ; see [17]. We show the following simpler lemma and then extend its proof to a proof of Lemma 2.5.

► **Lemma 2.7.** *3-PARTITION  $\leq_p$  INCREASING CHUNK ORDERING.*

**Proof.** Let  $I = (A, K)$  with  $A = \{a_1, \dots, a_{3m}\}$  be an instance of 3-PARTITION. We create for each element  $a_i$  a corresponding chunk  $C_i$  as follows. For two integers  $a < l$ , we denote the consecutive integer sequence  $(a, a + 1, \dots, a + l - 1)$  as the *incremental sequence* of length  $l$  starting at  $a$ .

We say that an incremental sequence *crosses a multiple of  $K$*  if it contains  $cK + 1$  and  $cK$  for some integer  $c$ . We take all the incremental sequences of length  $a_i$  that start at a value in  $\{1, \dots, mK\}$  except for those that cross a multiple of  $K$ . The chunk  $C_i$  is formed by concatenating these sequences in decreasing order of their first number. For example, for  $a_i = 3, m = 2, K = 6$ ,  $C_i$  is the concatenation of sequences  $(10, 11, 12), (9, 10, 11), (8, 9, 10), (7, 8, 9), (4, 5, 6), (3, 4, 5), (2, 3, 4), (1, 2, 3)$ .

We obtain an instance  $I' = (C, M)$  of INCREASING CHUNK ORDERING by setting  $C = \{C_1, \dots, C_{3m}\}$  and  $M := mK$ . We claim that  $I$  is a yes-instance of 3-PARTITION if and only if  $I'$  is a yes-instance of INCREASING CHUNK ORDERING. For the proof, we rely on the following observations:

- (i) every strictly increasing subsequence in  $C_i$  has length at most  $a_i$ .
- (ii) every strictly increasing subsequence in  $C_i$  of length  $a_i$  is consecutive and does not cross a multiple of  $K$ .
- (iii) every incremental sequence of  $\{1, \dots, mK\}$  that has length  $a_i$  and does not cross a multiple of  $K$  is a subsequence of  $C_i$ .

Assume there is a partition of the elements of  $A$  into  $m$  triples, each of which sums to  $K$ . We arbitrarily order these triples, and within each triplet, we order the elements according to their index. This defines a total ordering on the elements, and therefore on the chunks. Let  $T_i = \{a_x, a_y, a_z\}$  with  $x < y < z$  be the  $i$ th triplet and let  $C_x, C_y, C_z$  be the corresponding chunks. By observation (iii)  $C_x, C_y,$  and  $C_z$  contain respectively three incremental subsequences of length  $a_x, a_y,$  and  $a_z$  starting at  $iK + 1, iK + a_x + 1,$  and  $iK + a_x + a_y + 1$ . Concatenating the subsequences for all chunks hence gives the increasing subsequence  $1, \dots, mK$ .

Conversely, assume that there is a chunk ordering so that we obtain the incremental subsequence  $1, \dots, mK$ . By observation (i), each chunk  $C_i$  can contribute a subsequence of at most  $a_i$  elements; therefore each chunk  $C_i$  must contribute an increasing subsequence  $S_i$  of length  $a_i$ . By observation (ii), the subsequence  $S_i$  does not cross a multiple of  $K$ . Therefore, partitioning the sequence  $1, \dots, mK$  into  $k$  incremental sequences  $((c - 1)K + 1, \dots, cK)$  for  $c \in \{1, \dots, m\}$ , each of which corresponds to a triplet of  $A$  with the sum  $K$ . Together, these triplets define a solution of the instance  $I$  of 3-PARTITION. ◀

The proof of the stronger claim of Lemma 2.5 follows the same ideas but requires several additional ingredients. First of all, to achieve distinctness of the elements, we use strings of numbers, called *words*, which we order lexicographically. Then the main information is encoded in the first elements of the sequence, whereas the later entries are used to make the words pairwise distinct. At the end of the construction, each word can be replaced by its rank in a lexicographic ordering of all words that occur in the instance.

A second complication stems from the fact that chunks can be reversed. The chunks we construct in the proof of Lemma 2.7 contain a significantly longer increasing subsequence after reversal, as it may include one element from each incremental subsequence of the chunk,

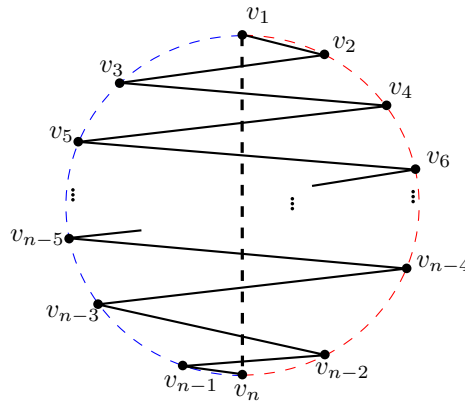
of which there may be  $mK$  many. To alleviate this, we add a sufficiently long *tailing sequence* of length  $X$  to each increasing subsequence so that one cannot benefit from a reversal. Then chunk  $C_i$  can provide an increasing subsequence of length  $a_i + X$ , and all chunks together shall provide an increasing subsequence of length  $mK + 3mX$ . Implementing this naively by simply adding  $X$  to each element in the 3-PARTITION instance does not work, as the possible starting positions for the increasing subsequences provided by a chunk then grows to  $mK + 3mX$ , thus providing an incremental sequence of length  $mK + 3mX$  after reversal. We can however observe that the only reasonable starting points for the increasing subsequence provided by a chunk  $C_i$  are the original  $mK$ , each of which can be shifted by  $cX$ , where  $c$  is the number of chunks placed before  $C_i$ . This makes for a total of only  $3m^2K$  possible starting values. By choosing  $X > 3m^2K$ , it is then ensured that reversing a chunk only provides a shorter increasing subsequence than  $a_i + X$ .

### 3 A Tight Upper Bound for Almost-Planar Drawings

Let  $G = (V, E)$  be an outerplanar graph, let  $\delta_G$  be an almost-planar circular drawing of  $G$ . In this section, we present an untangling procedure for such almost-planar circular drawings that provides a tight upper bound of  $\lfloor \frac{n}{2} \rfloor - 1$  on  $\text{shift}^\circ(\delta_G)$ .

► **Theorem 3.1.** *Given an almost-planar drawing  $\delta_G$  of an  $n$ -vertex outerplanar graph  $G$  the circular shifting number  $\text{shift}^\circ(\delta_G) \leq \lfloor \frac{n}{2} \rfloor - 1$ , and this bound is tight.*

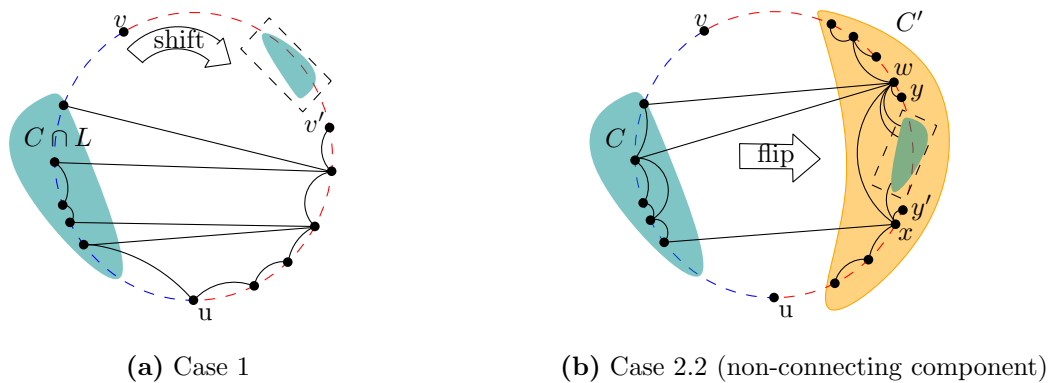
To see that the bound is tight, let  $n \geq 4$  be an even number and let  $G$  be the cycle on vertices  $v_1, \dots, v_n, v_1$  (in this order) and let  $\delta_G$  be a drawing with the clockwise order  $v_2, \dots, v_{2i} \dots, v_n, v_{n-1}, \dots, v_{2i+1}, \dots, v_1$ ; see Figure 2.



■ **Figure 2** An almost-planar drawing  $\delta_G$  with  $\text{shift}^\circ(\delta_G) = \frac{n}{2} - 1$ .

We claim that  $\text{shift}^\circ(\delta_G) \geq \frac{n}{2} - 1$ . Clearly, the clockwise circular ordering of its vertices in a crossing-free circle drawing is either  $v_1, v_2, \dots, v_n$  or its reversal. Assume that we turn it to the clockwise ordering  $v_1, v_2, \dots, v_n$ ; the other case is symmetric. In  $\delta_G$ , the  $\frac{n}{2}$  odd-index vertices  $v_1, \dots, v_{2i+1} \dots, v_{n-1}$  and  $v_n$  are ordered counterclockwise. To reach a clockwise ordering, we need to move all but two of these vertices. Thus, at least  $\frac{n}{2} - 1$  vertices in total are required to move.

The remainder of this section is devoted to proving the upper bound. Let  $e = uv$  be the edge of  $\delta_G$  that contains all the crossings, and let  $G' = G - e$  and  $\delta_{G'}$  be the circular drawing of  $G'$  by removing the edge  $e$  from  $\delta_G$ . The edge  $uv$  partitions the vertices in  $V \setminus \{u, v\}$  into the sets  $L$  and  $R$  that lie on the left and right side of the edge  $uv$  (directed from  $u$  to  $v$ ).



■ **Figure 3** Moving a left component, keeping/reversing the clockwise ordering of its vertices.

► **Theorem 3.2.** *Let  $\delta_G$  be an almost-planar drawing of an outerplanar graph  $G$ . An outerplanar drawing of  $G$  can be obtained by moving only vertices of  $L$  or only vertices of  $R$  to the other side in  $\delta_G$  and fixing all the remaining vertices. The untangling moves only  $\min\{|L|, |R|\}$  vertices and can be computed in linear time.*

This immediately implies the upper bound from Theorem 3.1, since  $|L \cup R| = n - 2$ , and therefore  $\min\{|L|, |R|\} \leq \lfloor \frac{n}{2} \rfloor - 1$ . To prove Theorem 3.2, we distinguish different cases based on the connectivity of  $u$  and  $v$  in  $G'$ .

**Case 1:  $u, v$  are not connected in  $G'$ .** Consider a connected component  $C$  of  $G'$  that contains vertices from  $L$  and from  $R$ .

► **Proposition 3.3.** *Suppose  $u, v$  are not connected in  $G'$ . Let  $C$  be a connected component of  $G'$  that contains vertices from  $L$  and from  $R$ . It is possible to obtain a new almost-planar drawing  $\delta'_G$  of  $G$  from  $\delta_G$  by moving only the vertices of  $C \cap L$  (resp.  $C \cap R$ ) such that  $C$  lies entirely on the right (resp. left) side of  $uv$ .*

**Proof.** Since  $u, v$  are not connected in  $G'$ ,  $C$  contains at most one of  $u, v$ . Without loss of generality, we assume that  $v \notin C$ ; see Figure 3a. Let  $v'$  be the first clockwise vertex after  $v$  that lies in  $C$ . Let  $\delta'_G$  be the drawing obtained from  $\delta_G$  by moving the vertices of  $C \cap L$  clockwise just before  $v'$  without changing their clockwise ordering. Observe that this removes all crossings of  $e$  with  $C$ . The choice of  $v'$  ensures that no edge of  $C$  alternates with an edge whose endpoints lie in  $V \setminus C$ . Finally, the vertices of  $C$  maintain their clockwise order. This shows that no new crossings are introduced, and the crossings between  $e$  and  $C$  are removed. ◀

By applying Proposition 3.3 for each connected component of  $G'$  that contains vertices from  $L$  and from  $R$ , we obtain an outerplanar drawing of  $G$ .

**Case 2:  $u, v$  are connected in  $G'$ .** Let  $C$  be the connected component in  $G'$  that contains both vertices  $u$  and  $v$ . Note that if  $C'$  is another connected component of  $G'$ , then it must lie entirely to the left or entirely to the right of edge  $e$ . Here, we ignore such components as they never need to be moved. We may hence assume that  $G'$  is connected.

**Case 2.1:  $u, v$  are 2-connected in  $G'$ .** We claim that in this case  $\delta_G$  is already planar.

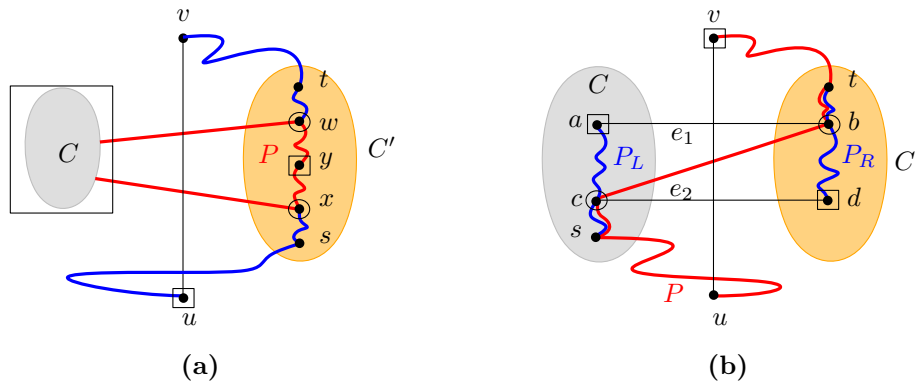
► **Proposition 3.4.** *If  $u$  and  $v$  are 2-connected in  $G'$ , then  $\delta_G$  is planar.*



**Proof.** If vertices  $u, v \in V$  are 2-connected in  $G'$ , then  $G'$  contains a cycle  $C$  that includes both  $u$  and  $v$ . In  $\delta_{G'}$ , this cycle is drawn as a closed curve. Any edge that intersects the interior region of this closed curve therefore has both endpoints on  $C$ . If there exists an edge  $e' = xy$  that intersects  $e = uv$ , then contracting the four subpaths of  $C$  connecting each of  $\{x, y\}$  to each of  $\{u, v\}$  yields a  $K_4$ -minor in  $G$ , which contradicts the outerplanarity of  $G$ . ◀

**Case 2.2:  $u, v$  are connected but not 2-connected in  $G'$ .** In this case  $G'$  contains at least one cut-vertex that separates  $u$  and  $v$ . Notice that each path from  $u$  to  $v$  visits all such cut-vertices between  $u$  and  $v$  in the same order. Let  $f$  and  $l$  be the first and the last cut-vertex on any  $uv$ -path. Additionally, add  $u$  to the set of  $L, R$  that contains  $f$  and likewise add  $v$  to the set of  $L, R$  that contains  $l$ . Let  $X$  denote the set of edges of  $G'$  that have one endpoint in  $L$  and the other in  $R$ . Each connected component of  $G' - X$  is either a subset of  $L$  or a subset of  $R$ , which are called *left* and *right components*, respectively. We call a component of  $G' - X$  *connecting* if it contains either  $u$  or  $v$ , or removing it from  $G'$  disconnects  $u$  and  $v$ . For a left component  $C_L$  and a right component  $C_R$ , we denote by  $E(C_L, C_R)$  the set of edges of  $G'$  that connect a vertex from  $C_L$  to a vertex in  $C_R$ . We can observe that since  $G'$  is connected, for any edge that connects a left and a right component, at least one of the components must be connecting. We use the following observation.

► **Observation 3.5.** *If  $P$  is an  $xy$ -path in a left (right) component  $C$ , then it contains all vertices of  $C$  that are adjacent to a vertex of a right (left) component and lie between  $x$  and  $y$  on the left (right) side.*



■ **Figure 4** The  $K_{2,3}$ -minors we use in the proofs of (a) Lemma 3.6 and (b) Lemma 3.8.

► **Lemma 3.6.** *Every non-connecting component  $C$  of  $G' - X$  is adjacent to exactly one component  $C'$  of  $G' - X$ . Moreover,  $C'$  is connecting, there are at most two vertices in  $C'$  that are incident to edges in  $E(C, C')$ , and if there are two such vertices  $w, x \in C'$ , then they are adjacent and removing  $wx$  disconnects  $C'$ .*

**Proof.** Without loss of generality, we assume that  $C$  is a left component. Since  $C$  is non-connecting, any component adjacent to it must be connecting. Moreover, if there are two distinct such components, they lie on the right side of the edge  $uv$ . Then either there is a path on the right side that connects them (but then they are not distinct), or removing  $C$  disconnects these components, and therefore  $uv$ , contradicting the assumption that  $C$  is a non-connecting component. Therefore  $C$  is adjacent to exactly one other component  $C'$ ,

which must be a right connecting component. Let  $w$  and  $x$  be the first and the last vertex in  $C'$  that are adjacent to vertices in  $C$  when sweeping the vertices of  $G$  clockwise in  $\delta_G$  starting at  $v$ ; see Figure 4a. The lemma holds trivially if  $w = x$ . Suppose  $w \neq x$ . Next we show that  $wx \in E$  and that  $wx$  is a bridge of  $C$ . Let  $P$  be an arbitrary path from  $w$  to  $x$  in  $C$ . If  $P$  contains an internal vertex  $y$ , then the path  $P$  together with a path from  $w$  to  $x$  whose internal vertices lie in  $C$  forms a cycle, where  $x$  and  $w$  are not consecutive. Note that at least one of  $u, v$ , say  $u$ , is not identical to  $w, x$ , otherwise,  $u, v$  are 2-connected. This cycle, together with disjoint paths from  $w$  to  $v$  and  $x$  to  $u$  and the edge  $uv$  yields a  $K_{2,3}$ -minor in  $G$ ; see Figure 4a. Such paths exist, by the outerplanarity of  $\delta_{G'}$  and the fact that  $C'$  is connecting, but  $C$  is not. Since  $G$  is outerplanar, and therefore cannot contain a  $K_{2,3}$ -minor, this immediately implies that  $P$  consists of the single edge  $wx$ , which must be a bridge of  $C'$  as otherwise there would be a  $wx$ -path with an internal vertex. Observation 3.5 implies that  $w$  and  $x$  are the only vertices of  $C$  that are adjacent to vertices in  $C'$ . ◀

► **Proposition 3.7.** *Let  $C$  be a left (right) non-connecting component of  $G' - X$ . It is always possible to obtain a new almost-planar drawing  $\delta'_G$  of  $G$  from  $\delta_G$  by moving only the vertices of  $C \setminus \{u, v\}$  to the right (left) side.*

**Proof.** Without loss of generality, we assume that  $C$  is a left component. Since  $C$  is non-connecting, then by Lemma 3.6, it is adjacent to at most two vertices on the right side. If there are two such vertices, denote them by  $w$  and  $x$  such that  $w$  occurs before  $x$  on a clockwise traversal from  $v$  to  $u$ . Note that  $wx$  is a bridge of a right component  $C'$  by Lemma 3.6; see Figure 3b. Consider the two components of  $C' - wx$  and let  $y$  be the last vertex that lies in the same component as  $w$  when traversing vertices clockwise from  $w$  to  $x$ . If  $C$  is connected to only one vertex, then we denote this by  $y$ . In both cases, let  $y'$  be the vertex of  $L$  that immediately succeeds  $y$  in clockwise direction (If  $y = u$ , let  $y'$  be the vertex that immediately precedes  $y$ ).

We obtain  $\delta'_G$  by moving all vertices of  $C \setminus \{u, v\}$  between  $y$  and  $y'$ , reversing their clockwise ordering. Observe that the choice of  $y$  and  $y'$  guarantees that  $\delta'_G$  is almost-planar and all crossings lie on  $uv$ . ◀

It remains to deal with connecting components.

► **Lemma 3.8.** *The connecting component of  $G' - X$  containing  $u$  or  $v$  is adjacent to at most one connecting component. Every other connecting component is adjacent to exactly two connecting components. Moreover, if  $C$  and  $C'$  are two adjacent connecting components, then there is a vertex  $w$  that is incident to all edges in  $E(C, C')$ .*

**Proof.** The claims concerning the adjacencies of the connecting components follows from the fact that every  $uv$ -path visits all connecting components in the same order. It remains to prove that all edges between two connecting components share a single vertex. If  $u$  and  $v$  are in one component, then this component is the only connecting component and there is nothing to show.

Now let  $C$  and  $C'$  be adjacent connecting components and assume that  $C$  or  $C'$  may contain one of  $u$  or  $v$  but not both. Furthermore, we assume without loss of generality, that  $C$  is a left and  $C'$  is a right component. For the sake of contradiction, assume there exist two edges  $e_1, e_2 \in E(C, C')$  that do not share an endpoint. Let  $e_1 = ab$  and  $e_2 = cd$  where  $a, c \in C$  and  $b, d \in C'$  such that their clockwise order is  $a, b, d, c$ ; see Figure 4b. Note that one of  $u, v$  is not in the set  $\{a, b, c, d\}$ . Otherwise,  $u$  and  $v$  are 2-connected, which contradicts our case assumption. In the following, we assume without loss of generality that  $a, b, c, d, v$

## 19:10 Untangling Circular Drawings: Algorithms and Complexity

are five distinct vertices in  $G'$ . Let  $P$  be a path from  $u$  to  $v$  in  $G'$ . Since  $C$  and  $C'$  are both connecting,  $P$  contains vertices from both components. When traversing  $P$  from  $u$  to  $v$ , let  $s$  and  $t$  denote the first and the last vertex of  $C \cup C'$  that is encountered, respectively. Here, we assume without loss of generality that  $s \in C$  and  $t \in C'$ . Let  $P_L$  be a path in  $C$  that connects  $s$  to  $a$  and let  $P_R$  be a path in  $C'$  that connects  $d$  to  $t$ . By Observation 3.5,  $P_L$  contains  $c$  and  $P_R$  contains  $b$ . We then obtain a  $K_{2,3}$ -minor of  $G$  by contracting each of the paths  $P_L[c, a]$ ,  $P_R[d, b]$ ,  $vuP[u, s]P_L[s, c]$ , and  $P_R[b, t]P[t, v]$  into a single edge. ◀

By Lemma 3.6 and Lemma 3.8, all vertices of a connecting component of  $G' - X$  can be moved to the other side, similarly as in Proposition 3.7.

► **Proposition 3.9.** (★) *Let  $C$  be a left (right) connecting component of  $G' - X$ . It is possible to obtain a new almost-planar drawing  $\delta'_G$  of  $G$  from  $\delta_G$  by moving only the vertices of  $C \setminus \{u, v\}$  to the right (left) side.*

Proposition 3.7 and Proposition 3.9 together imply Theorem 3.2.

## 4 Untangling Almost-Planar Drawings

In this section, we consider how to untangle an almost-planar circular drawing  $\delta_G$  of an  $n$ -vertex outerplanar graph  $G = (V, E)$  with the minimum number of vertex moves. Firstly, we study this problem in several restricted settings (Sections 4.1–4.3), which leads us to the design of an  $O(n^2)$ -time algorithm to compute  $\text{shift}^\circ(\delta_G)$  in Section 4.4. Let  $e = uv$  be the edge of  $\delta_G$  that contains all the crossings, and let  $G' = G - e$  and  $\delta_{G'}$  be the straight-line circular drawing of  $G'$  by removing the edge  $e$  from  $\delta_G$ . The edge  $uv$  partitions the vertices in  $V \setminus \{u, v\}$  into the sets  $L$  and  $R$  that lie on the left and right side of the edge  $uv$  (directed from  $u$  to  $v$ ). Let  $C_u$  and  $C_v$  be the connected components of  $G'$  that contain  $u$  and  $v$ , respectively. Note that  $C_u = C_v$  if  $u, v$  are connected.

### 4.1 Fixed Edge Untangling

Here we consider untangling under the restriction that the positions of  $u$  and  $v$  are fixed. We denote such untangling as *fixed edge untangling*. From very similar arguments as in Section 3, we derive the following statements.

► **Lemma 4.1.** (★) *Let  $C$  be a connected component of  $G'$ . It is always possible to obtain an almost-planar drawing  $\delta'_G$  of  $G$  from  $\delta_G$  by moving all vertices in  $L \cap C$  (resp.  $R \cap C$ ) to the right (resp. left) side.*

► **Theorem 4.2.** (★) *Given an almost-planar drawing  $\delta_G$  of an outerplanar graph  $G$ , a fixed edge untangling of  $\delta_G$  with the minimum number of vertex moves can be computed in linear time.*

### 4.2 Single Component Untangling

Next, we study an untangling variant, called *Single Component Untangling*, which moves vertices of one particular connected component of  $G'$  that contains the vertices  $u$  or  $v$ , while the other components remain fixed. We claim that  $\delta_G$  can always be untangled in this way.

► **Theorem 4.3.** *It is always possible to untangle  $\delta_G$  by moving only the vertices of  $C_u$  or only the vertices of  $C_v$  and such a single component untangling procedure can be found in linear time.*

**Proof.** If  $C_u = C_v$  the claim is trivially true. So let's consider the case that  $u$  and  $v$  are not connected in  $G'$  and assume that  $|C_u| \leq |C_v|$ . We move the vertices of  $C_u$  as follows. Let  $\sigma_u$  be the clockwise order of  $C_u$  in  $\delta_{G'}$ , starting with  $u$ . We insert the vertices of  $C_u$  in the order  $\sigma_u$  clockwise right after  $v$  to obtain a new drawing  $\delta'_{G'}$  of  $G'$ . Since  $C_u$  was crossing-free before and is placed consecutively on the circle, it remains crossing-free. No other edges have been moved. Furthermore,  $u$  and  $v$  are now neighbors on the circle, so we can insert the edge  $uv$  without crossings and have untangled  $\delta_G$  with  $\min\{|C_u|, |C_v|\}$  moves. ◀

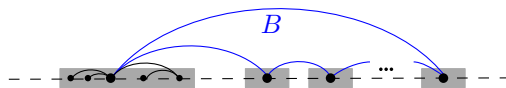
### 4.3 Component-Fixed Untangling

An untangling under the restriction that both of  $C_u$  and  $C_v$  must contain fixed vertices, is denoted as *Component-Fixed Untangling*.

We introduce some notions and provide basic observations. Let  $G$  be a connected outerplanar graph. Let  $B$  be a 2-connected component of  $G$  and  $E(B)$  the set of edges in  $B$ . Since  $G$  is connected and  $B$  is 2-connected, each connected component of  $G - E(B)$  contains exactly one vertex in  $B$ . Given a vertex  $b$  in  $B$ , let  $C_b$  be the connected component of  $G - E(B)$  that contains  $b$ . We denote  $C_b$  as the *attachment* of the 2-connected component  $B$  at the vertex  $b$ .

Let  $H(B)$  be the cyclic vertex ordering of  $B$  in the order of its Hamiltonian cycle<sup>1</sup>. We get Observation 4.4; see Figure 5.

► **Observation 4.4.** *Let  $\delta_G$  be an outerplanar drawing of an outerplanar graph  $G$  and  $B$  be a 2-connected component of  $G$ . Then, the clockwise cyclic vertex ordering of  $B$  in  $\delta_G$  is either  $H(B)$  or its reverse. Furthermore, for each attachment of  $B$ , its vertices appear consecutively on the circle in  $\delta_G$ .*



■ **Figure 5** A 2-connected component  $B$  (in blue) and its attachments (gray boxes) in an outerplanar drawing.

Given a connected outerplanar graph  $G$ , a 2-connected component  $B$  of  $G$  and a circular drawing  $\delta_G$ , we say a sequence  $S$  of vertex moves of  $G$  is *canonical*, associated with  $B$ , if in the drawing obtained by applying  $S$  to  $\delta_G$ , the clockwise cyclic vertex ordering of each attachment of  $B$  remains unchanged. Now we are ready to show that an optimal component-fixed untangling with the restriction that fixed vertices exist in both of  $C_u$  and  $C_v$  can be found in  $O(n^2)$  time; see Theorem 4.5.

► **Theorem 4.5.** *A component-fixed untangling procedure  $U$  with the minimum number of vertex moves can be found in  $O(n^2)$  time.*

The remainder of this section is devoted to describing the procedure  $U$ . We distinguish between the following two cases based on the connectivity of  $u, v$  in  $G'$ . In each case, we present a procedure that runs in  $O(n^2)$  time and reports an optimal component-fixed untangling procedure.

<sup>1</sup> In every outerplanar biconnected graph, there is a unique Hamiltonian cycle that visits each node exactly once [29].

**Case 1:  $u$  and  $v$  are connected in  $G'$ .** Let  $C$  be a connected component of  $G'$  that does not contain  $u, v$ . We claim now that  $C$  must lie entirely on one side of  $uv$  in  $\delta_G$ . Otherwise, let  $P$  be a path of  $\delta_{G'}$  that connects  $u$  and  $v$ . Then there would exist crossings between edges of  $P$  and edges of  $C$  in  $\delta_{G'}$  which contradicts the fact that  $\delta_{G'}$  has no crossings. Thus, we can ignore such components as they do not need to be involved in an untangling. Hence, we may assume  $G'$  is a connected graph. If  $u$  and  $v$  are 2-connected in  $G'$ , then  $\delta_G$  is already outerplanar; see Proposition 3.4. Now we consider the case that  $u$  and  $v$  are connected, but not 2-connected in  $G'$ . Note that  $u, v$  are 2-connected in  $G$ . Let  $B$  be the 2-connected component of  $G$  that contains  $u, v$ . We prove that each component-fixed untangling  $U$  can be transformed into a canonical untangling with smaller or the same number of vertex moves; see Lemma 4.6. Thus, we restrict our attention to canonical untanglings. Let  $H(B) = b_1, \dots, b_k$  be the cyclic vertex ordering of the Hamiltonian cycle of  $B$ . Let  $A_i$  be the attachment of  $B$  at the vertex  $b_i$  and let  $\sigma(A_i)$  be the clockwise vertex ordering of  $A_i$  in  $\delta_G$  for  $i \in \{1, \dots, k\}$ . We consider an optimal canonical component-fixed untangling  $U_o$  which orders  $B$  clockwise as  $H(B)$ . Let  $\delta_G''$  be the outerplanar drawing obtained by applying  $U_o$ . Then the clockwise vertex ordering of  $\delta_G''$  is exactly the concatenation of  $\sigma(A_1), \sigma(A_2), \dots, \sigma(A_k)$ . Given  $\delta_G''$ , an optimal untangling transforming  $\delta_G$  to  $\delta_G''$  can be computed in  $O(n^2)$  time; see [24]. Analogously, we obtain an optimal component-fixed untangling  $U_r$  which orders  $B$  counterclockwise as  $H(B)$ . From the two untanglings  $U_o$  and  $U_r$ , we report the one which moves less vertices as the optimal component-fixed untangling.

► **Lemma 4.6.** *Let  $B$  be the 2-connected component of  $G$  that contains  $u, v$ . Each component-fixed untangling  $U$  of  $\delta_G$  can be transformed into a canonical vertex move sequence  $U_c$  (associated with  $B$ ) that untangles  $\delta_G$ . Furthermore, the number of vertex moves in  $U_c$  is not greater than the number of vertex moves in  $U$ .*

**Proof.** Given a component-fixed untangling  $U$  of  $\delta_G$ , let  $\delta_G^U$  be the drawing obtained after applying  $U$  on  $\delta_G$ . In  $\delta_G^U$ , the cyclic vertex ordering of  $B$  (clockwise or counterclockwise) must correspond to its Hamiltonian cycle ordering  $H(B)$ . Furthermore, the vertices of each attachment of  $B$  appear consecutively in  $\delta_G^U$ , including one vertex of  $B$ ; see Observation 4.4. Let  $A_1, \dots, A_k$  be the attachments of  $B$  in  $G$  (indexed in clockwise order as in  $\delta_G^U$ ) and let  $\sigma(A_i)$  be the clockwise vertex ordering of  $A_i$  in  $\delta_G$  for  $i \in \{1 \dots k\}$ . Now consider the vertex ordering  $\sigma'_G = (\sigma(A_1), \dots, \sigma(A_k))$  and let  $\delta'_G$  be an arbitrary circular drawing where the vertices are ordered as  $\sigma'_G$ . Note that the vertex ordering of each attachment is  $\sigma(A_i)$  in  $\delta'_G$  as in the almost-planar drawing  $\delta_G$ , thus each attachment in  $\delta'_G$  is crossing-free. Moreover, in  $\delta'_G$  the vertices of  $B$  are ordered as in the planar drawing  $\delta_G^U$ , thus there is no crossing inside  $B$ . Overall,  $\delta'_G$  is a planar circular drawing. Let  $U_c$  be the untangling of  $\delta_G$  with minimum number of vertex moves such that the clockwise vertex ordering of the resulting drawing is  $\sigma'_G$ .

To see that  $U_c$  does not move more vertices than  $U$ , let  $\sigma_G$  and  $\sigma_G^U$  be the clockwise vertex orderings of  $\delta_G$  and  $\delta_G^U$ , respectively. We can observe that any common subsequence of  $\sigma_G, \sigma_G^U$  is a subsequence of  $\sigma'_G$ . ◀

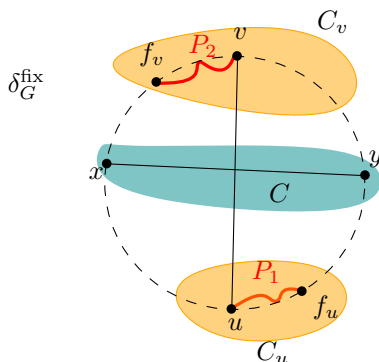
**Case 2:  $u$  and  $v$  are not connected in  $G'$ .** Note that a connected component of  $G'$  that lies entirely on one side of  $uv$  in  $\delta_G$  can be ignored, since there is no need to move any vertices in such components. After ignoring such components, we can assume that a connected component  $C$  of  $G'$  either contains  $u, v$  or  $C$  contains vertices from  $L$  and also vertices from  $R$ .

► **Observation 4.7.** In  $\delta_{G'}$ , vertices of  $C_u$  (resp.  $C_v$ ) lie consecutively on the cycle.

The first step of our untangling procedure  $U$  deals with the connected components of  $G'$  that neither contain  $u$  nor  $v$ . Let  $U^{\text{fix}}$  be an arbitrary component-fixed untangling of  $\delta_G$ , and let  $\delta_G^{\text{fix}}$  be the outerplanar drawing of  $G$  obtained from  $\delta_G$  by applying  $U^{\text{fix}}$ .

► **Lemma 4.8.** Let  $C$  be a connected component of  $G'$  that does not contain vertices  $u$  or  $v$ . Let  $f_u, f_v$  be two vertices in  $C_u$  and  $C_v$ , respectively, which are fixed in  $\delta_G^{\text{fix}}$ . Then,  $C$  must lie entirely on one side of  $f_u f_v$ <sup>2</sup> in  $\delta_G^{\text{fix}}$ .

**Proof.** In the graph  $G$ , due to the definition of  $f_u$  and  $f_v$ , there exists a path  $P_1$  in  $C_u$  connecting  $f_u$  to  $u$ , and a path  $P_2$  in  $C_v$  connecting  $v$  to  $f_v$ ; see Figure 6. Then, the path  $P = P_1 u v P_2$  in  $G$  connects  $f_u$  to  $f_v$ . In  $\delta_G^{\text{fix}}$ , suppose that the connected component  $C$  is not entirely on one side of  $f_u f_v$ , it implies that at least one edge  $xy$  in  $C$  has endpoints  $x, y$  alternate with  $f_u, f_v$  in clockwise ordering of  $\delta_G^{\text{fix}}$  and then has crossings with  $P$ . It contradicts the outerplanarity of the drawing  $\delta_G^{\text{fix}}$ . ◀

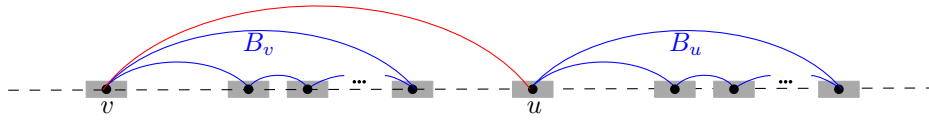


■ **Figure 6** An example illustration for the proof of Lemma 4.8.

Now let  $C$  be a connected component that does not contain  $u, v$ . Vertices  $f_u$  and  $f_v$  partition the vertices of  $C$  in drawing  $\delta_G$  into two sets  $L_C$  and  $R_C$  that are encountered clockwise and counter-clockwise from  $f_u$  to  $f_v$  in  $\delta_G$ , respectively. Observe that,  $L_C = L \cap C$  and  $R_C = R \cap C$ ; see Observation 4.7. Let  $m(C) = \min\{|L \cap C|, |R \cap C|\}$ . By Lemma 4.8,  $m(C)$  is a lower bound of the moved vertices in  $C$  in a component-fixed untangling. By Lemma 4.1, there is a procedure moving  $m(C)$  vertices of  $C$  such that  $C$  lies entirely on one side of  $uv$ . In the first step of our untangling procedure  $U$ , we repeat this step for each component not containing  $u, v$ . After that, an almost-planar drawing of  $G$  remains that has already each component not containing  $u, v$  placed entirely on one side of  $uv$ . We can ignore such components from now on since they never need to be moved again.

Now we assume that  $G'$  has exactly two connected components, namely  $C_u$  and  $C_v$ . Consider an arbitrary outerplanar drawing  $\delta'_G$  of  $G$ . Let  $\sigma(\delta'_G)$  be the circular ordering of vertices in  $\delta'_G$  encountered clockwise. Observe that, in  $\sigma(\delta'_G)$ , the vertices of  $C_u$  (resp.  $C_v$ ) are in a consecutive subsequence  $\sigma(C_u)$  (resp.  $\sigma(C_v)$ ). Otherwise, alternating vertices of two connected components would introduce crossings.

<sup>2</sup> Given a circular drawing of  $G = (V, E)$ , two vertices  $\overrightarrow{a, b}$  partitions the vertices in  $V \setminus \{a, b\}$  into two sets that lie on the left side and right side of the ray  $\overrightarrow{ab}$ .



■ **Figure 7** In any clockwise vertex ordering of an outerplanar drawing,  $u, v$  must be the extreme vertices in the 2-connected components  $B_v$  and  $B_u$ , respectively.

Given an edge  $e'$  in  $C_v$ , we say  $e'$  covers  $v$  if the endpoints of  $e$  alternate with  $u$  and  $v$  in  $\delta_{G'}$ . Note that there is no edge covering  $v$  in  $\sigma(C_v)$ . Otherwise, such an edge would cross with edge  $uv$ . Therefore, in a valid untangling of  $\delta_G$ , it is necessary to move vertices of  $C_v$  in  $\delta_G$  such that no crossing is introduced in  $C_v$  and  $v$  is not covered by any edges in  $C_v$ . Similarly, the same claim holds also for  $C_u$ . We call such vertex moves *vertex unwrapping*. In the following, we consider how to find a valid unwrapping of  $v$  with the minimum number of vertex moves. The same operation will be also applied to  $C_u$ . Observe that, once  $u, v$  are both unwrapped, adding the edge  $e$  into the drawing does not introduce any crossings. The combination of these two unwrappings makes an optimal untangling. Here, we also consider the canonical vertex sequences and get the following Lemma 4.10. The proof is quite similar to the proof of Lemma 4.6 which concerns canonical untanglings.

► **Observation 4.9.** *There exists at least one 2-connected component  $B$  of  $C_v$  such that  $B$  contains  $v$  and no edge in the attachment of  $v$  (associated with  $B$ ) covers  $v$  in  $\delta_G$ .*

The reason for this observation is that either no 2-connected component  $B$  containing  $v$  contains an edge covering  $v$ , in which case  $v$  is already unwrapped and the statement is true for any such  $B$ . Or some 2-connected component  $B$  does contain a covering edge, but then the attachment of  $v$  in  $B$  cannot cover  $v$  due to planarity of  $\delta_{G'}$ .

► **Lemma 4.10.** *Let  $B$  be a 2-connected component of  $C_v$  that contains  $v$  such that the attachment of  $v$  contains no edge covering  $v$ . Each unwrapping  $U$  of  $v$  can be transformed into a canonical unwrapping  $U_c$  (associated with  $B$ ). Furthermore, the number of vertex moves in  $U_c$  is not greater than the number of vertex moves in the original unwrapping  $U$ .*

**Proof.** Given a unwrapping procedure  $U$  of  $v$ , let  $\delta_G^U$  be the drawing obtained after applying  $U$  on  $\delta_G$ . In  $\delta_G^U$ , the cyclic vertex ordering of  $B$  (clockwise or counterclockwise) must correspond to its Hamiltonian cycle ordering  $H(B)$ . Furthermore, the vertices of each attachment of  $B$  appear consecutively in  $\delta_G^U$ , including one vertex of  $B$ ; see Observation 4.4. Let  $A_1, \dots, A_k$  be the attachments of  $B$  in  $C_v$  (in this clockwise order in  $\delta_G^U$ ), let  $\sigma(A_i)$  be the clockwise vertex ordering of  $A_i$  in  $\delta_G$  for  $i \in \{1 \dots k\}$ . Consider the clockwise vertex ordering  $\sigma'_G$  where the vertices of  $B \cup C_u$  are ordered as in  $\delta_G^U$ . Furthermore, for each attachment  $A_i$  the vertices of  $A_i$  appear consecutively in the clockwise ordering  $\sigma(A_i)$ . Let  $\delta'_G$  be an arbitrary circular drawing where the vertices are ordered as  $\sigma'_G$ . Note that the vertex ordering of each attachment of  $B$  is  $\sigma(A_i)$  in  $\delta'_G$  as in the almost-planar drawing  $\delta_G$ , thus each attachment in  $\delta'_G$  is crossing-free. Moreover, in  $\delta'_G$  the vertices of  $B$  are ordered as in the planar drawing  $\delta_G^U$ , thus there is no crossing inside  $B$ . Overall, the vertex  $v$  is unwrapped in  $\delta'_G$ . It remains to prove that the untangling  $U'$ , which transforms  $\delta_G$  to  $\delta'_G$ , moves less than or equally many vertices as  $U$ . By construction each common subsequence of  $\delta_G$  and  $\delta_G^U$  is also a subsequence of  $\delta'_G$ , which implies this fact. ◀

By Lemma 4.10, we restrict our attention to canonical unwrappings. Fixing a 2-connected component  $B_v$  of  $C_v$  containing  $v$  such that no edge in the attachment (associated with  $B_v$ ) of  $v$  covers  $v$ , we consider the two possible canonical unwrappings of  $v$ , which respectively



order vertices of  $B$  clockwise along  $H(B)$  or its reversal, and compute the corresponding resulting clockwise vertex ordering  $\sigma_v$  and  $\sigma_v^{rev}$  of  $C_v$ . With the same idea, we get the clockwise vertex orderings  $\sigma_u$  and  $\sigma_u^{rev}$  of  $C_u$  by the canonical unwrappings of  $u$ . We then get the four optimal unwrappings, each of them transforming  $\delta_G$  to one of the vertex orderings  $(\sigma_v, \sigma_u)$ ,  $(\sigma_v^{rev}, \sigma_u)$ ,  $(\sigma_v, \sigma_u^{rev})$  and  $(\sigma_v^{rev}, \sigma_u^{rev})$ . Such optimal unwrappings can be computed in  $O(n^2)$  time; see [24]. We report the one that moves the minimum number of vertices as an optimal component-fixed untangling.

#### 4.4 Circular Untangling

Given an almost-planar drawing  $\delta_G$ , we claim that it is always possible to compute an optimal untangling procedure for  $\delta_G$  in  $O(n^2)$  time, where  $n$  is the number of vertices of  $G$ . In our approach, we use procedures described in Sections 4.1–4.3 as subroutines.

**The Approach.** *Step 1:* we compute an optimal component-fixed untangling  $U$  by applying the approach described in Section 4.3. An optimal component-fixed untangling  $U$  can be reported in  $O(n^2)$  time (see Theorem 4.5). *Step 2:* let  $m(U)$  be the number of vertex moves in  $U$ . we compare  $m(U)$  with  $\min\{|C_u|, |C_v|\}$ . If  $m(U) \leq \min\{|C_u|, |C_v|\}$ , then we report  $U$ . Otherwise, if  $m(U) > \min\{|C_u|, |C_v|\}$ , we know  $U$  is not an optimal untangling procedure. Because there exists a specific untangling procedure  $U'$  which moves exactly  $\min\{|C_u|, |C_v|\}$  vertices; see its description in the proof of Theorem 4.3. In this case, we compute and report this procedure  $U'$ . The second step takes linear time. In total, the whole procedure needs  $O(n^2)$  time.

**Correctness.** Let  $U_a$  be the untangling reported by our approach. Now, we show that  $U_a$  is indeed an optimal untangling of  $\delta_G$  by contradiction. Note that  $U_a$  has size bounded by  $\min\{|C_u|, |C_v|\}$  (*Step 2*). Suppose there exists an untangling  $U_{a'}$  which moves less vertices than  $U_a$ . Then  $U_{a'}$  moves less vertices than  $\min\{|C_u|, |C_v|\}$ . If so, there are vertices in both of  $|C_u|, |C_v|$  that remain fixed in  $U_{a'}$ . Thus,  $U_{a'}$  is a component-fixed untangling. It leads to a contradiction to the fact that  $U_a$  has its size bounded by the size of optimal component-fixed untangling (*Step 1*). Therefore,  $U_a$  is indeed an untangling of  $\delta_G$  with the minimum number of vertex moves.

► **Theorem 4.11.** *Given an almost-planar drawing  $\delta_G$  of an outerplanar graph  $G$ , an untangling of  $\delta_G$  with the minimum number of vertex moves can be computed in  $O(n^2)$  time, where  $n$  denotes the number of vertices in  $G$ .*

## 5 Conclusions and Discussions

We introduced and investigated the problem of untangling non-planar circular drawings. First from the computational side, we demonstrated the NP-hardness of the problem CIRCULAR UNTANGLING. Second, we studied the almost-planar circular drawings, where all crossings involve a single edge. We gave a tight upper bound of  $\lfloor \frac{n}{2} \rfloor - 1$  on the shift number and an  $O(n^2)$ -time algorithm to compute it. Open problems for future work include: (i) The parameterized complexity of computing the circular shifting, e.g., with respect to the number of crossings or the number of connected components. (ii) Generalization of our results for almost-planar drawings. (iii) Investigation of minimum untangling by other elementary moves such as swapping vertex pairs or moving larger chunks of vertices.

## References

- 1 Jasine Babu, Areej Khoury, and Ilan Newman. Every property of outerplanar graphs is testable. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016*, volume 60 of *LIPICs*, pages 21:1–21:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.APPROX-RANDOM.2016.21.
- 2 Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. The state of the art in visualizing dynamic graphs. In *16th Eurographics Conference on Visualization, EuroVis 2014 – State of the Art Reports*. Eurographics Association, 2014. doi:10.2312/eurovisstar.20141174.
- 3 Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *J. Comb. Theory, Ser. B*, 27(3):320–331, 1979. doi:10.1016/0095-8956(79)90021-2.
- 4 Sujoy Bhore, Prosenjit Bose, Pilar Cano, Jean Cardinal, and John Iacono. Dynamic Schnyder woods. *CoRR*, abs/2106.14451, 2021. arXiv:2106.14451.
- 5 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for book embedding problems. *J. Graph Algorithms Appl.*, 24(4):603–620, 2020. doi:10.7155/jgaa.00526.
- 6 Sujoy Bhore, Guangping Li, Martin Nöllenburg, Ignaz Rutter, and Hsiang-Yun Wu. Untangling circular drawings: Algorithms and complexity. *CoRR*, abs/2111.09766, 2021. arXiv:2111.09766.
- 7 Prosenjit Bose, Vida Dujmovic, Ferran Hurtado, Stefan Langerman, Pat Morin, and David R. Wood. A polynomial bound for untangling geometric planar graphs. *Discret. Comput. Geom.*, 42(4):570–585, 2009. doi:10.1007/s00454-008-9125-3.
- 8 Javier Cano, Csaba D. Tóth, and Jorge Urrutia. Upper bound constructions for untangling planar geometric graphs. In *Graph Drawing (GD’11)*, volume 7034 of *LNCS*, pages 290–295. Springer, 2011. doi:10.1007/978-3-642-25878-7\_28.
- 9 Gary Chartrand and Frank Harary. Planar permutation graphs. *Annales de l’Institut Henri Poincaré. Probabilités et Statistiques*, 3:433–438, 1967.
- 10 Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Embedding graphs in books: a layout problem with applications to VLSI design. *SIAM Journal on Algebraic Discrete Methods*, 8(1):33–58, 1987. doi:10.1137/0608002.
- 11 Josef Cibulka. Untangling polygons and graphs. *Discret. Comput. Geom.*, 43(2):402–411, 2010. doi:10.1007/s00454-009-9150-x.
- 12 Robert F. Cohen, Giuseppe Di Battista, Roberto Tamassia, and Ioannis G. Tollis. Dynamic graph drawings: Trees, series-parallel digraphs, and planar st-digraphs. *SIAM J. Comput.*, 24(5):970–1001, 1995. doi:10.1137/S0097539792235724.
- 13 Stephan Diehl and Carsten Görg. Graphs, they are changing. In *Graph Drawing (GD’02)*, volume 2528 of *LNCS*, pages 23–30. Springer, 2002. doi:10.1007/3-540-36151-0\_3.
- 14 Mark N. Ellingham, Emily A. Marshall, Kenta Ozeki, and Shoichi Tsuchiya. A characterization of  $K_{2,4}$ -minor-free graphs. *SIAM J. Discret. Math.*, 30(2):955–975, 2016. doi:10.1137/140986517.
- 15 Fabrizio Frati. Planar rectilinear drawings of outerplanar graphs in linear time. In *Graph Drawing (GD’20)*, volume 12590 of *LNCS*, pages 423–435. Springer, 2020. doi:10.1007/978-3-030-68766-3\_33.
- 16 Greg N. Frederickson. Searching among intervals and compact routing tables. *Algorithmica*, 15(5):448–466, 1996. doi:10.1007/BF01955044.
- 17 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 18 Xavier Goaoc, Jan Kratochvíl, Yoshio Okamoto, Chan-Su Shin, Andreas Spillner, and Alexander Wolff. Untangling a planar graph. *Discrete and Computational Geometry*, 42(4):542–569, January 2009. doi:10.1007/s00454-008-9130-6.

- 19 Mihyun Kang, Oleg Pikhurko, Alexander Ravsky, Mathias Schacht, and Oleg Verbitsky. Untangling planar graphs from a specified vertex position – hard cases. *Discret. Appl. Math.*, 159(8):789–799, 2011. doi:10.1016/j.dam.2011.01.011.
- 20 Martin Krzywinski, Jacqueline Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: an information aesthetic for comparative genomics. *Genome research*, 19(9):1639–1645, 2009. doi:doi:10.1101/gr.092759.109.
- 21 Sylvain Lazard, William J. Lenhart, and Giuseppe Liotta. On the edge-length ratio of outerplanar graphs. *Theor. Comput. Sci.*, 770:88–94, 2019. doi:10.1016/j.tcs.2018.10.002.
- 22 Chun-Cheng Lin, Yi-Yi Lee, and Hsu-Chun Yen. Mental map preserving graph drawing using simulated annealing. *Inf. Sci.*, 181(19):4253–4272, 2011. doi:10.1016/j.ins.2011.06.005.
- 23 Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *J. Visual Languages and Computing*, 6(2):183–210, 1995. doi:10.1006/jvlc.1995.1010.
- 24 Andy Nguyen. Solving cyclic longest common subsequence in quadratic time. *CoRR*, abs/1208.0396, 2012. arXiv:1208.0396.
- 25 János Pach and Gábor Tardos. Untangling a polygon. *Discret. Comput. Geom.*, 28(4):585–592, 2002. doi:10.1007/s00454-002-2889-y.
- 26 Alexander Ravsky and Oleg Verbitsky. On collinear sets in straight-line drawings. In *Graph-Theoretic Concepts in Computer Science (WG'11)*, volume 6986 of LNCS, pages 295–306. Springer, 2011. doi:10.1007/978-3-642-25870-1\_27.
- 27 Janet M. Six and Ioannis G. Tollis. A framework and algorithms for circular drawings of graphs. *J. Discrete Algorithms*, 4(1):25–50, 2006. doi:10.1016/j.jda.2005.01.009.
- 28 Janet M. Six and Ioannis G. Tollis. Circular drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 285–315. Chapman and Hall/CRC, 2013.
- 29 Maciej M. Sysło. Characterizations of outerplanar graphs. *Discrete Mathematics*, 26(1):47–53, 1979. doi:10.1016/0012-365X(79)90060-8.
- 30 Oleg Verbitsky. On the obfuscation complexity of planar graphs. *Theor. Comput. Sci.*, 396(1-3):294–300, 2008. doi:10.1016/j.tcs.2008.02.032.
- 31 Manfred Wieggers. Recognizing outerplanar graphs in linear time. In *Graphtheoretic Concepts in Computer Science, International Workshop, WG '86, Germany, 1986, Proceedings*, volume 246 of LNCS, pages 165–176. Springer, 1986. doi:10.1007/3-540-17218-1\_57.
- 32 Hsiang-Yun Wu, Martin Nöllenburg, and Ivan Viola. Graph models for biological pathway visualization: State of the art and future challenges. In *The 1st Workshop on Multilayer Nets: Challenges in Multilayer Network Visualization and Analysis*, Vancouver, Canada, October 2019. URL: <http://yun-vis.net/projects/bionet/visworkshop2019.pdf>.



# Algorithms and Complexity on Indexing Elastic Founder Graphs

Massimo Equi  

Department of Computer Science, University of Helsinki, Finland

Tuukka Norri  



Department of Computer Science, University of Helsinki, Finland

Jarno Alanko  

Department of Computer Science, University of Helsinki, Finland  
Faculty of Computer Science, Dalhousie University, Halifax, Canada

Bastien Cazaux  

LIRMM, Univ. Montpellier, CNRS, France

Alexandru I. Tomescu  

Department of Computer Science, University of Helsinki, Finland

Veli Mäkinen  

Department of Computer Science, University of Helsinki, Finland

---

## Abstract

We study the problem of matching a string in a labeled graph. Previous research has shown that unless the *Orthogonal Vectors Hypothesis* (OVH) is false, one cannot solve this problem in strongly sub-quadratic time, nor index the graph in polynomial time to answer queries efficiently (Equi et al. ICALP 2019, SOFSEM 2021). These conditional lower-bounds cover even deterministic graphs with binary alphabet, but there naturally exist also graph classes that are easy to index: E.g. *Wheeler graphs* (Gagie et al. *Theor. Comp. Sci.* 2017) cover graphs admitting a Burrows-Wheeler transform-based indexing scheme. However, it is NP-complete to recognize if a graph is a Wheeler graph (Gibney, Thankachan, ESA 2019).

We propose an approach to alleviate the construction bottleneck of Wheeler graphs. Rather than starting from an arbitrary graph, we study graphs induced from *multiple sequence alignments*. *Elastic degenerate strings* (Bernadini et al. SPIRE 2017, ICALP 2019) can be seen as such graphs, and we introduce here their generalization: *elastic founder graphs*. We first prove that even such induced graphs are hard to index under OVH. Then we introduce two subclasses that are easy to index. Moreover, we give a near-linear time algorithm to construct indexable elastic founder graphs. This algorithm is based on an earlier segmentation algorithm for gapless multiple sequence alignments inducing non-elastic founder graphs (Mäkinen et al., WABI 2020), but uses more involved techniques to cope with repetitive string collections synchronized with gaps. Finally, we show that one of the subclasses admits a reduction to Wheeler graphs in polynomial time.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis; Theory of computation → Pattern matching; Theory of computation → Sorting and searching; Theory of computation → Dynamic programming; Applied computing → Genomics

**Keywords and phrases** orthogonal vectors hypothesis, multiple sequence alignment, segmentation

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.20

**Related Version** *Full Version*: <https://arxiv.org/abs/2102.12822>

**Funding** This work was partly funded by the Academy of Finland (grants 309048 and 322595), by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFE BIO), and by the Helsinki Institute for Information Technology.



© Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 20; pp. 20:1–20:18

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

In string research, many different problems relate to the common question of how to handle a collection of strings. When such a collection contains very similar strings, it can be represented as some “high scoring” *Multiple Sequence Alignment (MSA)*, i.e., as a matrix  $\text{MSA}[1..m, 1..n]$  whose  $m$  rows are the individual strings each of length  $n$ , which may include special “gap” symbols such that the columns represent the aligned positions. While it is NP-hard to find an optimal MSA even under the simplest score of maximizing the number of identity columns (i.e., longest common subsequence length) [21], the central role of MSA as a model of biological evolution has resulted into numerous heuristics to solve this problem in practice [9]. In this paper, we assume an MSA as an input.

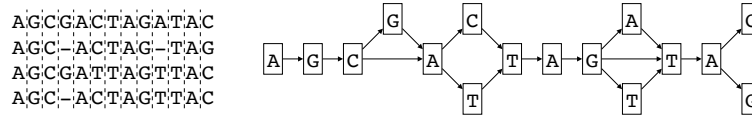
A simple way to define the problem of finding a match for a given string in the MSA is to ask whether the string matches a substring of some row (ignoring gap symbols). This leads to the widely studied problem of indexing repetitive text collections, see, e.g., references [24, 29, 30, 28, 27, 14, 15]. These approaches reducing an MSA to plain text reach algorithms with linear time complexities.

One feature worth considering is the possibility to allow a match to jump from any row to any other row of the MSA between consecutive columns. This property is usually referred to as *recombination* due to its connection to evolution, and leads to the graph representation of the MSA [26]. Figure 1a shows a simple solution, which consists in turning distinct characters of each column into nodes, and then adding the edges supported by row-wise connections. In this graph, a path whose concatenation of node labels matches a given string represents a match in the original MSA (ignoring gaps).

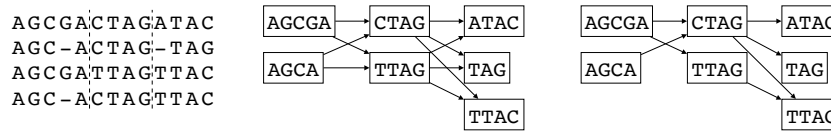
Aligning a sequence against a graph is not a trivial task. Only quadratic solutions are known [3, 25, 32], and this was recently proved to be a conditional lower bound for the problem [10]. Moreover, even attempting to index the graph to query the string faster presents significant difficulties. On one hand, indexes constructed in polynomial time still require quadratic-time queries in the worst case [35]. On the other hand, worst-case linear-time queries are possible, but this has the potential to make the index grow exponentially [34]. These might be the best results possible for general graphs and DAGs without any specific structural property, as the need for exponential indexing time to achieve sub-quadratic time queries constitutes another conditional lower bound for the problem [11].

Thus, if we want to achieve better performances, we have to make more assumptions on the structure of the input, so that the problem might become tractable. Following this line, a possible solution consists in identifying special classes of graphs that, while still able to represent any MSA, have a more limited amount of recombination, thus allowing for fast matching or fast indexing. This is the case for *Elastic Degenerate Strings (EDS)* [4, 5, 7, 6, 18], which can represent an MSA as a sequence of sets of strings, in which a match can span consecutive sets, using any one string in each of these (see Figure 1b, graph in the center). The advantage of this structure is that it is possible to perform expected-case subquadratic time queries [5]. However, EDS are still hard to index [16], and there is a lack of results on how to derive a “suitable” EDS from an MSA.

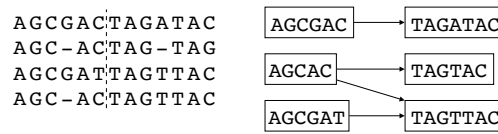
In this context, we propose a generalization of an EDS to what we call an *Elastic Founder Graph (EFG)*. An EFG is a DAG that, as an EDS, represents an MSA as a sequence of sets of strings; each set is called a *block*, and each string inside a block is represented as a labeled node. The difference with EDSes is that the nodes of two consecutive blocks are not forced to be fully connected. This means that, while in an EDS a match can always pair any string of a set with any string of the next set, in an EFG it might be the case that only some of



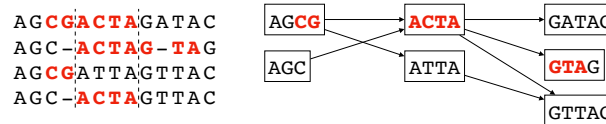
(a) A column-by-column segmentation of an MSA on the left, leading to the variation graph on the right.



(b) A different segmentation of the MSA, leading to the EDS in the center, and the EFG on the right. Notice that in an EDS every node is connected with all nodes to the right, while in an EFG edges are added only if their endpoints are consecutive in some row of the MSA (as in the case of variation graphs).



(c) A segmentation of the MSA that leads to a repeat-free EFG (i.e. no node label has another occurrence on some path of the EFG).



(d) A segmentation of the MSA that leads to a semi-repeat-free EFG (i.e. no node label has another occurrence on some path of the EFG, except as a prefix of another node in the same segment). An occurrence of query  $Q = \text{CGACTAGTA}$  in EFG is depicted in red. As can be seen, such query does not have an occurrence in a single row of the MSA.

■ **Figure 1** An MSA on the left, and various graph-based representations of it on the right. Notice that in all graphs (except the EDS) edges are added only between nodes that are observed as consecutive in some row of the MSA.

these pairings are allowed. Figure 1b illustrates these differences. Allowing for more selective connectivity between consecutive blocks also means that finding a match for a string in an EFG is harder than in an EDS. This is because EDSes are a special case of EFGs, hence the hardness results for the former carry to the latter. Specifically, a previous work [17] showed that, under the *Orthogonal Vectors Hypothesis* (OVH), no index for EDSes constructed in polynomial time can provide queries in time  $O(|Q| + |\tilde{T}|^\delta |Q|^\beta)$ , where  $|\tilde{T}|$  is the number of sets of strings,  $|Q|$  is the length of the pattern and  $\beta < 1$  or  $\delta < 1$ . Nevertheless, in this work we present an even tighter quadratic lower bound for EFGs, proving that, under OVH, an index built in time  $O(|E|^\alpha)$  cannot provide queries in time  $O(|Q| + |E|^\delta |Q|^\beta)$ , where  $|E|$  is the number of edges and  $\beta < 1$  or  $\delta < 1$ . Notice that  $|\tilde{T}|$  could even be  $o(|E|)$  (e.g. an EFG of two fully connected blocks), hence our lower bound more closely relates to the total size of an EFG. Additionally, the earlier lower bound [17] naturally applies only to indexing EDSes, and is obtained by performing many hypothetical fast queries; ours is derived by first proving a quadratic OVH-based lower bound for the *online* string matching problem in EFGs, and then using a general result [11] to simply translate this into an indexing lower bound.



Then, in order to break through these lower bounds, we identify two natural classes of EFGs, which respect what we call *repeat-free* and *semi-repeat-free* properties. The repeat-free property (Figure 1c) forces each string in each block to occur only once in the entire graph, and the semi-repeat-free property (Figure 1d) is a weaker form of this requirement. Thanks to these properties, we can more easily locate substrings of a query string in repeat-free EFGs and semi-repeat-free EFGs. In particular, (semi-)repeat-free EFGs and EDSes can be indexed in polynomial time for linear time string matching.

One might think that these time speedups come with a significant cost in terms of flexibility. Instead, the special structure of these EFGs do not hinder their expressive power. Indeed, we show that an MSA can be “optimally” segmented into blocks inducing a repeat-free or semi-repeat-free EFG. Clearly, this depends on how one chooses to define optimality. We consider three optimality notions: maximum number of blocks, minimum maximum block width, and minimum maximum block length. In Figure 1d, the first score is 3, second is 3, and the third is 5. The two latter notions stem from the earlier work on segmentations [31, 8], now combined with the (semi-)repeat-free constraint. The first is the simplest optimality notion, now making sense combined with the (semi-)repeat-free constraint.

For each of these optimality notions, we give a polynomial-time dynamic programming algorithm that converts an MSA into an optimal (semi-)repeat-free EFG if such exists. For the first and the third notion combined with semi-repeat-free constraint, we derive more involved solutions with almost optimal  $O(mn \log m)$  and  $O(mn \log m + n \log \log n)$  running time, respectively. In previous work [23], an (optimal)  $O(mn)$  time solution was given for the special case of MSA without gap symbols. Our new solution does not much build on the previous approach, which was based on a monotonicity property not anymore holding with gaps. Instead, we delve into the combinatorial properties of repetitive string collections synchronized with gaps and show how to use string data structures in this setting. The techniques can be easily adapted for other notions of optimality.

Another class of graphs that admits efficient indexing are Wheeler graphs [13], which offer an alternative way to model an EFG and thus a MSA. However, it is NP-complete to recognize if a given graph is a Wheeler graph [17], and thus, to use the efficient algorithmic machinery around Wheeler graphs [1] one needs to limit the focus on indexable graphs that admit efficient construction. Indeed, we show that any EFG that respects the repeat-free property can be reduced to a Wheeler graph in polynomial time. Interestingly, we were not able to modify this reduction to cover the semi-repeat-free case, leaving it open if these two notions of graph indexability have indeed different expressive power, and whether there are more graph classes with distinctive properties in this context.

## 2 Definitions

**Strings.** We denote integer intervals by  $[i..j]$ . Let  $\Sigma = \{1, \dots, \sigma\}$  be an alphabet of size  $|\Sigma| = \sigma$ . A *string*  $T[1..n]$  is a sequence of symbols from  $\Sigma$ , i.e.  $T \in \Sigma^n$ , where  $\Sigma^n$  denotes the set of strings of length  $n$  under the alphabet  $\Sigma$ . A *suffix* of string  $T[1..n]$  is  $T[i..n]$  for  $1 \leq i \leq n$ . A *prefix* of string  $T[1..n]$  is  $T[1..i]$  for  $1 \leq i \leq n$ . A *substring* of string  $T[1..n]$  is  $T[i..j]$  for  $1 \leq i \leq j \leq n$ . The *length* of a string  $T$  is denoted  $|T|$ . The *empty string* is the string of length 0. In particular, substring  $T[i..j]$  where  $j < i$  is the empty string. The *lexicographic order* of two strings  $A$  and  $B$  is naturally defined by the order of the alphabet:  $A < B$  iff  $A[1..i] = B[1..i]$  and  $A[i+1] < B[i+1]$  for some  $i \geq 0$ . If  $i+1 > \min(|A|, |B|)$ , then the shorter one is regarded as smaller. However, we usually avoid this implicit comparison by adding *end marker*  $\mathbf{0}$  to the strings. Concatenation of strings  $A$  and  $B$  is denoted  $AB$ .

**Elastic founder graphs.** As mentioned in the introduction, our goal is to compactly represent an MSA using an elastic founder graph. In this section we formalize these concepts.

A *multiple sequence alignment*  $\text{MSA}[1..m, 1..n]$  is a matrix with  $m$  strings drawn from  $\Sigma \cup \{-\}$ , each of length  $n$ , as its rows. Here  $- \notin \Sigma$  is the *gap* symbol. For a string  $X \in (\Sigma \cup \{-\})^*$ , we denote  $\text{spell}(X)$  the string resulting from removing the gap symbols from  $X$ .

Let  $\mathcal{P}$  be a *partitioning* of  $[1..n]$ , that is, a sequence of subintervals  $\mathcal{P} = [x_1..y_1], [x_2..y_2], \dots, [x_b..y_b]$ , where  $x_1 = 1$ ,  $y_b = n$ , and for all  $j > 2$ ,  $x_j = y_{j-1} + 1$ . A *segmentation*  $S$  of  $\text{MSA}[1..m, 1..n]$  based on partitioning  $\mathcal{P}$  is a sequence of  $b$  sets  $S^k = \{\text{spell}(\text{MSA}[i, x_k..y_k]) \mid 1 \leq i \leq m\}$  for  $1 \leq k \leq b$ ; in addition, we require for a (proper) segmentation that  $\text{spell}(\text{MSA}[i, x_k..y_k])$  is not an empty string for any  $i$  and  $k$ . We call set  $S^k$  a *block*, while  $\text{MSA}[1..m, x_k..y_k]$  or just  $[x_k..y_k]$  is called a *segment*. The *length* of block  $S^k$  is  $L(S^k) = y_k - x_k + 1$  and the *width* of block  $S^k$  is  $W(S^k) = |S^k|$ . Segmentation naturally leads to the definition of a founder graph through the block graph concept:

► **Definition 1** (Block Graph). A block graph is a graph  $G = (V, E, \ell)$  where  $\ell : V \rightarrow \Sigma^+$  is a function that assigns a string label to every node and for which the following properties hold.

1. Set  $V$  can be partitioned into a sequence of  $b$  blocks  $V^1, V^2, \dots, V^b$ , that is,  $V = V^1 \cup V^2 \cup \dots \cup V^b$  and  $V^i \cap V^j = \emptyset$  for all  $i \neq j$ ;
2. If  $(v, w) \in E$  then  $v \in V^i$  and  $w \in V^{i+1}$  for some  $1 \leq i \leq b - 1$ ; and
3. if  $v, w \in V^i$  then  $|\ell(v)| = |\ell(w)|$  for each  $1 \leq i \leq b$  and if  $v \neq w$ ,  $\ell(v) \neq \ell(w)$ .

With *gapless* MSAs, block  $S^k$  equals segment  $\text{MSA}[1..m, x_k..y_k]$ , and in that case the *founder graph* is a block graph induced by segmentation  $S$  [23]. The idea is to have a graph in which the nodes represent the strings in  $S$  while the edges retain the information of how such strings can be recombined to spell any sequence in the original MSA. With general MSAs with gaps, we consider the following extension, with an analogy to EDSes [5]:

► **Definition 2** (Elastic block and founder graphs). We call a block graph *elastic* if its third condition is relaxed in the sense that each  $V^i$  can contain non-empty variable-length strings. An *elastic founder graph* (EFG) is an elastic block graph  $G(S) = (V, E, \ell)$  induced by a segmentation  $S$  as follows: For each  $1 \leq k \leq b$  we have  $S^k = \{\text{spell}(\text{MSA}[i, x_k..y_k]) \mid 1 \leq i \leq m\} = \{\ell(v) : v \in V^k\}$ . It holds  $(v, w) \in E$  if and only if there exists  $k \in [1..b - 1]$  and  $t \in [1..m]$  such that  $v \in V^k$ ,  $w \in V^{k+1}$  and  $\text{spell}(\text{MSA}[t, x_k..y_{k+1}]) = \ell(v)\ell(w)$ .

By definition, (elastic) founder and block graphs are acyclic. For convention, we interpret the direction of the edges as going from left to right. Consider a path  $P$  in  $G(S)$  between any two nodes. The label  $\ell(P)$  of  $P$  is the concatenation of labels of the nodes in the path. Let  $Q$  be a query string. We say that  $Q$  *occurs* in  $G(S)$  if  $Q$  is a substring of  $\ell(P)$  for any path  $P$  of  $G(S)$ . Figure 1 illustrates such a query.

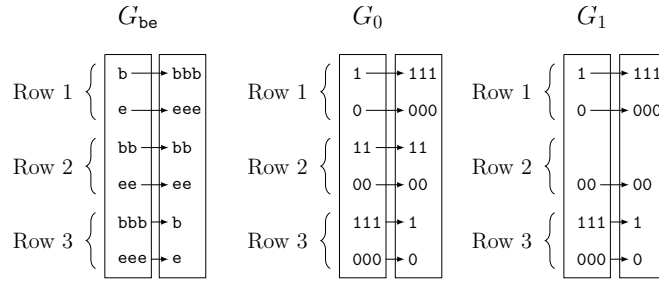
We use the same repeat-free definition as in the non-elastic case [23]:

► **Definition 3.** EFG  $G(S)$  is *repeat-free* if each  $\ell(v)$  for  $v \in V$  occurs in  $G(S)$  only as prefix of paths starting with  $v$ .

We also consider a variant that is relevant due to variable-length strings in the blocks:

► **Definition 4.** EFG  $G(S)$  is *semi-repeat-free* if each  $\ell(v)$  for  $v \in V$  occurs in  $G(S)$  only as prefix of paths starting with  $w \in V$ , where  $w$  is from the same block as  $v$ .

These definitions also apply to general elastic block graphs and to elastic degenerate strings as their special case.



■ **Figure 2** Gadgets  $G_{be}$ ,  $G_0$  and  $G_1$ . Each gadget is organized into three rows, each row encoding a different partitioning of the strings  $bbbb$ ,  $eeee$ ,  $0000$ ,  $1111$ . This ensures that, when combining these gadgets in Figure 3, edges can be controlled to go within the same row, or to the row below.

We note that not all MSAs admit a segmentation leading to a (semi-)repeat-free EFG, e.g. an alignment with rows  $-A$  and  $AA$ . However, our algorithms detect such cases, thus one can build an EFG consisting of just one block with the rows of the MSA (with gaps removed). Such EFGs can be indexed using standard string data structures to support efficient queries.

### 3 Conditional Hardness of Indexing EFGs

We show a reduction from *Orthogonal Vectors* (OV) to the problem of matching a query string in an EFG, continuing the line of research conducted on many related (degenerate) string problems [19, 10, 2, 16]. The OV problem is to find out if there exist  $x \in X$  and  $y \in Y$  such that  $x \cdot y = 0$ , given two sets  $X$  and  $Y$  of  $n$  binary vectors each. We construct string  $Q$  using  $X$  and graph  $G$  using  $Y$ . Then, we show that  $Q$  has a match in  $G$  if and only if  $X$  and  $Y$  form a “yes”-instance of OV. We condition our results on the following OV hypothesis, which is implied by the *Strong Exponential Time Hypothesis* [20].

► **Definition 5** (Orthogonal Vectors Hypothesis (OVH) [36]). *Let  $X, Y$  be the two sets of an OV instance, each containing  $n$  binary vectors of length  $d$ .<sup>1</sup> For any constant  $\epsilon > 0$ , no algorithm can solve OV in time  $O(\text{poly}(d)n^{2-\epsilon})$ .*

**Query String.** We build string  $Q$  by combining string gadgets  $Q_1, \dots, Q_n$ , one for each vector in  $X$ , plus some additional characters. To build string  $Q_i$ , first we place four  $b$  characters, then we scan vector  $x_i \in X$  from left to right. For each entry of  $x_i$ , we place substring  $Q_{i,h}$  consisting of four  $0$  characters if  $x_i[h] = 0$ , or four  $1$  characters if  $x_i[h] = 1$ . Finally, we place four  $e$  characters. For example, vector  $x_i = 101$  results into string

$$Q_i = bbbb Q_{i,1} Q_{i,2} Q_{i,3} eeee, \text{ where } Q_{i,1} = 1111, \quad Q_{i,2} = 0000, \quad Q_{i,3} = 1111.$$

Full string  $Q$  is then the concatenation  $Q = bbbb Q_1 Q_2 \dots Q_n eeee$ . The reason behind these specific quantities will be clear when discussing the structure of the graph.

**Elastic Founder Graph.** We build graph  $G$  combining together three different sub-graphs:  $G_L, G_M, G_R$  (for *left, middle* and *right*). Our final goal is to build a graph structured in three logical “rows”. We denote the three rows of  $G_M$  as  $G_{M1}, G_{M2}, G_{M3}$ , respectively. The

<sup>1</sup> In this section, keeping in line with the usual notation in the OV problem, we use  $n$  to denote the size of  $X$  and  $Y$ , instead of the number of columns of the MSA.

first and the third rows of  $G$ , along with sub-graphs  $G_L$  and  $G_R$  (introduced to allow slack), can match any vector. The second row matches only sub-patterns encoding vectors that are orthogonal to the vectors of set  $Y$ . The key is to structure the graph such that the pattern is forced to utilize the second row to obtain a full match. We present the full structure of the graph in Figure 3, which shows the graph built on top of vector set  $\{100, 011, 010\}$ . In particular,  $G_M$  consists of  $n$  gadgets  $G_M^j$ , one for each vector  $y_j \in Y$ . The key elements of these sub-graphs are gadgets  $G_{be}$ ,  $G_0$  and  $G_1$  (see Figure 2), which allow to stack together multiple instances of strings  $b^4$ ,  $e^4$ ,  $1^4$ ,  $0^4$ . The overall structure mimics the one by Equi et al. [10], except for the new idea from Figure 2.

**Detailed structure of the graph.** **Sub-graph  $G_L$**  (Figure 3a) consists of a starting segment with a single node labeled  $b^4$ , followed by  $n - 1$  sub-graphs  $G_L^1, \dots, G_L^{n-1}$ , in this order. Each  $G_L^i$  has  $d + 2$  segments, and is obtained as follows. First, we place a segment containing only one node with label  $b^4$ , then we place  $d$  other segments, each one containing two nodes with labels  $1^4$  and  $0^4$ . Finally, we place a segment containing two nodes with labels  $b^4$  and  $e^4$ .

The nodes in each segment are connected to all nodes in the next segment, with the exception of the last segment of each  $G_L^i$ : in this case, the node with label  $1^4$  and the one with label  $0^4$  are connected only to the  $e^4$ -node of the next (and last) segment of such  $G_L^i$ .

**Sub-graph  $G_R$**  (Figure 3c) is similar to sub-graph  $G_L$ , and it consists in  $n - 1$  parts  $G_R^1, \dots, G_R^{n-1}$ , followed by a segment with a single node labeled  $e^4$ . Part  $G_R^i$  has  $d + 2$  segments, and is constructed almost identically to  $G_L^i$ . The differences are that, in the first segment of  $G_R^i$ , we place two nodes labeled  $b^4$  and  $e^4$ , while in the last segment we place only one node, which we label  $e^4$ .

As in  $G_L$ , the nodes in each segment are connected to all nodes in the next segment, with the exception of the first segment of each  $G_R^i$ : in this case, the node labeled  $e^4$  has no outgoing edge.

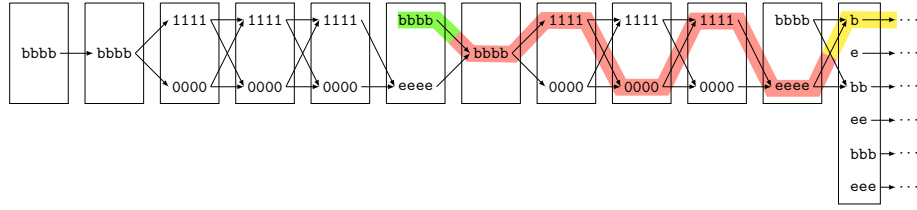
**Sub-graph  $G_M$**  (Figure 3b) implements the main logic of the reduction, and it uses three building blocks,  $G_{be}$ ,  $G_0$  and  $G_1$ , which are organized in three rows, as shown in Figure 2.

Sub-graph  $G_M$  has  $n$  parts,  $G_M^1, \dots, G_M^n$ , one for each of the vectors  $y_1, \dots, y_n$  in set  $Y$ . Each  $G_M^j$  is constructed, from left to right, as follows. First, we place a  $G_{be}$  gadget. Then, we scan vector  $y_j$  from left to right and, for each position  $h \in \{1, \dots, d\}$ , we place a  $G_0$  gadget if the  $h$ -th entry is  $y_j[h] = 0$ , or a  $G_1$  if  $y_j[h] = 1$ . Finally, we place another  $G_{be}$  gadget.

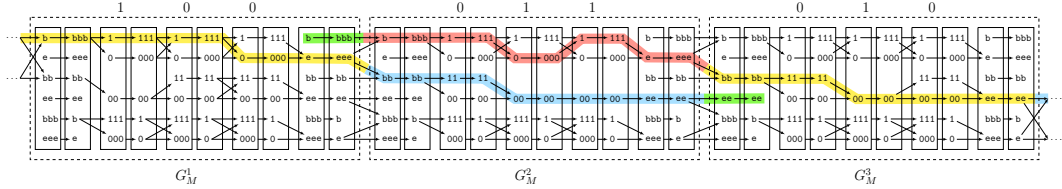
For the edges, we first consider each gadget  $G_M^j$  separately. Let  $G_h$  and  $G_{h+1}$ , be the gadgets encoding  $y_j[h]$  and  $y_j[h + 1]$ , respectively. We fully connect the nodes of  $G_h$  to the nodes of  $G_{h+1}$  row by row, respecting the structure of the segments. Then we connect, row by row, the  $b$ -nodes of the left  $G_{be}$  to the leftmost  $G_h$ , which encodes  $y_j[1]$ , and the nodes of the rightmost  $G_h$ , which encodes  $y_j[d]$ , to the  $e$ -nodes of the right  $G_{be}$ , again row by row. We repeat the same placement of the edges for every vector  $G_h, G_{h+1}$ ,  $1 \leq h \leq d - 1$ ; this construction is shown in Figure 3b.

To conclude the construction of  $G_M$ , we need to connect all the  $G_M^j$  gadgets together. Consider the right  $G_{be}$  of gadget  $G_M^j$ , and the left  $G_{be}$  of gadget  $G_M^{j+1}$ . The edges connecting these two gadgets are depicted in Figure 3b, which shows that following a path we can either remain in the same row or move to the row below, but we cannot move to the row above. Moreover, sub-pattern  $b^8$  can be matched only in the first and second row, while sub-pattern  $e^8$  only in the second and third rows.

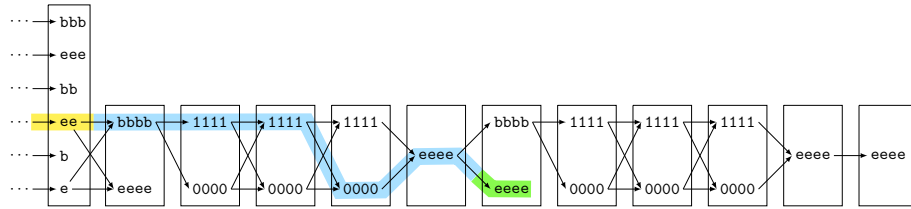
In proving the correctness of the reduction, we will refer to graphs  $G_{M1}$ ,  $G_{M2}$  and  $G_{M3}$  as the sub-graphs of  $G_M$  consisting of only the nodes and edges of the first, second and third row, respectively. Formally, for  $t \in \{1, 2, 3\}$ ,  $V_{Mt} \subset V$   $V_{Mt} \subset V$  is the set of



(a) Sub-graph  $G_L$ . The last segment belongs to sub-graph  $G_M$  and shows the connection.



(b) Sub-graph  $G_M$  for vectors  $y_1 = 100$ ,  $y_2 = 011$  and  $y_3 = 010$ . The dashed rectangles highlight the single  $G_M^j$  gadgets.



(c) Sub-graph  $G_R$ . The first segment belongs to sub-graph  $G_M$  and shows the connection.

**Figure 3** An example of graph  $G$ . To visualize the entire graph, watch the three sub-figures from top to bottom and from left to right. We also show two example occurrences of a query string  $Q$  constructed from  $x_1 = 101$ ,  $x_2 = 110$ ,  $x_3 = 100$  (left-most), and from  $x_1 = 101$ ,  $x_2 = 100$ ,  $x_3 = 110$  (right-most), respectively. We highlight each  $Q_i$  with a different color. Any such occurrence must pass through the middle row of  $G_M$ .

nodes placed in the  $t$ -th row of each  $G_{be}$ ,  $G_0$  or  $G_1$  gadget belonging to sub-graph  $G_M$ , and  $E_{Mt} = \{(v, w) \in E \mid v, w \in V_{Mt}\}$ . Thus,  $G_{Mt} = (V_{Mt}, E_{Mt})$ . We will use the notation  $G_{M2}^j$  to refer to the nodes belonging to both  $G_M^j$  and  $G_{M2}$ , excluding the ones in  $G_{M1}$  and  $G_{M3}$ , and the edges connecting them.

**Final graph  $G$**  is obtained by combining sub-graphs  $G_L$ ,  $G_M$  and  $G_R$ . To this end, we connect the nodes in the last segment of  $G_L$  with the  $b$ -nodes in the first and second row of the left  $G_{be}$  gadget of  $G_M^1$ . Finally, we connect the  $e$ -nodes in the second and third row of the right  $G_{be}$  gadget of  $G_M^n$  with both the  $b^4$ -node and  $e^4$ -node in the first segment of  $G_R$ . Figures 3a, 3b and 3c can be visualized together, in this order, as one big picture of final graph  $G$ . In Figures 3a and 3c we placed the adjacent segment of  $G_M$  to show the connection.

**OVH Conditional Hardness.** The proof of correctness is similar to the one by Equi et al. [10], but with adaptations to the elastic founder graph. The following technical lemma (whose proof is deferred to the full version of this paper) summarizes the structure of the possible matches of  $Q$  inside  $G$ . In it, we use the notation  $G_{M2}^j$  to indicate the nodes and edges that belong to the second row of  $G_M^j$ .

► **Lemma 6.**

- If string  $Q_i$  has a match in  $G_{M2}$ , then the path matching  $Q_i$  is fully contained in  $G_{M2}^j$ , for some  $1 \leq j \leq n$ . Moreover, each  $Q_{i,h}$  substring matches a path of two nodes which belong to the  $G_0$  or  $G_1$  gadget encoding  $y_j[h]$ .
- String  $Q_i$  has a match in  $G_{M2}$  if and only if there exist  $y_j \in Y$  such that  $x_i \cdot y_j = 0$ .
- String  $Q$  has a match in  $G$  if and only if a substring  $Q_i$  of  $Q$  has a match in the underlying sub-graph  $G_{M2}$  of  $G_M$ .

Our first lower bound is on matching a query string in an EFG without indexing.

► **Theorem 7.** For any constant  $\epsilon > 0$ , it is not possible to find a match for a query string  $Q$  into an EFG  $G = (V, E, \ell)$  in either  $O(|E|^{1-\epsilon}|Q|)$  or  $O(|E||Q|^{1-\epsilon})$  time, unless OVH fails. This holds even if restricted to an alphabet of size 4.

**Proof sketch.** First, Lemma 6 guarantees that string  $Q$  has a match in  $G$  if and only if there exist orthogonal vectors  $x_i \in X$  and  $y_j \in Y$ . Second, it is easy to check that the reduction requires linear time and space in the size  $O(nd)$  of the OV problem. Third, if we find a match for  $Q$  in  $G$  in  $O(|E|^{1-\epsilon}|Q|)$  or  $O(|E||Q|^{1-\epsilon})$  time, then we can decide if there is a pair of orthogonal vectors in  $O(nd \cdot (nd)^{1-\epsilon}) = O(n^{2-\epsilon}\text{poly}(d))$  time, contradicting OVH. ◀

We obtain the indexing lower bound by proving that the above reduction is a *linear independent-components (lic)* reduction, as defined by [11, Definition 3].

► **Theorem 8.** For any  $\alpha, \beta, \delta > 0$  such that  $\beta + \delta < 2$ , there is no algorithm preprocessing an EFG  $G = (V, E, \ell)$  in time  $O(|E|^\alpha)$  such that for any query string  $Q$  we can find a match for  $Q$  in  $G$  in time  $O(|Q| + |E|^\delta|Q|^\beta)$ , unless OVH is false. This holds even if restricted to an alphabet of size 4.

**Proof.** It is enough to notice that the reduction from OV that we presented is a *lic* reduction. Namely, (1) the reduction is correct and can be performed in linear time and space  $O(nd)$  (recall the proof of Theorem 7), and (2) query string  $|Q|$  is defined using only vector set  $X$  and it is independent from vector set  $Y$ , while elastic founder graph  $G$  is built using only vector set  $Y$  and it is independent from vector set  $X$ . Hence, Corollary 1 in [11] can be applied, proving our thesis. ◀

## 4 Indexing (Semi-)Repeat-Free EFGs

Since indexing a general EFG is hard, we turn our attention to repeat-free and semi-repeat-free EFGs. We show in this section that such EFGs are easy to index.

Let  $G = (V, E)$  be a (semi-)repeat-free founder/block graph. We show that it is sufficient to build an index on a string formed by concatenating labels of neighboring nodes. Namely, consider string  $D = \prod_{i \in \{1, 2, \dots, b\}} \prod_{v \in V^i, (v, w) \in E} \ell(w)^{-1} \ell(v)^{-1} \mathbf{0}$ , where  $X^{-1}$  is the reverse  $x_m x_{m-1} \cdots x_1$  of string  $X = x_1 x_2 \cdots x_m$ . The key feature requiring this reversed direction is that the lexicographic ranges of suffixes of  $D$  starting with  $\ell(v)^{-1}$  for each  $v \in V$  are distinct; this would not (on all inputs) hold on suffixes starting with  $\ell(v)$  in a forward concatenation if  $\ell(v)$  is a prefix of some  $\ell(u)$ , as it can be in the semi-repeat-free case.

As the reader can easily verify, the *expanded backward search* [23] developed for the case of gapless MSAs applied on  $D$  (in place of the forward concatenation therein) works also for repeat-free founder/block graphs; the feature of having variable-length of strings in a block is not used in the correctness analysis. In the following, we give an alternative solution for the semi-repeat-free case using suffix trees.



Let us consider a solution based on the *descending suffix walk*. This search can be supported e.g. by any (compressed) suffix tree [12, 33]. In the following, we assume the reader is familiar with the basic notions on suffix tree [22, Chapter 8]. Consider searching query  $Q[1..q]$  from right to left from the suffix tree of  $D$ . Consider finding a match for  $Q[j..q]$ , but there is no branch with  $Q[j-1]$ . If there is a branch with  $\mathbf{0}$ , there may be  $j'$  such that  $Q[j..j']$  is a label of some node  $v$  of the semi-repeat-free EFG, for some  $j' \leq q$ . To find such  $j'$  (or to find out it does not exist), we can then take suffix links to reach the locus corresponding to  $Q[j..j']$ , and continue the search with  $Q[1..j'-1]$ . This process is repeated until  $Q$  is found, or, if one cannot proceed,  $Q$  does not occur in  $G$ . For this to work, we need to have marked all loci of the suffix tree reached by  $\ell(v)^{-1}$  for all nodes  $v$  of the semi-repeat-free EFG. We stop taking suffix links when reaching a marked loci.

► **Theorem 9.** *A (semi-)repeat-free founder/block graph  $G = (V, E)$  or a (semi-)repeat-free elastic degenerate string can be indexed in polynomial time into a data structure occupying  $O(|D| \log |D|)$  bits of space, where  $|D| = O(L|E|)$  and  $L$  is the maximum length of a node label. Later, one can find out in  $O(|Q|)$  time if a given query string  $Q$  occurs in  $G$ .*

**Proof.** Consider the approach above. If and only if  $Q$  is reported to be found, there is a path in  $G$  that spells  $Q$ : If  $Q$  is found without taking any suffix links, it is a substring of  $D$  and thus a substring of a concatenation of labels of two neighboring nodes of  $G$ . Otherwise, suffix  $Q[j..q]$  is a prefix of a path starting with  $\ell(w) = Q[j..j']$  for some  $w \in V$ . Since  $\ell(w)^{-1}$  occurs in  $D$  only when followed by  $\mathbf{0}$  or  $\ell(v)^{-1}$  for  $(v, w) \in E$ , the search continues only on the in-neighbors of  $w$ . If  $Q$  is found before taking suffix links again,  $Q[1..j-1]$  is a suffix of  $\ell(v)$  for some  $v$  such that  $(v, w) \in E$ , and thus there is a path in  $G$  spelling  $Q$ . Otherwise, suffix  $Q[k..q]$  is a prefix of a path starting with  $\ell(v)\ell(w)$ , for some  $k \leq j-1$ . Continuing this shows that the claim is correct.

Clearly, the length of  $D$  is bounded by  $O(L|E|)$ . Construction of suffix tree on  $D$  can be done in linear time [12]. In polynomial time, the nodes of the suffix tree can be preprocessed with perfect hash functions, such that following a downward path takes constant time per step. Following a suffix link takes amortized constant time. ◀

Observe that  $|D| \leq 2mn$ , where  $m$  and  $n$  are the number of rows and number of columns, respectively, in the MSA from where the elastic founder graph is induced. That is, the index size is linear in the (original) input size. We also note that the index can be modified to report only matches that are (gap-oblivious) substrings of the MSA rows: Short patterns spanning only one edge are already such. Longer patterns can have only one occurrence in  $G$ , and it suffices to verify them with a regular string index on the MSA. Such modified scheme makes the approach functionality equivalent with wide range of indexes designed for repetitive collections [24, 29, 30, 28, 27, 14, 15] and shares the benefit of alignment-based indexes of Na et al. [29, 30, 28, 27] in reporting the aligned matches only once, where e.g. r-index [15] needs to report all occurrences.

Using compressed suffix trees, different space-time tradeoffs can be achieved. We develop an alternative compressed indexing scheme in Section 6 using Wheeler graphs.

## 5 Construction of (Semi-)Repeat-Free EFGs

Now that we have seen that (semi-)repeat-free EFGs are easy to index, it remains to consider their construction.

Consider a segmentation  $S = S^1, S^2, \dots, S^b$  that induces an EFG  $G(S) = (V, E)$ . We call such a segmentation (semi-)repeat-free or simply *valid*, if the resulting EFG is (semi-)repeat-free. A segment  $\text{MSA}[1..m, x_k..y_k]$  corresponding to block  $S^k = \text{spell}(\text{MSA}[1..m, x_k..y_k])$  of



such (semi-)repeat-free  $S$  is then analogously called a (semi-)repeat-free segment. In an earlier work on gapless MSAs [23], it was observed that a sufficient and necessary condition to check if a segment  $\text{MSA}[1..m, x_k..y_k]$  is repeat-free is to check that no string  $\text{MSA}[i, x_k..y_k]$ ,  $1 \leq i \leq m$ , occurs elsewhere in the MSA except at  $\text{MSA}[i', x_k..y_k]$ , for some  $1 \leq i' \leq m$ . Let us refine this condition with gaps. We say that segment  $\text{MSA}[1..m, x_k..y_k]$  is repeat-free (semi-repeat-free) if no string  $\text{spell}(\text{MSA}[i, x_k..y_k])$ ,  $1 \leq i \leq m$ , occurs elsewhere in the MSA except at (as a prefix of, respectively)  $\text{spell}(\text{MSA}[i', 1..n])[g(i', x_k)..g(i', y_k)]$ , for some  $1 \leq i' \leq m$ , where  $g(i', j)$  is  $j$  subtracted with the number of gaps at row  $i'$  of MSA up to column  $j$ . The earlier arguments [23, Sect. 5.1] carry over to showing that these are sufficient and necessary conditions for inducing a repeat-free and semi-repeat-free EFGs, respectively.

We consider three score functions for the valid segmentations, one maximizing the number of blocks, one minimizing the maximum width of a block, and one minimizing the maximum length of a block. The latter two have been studied earlier without the (semi-)repeat-free constraint, and non-trivial linear time solutions have been found [31, 8], while the first score function makes sense only with this new constraint.

Let  $s(j')$  be the optimal score of a semi-repeat-free segmentation  $S^1, S^2, \dots, S^b$  of prefix  $\text{MSA}[1..m, 1..j']$  for a selected scoring scheme. Then

$$s(j) = \bigoplus_{\substack{j' : 0 \leq j' < j, \\ \text{MSA}[1..m, j' + 1..j] \text{ is semi-repeat-free segment}}} w(s(j'), j', j), \quad (1)$$

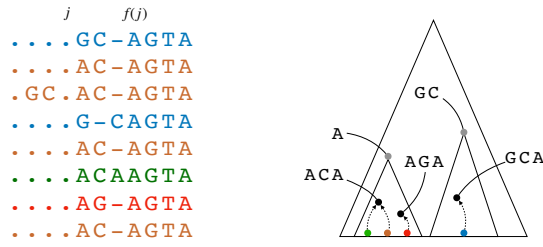
gives the optimal score of a semi-repeat-free segmentation  $S^1, S^2, \dots, S^b, S^{b+1}$  of  $\text{MSA}[1..m, 1..j]$ , where  $\bigoplus$  is an operator depending on the scoring scheme and  $w(x, j', j)$  is a function on the score  $x$  of the segmentation of  $S^1, S^2, \dots, S^b$  and on the last block  $S^{b+1}$  corresponding to  $\text{MSA}[1..m, j' + 1..j]$ . To fix this recurrence so that  $s(n)$  equals the maximum number of blocks over valid segmentations of  $\text{MSA}[1..m, 1..n]$ , set  $\bigoplus = \max$  and  $w(x, j', j) = x + 1$ . For initialization, set  $s(j) = 0$ . Moreover, when there is no valid segmentation for some  $j$ , set  $s(j) = -\infty$ . To fix this recurrence so that  $s(n)$  equals the minimum of maximum widths of blocks over valid segmentations of  $\text{MSA}[1..m, 1..n]$ , set  $\bigoplus = \min$  and  $w(x, j', j) = \max(x, |\{\text{spell}(\text{MSA}[i, j' + 1..j]) \mid 1 \leq i \leq m\}|)$ . For initialization, set  $s(j) = 0$ . Moreover, when there is no valid segmentation for some  $j$ ,  $s(j) = \infty$ . Finally, to fix this recurrence so that  $s(n)$  equals the minimum of maximum length of blocks over valid segmentations of  $\text{MSA}[1..m, 1..n]$ , set  $\bigoplus = \min$  and  $w(x, j', j) = \max(x, j - j')$ . For initialization, set  $s(j) = 0$ . Moreover, when there is no valid segmentation for some  $j$ , set  $s(j) = \infty$ .

The recurrence fixed for the latter case can be solved in  $O(mn)$  time when the input is a gapless MSA [23]. However, gaps affect most of the more involved techniques [23, 31, 8], so that we only know of a rather straightforward solution working in  $O(mn^2 \log m)$  time for this general case with gaps, for all three score functions and also for the repeat-free case (explicit proof omitted here, but the techniques developed later in this paper are sufficient for deriving such result). In what follows, we develop a different approach that works in  $O(mn \log m)$  time for the first and the last score function in the semi-repeat-free case. We leave it as an open question to obtain a faster algorithm for the second score function, and for the repeat-free case. We start with a simple observation:

► **Observation 10.** *If segment  $\text{MSA}[1..m, j + 1..f(j)]$  is semi-repeat-free, then segment  $\text{MSA}[1..m, j + 1..j']$  is semi-repeat-free for all  $j'$  such that  $f(j) < j' \leq n$ .*

Our goal is to compute for each  $j$  the smallest integer  $f(j)$  such that  $\text{MSA}[1..m, j + 1..f(j)]$  is a semi-repeat-free segment. These values can then be used for efficient evaluation of Eq. (1).

## 20:12 Indexing Elastic Founder Graphs



■ **Figure 4** Illustrating the  $O(m \log m)$  time algorithm to compute value  $f(j)$  for a given  $j$ . Node labels correspond to the string spelled from the root to the node. We assume ACA, AGA, and GCA only appear in the region of the MSA visualized, while GC and A appear also elsewhere.

► **Lemma 11.** *Let  $f(j)$  be the smallest integer such that  $\text{MSA}[1..m, j+1..f(j)]$  is a semi-repeat-free segment. We can compute all values  $f(j)$  in  $O(mn \log m)$  time.*

**Proof sketch.** Full proof is deferred to the full version of this paper. Here we explain the algorithm through the example of Fig. 4. We build a compact trie on the suffixes of the concatenation of MSA rows with gaps removed and special markers added between rows, that is, a *generalized suffix tree* [22, Chapter 8] on set  $\{\text{spell}(\text{MSA}[i, 1..n]) \mid 1 \leq i \leq m\}$ . For each column  $j$ , locate the subset  $W$  of (implicit) suffix tree nodes corresponding to  $\{\text{spell}(\text{MSA}[i, j+1..n]) \mid 1 \leq i \leq m\}$ ; these are the colored nodes in Fig. 4. If the number of leaves covered by the subtrees rooted at  $W$  is greater than  $m$ ,  $f(j)$  remains undefined. Otherwise, we know that  $f(j) \leq n$ , and our aim is to decrease the right boundary, starting with  $n$ , until we have reached column  $f(j)$ . The algorithm picks an arbitrary node in  $W$  and tries to replace it with its parent. This replacement is safe, if the new subtree covers only leaves already covered by  $W$ . Safe replacements are continued as long as possible; these are the black nodes in Fig. 4, while the gray nodes are unsafe replacements. These replacements place the rows into equivalence classes, each sharing the identified common prefix. For row  $i$  in an equivalence class with common prefix length  $k$ , one can then locate the smallest column  $f^i(j)$  with  $|\text{spell}(\text{MSA}[i, j+1..f^i(j)])| = k$ . E.g. for row  $i = 2$  in Fig. 4, we have  $f^2(j) = j + 3$ , as  $|\text{spell}(\text{AC} - \text{A})| = 3 = |\text{ACA}|$ . Finally,  $f(j) = \max_{i:1 \leq i \leq m} f^i(j)$ . At each column, at most  $m$  replacements are required and each replacement can be done in  $O(\log m)$  time, by maintaining the non-overlapping lexicographic intervals corresponding to the suffix tree nodes in a balanced search tree. ◀

Using these precomputed  $f(j)$  values, Algorithms 1 and 2 compute the scores of an optimal semi-repeat-free segmentation under the maximum number of blocks score and minimum of maximum block length score, respectively.

► **Theorem 12.** *After an  $O(mn \log m)$  time preprocessing, Algorithms 1 and 2 compute the scores  $\text{maxblocks}(n) = b$  and  $\text{minmaxlength}(n) = \max_{i:1 \leq i \leq b} L(S^i)$  of optimal semi-repeat-free segmentations  $S^1, S^2, \dots, S^b$  of  $\text{MSA}[1..m, 1..n]$  in  $O(n)$  and  $O(n \log \log n)$  time, respectively.*

**Proof sketch.** Regarding running time, Algorithm 1 and Algorithm 2 clearly take  $O(n)$  and  $O(n \log n)$  time, respectively, when implemented as described in their pseudo-codes. The correctness for Algorithm 1 follows from the fact that when computing the score at column  $j$ , all earlier segmentations that are safe to be extended with a new segment ending at  $j$  are considered. This argument can be formalized analogously for Algorithm 2, whose detailed proof of correctness, as well as running time improvement to  $O(n \log \log n)$  (by using a more efficient data structure for semi-infinite range queries) are given in the full version of this paper. ◀

■ **Algorithm 1** An  $O(n)$  time algorithm for finding an optimal semi-repeat-free segmentation maximizing the number of blocks.

---

**Input:** Right-extensions  $(j, f(j))$  sorted from smallest to largest order by second component:  $(j_1, f(j_1)), (j_2, f(j_2)), \dots, (j_{n-J}, f(j_{n-J}))$ , where  $J$  is such that  $f(j_{n-J+1}), f(j_{n-J+2}), \dots, f(j_n)$  are not defined.

**Output:** Score of an optimal semi-repeat-free segmentation maximizing the number of blocks.

```

1  $x \leftarrow 1$ ;  $\text{maxblocks}(0) \leftarrow 0$ ;  $\text{maxblocks}(j) = \text{maxscore} = -\infty$  for all  $0 < j \leq n$ ;
2 for  $j \leftarrow 1$  to  $n$  do
3   while  $j = f(j_x)$  do
4      $\text{maxscore} \leftarrow \max(\text{maxscore}, \text{maxblocks}(j_x))$ ;
5      $x \leftarrow x + 1$ ;
6    $\text{maxblocks}(j) \leftarrow \text{maxscore} + 1$ ;
7 return  $\text{maxblocks}(n)$ ;
```

---

■ **Algorithm 2** An  $O(n \log n)$  time algorithm for finding an optimal semi-repeat-free segmentation minimizing the maximum segment length. Minimization over an empty set is assumed to return  $\infty$ . Operation **Upgrade** $(k, v)$  sets key  $k$  to value  $v$  if the previous value is larger. Operation **RangeMin** $(a, b)$  returns the smallest value associated with keys in range  $[a..b]$ . Both operations can be supported in  $O(\log n)$  time with standard balanced search trees.

---

**Input:** Right-extensions  $(j, f(j))$  sorted from smallest to largest order by second component:  $(j_1, f(j_1)), (j_2, f(j_2)), \dots, (j_{n-J}, f(j_{n-J}))$ , where  $J$  is such that  $f(j_{n-J+1}), f(j_{n-J+2}), \dots, f(j_n)$  are not defined.

**Output:** Score of an optimal semi-repeat-free segmentation minimizing the maximum segment length.

```

1 Initialize one-dimensional search trees  $\mathcal{T}$  and  $\mathcal{I}$  with keys  $0, 1, 2, \dots, 2n$ , with all keys associated with values  $\infty$ ;
2  $x \leftarrow 1$ ;
3  $\text{minmaxlength}(0) \leftarrow 0$ ;
4 for  $j \leftarrow 1$  to  $n$  do
5   while  $j = f(j_x)$  do
6      $\mathcal{T}.\text{Upgrade}(j_x + \text{minmaxlength}(j_x), \text{minmaxlength}(j_x))$ ;
7      $\mathcal{I}.\text{Upgrade}(j_x + \text{minmaxlength}(j_x), -j_x)$ ;
8      $x \leftarrow x + 1$ ;
9    $\text{minmaxlength}(j) \leftarrow \min(\mathcal{T}.\text{RangeMin}(j + 1, \infty), \mathcal{I}.\text{RangeMin}(-\infty, j) + j)$ ;
10 return  $\text{minmaxlength}(n)$ ;
```

---

## 6 Connection to Wheeler Graphs

Wheeler graphs, also known as Wheeler automata, are a class of labeled graphs that admit an efficient index for path queries [13]. We now give an alternative way to index repeat-free elastic block graphs by transforming the graph into an equivalent Wheeler automaton.

We view a block graph as a nondeterministic finite automaton (NFA) by adding a new initial state and edges from the source node to the starts of the first block, and expanding each string of each block to a path of states. To conform with automata notions, we define that the label of an edge is the label of the destination node.

We denote the repeat-free NFA with  $F$ . First we determinize it with the standard subset construction for the reachable subsets of states. The states of the DFA are subsets of states of the NFA such that there is an edge from subset  $S_1$  to subset  $S_2$  with label  $c$  iff  $S_2$  is the set of states at the destinations of edges labeled with  $c$  from  $S_1$ . We only represent the subsets of states reachable from the subset containing only the initial state. We call the deterministic graph  $G$ . See Figures 5 and 6 for an example.

A DFA is indexable as a Wheeler graph if there exists an order  $<$  on the nodes such that if  $u < v$ , then every incoming path label to  $u$  is colexicographically smaller than every incoming path label to  $v$  (recall that the colexicographic order of strings is the lexicographic order of the reverses of the strings). The repeat-free property guarantees that the nodes at the ends of the blocks can be ordered among themselves by picking an arbitrary incoming path as the sorting key.

To make sure that the rest of the nodes are sortable, we modify the graph so that if a node is not at the end of a block, we make it so that the incoming paths to the node do not branch backward before the backward path reaches the end of a previous block. This is done by turning each block into a set of disjoint trees, where the roots of the trees are the ending nodes of the previous block, in a way that preserves the language of the automaton. The roots may have multiple incoming edges from the leaves of the previous tree. See Figure 7 for an example. The formal definition of the transformation and the proof of sortability are deferred to the full version of this paper. We denote the transformed graph with  $G'$  and obtain the following result:

► **Lemma 13.** *The number of nodes in  $G'$  is at most  $O(NW)$ , where  $W$  is the maximum number of strings in a block of  $F$  and  $N$  is the total number of nodes in  $F$ .*

The Wheeler order  $<$  of the transformed graph can be found by running the XBWT sorting algorithm on a spanning tree of the graph, as shown by Alanko et al. [1]. Finally, we can find the minimum equivalent Wheeler graph by running the general Wheeler graph minimization algorithm of Alanko et al. [1].

With the input graph now converted into a Wheeler graph, one can deploy succinct data structures supporting fast pattern matching [13, Lemma 4], leading to the following result:

► **Corollary 14.** *A repeat-free founder/block graph  $G$  or a repeat-free elastic degenerate string can be indexed in  $O(NW)$  time into a Wheeler-graph-based data structure occupying  $O(NW \log |\Sigma|)$  bits of space, where  $N$  is the total number of characters in the node labels of  $G$ ,  $W$  is the width of  $G$  (maximum number of strings in a block of  $G$ ), and  $\Sigma$  is the alphabet. Later, using the data structure, one can find out in  $O(|Q| \log |\Sigma|)$  time if a given query string  $Q$  occurs in  $G$ .*

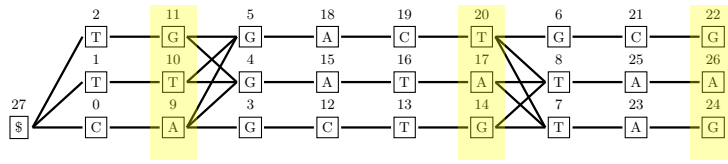


Figure 5 Repeat-free block NFA. The last columns of each block are highlighted.

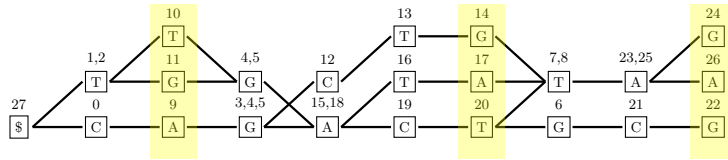


Figure 6 The DFA resulting from the subset construction for the NFA in Figure 5. The numbers above the nodes specify the subset of NFA states corresponding to the DFA state.

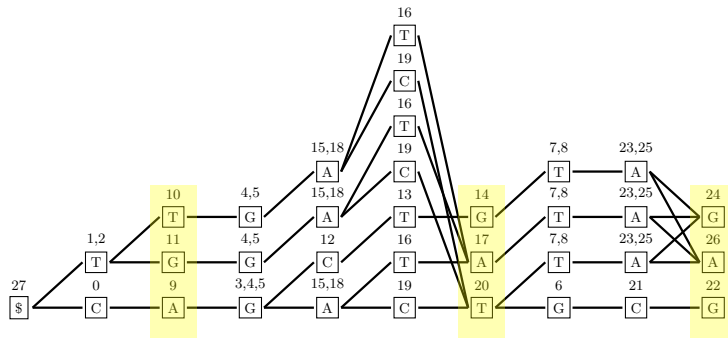


Figure 7 The Wheeler DFA resulting from running our Wheeler expansion algorithm on the DFA in Figure 6.

## 7 Discussion

There are many options how to optimize among the valid segmentations [31, 8]. We studied some of these here under the (semi-)repeat-free indexability constraint, but left open how to e.g. minimize the maximum number of distinct strings in a segment (i.e. *width* of the graph) [31], or how to control the over-expressiveness of the graph, in this context.

Other open problems include strengthening the conditional indexing lower bound to cover non-elastic founder graphs, and improving the running time for constructing (semi-)repeat-free elastic founder graphs.

We focused on the theoretical aspects of indexable founder graphs. Our preliminary experiments [23] show that the approach works well in practice on multiple sequence alignments without gaps. In our future work, we will focus on making the approach practical also in the general case.

## References

- 1 Jarno Alanko, Giovanna D’Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 911–930. SIAM, 2020.

- 2 Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Comparing degenerate strings. *Fundam. Informaticae*, 175(1-4):41–58, 2020.
- 3 Amihoud Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. *J. Algorithms*, 35(1):82–99, 2000.
- 4 Kotaro Aoyama, Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Online Elastic Degenerate String Matching. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:10, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CPM.2018.9.
- 5 Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even Faster Elastic-Degenerate String Matching via Fast Matrix Multiplication. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.21.
- 6 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Pattern matching on elastic-degenerate text with errors. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Palermo, Italy, September 26-29, 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2017. doi:10.1007/978-3-319-67428-5\_7.
- 7 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.*, 812:109–122, 2020. doi:10.1016/j.tcs.2019.08.012.
- 8 Bastien Cazaux, Dmitry Kosolobov, Veli Mäkinen, and Tuukka Norri. Linear time maximum segmentation problems in column stream model. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *String Processing and Information Retrieval - 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7-9, 2019, Proceedings*, volume 11811 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2019.
- 9 Maria Chatzou, Cedrik Magis, Jia-Ming Chang, Carsten Kemena, Giovanni Bussotti, Ionas Erb, and Cedric Notredame. Multiple sequence alignment modeling: methods and applications. *Briefings in Bioinformatics*, 17(6):1009–1023, November 2015.
- 10 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 11 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In Tomás Bures, Riccardo Dondi, Johann Gamper, Giovanna Guerrini, Tomasz Jurdzinski, Claus Pahl, Florian Sikora, and Prudence W. H. Wong, editors, *SOFSEM 2021: Theory and Practice of Computer Science - 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25-29, 2021, Proceedings*, volume 12607 of *Lecture Notes in Computer Science*, pages 608–622. Springer, 2021. doi:10.1007/978-3-030-67731-2\_44.
- 12 Martin Farach. Optimal suffix tree construction with large alphabets. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 137–143. IEEE, 1997.
- 13 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017.



- 14 Travis Gagie and Gonzalo Navarro. Compressed indexes for repetitive textual datasets. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019.
- 15 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in bwt-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020.
- 16 Daniel Gibney. An efficient elastic-degenerate text index? not likely. In *International Symposium on String Processing and Information Retrieval*, pages 76–88. Springer, 2020.
- 17 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 18 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate texts. In Frank Drewes, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden, March 6-9, 2017, Proceedings*, volume 10168 of *Lecture Notes in Computer Science*, pages 131–142, 2017. doi:10.1007/978-3-319-53733-7\_9.
- 19 Costas S. Iliopoulos and Jakub Radoszewski. Truly subquadratic-time extension queries and periodicity detection in strings with uncertainties. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPICs*, pages 8:1–8:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- 20 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.
- 21 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978. doi:10.1145/322063.322075.
- 22 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015. doi:10.1017/CB09781139940023.
- 23 Veli Mäkinen, Bastien Cazaux, Massimo Equi, Tuukka Norri, and Alexandru I. Tomescu. Linear time construction of indexable founder block graphs. In Carl Kingsford and Nadia Pisanti, editors, *20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 172 of *LIPICs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.WABI.2020.7.
- 24 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology*, 17(3):281–308, 2010.
- 25 U. Manber and S. Wu. Approximate string matching with arbitrary costs for text and hypertext. In *IAPR Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland*, pages 22–33, 1992.
- 26 Tobias Marschall, Manja Marz, Thomas Abeel, Louis Dijkstra, Bas E Dutilh, Ali Ghaffaari, Paul Kersey, Wigard Kloosterman, Veli Mäkinen, Adam Novak, et al. Computational pan-genomics: status, promises and challenges. *BioRxiv*, page 043430, 2016.
- 27 Joong Na, Hyunjoon Kim, Seunghwan Min, Heejin Park, Thierry Lecroq, Martine Leonard, Laurent Mouchard, and Kunsoo Park. FM-index of alignment with gaps. *Theoretical Computer Science*, 710, June 2016. doi:10.1016/j.tcs.2017.02.020.
- 28 Joong Chae Na, Hyunjoon Kim, Heejin Park, Thierry Lecroq, Martine Léonard, Laurent Mouchard, and Kunsoo Park. FM-index of alignment: A compressed index for similar strings. *Theoretical Computer Science*, 638:159–170, 2016. Pattern Matching, Text Data Structures and Compression. doi:10.1016/j.tcs.2015.08.008.
- 29 Joong Chae Na, Heejin Park, Maxime Crochemore, Jan Holub, Costas S. Iliopoulos, Laurent Mouchard, and Kunsoo Park. Suffix tree of alignment: An efficient index for similar data. In Thierry Lecroq and Laurent Mouchard, editors, *Combinatorial Algorithms - 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, Revised Selected Papers*, volume 8288 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.



## 20:18 Indexing Elastic Founder Graphs

- 30 Joong Chae Na, Heejin Park, Sunho Lee, Minsung Hong, Thierry Lecroq, Laurent Mouchard, and Kunsoo Park. Suffix array of alignment: A practical index for similar data. In Oren Kurland, Moshe Lewenstein, and Ely Porat, editors, *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, volume 8214 of *Lecture Notes in Computer Science*, pages 243–254. Springer, 2013.
- 31 Tuukka Norri, Bastien Cazaux, Dmitry Kosolobov, and Veli Mäkinen. Linear time minimum segmentation enables scalable founder reconstruction. *Algorithms Mol. Biol.*, 14(1):12:1–12:15, 2019.
- 32 Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in  $O(V + mE)$  time. *bioRxiv*, pages 216–127, 2017.
- 33 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.
- 34 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014.
- 35 Chris Thachuk. Indexing hypertext. *Journal of Discrete Algorithms*, 18:113–122, 2013. Selected papers from the 18th International Symposium on String Processing and Information Retrieval (SPIRE 2011).
- 36 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

# Partitioning $H$ -Free Graphs of Bounded Diameter

Christoph Brause ✉

Technische Universität Bergakademie Freiberg, Germany

Petr Golovach ✉ 

University of Bergen, Norway

Barnaby Martin ✉

Department of Computer Science, Durham University, UK

Daniël Paulusma ✉ 

Department of Computer Science, Durham University, UK

Siani Smith ✉

Department of Computer Science, Durham University, UK

---

## Abstract

A natural way of increasing our understanding of NP-complete graph problems is to restrict the input to a special graph class. Classes of  $H$ -free graphs, that is, graphs that do not contain some graph  $H$  as an induced subgraph, have proven to be an ideal testbed for such a complexity study. However, if the forbidden graph  $H$  contains a cycle or claw, then these problems often stay NP-complete. A recent complexity study (MFCS 2019) on the  $k$ -COLOURING problem shows that we may still obtain tractable results if we also bound the diameter of the  $H$ -free input graph. We continue this line of research by initiating a complexity study on the impact of bounding the diameter for a variety of classical vertex partitioning problems restricted to  $H$ -free graphs. We prove that bounding the diameter does not help for INDEPENDENT SET, but leads to new tractable cases for problems closely related to 3-COLOURING. That is, we show that NEAR-BIPARTITENESS, INDEPENDENT FEEDBACK VERTEX SET, INDEPENDENT ODD CYCLE TRANSVERSAL, ACYCLIC 3-COLOURING and STAR 3-COLOURING are all polynomial-time solvable for chair-free graphs of bounded diameter. To obtain these results we exploit a new structural property of 3-colourable chair-free graphs.

**2012 ACM Subject Classification** Mathematics of computing → Graph theory

**Keywords and phrases** vertex partitioning problem,  $H$ -free, diameter, complexity dichotomy

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.21

**Funding** *Daniël Paulusma*: Supported by the Leverhulme Trust (RPG-2016-258).

## 1 Introduction

Many well-known graph problems are NP-complete in general but become polynomial-time solvable under input restrictions. We focus on problems that partition the vertex set  $V$  of a graph  $G$  into sets  $V_1, \dots, V_k$  such that each  $V_i$  satisfies some property  $\pi_i$  and where  $V_1$  might have the extra condition of being large or being small. For instance, the  $k$ -COLOURING problem is to decide if  $V$  can be partitioned into sets  $V_1, \dots, V_k$ , called *colour classes*, such that each  $V_i$  is an independent set. To give another example, the INDEPENDENT SET problem is to decide if  $V$  can be partitioned into sets  $V_1$  and  $V_2$  where  $V_1$  is independent and  $|V_1| \geq p$  for some given integer  $p$ . Our underlying goal is to understand which graph properties ensure tractability of these problems and which properties cause the computational hardness. In the literature, input is restricted in various ways. In particular, *hereditary* graph classes have been considered.

Hereditary graph classes are the classes of graphs closed under vertex deletion. They form a natural and rich framework that cover many well-known graph classes (see, for example, [7]). Moreover, they enable a systematic study on the computational complexity of graph problems



© Christoph Brause, Petr Golovach, Barnaby Martin, Daniël Paulusma, and Siani Smith; licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 21; pp. 21:1–21:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

under input restrictions. The reason is that a graph class  $\mathcal{G}$  is hereditary if and only if  $\mathcal{G}$  can be characterized by a set  $\mathcal{F}_{\mathcal{G}}$  of forbidden induced subgraphs; we also say that  $\mathcal{G}$  is  $\mathcal{F}_{\mathcal{G}}$ -free. A natural starting point for a systematic study is the case where  $\mathcal{F}_{\mathcal{G}}$  has size 1, say  $\mathcal{F}_{\mathcal{G}} = \{H\}$  for some graph  $H$ . In this case the graphs in  $\mathcal{G}$  are said to be  $H$ -free. In other words, no graph  $G \in \mathcal{G}$  can be modified into  $H$  by a sequence of vertex deletions.

In the literature, there are extensive studies on  $H$ -free graphs; for example, on bull-free graphs [11] and claw-free graphs [12, 17]. There also exist several surveys on graph problems or graph parameters for hereditary graph classes that are characterized by a small set of forbidden induced subgraphs, for example, on COLOURING [16, 27] and clique-width [13].

A well-known dichotomy on COLOURING restricted to  $H$ -free graphs is due to Král', Kratochvíl, Tuza, and Woeginger [19]. Namely, COLOURING on  $H$ -free graphs is polynomial-time solvable if  $H$  is an induced subgraph of  $P_4$  (the 4-vertex path) or of  $P_1 + P_3$  (the disjoint union of  $P_1$  and  $P_3$ ) and it is NP-complete otherwise. Recently, similar but almost-complete dichotomies (up to one missing case each) were established for ACYCLIC COLOURING, STAR COLOURING and INJECTIVE COLOURING [4]. In particular, all these problems stay NP-complete if the forbidden induced subgraph  $H$  has a cycle or claw (the 4-vertex star  $K_{1,3}$ ). Moreover, the latter holds even if the number of colours  $k$  is fixed, i.e., not part of the input.

Several other vertex partitioning problems on  $H$ -free graphs stay NP-complete as well if  $H$  has a cycle or claw. Examples of such problems include (INDEPENDENT) FEEDBACK VERTEX SET [6, 23, 26], (INDEPENDENT) ODD CYCLE TRANSVERSAL [6, 10] and EVEN CYCLE TRANSVERSAL [24]. Hence, for all these problems, if  $H$  is a cycle or claw, then we need to add more structure to the class of input graphs in order to find tractable results for  $H$ -free graphs. One way of doing this is to bound the *diameter* of the input graph  $G$  for some problem. Our research question then becomes:

*Does bounding the diameter of an  $H$ -free graph lead to new tractability results?*

We note that graph classes of diameter at most  $d$  are hereditary if and only if  $d \leq 1$ . Many graph problems, such as COLOURING, ACYCLIC COLOURING, STAR COLOURING, CLIQUE and INDEPENDENT SET stay NP-complete even for graphs of diameter 2. The reason is that we can take an arbitrary graph  $G$  from such a problem and add a dominating vertex: the graph  $G$  is a yes-instance if and only if the new graph  $G'$  is a yes-instance.

This approach of adding a dominating vertex does not work if we consider 3-COLOURING. Mertzios and Spirakis [22] proved in a highly nontrivial way that 3-COLOURING is NP-complete even for triangle-free graphs of diameter at most 3. However, determining the complexity of 3-COLOURING for graphs of diameter 2 is a notoriously open problem (see [3, 9, 20, 22, 25]); we refer to [22] and [14] for subexponential-time algorithms for LIST 3-COLOURING on graphs of diameter at most 2. It is also known that ACYCLIC 3-COLOURING and STAR COLOURING, restricted to graphs of diameter at most  $d$ , are polynomial-time solvable if  $d = 2$  or  $d = 3$ , respectively, but NP-complete if  $d = 5$  or  $d = 8$ , respectively [8]. Moreover, the related problems NEAR BIPARTITENESS and INDEPENDENT FEEDBACK VERTEX SET, which we define below, are polynomial-time solvable for  $d = 2$  but NP-complete for  $d = 3$  [5]. These results also show that bounding the diameter on its own (without forbidding any induced graph  $H$ ) does not suffice.

We refer to [20, 21] for a number of results on 3-COLOURING and LIST 3-COLOURING for  $H$ -free graphs of bounded diameter, where  $H$  is a cycle or a *polyad*, which is a tree where exactly one vertex has degree at least 3 (polyads are also known as *subdivided stars*). One crucial observation in [20], based on an application of Ramsey's Theorem, was the starting point of this investigation: for all integers  $d, k$ , the  $k$ -COLOURING problem is constant-time solvable on claw-free graphs of diameter at most  $d$ . In the same paper [20], this result was generalized to the case where  $H$  is the *chair*, which is the graph obtained from the claw after subdividing exactly one of its edges. The chair is also known as the *fork*.

**Our Results.** We first consider the INDEPENDENT SET problem. For this problem we will prove new NP-hardness results that show that bounding the diameter does not help. To explain this, let the graph  $S_{h,i,j}$ , for  $1 \leq h \leq i \leq j$ , be the *subdivided claw*, which is the tree with one vertex  $x$  of degree 3 and exactly three leaves, which are at distance  $h$ ,  $i$  and  $j$  from  $x$ , respectively. Note that  $S_{1,1,1}$  is the claw  $K_{1,3}$ , the graph  $S_{1,1,2}$  is the chair and that every subdivided claw is a polyad. Let  $\mathcal{S}$  be the set of graphs, each connected component of which is a subdivided claw or a path. Alekseev [1] proved that for every finite set of graphs  $\mathcal{F}$ , if no graph from  $\mathcal{F}$  belongs to  $\mathcal{S}$ , then INDEPENDENT SET is NP-complete for the class of  $\mathcal{F}$ -free graphs. In Section 2, we show that exactly the same NP-completeness result holds for the class of  $\mathcal{F}$ -free graphs of diameter 2 if and only if  $|\mathcal{F}| = 1$ .

We then turn to the class of  $H$ -free graphs where  $H$  is a polyad. First, we focus on the case where  $H$  is the chair. In Section 3, we prove that for every integer  $d \geq 1$ , a number of vertex partitioning problems that require yes-instances to be 3-colourable become polynomial-time solvable on chair-free graphs of diameter at most  $d$ . The problems are ACYCLIC 3-COLOURING, STAR 3-COLOURING, NEAR-BIPARTITENESS, INDEPENDENT FEEDBACK VERTEX SET and INDEPENDENT ODD CYCLE TRANSVERSAL. We define these problems below.

Our proof is based on a common strategy. Namely, we determine the following for every chair-free 3-colourable non-bipartite input graph  $G$  of bounded diameter: either  $G$  has a constant number of 3-colourings or there exists a set  $S$  such that  $G - S$  has this property. We prove that we can let  $S$  be the set of private neighbours of some vertex  $u$  of a triangle on vertices  $u, v, w$ , that is, the vertices of  $S$  are adjacent to neither  $v$  nor  $w$ . We then consider each constructed 3-colouring  $c$  and determine in polynomial time if we can extend  $c$  to a solution for the vertex partitioning problem under consideration.

In Section 4 we prove that there is little hope of a full extension from the chair to arbitrary polyads  $H$ . To be more precise, we prove that for ACYCLIC 3-COLOURING, STAR 3-COLOURING and INDEPENDENT ODD CYCLE TRANSVERSAL, there exists a polyad  $H$  and a constant  $d$  such that each of these problems is NP-complete for the class of  $H$ -free graphs of diameter at most  $d$ . In the same section we give some relevant open problems.

**Additional Terminology.** A graph is *acyclic 3-colourable* or *star 3-colourable* if it is 3-colourable and every two colour classes induce a forest or a star forest, respectively (in this context, the  $P_1$  and  $P_2$  are seen as stars). The corresponding decision problems are ACYCLIC 3-COLOURING and STAR 3-COLOURING. A graph  $G$  is *near-bipartite* if its vertex set can be partitioned into an independent set  $I$  and a forest  $F$ ; we also say that  $I$  is an *independent feedback vertex set* of  $G$ . The problems NEAR-BIPARTITENESS and INDEPENDENT FEEDBACK VERTEX SET are to decide if a graph is near-bipartite or has an independent feedback vertex set of size at most  $k$  for some given integer  $k$ . A subset  $S \subseteq V$  of a graph  $G = (V, E)$  is an *independent odd cycle transversal* if  $S$  is independent and  $G - S$  is bipartite. Note that a graph is 3-colourable if and only if it has an independent odd cycle transversal. The INDEPENDENT ODD CYCLE TRANSVERSAL problem is to decide if a given graph has an independent odd cycle transversal of size at most  $k$  for some given integer  $k$ .

Let  $C_r$ ,  $P_r$  and  $K_r$  be the cycle, path and complete graph on  $r$  vertices. The graph  $G + H = (V(G) \cup V(H), E(G) \cup E(H))$  is the disjoint union of graphs  $G$  and  $H$ , and  $sG$  is the disjoint union of  $s$  copies of  $G$ . A graph  $G$  is  $\mathcal{H}$ -free if  $G$  is  $H$ -free for every  $H \in \mathcal{H}$ .

## 2 Independent Set

We let  $\mathcal{S}$  denote the set of graphs, each connected component of which is either a subdivided claw or a path. The following well-known result is due to Alekseev.

► **Theorem 1** ([1]). *Let  $\mathcal{F}$  be a finite set of graphs. If no graph from  $\mathcal{F}$  belongs to  $\mathcal{S}$ , then INDEPENDENT SET is NP-complete for  $\mathcal{F}$ -free graphs.*

We strengthen Theorem 1 to  $\mathcal{F}$ -free graphs of diameter 2 if  $|\mathcal{F}| = 1$ . We need two lemmas. We sketch the proof of the first lemma.

► **Lemma 2.** *For every  $r \geq 3$ , INDEPENDENT SET is NP-complete for  $C_r$ -free graphs of diameter 2.*

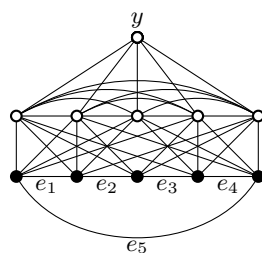
**Proof.** First suppose that  $r = 3$ . By Theorem 1, INDEPENDENT SET is NP-complete for  $C_3$ -free graphs. Let  $(G, k)$  be an instance of INDEPENDENT SET, where  $G$  is an  $n$ -vertex  $C_3$ -free graph. We may assume without loss of generality that  $n \geq 2$  and  $k \geq 2$ ; otherwise, the problem is trivial. We also assume that  $G$  has no dominating vertex. Otherwise, if  $u$  is a dominating vertex, then  $G$  has an independent set of size at least  $k \geq 2$  if and only if  $G - u$  has an independent set of size at least  $k$ . Hence, by the iterative deletion of universal vertices, we either solve the problem or obtain an equivalent instance without universal vertices. From  $G$  we now construct a graph  $G'$  as follows:

- Construct a copy of  $G$ .
- For every pair  $\{u, v\}$  of nonadjacent vertices of  $G$ ,
  - construct an arbitrary inclusion maximal independent set  $I_{uv}$  of  $G$  containing  $u$  and  $v$ ,
  - construct a vertex  $x_{uv}$  and make  $x_{uv}$  adjacent to every vertex of  $I_{uv}$ .
 Denote by  $X$  the set of all vertices  $x_{uv}$  constructed for the pairs of nonadjacent  $u$  and  $v$ .
- Construct an independent set of  $n^2$  vertices  $Y$  and make every vertex of  $Y$  adjacent to every vertex of  $X$ .

Observe that for every pair  $\{u, v\}$  of nonadjacent vertices of  $G$ ,  $I_{uv}$  can be constructed by the straightforward greedy procedure starting from the set  $\{u, v\}$ . This implies that  $G$  can be constructed in polynomial time. Our construction immediately implies that  $G'$  is triangle free, because the initial graph  $G$  is triangle-free and the neighbourhood of every vertex from  $X \cup Y$  is an independent set. We claim that the diameter of  $G'$  is at most 2, and moreover that  $G$  has an independent set of size at least  $k$  if and only if  $G'$  has an independent set of size at least  $k'$ . Proof details are omitted. ◀

► **Lemma 3.** *INDEPENDENT SET is NP-complete for  $K_{1,4}$ -free graphs of diameter 2.*

**Proof.** By Theorem 1, INDEPENDENT SET is NP-hard for  $(C_3, K_{1,4})$ -free graphs. Let  $(G, k)$  be an instance of INDEPENDENT SET, where  $G$  is a  $(C_3, K_{1,4})$ -free graph of order  $n$  and size  $m$ . We may assume without loss of generality that  $G$  is connected. Note that  $G$  is subcubic, that is, has maximum degree at most 3. We may assume without loss of generality that  $G$  has no vertices of degree 1 (as we can pick such vertices to be in the independent set and remove their neighbours from  $G$  until this operation can no longer be applied). To  $G$ , we add for every pair of edges  $e_1$  and  $e_2$  of  $G$  that do not share an end-vertex, a new vertex  $x_{e_1, e_2}$  and an edge between  $x_{e_1, e_2}$  and the two end-vertices of both  $e_1$  and  $e_2$ . We also add a new vertex  $y$  and all edges such that  $\{x_{e_1, e_2} : e_1, e_2\} \cup \{y\}$  induces a complete subgraph. Let  $G'$  be the resulting graph and let  $X := \{x_{e_1, e_2} : e_1, e_2\} \cup \{y\}$ . See also Fig. 1.



■ **Figure 1** The graph  $G'$  in the proof of Lemma 3 if  $G = C_5$ . The black vertices are those of  $G$  and the white vertices in the middle are  $x_{e_1e_3}, x_{e_2e_5}, x_{e_1e_4}, x_{e_3e_5},$  and  $x_{e_2e_4}$  from left to right.

For a contradiction, assume that there is some induced  $K_{1,4}$  in  $G'$ , say  $z$  is its centre vertex and  $z_1, z_2, z_3, z_4$  are its leaves. Since  $N_{G'}(x)$  can be partitioned into at most 3 cliques for each  $x \in X$ , we have  $z \in V(G)$ . Furthermore, since  $G$  is subcubic and  $X$  is a clique,  $X$  contains at least and at most one vertex of  $\{z_1, z_2, z_3, z_4\}$ , respectively. As we see now,  $N_G(z) \subseteq \{z_1, z_2, z_3, z_4\}$ . However, each vertex of  $X$  that is adjacent to  $z$ , has a neighbour in  $N_G(z)$ , which contradicts our supposition that  $z$  is the centre vertex of an induced  $K_{1,4}$ .

To show that  $G'$  has diameter 2, first consider an arbitrary vertex  $u$  of  $G$ . As  $G$  is connected, there is a vertex  $v \in N_G(u)$ . As  $G$  has minimum degree at least 2 and  $G$  is triangle-free, there are vertices  $u' \in N_G(u) \setminus \{v\}$  and  $v' \in N_G(v) \setminus \{u, u'\}$ . For  $e_1 = uu'$  and  $e_2 = vv'$ ,  $e_1$  and  $e_2$  do not have a common end-vertex,  $u, x \in N_{G'}[x_{e_1, e_2}]$ , and  $dist_{G'}(u, x) \leq 2$  for each  $x \in X$ . Additionally, if  $w \in V(G) \setminus \{u\}$  is a vertex with  $dist_G(u, w) \geq 3$ , then, since  $G$  is connected, there are edges  $e_3, e_4 \in E(G)$  such that  $u$  is incident to  $e_3$ ,  $w$  is incident to  $e_4$ , and the two edges  $e_3, e_4$  have no common incident vertex. Thus,  $u, w \in N_G(x_{e_3, e_4})$  and so  $dist_{G'}(u, w) \leq 2$ . As  $X$  is a clique, we conclude that  $G'$  has diameter 2.

We observe that  $I \cup \{y\}$  is an independent set of size  $k + 1$  in  $G'$  if  $I$  is an independent set of size  $k$  in  $G$ . Vice versa, given an independent set  $I'$  of size  $k + 1$  in  $G'$ , at most one of its vertices belongs to  $Y$  and  $I' \setminus X$  is an independent set of size at least  $k$  in  $G$ .

The graph  $G'$  can be constructed in time  $\mathcal{O}(m^2)$  and has at most  $n + m^2$  vertices. Vice versa, given an independent set  $I'$  in  $G'$ , the set  $I' \setminus X$  can be constructed in time  $\mathcal{O}(m^2)$ . ◀

We now strengthen Theorem 1, but can only do this for the case where  $|\mathcal{F}| = 1$ . For example, if  $\mathcal{F} = \{C_3, K_{1,4}\}$ , every  $\mathcal{F}$ -free graph has maximum degree 3. Then every  $\mathcal{F}$ -free graph with bounded diameter has constant size. Hence, INDEPENDENT SET is NP-complete for  $\mathcal{F}$ -free graphs by Theorem 1 but constant-time solvable for  $\mathcal{F}$ -free graphs of bounded diameter.

► **Theorem 4.** *Let  $H$  be a graph. If  $H \notin \mathcal{S}$ , then INDEPENDENT SET is NP-complete for  $H$ -free graphs of diameter at most 2. If  $H \in \mathcal{S}$ , then INDEPENDENT SET for  $H$ -free graphs is polynomially equivalent to INDEPENDENT SET for  $H$ -free graphs of diameter at most 2.*

**Proof.** Let  $H$  be a graph. First assume that  $H \notin \mathcal{S}$ . If  $H$  contains an induced cycle  $C_r$  for some  $r \geq 3$ , then the class of  $H$ -free graphs contains the class of  $C_r$ -free graphs, and we use Lemma 2. Otherwise  $H$  is a forest with either an induced  $K_{1,4}$  or a connected component that has at least two vertices of degree 3. In the first case, we use Lemma 3. In the second case, we reduce from INDEPENDENT SET for  $H$ -free graphs, which is NP-complete by Theorem 1. Let  $(G, k)$  be an instance of INDEPENDENT SET, where  $G$  is an  $H$ -free graph. We add a dominating vertex to  $G$  to obtain a graph  $G'$ . As  $H$  has no dominating vertex,  $G'$  is  $H$ -free. We also note that  $(G, k)$  and  $(G', k)$  are equivalent instances.

Now assume that  $H \in \mathcal{S}$ . Any polynomial-time algorithm for INDEPENDENT SET on  $H$ -free graphs can be used on  $H$ -free graphs of diameter 2. As INDEPENDENT SET is polynomial-time solvable for  $K_{1,3}$ -free graphs [28], we may assume that  $H \notin \{K_{1,3}, P_1, P_2, P_3\}$ . Any polynomial-time algorithm for INDEPENDENT SET on  $H$ -free graphs of diameter 2 can be used on  $H$ -free graphs of arbitrary diameter as follows. To the  $H$ -free input graph  $G$  we add a dominating vertex. As  $H \in \mathcal{S} \setminus \{K_{1,3}, P_1, P_2, P_3\}$ , this yields an  $H$ -free graph  $G'$  of diameter 2. We then observe that  $G$  has an independent set of size at least  $k$  if and only if  $G'$  has an independent set of size at least  $k$ . ◀

### 3 Chair-Free Graphs of Bounded Diameter

It follows from Ramsey's Theorem that every  $k$ -colourable  $K_{1,r}$ -free graph of diameter at most  $d$  has order bounded by a function in  $d, k, r$ ; see [20] for a proof of this observation. As a consequence, every problem that has the property that all its yes-instances are  $k$ -colourable for some constant  $k$  is constant-time solvable on  $K_{1,r}$ -free graphs of diameter at most  $d$ . We aim to extend the above observation to  $H$ -free graphs of bounded diameter when  $H$  is obtained from a star after at least one edge subdivision. In [20], a number of results are given for 3-COLOURING for such graph classes. We consider a variety of problems that require all the yes-instances to be 3-colourable. We focus on the first interesting case which is where  $H$  is the chair  $S_{1,1,2}$  (recall that the chair is obtained from the claw after subdividing one edge).

We need the following characterization of bipartite chair-free graphs, due to Alekseev. A *complex* is a bipartite graph that can be obtained by removing the edges of a possibly empty matching from a complete bipartite graph.

► **Theorem 5** ([2]). *If  $G$  is a connected bipartite chair-free graph, then  $G$  is a cycle or a path or a complex.*

It is well-known (cf. [18]) that finding the components of a graph by breadth-first search takes  $O(n + m)$  time. Let  $p$  be the number of components of a graph  $G$ ,  $n$  be its order, and  $m$  be its size. Then  $G$  is a forest if and only if  $p = n - m$ , which implies the following result.

► **Observation 6.** *If  $G$  is a graph, then we can decide if  $G$  is a forest in  $O(n + m)$  time.*

If  $T$  is a tree of order  $n$ , then its diameter is at most 2 if and only if its maximum degree equals  $n - 1$ . Therefore, we can decide whether a given graph is a forest each component of which is of diameter at most 2 in  $O(n + m)$  time. When working with vertex labellings, our findings imply the following observation.

► **Observation 7.** *If  $G$  is a graph and  $\ell$  is a vertex labelling of  $G$  with labels 1, 2, and 3, then we can decide whether  $\ell$  is a 3-colouring, star 3-colouring, or acyclic 3-colouring of  $G$  in  $O(n + m)$  time.*

It is also well-known that we can use breadth-first search for deciding whether a given graph  $G$  is bipartite and, if so, we are in a position to determine its parts in the same time. By this fact, we obtain the following result.

► **Observation 8.** *If  $G$  is a graph,  $k$  is an integer, and  $\ell$  is a vertex labelling of  $G$  with labels 1, 2, and 3, then we can decide whether one colour class of  $\ell$  is an independent feedback vertex set or an independent odd cycle transversal (of size at most  $k$ ) in  $O(n + m)$  time.*

To prove our results we need some more terminology. A *list assignment* of a graph  $G$  is a function  $L$  that gives each vertex  $u \in V(G)$  a (finite) *list of admissible colours*  $L(u) \subseteq \{1, 2, \dots\}$ . A colouring  $c$  *respects*  $L$  if  $c(u) \in L(u)$  for every  $u \in V(G)$ . If  $|L(u)| \leq 2$



for each  $u \in V(G)$ , then  $L$  is a 2-list assignment. The 2-LIST COLOURING problem is to decide if a graph  $G$  with a 2-list assignment  $L$  has a colouring that respects  $L$ . We use the following result.

► **Lemma 9** ([15]). *The 2-LIST COLOURING problem is solvable in  $O(n + m)$  time on graphs with  $n$  vertices and  $m$  edges.*

Let  $G$  be a graph of diameter  $d$  for some  $d \geq 1$  and  $S$  be some set of vertices of  $G$ . A vertex  $u \notin S$  is a *private* neighbour of a vertex  $v \in S$  with respect to  $S$  if  $u$  is adjacent to  $v$  but non-adjacent to any other vertex of  $S$ . We let  $P(v)$  be the set of private neighbours of  $v$  with respect to  $S$ . Let  $N_0 = S$  and, for  $i \in \{1, \dots, d\}$ , let  $N_i$  be the set of vertices that do not belong to  $N_0 \cup N_1 \cup \dots \cup N_{i-1}$  but do have a neighbour in  $N_{i-1}$ . As  $G$  has diameter at most  $d$ , we partition  $V(G)$  into the sets  $N_0, N_1, \dots, N_d$  (where some sets might be empty). We say that we *partition*  $V(G)$  *from*  $S$ . Note that this partitioning takes  $O(n + m)$  time by breadth-first search on the graph  $G'$  which is obtained from  $G$  by adding a new vertex  $u$  and all edges from  $u$  to every vertex of  $S$ .

In [20], it was shown that 3-COLOURING is polynomial-time solvable for chair-free graphs of bounded diameter. The first statement of Theorem 10 below is proven by similar but more precise arguments. The second statement is a new result that requires new arguments.

► **Theorem 10.** *Let  $d \geq 1$  be an integer and  $G$  be a chair-free non-bipartite graph of diameter  $d$  with  $n$  vertices and  $m$  edges.*

1. *We can decide whether  $G$  is 3-colourable in  $O(n + m)$  time.*
2. *If  $G$  is 3-colourable, then we find in  $O(n + m)$  time either all 3-colourings of  $G$ , or a triangle  $xyzx$  in  $G$  with exactly one vertex, say  $x$ , that has private neighbours and all 3-colourings of  $G - P(x)$  that can be extended to 3-colourings of  $G$ . In both cases, we find at most  $3^{9 \cdot 2^d + 8}$  3-colourings.*

**Proof.** We first check in constant time whether  $G$  is of order at most  $2d + 1$ . If so, then we can determine in constant time all 3-colourings of  $G$  and these are at most  $3^{2d+1}$ . Note that  $3^{2d+1} < 3^{9 \cdot 2^d + 8}$ . We proceed by assuming that  $G$  is of order at least  $2d + 2$  and claim that  $G$  contains a triangle. We prove this claim by contradiction: assume that  $G$  is triangle-free. As  $G$  is not bipartite, there is an odd cycle in  $G$ . Let  $x_1 x_2 \dots x_p x_1$  be a shortest one. As  $G$  is triangle-free and of diameter  $d$ , we find  $5 \leq p \leq 2d + 1$ , respectively. Moreover, as  $G$  is of order at least  $2d + 2$ , there is some vertex outside this cycle that has a neighbour on this cycle. Without loss of generality let us assume  $y$  with  $y \notin \{x_1, x_2, \dots, x_p\}$  is adjacent to  $x_1$ . As  $G$  is triangle-free,  $y$  does not have two consecutive neighbours on  $x_1 x_2 \dots x_p x_1$ . As  $G$  is chair-free and  $y$  is neither adjacent to  $x_2$  nor to  $x_p$ , we find that  $y$  must be adjacent to  $x_3$ . We repeat this argument and obtain that  $y$  is adjacent to  $x_{2q+1}$  for every  $0 \leq q \leq \lfloor \frac{p}{2} \rfloor$ . In particular  $y$  is adjacent to the two consecutive vertices  $x_1$  and  $x_p$ , a contradiction. We conclude that our assumption is false and that  $G$  contains a triangle.

We continue and show that we can compute a triangle, say  $T$ , of  $G$  in  $O(n + m)$  time. Let  $u$  be a vertex of  $G$ . We partition  $V(G)$  from  $\{u\}$  and note that breadth-first search computes a breadth-first tree  $F$ , that is,  $F$  is a spanning tree of  $G$  such that each vertex of  $N_i$  has distance  $i$  to  $u$  in  $F$  for any  $i$ . As  $G$  is not bipartite, there has to be an edge  $e$  and an integer  $i$  such that  $e$  is incident to two vertices of  $N_i$ . We can compute such an edge that additionally minimizes  $i$  in  $O(n + m)$  time. By adding this edge to  $F$ , we find an odd cycle  $C$  in  $G$ . As  $F$  is of diameter at most  $2d$ , we find that  $C$  has at most  $2d + 1$  vertices. Hence, we can determine in constant time a shortest induced odd cycle, say  $C'$ , in  $G[V(C)]$ . We check in constant time whether  $C'$  is a triangle. If not, then  $C'$  is of order at least 5. As

$G$  is of order at least  $2d + 2$ , there is a vertex outside  $C'$  that has a neighbour on  $C'$ . We compute such a vertex, say  $y$ , in  $O(n + m)$  time. As shown above,  $y$  has two consecutive neighbours on  $C'$ . As  $C'$  has at most  $2d + 1$  vertices, we can find such two vertices, and thus a triangle in  $G$ , in constant time.

Let  $\{x, y, z\}$  be the vertex set of the triangle  $T$ . We partition  $V(G)$  from  $V(T)$  in  $O(n + m)$  time. We additionally determine all private neighbours of the vertices of  $T$  and all vertices of  $N_1$  that are adjacent to all vertices of  $T$  in linear time. If there is a vertex of the latter type, then  $G$  is not 3-colourable. Thus, we focus on the case where each vertex of  $N_1$  is adjacent to at most two vertices of  $T$ . We compute in linear time the set  $N_1^*$  of all vertices of  $N_1$  that have two neighbours of  $T$ . Clearly,  $S = N_1 \setminus N_1^*$  consists of all private neighbours of the vertices of  $T$ , and its computation takes linear time. We proceed by considering  $G - N_1$ . We check in linear time if this graph has at most  $9 \cdot 2^d + 2$  vertices.

Let us consider the subcase where  $G - N_1$  has at least  $9 \cdot 2^d + 3$  vertices. We claim that  $G$  is not 3-colourable and prove this claim by contradiction: assume that  $G$  is 3-colourable. Hence,  $G$  is  $K_4$ -free. Recall that every vertex of  $N_1$  has at most two neighbours on  $T$ . Let  $i \geq 1$  and  $u$  be a vertex of  $N_i$ . As  $G$  is chair-free, the neighbours of  $u$  in  $N_{i+1}$  form a clique. As  $G$  is  $K_4$ -free, we obtain that  $u$  has at most 2 neighbours in  $N_{i+1}$ . It follows that

$$3 + 9 \cdot 2^d \leq |N_0| + |N_2| + |N_3| + \dots + |N_d| \leq 3 + |N_2| \cdot \sum_{i=2}^d 2^{i-2} < 3 + |N_2| \cdot 2^{d-1}.$$

Hence,  $|N_2| > 18$ . We let  $N_2^*$  be the neighbours of  $N_1^*$  in  $N_2$ . Consider the set  $N_{xy}$  of common neighbours of  $x$  and  $y$  in  $N_1^*$ . The set  $N_{xy}$  is an independent set as  $G$  is  $K_4$ -free. Every vertex  $u \in N_2^*$  with a neighbour  $v$  in  $N_{xy}$  must be adjacent to every vertex in  $N_{xy}$ , as  $G$  is chair-free. For the same reason, no vertex of  $N_1^*$  has two non-adjacent neighbours in  $N_2^*$ . As  $G$  is  $K_4$ -free, this means that there are at most two vertices in  $N_2^*$  that are adjacent to the vertices of  $N_{xy}$ . By applying the same reasoning for every other pair of vertices of  $T$ , we find that  $N_2^*$  has size at most 6. Thus,  $|N_2 \setminus N_2^*| > 12$ . As every vertex of  $N_1$  has at most two neighbours in  $N_2$ , it follows that  $|S| > 6$ . We consider the subcase where at least two vertices, say  $x$  and  $y$ , of  $T$  have a private neighbour. Assume that  $x$  has two non-adjacent private neighbours  $u$  and  $v$  in  $S$ . Then these three vertices, together with  $y$  and a private neighbour  $w \in S$  of  $y$  induce a chair unless  $w$  is adjacent to at least one of  $u$  and  $v$ . If  $w$  is adjacent to  $u$  but not to  $v$ , then  $\{u, v, w, x, z\}$  induces a chair. Hence,  $w$  is adjacent to both  $u$  and  $v$ , but then  $\{u, v, w, y, z\}$  induces a chair. Therefore, the private neighbours of every vertex of  $T$  form a clique. As  $G$  is 3-colourable, we find  $|S| \leq 6$  if at least two vertices of  $T$  have private neighbours. Thus, we obtain that all vertices of  $S$  are adjacent to a common vertex, say  $x$ , of  $T$ . As  $G$  is 3-colourable, we find that  $G[S]$  is bipartite. Therefore, we partition  $S$  into two independent sets  $A$  and  $B$  (one of these two sets might be empty). As  $G$  is chair-free and as  $A$  is independent, the vertices of  $A$  share the same set of neighbours in  $N_2$ . Similarly, the vertices of  $B$  share the same set of neighbours in  $N_2$ . As  $G$  is chair-free and  $K_4$ -free, the neighbourhood of every vertex of  $A \cup B$  is a clique of size at most 2. We conclude that the total number of vertices in  $N_2$  with a neighbour in  $S$  is at most 4, a contradiction as  $|N_2 \setminus N_2^*| > 12$ . We find that  $G$  is not 3-colourable and proceed by assuming that  $G - N_1$  has at most  $9 \cdot 2^d + 2$  vertices.

We consider every vertex labelling of  $G - N_1$  with labels 1, 2, 3 and determine in  $O(n + m)$  time which ones lead to a 3-colouring of  $G$ . We discard those labellings which are not a 3-colouring of  $G - N_1$ . Given a 3-colouring of  $G - N_1$ , each vertex of  $N_1^*$  receives the remaining available label that is not used for its neighbours of  $T$ . Note that this assignment takes linear time. We discard in  $O(n + m)$  time those labellings which do not lead to a

3-colouring of  $G - S$ . Let us take an arbitrary 3-colouring of  $G - S$ . We assign lists to the vertices of  $G$  as follows: we set  $L(u) = \{i\}$ , where  $i$  is the label of  $u$ , if  $u \notin S$  and we set  $L(u) = \{1, 2, 3\} \setminus \{i\}$ , where  $i$  is the label of the unique neighbour of  $u$  on  $T$ , if  $u \in S$ . Thus, checking whether a given 3-colouring of  $G - S$  leads to a 3-colouring of  $G$  takes  $O(n+m)$  time as  $(G, L)$  is an instance of 2-LIST COLOURING (cf. Lemma 9). We discard those 3-colouring of  $G - S$  which do not lead to a 3-colouring of  $G$ . If no 3-colouring of  $G - S$  lead to a 3-colouring of  $G$ , then  $G$  is not 3-colourable. Hence, we proceed by assuming that at least one does, and so we find that  $G$  is 3-colourable. As there are at most  $3^{9 \cdot 2^d + 2}$  vertex labellings of  $G - N_1$ , we can determine all 3-colourings of  $G - S$  that can be extended to 3-colourings of  $G$  in  $O(n+m)$  time, and there are at most  $3^{9 \cdot 2^d + 2}$  such colourings of  $G - S$ .

If  $S = \emptyset$ , then  $G - S$  equals  $G$ . We consider the subcase where at least two vertices of  $T$  have a private neighbour. As shown above, the private neighbours of every vertex of  $T$  form a clique. If  $|S| > 6$ , which we check in constant time, then  $G$  is not 3-colourable. Otherwise, as we have at most  $3^{9 \cdot 2^d + 2}$  3-colourings of  $G - S$ , we find at most  $3^{9 \cdot 2^d + 8}$  3-colourings of  $G$  and their computation takes  $O(n+m)$  time. We finally consider the subcase where all vertices of  $S$  are adjacent to a single vertex, say  $x$ , of  $T$ . We conclude that  $S = P(x)$ , which completes our proof.  $\blacktriangleleft$

We are now in a position to prove our main result of this section.

**► Theorem 11.** *If  $d \geq 1$ , then 3-COLOURING, ACYCLIC 3-COLOURING, STAR 3-COLOURING, INDEPENDENT ODD CYCLE TRANSVERSAL, INDEPENDENT FEEDBACK VERTEX SET, and NEAR-BIPARTITENESS can be solved in  $O(n+m)$  time for chair-free graphs of diameter at most  $d$ .*

**Proof.** Let  $G$  be a chair-free graph of diameter at most  $d$  with  $n$  vertices and  $m$  edges. Note that  $G$  is acyclic 3-colourable or star 3-colourable only if  $G$  is 3-colourable. Moreover, if  $I$  is an independent set of  $G$  for which  $G - I$  is a bipartite graph, then  $G$  is 3-colourable. Hence, our problems require all the yes-instances to be 3-colourable. If  $d = 1$ , then  $G$  is 3-colourable if and only if  $G$  has at most 3 vertices, and so each of our problems can be solved in constant time. We proceed by assuming  $d \geq 2$  and check in  $O(n+m)$  time whether  $G$  is bipartite.

**Case 1.**  $G$  is bipartite.

Note that  $G$  is 3-colourable, near-bipartite, and has an independent odd cycle transversal of size at most  $k$  for any integer  $k$ . We can determine the parts, say  $S_1$  and  $S_2$ , of  $G$  in  $O(n+m)$  time. We may assume without loss of generality that  $|S_1| \geq |S_2|$ . We check in constant time whether  $|S_1| + |S_2| \leq \max\{8, 2d\}$  and if so, then we can solve each of our problems in constant time. Otherwise, we find that  $|S_1| \geq 5$ . As bipartite graphs of maximum degree at most 2 and diameter at most  $d$  are paths or cycles of at most  $2d$  vertices, we find that  $G$  has a vertex of degree at least 3, and so  $G$  is a complex by Theorem 5.

We first claim that in the case where  $G$  is a complex with  $|S_1| \geq 5$ ,  $G$  is star 3-colourable if  $|S_2| \leq 2$  and acyclic 3-colourable only if  $|S_2| \leq 2$ . Note that this claim completes the bipartite case for ACYCLIC 3-COLOURING and STAR 3-COLOURING as we can decide whether  $|S_2| \leq 2$  or not in constant time and as every star 3-colouring of a graph is acyclic. We prove our claim as follows: If  $|S_2| \leq 2$ , then, for any  $s \in S_2$ ,  $G - s$  is a forest each component of which is of diameter at most 2, and thus  $G$  is star 3-colourable with colour classes  $S_1, S_2 \setminus \{s\}$ , and  $\{s\}$ . If  $|S_2| \geq 3$ , then let  $c$  be an arbitrary 3-colouring of  $G$ . By the pigeonhole principle there exists a colour class  $X$  of  $c$  that contains at least two vertices of  $S_1$ , and so  $X \cap S_2 = \emptyset$ . As  $|S_2| \geq 3$ , there are two vertices  $s_2, s'_2 \in S_2$  that are coloured alike. As  $|S_1| \geq 5$ , and as  $s_2$  and  $s'_2$  are of degree at least  $|S_1| - 1$ , we find that  $s_2$  and  $s'_2$  have at least three common

neighbours in  $S_1$  two of which, say  $s_1$  and  $s'_1$ , are coloured alike. Hence,  $s_1 s_2 s'_1 s'_2 s_1$  is a bichromatic 4-cycle. We conclude that every 3-colouring of  $G$  is not acyclic, which completes the proof of our claim.

It remains to consider INDEPENDENT FEEDBACK VERTEX SET for complexes with at least 9 vertices. Let  $k$  be an arbitrary integer. We claim that in the case where  $G$  is a complex with  $|S_1| \geq 5$ ,  $G$  has an independent feedback vertex set of size at most  $k$  if and only if  $k \geq |S_2| - 1$ . Note that the latter can be decided in linear time. We prove our claim as follows: If  $|S_2| \leq 2$ , then  $G - s$  is a forest for any  $s \in S_2$  and  $G$  has an independent feedback vertex set of size at most  $k$ . Hence, we may assume  $|S_2| \geq 3$ . Let  $I$  be a minimum independent feedback vertex set in  $G$ . Such a set exists as  $G$  is bipartite. As  $S_2 \setminus \{s\}$  is independent and as  $G[S_1 \cup \{s\}]$  is a forest for each vertex  $s \in S_2$ , we find  $|I| \leq |S_2| - 1$ . For the sake of a contradiction, let us assume  $|I| \leq |S_2| - 2$ . Hence, any two vertices of  $S_2 \setminus I$  have at least  $|S_1| - 2$  common neighbours in  $S_1$ , and so  $|I \cap S_1| \geq |S_1| - 3 \geq 2$ . Moreover,  $I = I \cap S_1$  as every vertex of  $S_2$  has a neighbour in  $I \cap S_1$  and as  $I$  is independent. As  $I$  is an independent feedback vertex set with  $|I| \leq |S_1| - 2$ , any two vertices of  $S_1 \setminus I$  do not have two common neighbours in  $S_2$  and so  $|S_2| \leq 3$ . Hence,  $5 \leq |S_1| \leq |I| + 3 \leq |S_2| + 1 \leq 4$ , a contradiction. As  $|I| = |S_2| - 1$ , the proof of our claim is complete.

**Case 2.**  $G$  is not bipartite.

*Outline.* As our problems require all the yes-instances to be 3-colourable, we check first whether  $G$  is 3-colourable. If so, then we compute an induced subgraph  $H$  of  $G$  and determine the set  $\mathcal{C}$  of all its 3-colourings that can be extended to 3-colourings of  $G$ . As we compute  $H$  by applying Theorem 10, we find that  $|\mathcal{C}| \leq 3^{9 \cdot 2^d + 8}$ . We then distinguish some subcases. In some of them we further branch by extending our 3-colourings. However, in some of them we find that  $H$  equals  $G$ , and so Observations 7 and 8 imply that our six problems are solvable in  $O(n + m)$  time as  $\mathcal{C}$  is of constant size. As an implicit step, we apply this finding whenever  $H$  is the whole graph  $G$ .

*Full Proof.* We first apply Theorem 10. We continue by assuming that  $G$  is 3-colourable. In fact, the only remaining case is that where the lemma provides a triangle  $T$  on vertex set  $\{x, y, z\}$ , a vertex  $x$  of  $T$  that has private neighbours, and the set of all 3-colourings of  $G - P(x)$  that can be extended to 3-colourings of  $G$ . Note that we have at most  $3^{9 \cdot 2^d + 8}$  such 3-colourings. We partition  $V(G)$  from  $V(T)$ .

We find that  $G[P(x)]$  is bipartite, as  $G$  is 3-colourable, but not necessarily connected. We extend each 3-colouring of  $G - P(x)$  that can be extended to a 3-colouring of  $G$  to some vertices of  $P(x)$ . Let  $c$  be an arbitrary 3-colouring of  $G - P(x)$  that can be extended to a 3-colouring of  $G$ . For  $i \in \{0, 1, 2\}$ , we compute in  $O(n + m)$  time the set  $S_i$  of all vertices of  $P(x)$  which have  $i$  available colours with respect to  $c$ , that is,  $S_i$  is the set of all vertices of  $P(x)$  which have neighbours in  $3 - i$  colours. As  $c$  can be extended to a 3-colouring of  $G$ , we find that  $S_0$  is empty. It takes  $O(n + m)$  time to determine the available colour of each vertex in  $S_1$ . Furthermore, we can extend  $c$  by breadth-first search in the same time to the vertices of those components of  $G[P(x)]$  that contain at least one vertex of  $S_1$ .

Let  $S_c$  be the set of vertices that induce those components of  $G[P(x)]$  that do not contain a vertex of  $S_1$ . Note that  $S_c$  can be computed in  $O(n + m)$  time and that all neighbours of all vertices of  $S_c$  in  $V(G) \setminus S_c$  are coloured alike. Moreover, every vertex of  $S_c$  has its neighbours in  $[N(y) \cap N(z)] \cup N_2 \cup S_c \cup \{x\}$  by definition. As  $c$  can be extended to a 3-colouring of  $G$ , we find that our approach leads to a 3-colouring, say  $c'$ , of  $G - S_c$ . As there are at most  $3^{9 \cdot 2^6 + 2}$  3-colourings of  $G - P(x)$ , we find at most  $3^{9 \cdot 2^6 + 2}$  such triples  $(c, c', S_c)$ . Furthermore, for each 3-colouring  $c_s$  of  $G - P(x)$ , there exists a triple  $(c_s, c'_s, S_{c_s})$  if  $c_s$  can

be extended to a 3-colouring of  $G$ . We proceed by considering the case where  $S_c \neq \emptyset$  as otherwise  $G = G - S_c$ . We continue by distinguishing on the problems we are considering. Recall that  $G$  is 3-colourable.

**Subcase 2.1.** ACYCLIC 3-COLOURING and STAR 3-COLOURING

We check whether for some triple  $(c, c', S_c)$ , the 3-colouring  $c'$  of  $G - S_c$  that can be extended to an acyclic 3-colouring or star 3-colouring of  $G$ . By this approach, we clearly solve ACYCLIC 3-COLOURING and STAR 3-COLOURING.

Let  $(c, c', S_c)$  be an arbitrary triple as defined above. Recall that a star 3-colouring of a graph is acyclic. In time  $O(n + m)$ , we can determine the components of  $G[S_c]$  and check whether  $G[S_c]$  is a forest. If not, then  $G[S_c \cup \{x\}]$ , and thus  $G$ , is not acyclic 3-colourable. We continue and assume that  $G[S_c]$  is a forest. We check in  $O(n + m)$  time if a vertex of  $S_c$  has a neighbour in  $N(y) \cap N(z)$ . If so, say  $s \in S_c$  is adjacent to  $v \in N(y) \cap N(z)$ , then  $c'$  cannot be extended to an acyclic 3-colouring of  $c$  as either  $s$  and  $x$  are coloured alike or one of  $\{svyxs, svzxs\}$  is a bichromatic 4-cycle. We proceed by assuming that  $S_c$  has its neighbours in  $N_2 \cup S_c \cup \{x\}$ . As  $G$  is chair-free, every two non-adjacent vertices of  $S_c$  share the same neighbours in  $N_2$  and, if there exists such a neighbour, then these two vertices have to be coloured differently to avoid a bichromatic 4-cycle. Therefore, in any acyclic extension of  $c'$  to  $G$ , each of the two colour classes in  $S_c$  either has size at most 1 or has no neighbour in  $N_2$ . We check in constant time if  $S_c$  is of size at most 2. If so, then there are at most 4 possibilities to extend  $c'$  to a 3-colouring of  $G$  and for each we apply Observation 7. Hence, we may assume  $|S_c| \geq 3$ . We check in  $O(n + m)$  time if a vertex of  $S_c$  has a neighbour in  $N_2$ .

Let us consider the subcase where  $s \in S_c$  has a neighbour, say  $v$ , in  $N_2$ . Let  $G_s$  be the component of  $G[P(x)]$  that contains  $s$ . Note that there are at most two possibilities to extend  $c'$  to the vertices of  $G_s$ . We check in linear time if  $S_c \setminus V(G_s)$  is of size at least 2. If so, say  $s_1, s_2 \in S_c \setminus V(G_s)$ , then  $v$  is a neighbour of  $s, s_1$ , and  $s_2$ . Thus,  $xs'_1vs'_2x$  is a bichromatic 4-cycle for two vertices  $s'_1$  and  $s'_2$  of  $\{s, s_1, s_2\}$ . We conclude that  $c'$  cannot be extended to an acyclic 3-colouring of  $G$ . Hence, we may assume  $|S_c \setminus V(G_s)| \leq 1$ , and so there are at most four possibilities to extend  $c'$  to a 3-colouring of  $G$  each of which can be obtained in  $O(n + m)$  time. We apply Observation 7 for each. We proceed by assuming that no vertex of  $S_c$  has a neighbour in  $N_2$ . In other words, each vertex of  $S_c$  has its neighbours in  $S_c \cup \{x\}$ .

As  $x$  is a cut-vertex of  $G$ , any extension of  $c'$  to a 3-colouring of  $G$  is acyclic if and only if  $c'$  is acyclic. Hence, we apply Observation 7 on  $G - S_c$  and  $c'$  in order to solve ACYCLIC 3-COLOURING.

We now check in  $O(n + m)$  time if each component of  $G[S_c]$  is of diameter at most 2. If not, then  $G[S_c \cup \{x\}]$ , and thus  $G$  is not star 3-colourable. Let us proceed by assuming that each component of  $G[S_c]$  is of diameter at most 2. We find that every 3-colouring of  $G[S_c \cup \{x\}]$  is a star 3-colouring. In other words, we can restrict ourselves to those 3-colouring extensions of  $c'$  to  $G$  that assign one colour to all vertices of  $S_c$  if  $S_c$  is independent, and an arbitrary 3-colouring extensions of  $c'$  to  $G$  if  $S_c$  is not independent. Note that we can check in  $O(n + m)$  time whether  $S_c$  is independent. We find in both subcases at most two extensions of  $c'$  to  $G$  and apply Observation 7 for each in order to solve STAR 3-COLOURING.

**Subcase 2.2.** INDEPENDENT ODD CYCLE TRANSVERSAL

Let  $k$  be an arbitrary integer. We check whether some triple  $(c, c', S_c)$  consists of a 3-colouring  $c'$  of  $G - S_c$  that can be extended to a 3-colouring of  $G$  whose one colour class is an independent odd cycle transversal of size at most  $k$ . As all the yes-instances require  $G$  to be 3-colourable, this approach clearly solves INDEPENDENT ODD CYCLE TRANSVERSAL.

## 21:12 Partitioning $H$ -Free Graphs of Bounded Diameter

Let  $(c, c', S_c)$  be an arbitrary triple as defined above. Moreover, let  $X, Y, Z$  be the colour classes of  $c'$  with  $x \in X$ ,  $y \in Y$ , and  $z \in Z$ . Clearly,  $X, Y$ , and  $Z$  can be computed in linear time. We decide in linear time which of  $\{Y, Z\}$  is of smaller size, say  $|Y| \leq |Z|$ . Recall that all vertices of  $S_c$  have their neighbours in  $S_c \cup X$ . Note that  $c'$  can be extended to a 3-colouring of  $G$  by 2-colourings of  $G[S_c]$  on the colours that  $c'$  assigns to  $y$  and  $z$ , and these are the only possibilities. We find that the smallest possible colour class of a 3-colouring of  $G$  that extends  $c'$  consists of the vertices either in  $X$  or in  $Y \cup W$ , where  $W$  is the smallest possible colour class of a 2-colouring of  $G[S_c]$ . As we can compute the components of  $G[S_c]$  and its parts in  $O(n + m)$  time, we find  $W$  in the same time. Hence, the smallest possible independent odd cycle transversal of  $G$  that is a colour class of an extension of  $c'$  to a 3-colouring of  $G$  is of size  $\min\{|X|, |Y \cup W|\}$ . We can compare the sizes of  $X$  and  $Y \cup W$  with  $k$  in linear time.

### Subcase 2.3. INDEPENDENT FEEDBACK VERTEX SET and NEAR-BIPARTITENESS

Let  $k$  be an arbitrary integer. We check whether some triple  $(c, c', S_c)$  consists of a 3-colouring  $c'$  of  $G - S_c$  that can be extended to a 3-colouring of  $G$  whose one colour class is an independent feedback vertex set (of size at most  $k$ ). As all the yes-instances require  $G$  to be 3-colourable, this approach clearly solves INDEPENDENT FEEDBACK VERTEX SET and NEAR-BIPARTITENESS.

Let  $(c, c', S_c)$  be an arbitrary triple as defined above. Moreover, let  $X, Y, Z$  be the colour classes of  $c'$  with  $x \in X$ ,  $y \in Y$ , and  $z \in Z$ . Clearly,  $X, Y$ , and  $Z$  can be computed in linear time. We check first whether  $G - X$  is a forest in  $O(n + m)$  time. If so, then we find that  $X$  is an independent feedback vertex set of  $G$  and we can determine its size in linear time. Hence, we proceed by assuming that  $G - X$  contains a cycle or  $|X| > k$ . As we aim to find an extension of  $c'$  to a 3-colouring of  $G$  whose one colour class is an independent feedback vertex set (of size at most  $k$ ), we find that such a set consists of the vertices of  $Y$  or of  $Z$ , and the vertices of some set  $A \subseteq S_c$ . Recall that all vertices of  $S_c$  have their neighbours in  $[N(y) \cap N(z)] \cup N_2 \cup S_c \cup \{x\}$  and their neighbours in  $[N(y) \cap N(z)] \cup N_2 \cup \{x\}$  form an independent set. Note that  $c'$  can be extended to a 3-colouring of  $G$  by 2-colourings of  $G[S_c]$  on the colours that  $c'$  assigns to  $y$  and  $z$ , and these are the only possibilities. If  $G[S_c]$  is connected, which can be tested in  $O(n + m)$  time, then there are at most two such possibilities, and so we apply Observation 8 for each. We proceed by assuming that  $G[S_c]$  is disconnected, and so  $|S_c| \geq 2$ .

We claim that all vertices of  $S_c$  have the same neighbours in  $N_2$ . Let us assume that  $v$  is an arbitrary vertex of  $N_2$  that is adjacent to some vertex of  $S_c$ . Let  $S_v$  be the set of neighbours of  $v$  in  $S_c$ . By definition, we find that  $S_v$  is non-empty. As  $G$  is chair-free, we obtain that every vertex of  $S_v$  is adjacent to every vertex of  $S_c \setminus S_v$  as otherwise  $\{s_1, s_2, v, x, y\}$  would induce a chair for some possible vertices  $s_1 \in S_v$  and  $s_2 \in S_c \setminus S_v$ . As  $G[S_c]$  is disconnected, we find that  $S_c \setminus S_v = \emptyset$ , which completes the proof of our claim as  $v$  is arbitrarily chosen.

We can check if there is a vertex in  $N(y) \cap N(z)$  in  $O(n + m)$  time. First assume there is such a vertex, say  $w$ . As  $\{s_1, s_2, w, x, y\}$  does not induce a chair for each two vertices  $s_1, s_2$  of an independent set  $I$  of  $G[S_c]$ , we find that  $w$  is adjacent to all but at most one vertex of  $I$ . As  $G[S_c]$  is bipartite, it follows that  $w$  has at least  $|S_c| - 2$  neighbours in  $S_c$ . For each  $s \in N(w) \cap S_c$ , we find  $s \in A$  as  $sxyws$  and  $szxws$  are 4-cycles. Note that  $N(w) \cap S_c$  can be computed in  $O(n + m)$  time. As  $|N(w) \cap S_c| \geq |S_c| - 2$ , we find at most eight possibilities to extend  $c'$  to a 3-colouring of  $G$  by a 2-colouring of  $G[S_c]$  in which one colour class contains all the vertices of  $N(w) \cap S_c$ . We apply Observation 8 for each. Hence, we may assume that  $N(y) \cap N(z) = \emptyset$ , and so every two vertices of  $S_c$  share the same neighbours in  $V(G) \setminus S_c$ .



If no vertex of  $N_2$  has a neighbour in  $S_c$ , then  $x$  is a cut-vertex. In this case we find that  $G$  has an independent feedback vertex set of size at most  $k$  if and only if  $G - S_c$  has an independent feedback vertex set (of size at most  $k - |W|$ , where  $W$  is the smallest possible colour class of a 2-colouring of  $G[S_c]$ ). As  $W$  can be computed in  $O(n + m)$  time, we apply Observation 8 for  $G - S_c$  and  $c'$ .

We proceed by considering the situation where  $v \in N_2$  has a neighbour in  $S_c$ . Recall that all vertices of  $S_c$  are adjacent to  $v$ . As  $xs_1vs_2x$  is a 4-cycle for any two vertices  $s_1, s_2 \in S_c$ , we find that  $A$  has size at least  $|S_c| - 1$ . In other words, we aim for such a 2-colouring of  $G[S_c]$  whose one colour class is of size at most 1. If  $S_c$  is not independent, we have at most two such possibilities, and each leads to a 3-colouring of  $G$ . We apply Observation 8 for each. Now suppose that  $S_c$  is independent. We find that any two vertices of  $S_c$  have the same neighbours in  $G$ . Let us fix one vertex, say,  $s$  of  $S_c$ . As there is at most one vertex of  $S_c$  that is not in the independent feedback vertex set, we may assume that  $s$  is that vertex. We have four ways of colouring the vertices of  $S_c$  such that all vertices of  $S_c \setminus \{s\}$  receive the same colour. It remains to apply Observation 8 for each case. ◀

#### 4 A Final Result and Some Open Problems

We showed that bounding the diameter does not help for INDEPENDENT SET for  $H$ -free graphs. We proved that this does help for some problems related to 3-COLOURING if  $H$  is the chair. Whether these results can be extended to larger polyads  $H$  is an interesting but challenging task. For three of these problems, however, we should not seek to extend the theorems from the previous section to omission of arbitrary polyads: in our next result, we give a polyad  $H$  such that these problems are NP-complete for  $H$ -free graphs of diameter  $d$  for some constant  $d$ . We reduce from NOT-ALL-EQUAL 3-SAT, which is well-known to be NP-complete [29]. We omit the proof details.

► **Theorem 12.** *For STAR 3-COLOURING, INDEPENDENT ODD CYCLE TRANSVERSAL and ACYCLIC 3-COLOURING, there exists a polyad  $H$  and integer  $d$  so that the problem remains NP-complete on  $H$ -free graphs of diameter  $d$ .*

Finally, we ask if there exists a polyad  $H$  and an integer  $d$  such that NEAR-BIPARTITENESS and INDEPENDENT FEEDBACK VERTEX SET are NP-complete for  $H$ -free graphs of diameter at most  $d$ . Such a polyad  $H$  was already known to exist for 3-COLOURING of graphs of diameter at most 4 [20].

---

#### References

- 1 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-Algebraic Methods in Applied Mathematics*, pages 3–13, 1982 (in Russian).
- 2 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135:3–16, 2004.
- 3 Manuel Bodirsky, Jan Kára, and Barnaby Martin. The complexity of surjective homomorphism problems – A survey. *Discrete Applied Mathematics*, 160:1680–1690, 2012.
- 4 Jan Bok, Nikola Jedličková, Barnaby Martin, Daniël Paulusma, and Pascal Ochem Siani Smith. Acyclic, star and injective colouring: A complexity picture for  $H$ -free graphs. *CoRR*, abs/2008.09415 (conference version in Proc. ESA 2020, LIPIcs 173, 22:1–22:22), 2021. [arXiv: 2008.09415](https://arxiv.org/abs/2008.09415).
- 5 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex sets for graphs of bounded diameter. *Information Processing Letters*, 131:26–32, 2018.



- 6 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex set for  $P_5$ -free graphs. *Algorithmica*, 81:1342–1369, 2019.
- 7 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 1999.
- 8 Christoph Brause, Petr A. Golovach, Barnaby Martin, Daniël Paulusma, and Siani Smith. Acyclic, star and injective colouring: bounding the diameter. *Proc. WG 2021, LNCS*, 12911:336–348, 2021.
- 9 Hajo Broersma, Fedor V. Fomin, Petr A. Golovach, and Daniël Paulusma. Three complexity results on coloring  $P_k$ -free graphs. *European Journal of Combinatorics*, 34(3):609–619, 2013.
- 10 Nina Chiarelli, Tatiana R. Hartinger, Matthew Johnson, Martin Milanič, and Daniël Paulusma. Minimum connected transversals in graphs: new hardness results and tractable cases using the price of connectivity. *Theoretical Computer Science*, 705:75–83, 2018.
- 11 Maria Chudnovsky. The structure of bull-free graphs II and III – A summary. *Journal of Combinatorial Theory, Series B*, 102:252–282, 2012.
- 12 Maria Chudnovsky and Paul D. Seymour. The structure of claw-free graphs. *Surveys in Combinatorics, London Mathematical Society Lecture Note Series*, 327:153–171, 2005.
- 13 Konrad K. Dabrowski, Matthew Johnson, and Daniël Paulusma. Clique-width for hereditary graph classes. *Proc. BCC 2019, London Mathematical Society Lecture Note Series*, 456:1–56, 2019.
- 14 Michał Dębski, Marta Piecyk, and Paweł Rzażewski. Faster 3-coloring of small-diameter graphs. *Proc. ESA 2021, LIPIcs*, 204:37:1–37:15, 2021.
- 15 Keith Edwards. The complexity of colouring problems on dense graphs. *Theoretical Computer Science*, 43:337–343, 1986.
- 16 Petr A. Golovach, Matthew Johnson, Daniël Paulusma, and Jian Song. A survey on the computational complexity of colouring graphs with forbidden subgraphs. *Journal of Graph Theory*, 84:331–363, 2017.
- 17 Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. *ACM Transactions on Algorithms*, 15:25:1–25:90, 2019.
- 18 John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.
- 19 Daniel Král', Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. *Proc. WG 2001, LNCS*, 2204:254–262, 2001.
- 20 Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring  $H$ -free graphs of bounded diameter. *Proc. MFCS 2019, LIPIcs*, 138:14:1–14:14, 2019.
- 21 Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring graphs of bounded diameter in the absence of small cycles. *Proc. CIAC 2021, LNCS*, 12701:367–380, 2021.
- 22 George B. Mertzios and Paul G. Spirakis. Algorithms and almost tight results for 3-Colorability of small diameter graphs. *Algorithmica*, 74:385–414, 2016.
- 23 Andrea Munaro. On line graphs of subcubic triangle-free graphs. *Discrete Mathematics*, 340:1210–1226, 2017.
- 24 Giacomo Paesani, Daniël Paulusma, and Paweł Rzażewski. Feedback Vertex Set and Even Cycle Transversal for  $H$ -free graphs: finding large block graphs. *Proc. MFCS 2021, LIPIcs*, 202:82:1–82:14, 2021.
- 25 Daniël Paulusma. Open problems on graph coloring for special graph classes. *Proc. WG 2015, LNCS*, 9224:16–30, 2015.
- 26 Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15:307–309, 1974.
- 27 Bert Randerath and Ingo Schiermeyer. Vertex colouring and forbidden subgraphs – A survey. *Graphs and Combinatorics*, 20:1–40, 2004.
- 28 Najiba Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoilé. *Discrete Mathematics*, 29:53–76, 1980.
- 29 Thomas J. Schaefer. The complexity of satisfiability problems. *Proc. STOC 1978*, pages 216–226, 1978.

# Clique-Based Separators for Geometric Intersection Graphs

Mark de Berg 

Department of Computer Science, TU Eindhoven, The Netherlands

Sándor Kisfaludi-Bak<sup>1</sup> 

Institute for Theoretical Studies, ETH Zürich, Switzerland

Morteza Monemizadeh 

Department of Computer Science, TU Eindhoven, The Netherlands

Leonidas Theocharous 

Department of Computer Science, TU Eindhoven, The Netherlands

---

## Abstract

---

Let  $F$  be a set of  $n$  objects in the plane and let  $\mathcal{G}^\times(F)$  be its intersection graph. A balanced clique-based separator of  $\mathcal{G}^\times(F)$  is a set  $\mathcal{S}$  consisting of cliques whose removal partitions  $\mathcal{G}^\times(F)$  into components of size at most  $\delta n$ , for some fixed constant  $\delta < 1$ . The weight of a clique-based separator is defined as  $\sum_{C \in \mathcal{S}} \log(|C| + 1)$ . Recently De Berg et al. (SICOMP 2020) proved that if  $S$  consists of convex fat objects, then  $\mathcal{G}^\times(F)$  admits a balanced clique-based separator of weight  $O(\sqrt{n})$ . We extend this result in several directions, obtaining the following results.

- Map graphs admit a balanced clique-based separator of weight  $O(\sqrt{n})$ , which is tight in the worst case.
- Intersection graphs of pseudo-disks admit a balanced clique-based separator of weight  $O(n^{2/3} \log n)$ . If the pseudo-disks are polygonal and of total complexity  $O(n)$  then the weight of the separator improves to  $O(\sqrt{n} \log n)$ .
- Intersection graphs of geodesic disks inside a simple polygon admit a balanced clique-based separator of weight  $O(n^{2/3} \log n)$ .
- Visibility-restricted unit-disk graphs in a polygonal domain with  $r$  reflex vertices admit a balanced clique-based separator of weight  $O(\sqrt{n} + r \log(n/r))$ , which is tight in the worst case.

These results immediately imply sub-exponential algorithms for MAXIMUM INDEPENDENT SET (and, hence, VERTEX COVER), for FEEDBACK VERTEX SET, and for  $q$ -COLORING for constant  $q$  in these graph classes.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Computational geometry, intersection graphs, separator theorems

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.22

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.09874>

**Funding** The work in this paper is supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

---

<sup>1</sup> The research was conducted while the author was at the Max Planck Institute for Informatics, Saarbrücken, Germany.



## 1 Introduction

The famous Planar Separator Theorem states that any planar graph  $\mathcal{G} = (V, E)$  with  $n$  nodes<sup>2</sup> admits a subset  $\mathcal{S} \subset V$  of size  $O(\sqrt{n})$  nodes whose removal decomposes  $\mathcal{G}$  into connected components of size at most  $2n/3$ . The subset  $\mathcal{S}$  is called a balanced<sup>3</sup> *separator* of  $\mathcal{G}$ . The theorem was first proved in 1979 by Lipton and Tarjan [19], and it has been instrumental in the design of algorithms for planar graphs: it has been used to design efficient divide-and-conquer algorithms, to design sub-exponential algorithms for various NP-hard graph problems, and to design approximation algorithms for such problems.

The Planar Separator Theorem has been extended to various other graph classes. Our interest lies in *geometric intersection graphs*, where the nodes correspond to geometric objects and there is an arc between two nodes iff the corresponding objects intersect. If the objects are disks, the resulting graph is called a *disk graph*. Disk graphs, and in particular unit-disk graphs, are a popular model for wireless communication networks and have been studied extensively. Miller et al. [22] and Smith and Wormald [27] showed that if  $F$  is a set of balls in  $\mathbb{R}^d$  of ply at most  $k$  – the *ply* of  $F$  is the maximum number of objects in  $F$  with a common intersection – then the intersection graph of  $F$  has a separator of size  $O(k^{1/d}n^{1-1/d})$ . This was generalized by Chan [4] and Har-Peled and Quanrud [14] to intersection graphs of so-called low-density sets. Separators for string graphs – a string graph is an intersection graph of sets of curves in the plane – have also been considered [13, 18, 21], with Lee [18] showing that a separator of size  $O(\sqrt{m})$  exists, where  $m$  is the number of arcs of the graph.

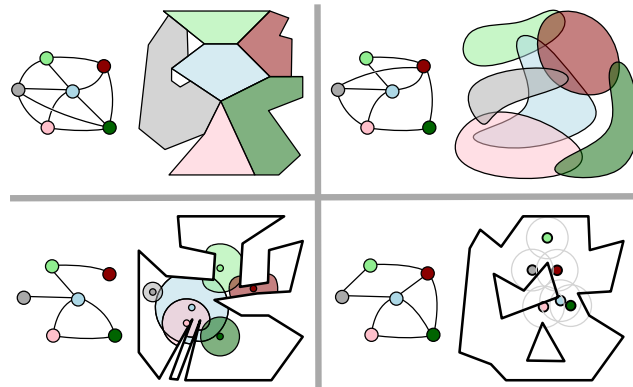
Even for simple objects such as disks or squares, one must restrict the ply to obtain a separator of small size. Otherwise the objects can form a single clique, which obviously does not have a separator of sublinear size. To design subexponential algorithms for problems such as MAXIMUM INDEPENDENT SET, however, one can also work with a separator consisting of a small number of cliques instead of a small number of nodes. Such *clique-based separators* were introduced recently by De Berg et al. [8]. Formally, a clique-based separator of a graph  $\mathcal{G}$  is a collection  $\mathcal{S}$  of node-disjoint cliques whose union is a balanced separator of  $\mathcal{G}$ . The *weight* of  $\mathcal{S}$  is defined as  $\text{weight}(\mathcal{S}) := \sum_{C \in \mathcal{S}} \log(|C| + 1)$ . De Berg et al. [8] proved that the intersection graph of any set  $F$  of  $n$  convex fat objects in the plane admits a clique-based separator of weight  $O(\sqrt{n})$ , and they used this to obtain algorithms with running time  $2^{O(\sqrt{n})}$  for many classic NP-hard problems on such graphs. This running time is optimal, assuming the Exponential-Time Hypothesis (ETH). The result generalizes to convex fat objects in  $\mathbb{R}^d$ , where the bound on the weight of the clique-based separator becomes  $O(n^{1-1/d})$ .

The goal of our paper is to investigate whether similar results are possible for non-fat objects in the plane. Note that not all intersection graphs admit clique-based separators of small weight. String graphs, for instance, can have arbitrarily large complete bipartite graphs as induced subgraphs, in which case any balanced clique-based separator has weight  $\Omega(n)$ .

The first type of intersection graphs we consider are map graphs, which are a natural generalization of planar graphs. The other types are generalizations of disk graphs. One way to generalize disk graphs is to consider fat objects instead of disks, as done by De Berg et al. [8]. We will study three other generalizations, involving non-fat objects: pseudo-disks, geodesic disks, and visibility-restricted unit disks. Next we define the graph classes we consider more precisely; see Fig. 1 for an example of each graph class.

<sup>2</sup> We use the terms *node* and *arc* when talking about graphs, and *vertex* and *edge* for geometric objects.

<sup>3</sup> For a separator to be balanced it suffices that the components have size at most  $\delta n$  for some constant  $\delta < 1$ . When we speak of separators, we always mean balanced separators, unless stated otherwise.



■ **Figure 1** A map graph, a pseudo-disk graph, a geodesic-disk graph, and a visibility restricted unit-disk graph. For the latter class, the grey disks in the picture have radius  $\frac{1}{2}$ .

In the following, we use  $\mathcal{G}^\times(F)$  to denote the intersection graph induced by a set  $F$  of objects. For convenience, we do not distinguish between the objects and the corresponding nodes, so we use  $F$  to denote the set of objects as well as the set of nodes in  $\mathcal{G}^\times(F)$ . We assume that the objects in  $F$  are connected, bounded, and closed.

**Map graphs.** Let  $\mathcal{M}$  be a planar subdivision and  $F$  be its set of faces. The graph with node set  $F$  that has an arc between every pair of neighboring faces is called the *dual graph* of  $\mathcal{M}$ , and it is planar. Here two faces are neighbors if their boundaries have an edge of the subdivision in common. A *map graph* [6] is defined similarly, except now two faces are neighbors even if their boundaries meet in a single point. Alternatively, we can define a map graph as the intersection graph of a set  $F$  of interior-disjoint regions in the plane. Since arbitrarily many faces can share a vertex on their boundary, map graphs can contain arbitrarily large cliques. If at most  $k$  faces meet at each subdivision vertex, the graph is called a *k-map graph*. Chen [5] proved that any  $k$ -map graph has a (normal, not clique-based) separator of size  $O(\sqrt{kn})$ , which is also implied by Lee’s recent result on string graphs [18].

**Pseudo-disk graphs.** A set  $F$  of objects is a set of pseudo-disks if for any  $f, f' \in F$  the boundaries  $\partial f$  and  $\partial f'$  intersect at most twice. Pseudo-disks were introduced in the context of motion planning by Kedem et al. [15], who proved that the union complexity of  $n$  pseudo-disks is  $O(n)$ . Since then they have been studied extensively. We consider two types of pseudo-disks: polygonal pseudo-disks with  $O(n)$  vertices in total, and arbitrary pseudo-disks.

**Geodesic-disk graphs and visibility-restricted unit-disk graphs.** As mentioned, unit-disk graphs are popular models for wireless communication networks. We consider two natural generalizations of unit-disk graphs, which can be thought of as communication networks in a polygonal environment that may obstruct communication.

- *Geodesic-disk graphs* in a simple polygon  $P$  are intersection graphs of geodesic disks inside  $P$ . (The *geodesic disk* with center  $q \in P$  and radius  $r$  is the set of all points in  $P$  at geodesic distance at most  $r$  from  $q$ , where the geodesic distance between two points is the length of the shortest path between them inside  $P$ .)

- In *visibility-restricted unit-disk graphs* the nodes correspond to a set  $Q$  of  $n$  points inside a polygon  $P$ , which may have holes, and two points  $p, q \in Q$  are connected by an arc iff  $|pq| \leq 1$  and  $p$  and  $q$  see each other (meaning that  $pq \subset P$ ).<sup>4</sup> A more general, directed version of such graphs was studied by Ben-Moshe et al. [2] under the name range-restricted visibility graph. They presented an output-sensitive algorithm to compute the graph.

### Our results: clique-based separator theorems

So far, clique-based separators were studied for fat objects: De Berg et al. [8] consider convex or similarly-sized fat objects, Kisfaludi-Bak et al. [17] study how the fatness of axis-aligned fat boxes impacts the separator weight, and Kisfaludi-Bak [16] studies balls in hyperbolic space. The  $O(\sqrt{n})$  bound on the separator weight is tight even for unit-disk graphs. Indeed, a  $\sqrt{n} \times \sqrt{n}$  grid graph can be realized as a unit-disk graph, and any separator of such a grid graph must contain  $\Omega(\sqrt{n})$  nodes. Since the maximum clique size in a grid graph is two, any separator must contain  $\Omega(\sqrt{n})$  cliques. All graph classes we consider can realize a  $\sqrt{n} \times \sqrt{n}$  grid graph, so  $\Omega(\sqrt{n})$  is a lower bound on the weight of the clique-based separators we consider. We obtain the following results.

In Section 2 we show that any map graph has a clique-based separator of weight  $O(\sqrt{n})$ . This gives the first ETH-tight algorithms for MAXIMUM INDEPENDENT SET (and, hence, VERTEX COVER), FEEDBACK VERTEX SET, and COLORING in map graphs; see below.

In Section 3 we show that any intersection graph of pseudo-disks has a clique-based separator of weight  $O(n^{2/3} \log n)$ . If the pseudo-disks are polygonal and of total complexity  $O(n)$  then the weight of the separator improves to  $O(\sqrt{n} \log n)$ .

In Section 4 we consider intersection graphs of geodesic disks inside a simple polygon. At first sight, geodesic disks seem not much harder to deal with than fat objects: they can have skinny parts only in narrow corridors and then packing arguments may still be feasible. Unfortunately another obstacle prevents us from applying a packing argument: geodesic distances in a simply connected polygon induce a metric space whose doubling dimension depends on the number of reflex vertices of the polygon. Nevertheless, by showing that geodesic disks inside a simple polygon behave as pseudo-disks, we are able to obtain a clique-based separator of weight  $O(n^{2/3} \log n)$ , independent of the number of reflex vertices.

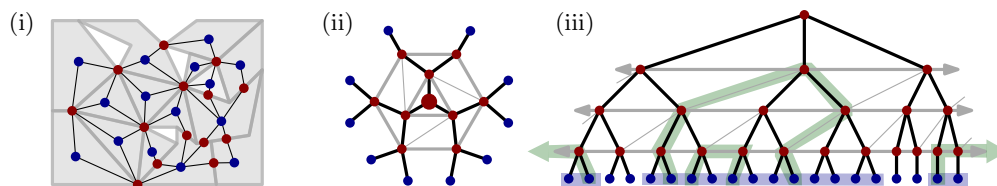
In Section 5 we study visibility-restricted unit-disk graphs. We give an  $\Omega(\min(n, r \log(n/r)) + \sqrt{n})$  lower bound for the separator weight, showing that a clique-based separator whose weight depends only on  $n$ , the number of points defining the visibility graph, is not possible. We then show how to construct a clique-based separator of weight  $O(\min(n, r \log(n/r)) + \sqrt{n})$ .

All separators can be computed in polynomial time. For map graphs and for the pseudo-disk intersection graphs, we assume the objects have total complexity  $O(n)$ . If the objects have curved edges, we assume that basic operations (such as computing the intersection points of two such curves) take  $O(1)$  time.

### Applications

We apply our separator theorems to obtain subexponential algorithms in the graph classes discussed above, for MAXIMUM INDEPENDENT SET, FEEDBACK VERTEX SET, and  $q$ -COLORING for constant  $q$ . The crucial property of these problems that makes our separator

<sup>4</sup> Visibility-restricted unit-disk graphs are, strictly speaking, not intersection graphs. In particular, if  $R_q$  is defined as the region of points within  $P$  that are visible from  $q$  and lie within distance  $1/2$ , then the visibility-restricted unit-disk graph is *not* the same as the intersection graph of the objects  $R_q$ .



■ **Figure 2** (i) A witness graph for the map graph induced by the grey regions. Points in  $P$  are blue, points in  $Q$  are red. (ii) The gadget used to replace a witness point. The edges of  $\mathcal{T}_q$  are black, the cycles connecting nodes at the same level are grey and thick, the edges to triangulate the 4-cycle are grey and thin. (iii) The green paths show an example of how the separator can intersect a gadget. (Note that the tree “wraps around”, as in part (ii) of the figure; see also one of the green paths.) The objects added to the clique  $C_q$  correspond to the leaves indicated by the blue rectangles.

applicable, is that the possible ways in which a solution can “interact” with a clique of size  $k$  is polynomial in  $k$ . We use known techniques (mostly from De Berg et al. [8]) to solve the three problems on any graph class that has small clique-based separators.

All our graph classes are subsumed by string graphs. Bonnet and Rzazewski [3] showed that string graphs have  $2^{O(n^{2/3} \log n)}$  algorithms for MAXIMUM INDEPENDENT SET and 3-COLORING, and a  $2^{n^{2/3} \log^{O(1)} n}$  algorithm for FEEDBACK VERTEX SET, and that string graphs do not have subexponential algorithms for  $q$ -COLORING with  $q \geq 4$  under ETH. One can also obtain subexponential algorithms in some of our classes from results of Fomin et al. [12, 5] or Marx and Philipczuk [20]. The running times we obtain match or slightly improve the results that can be obtained from these existing results. It should be kept in mind, however, that the existing results are for more general graph classes. An exception are our results on map graphs, which were explicitly studied before and where we improve the running time for MAXIMUM INDEPENDENT SET and FEEDBACK VERTEX SET from  $2^{O(\sqrt{n} \log n)}$  to  $2^{O(\sqrt{n})}$ . (But, admittedly, the existing results apply in the parameterized setting while ours don’t.) In any case, the main advantage of our approach is that it allows us to solve MAXIMUM INDEPENDENT SET, FEEDBACK VERTEX SET and  $q$ -COLORING on each of the mentioned graph classes in a uniform manner. Refer to the arXiv version [10] for more details on the state of the art, and the specific results we obtain.

## 2 Map graphs

Recall that a map graph is the intersection graph of a set  $F$  of interior-disjoint objects in the plane. We construct a clique-based separator for  $\mathcal{G}^\times(F)$  in four steps. First, we construct a bipartite plane *witness graph*  $\mathcal{H}_1$  with node set  $P \cup Q$ , where the nodes in  $P$  correspond to the objects in  $F$  and the nodes in  $Q$  (with their incident arcs) model the adjacencies in  $\mathcal{G}^\times(F)$ . Next, we replace each node  $q \in Q$  by a certain gadget whose “leaves” are the neighbors of  $q$ , and we triangulate the resulting graph. We then apply the Planar Separator Theorem to obtain a separator for the resulting graph  $\mathcal{H}_2$ . Finally, we turn the separator for  $\mathcal{H}_2$  into a clique-based separator for  $\mathcal{G}^\times(F)$ . Next we explain these steps in detail.

### Step 1: Creating a witness graph

To construct a witness graph for  $\mathcal{G}^\times(F)$  we use the method of Chen et al. [6]: take a point  $p_f$  in the interior of each object  $f \in F$ , and take a *witness point*  $q \in \partial f \cap \partial f'$  for each pair of touching objects  $f, f' \in F$  and add arcs from  $q$  to the points  $p_f$  and  $p_{f'}$ . Let  $P = \{p_f : f \in F\}$  and let  $Q$  be the set of all witness points added. We denote the resulting



bipartite graph with node set  $P \cup Q$  by  $\mathcal{H}_1$ ; see Figure 2(i) for an example. Observe that points where many objects meet can serve as witness points for many neighboring pairs in  $\mathcal{G}^\times(F)$ . Chen et al. [6, Lemma 2.3] proved that any map graph admits a witness set  $Q$  of size  $O(n)$ . If the objects in  $F$  are polygons with  $O(n)$  vertices in total then  $Q$  can be found in  $O(n)$  time (since the vertices can serve as the set  $Q$ .)

### Step 2: Replacing witness points by gadgets and triangulating

We would like to construct a separator for  $\mathcal{H}_1$  using the Planar Separator Theorem, and convert it to a clique-based separator for  $\mathcal{G}^\times(F)$ . For every witness point  $q \in Q$  in the separator for  $\mathcal{H}_1$ , the conversion would add a clique  $C_q$  to the clique-based separator, namely, the clique corresponding to all objects  $f \in F$  such that  $p_f$  is adjacent to  $q$ . However, the node  $q$  adds 1 to the separator size, but the clique  $C_q$  adds  $\log(|C_q| + 1)$  to the weight of the clique-based separator. To deal with this we modify  $\mathcal{H}_1$ , as follows.

Consider a node  $q \in Q$ . Let  $N(q) \subseteq P$  denote the set of neighbors of  $q$ . For all nodes  $q \in Q$  with  $|N(q)| \geq 3$ , we replace the star induced by  $\{q\} \cup N(q)$  by a gadget  $G_q$ , which is illustrated in Figure 2(ii) and defined as follows.

First, we create a tree  $\mathcal{T}_q$  with root  $q$  and whose leaves are the nodes in  $N(q)$ , as follows. Define the *level*  $\ell(v)$  of a node  $v$  in  $\mathcal{T}_q$  to be the distance of  $v$  to the root; thus the root has level 0, its children have level 1, and so on. All leaves in  $\mathcal{T}_q$  are at the same level, denoted  $\ell_{\max}$ . The root has degree 3, nodes at level  $\ell$  with  $1 \leq \ell < \ell_{\max} - 1$  have degree 2, and nodes at level  $\ell_{\max} - 1$  have degree 2 or 1. For each  $\ell < \ell_{\max}$  we connect the nodes at level  $\ell$  into a cycle. After doing so, all faces in the gadget (except the outer face) are triangles or 4-cycles. We finish the construction by adding a diagonal in each 4-cycle. Define the *height* of a node  $v$  as  $\text{height}(v) := \ell_{\max} - \ell(v)$ . The following observation follows from the construction.

► **Observation 1.** *Let  $v$  be a node at height  $h > 0$  in the gadget  $G_q$ .*

- (i) *The subtree of  $\mathcal{T}_q$  rooted at  $v$ , denoted  $\mathcal{T}_q(v)$ , has at most  $3 \cdot 2^{h-1}$  leaves.*
- (ii) *The distance from  $v$  to any leaf in  $\mathcal{T}_q$  is at least  $h$ .*

To unify the exposition, it will be convenient to also create a gadget for the case where  $q$  has only two neighbors in  $\mathcal{H}_1$ , say  $p_f$  and  $p_{f'}$ . We then define  $\mathcal{T}_q$  to consist of the arcs  $(q, p_f)$  and  $(q, p_{f'})$ . Note that Observation 1 holds for this gadget as well.

By replacing each witness point  $q \in Q$  with a gadget  $G_q$  as above, we obtain a (still planar) graph. We triangulate this graph to obtain a maximal planar graph  $\mathcal{H}_2$ .

### Step 3: Constructing a separator for $\mathcal{H}_2$

We now want to apply the Planar Separator Theorem to  $\mathcal{H}_2$ . Our final goal is to obtain a balanced clique-based separator for  $\mathcal{G}^\times(F)$ . Hence, we want the separator for  $\mathcal{H}_2$  to be balanced with respect to  $P$ . We will also need the separator for  $\mathcal{H}_2$  to be connected. Both properties are guaranteed by the following version of the Planar Separator Theorem, which was proved by Djidjev and Venkatesan [11].

**Planar Separator Theorem.** Let  $\mathcal{G} = (V, E)$  be a maximal planar graph with  $n$  nodes. Let each node  $v \in V$  have a non-negative cost, denoted  $\text{cost}(v)$ , with  $\sum_{v \in V} \text{cost}(v) = 1$ . Then  $V$  can be partitioned in  $O(n)$  time into three sets  $A, B, \mathcal{S}$  such that (i)  $\mathcal{S}$  is a simple cycle of size  $O(\sqrt{n})$ , (ii)  $\mathcal{G}$  has no arcs between a node in  $A$  and a node in  $B$ , and (iii)  $\sum_{v \in A} \text{cost}(v) \leq 2/3$  and  $\sum_{v \in B} \text{cost}(v) \leq 2/3$ .



When applying the Planar Separator Theorem to  $\mathcal{H}_2$ , we set  $\text{cost}(p) := 1/n$  for all nodes  $p_f \in P$  and  $\text{cost}(v) := 0$  for all other nodes. We denote the resulting separator for  $\mathcal{H}_2$  by  $\mathcal{S}(\mathcal{H}_2)$  and the node sets inside and outside the separator by  $A(\mathcal{H}_2)$  and  $B(\mathcal{H}_2)$ , respectively.

#### Step 4: Turning the separator for $\mathcal{H}_2$ into a clique-based separator for $\mathcal{G}^\times(F)$

We convert  $\mathcal{S}(\mathcal{H}_2)$  into a clique-based separator  $\mathcal{S}$  for  $\mathcal{G}^\times(F)$  as follows.

- For each node  $p_f \in \mathcal{S}(\mathcal{H}_2) \cap P$  we put the (singleton) clique  $\{f\}$  into  $\mathcal{S}$ .
- For each gadget  $G_q$  we proceed as follows. Let  $V_q$  be the set of all nodes  $v \in \mathcal{T}_q$  that are in  $\mathcal{S}(\mathcal{H}_2)$ , and define  $C_q := \{f \in F : p_f \text{ is a leaf of } \mathcal{T}_q(v) \text{ that has an ancestor in } V_q\}$ ; see Figure 2(iii). Observe that  $C_q$  is a clique in  $\mathcal{G}^\times(F)$ . We add<sup>5</sup>  $C_q$  to  $\mathcal{S}$ .

The clique-based separator  $\mathcal{S}$  induces a partition of  $F \setminus \bigcup_{C \in \mathcal{S}} C$  into two parts  $A$  and  $B$ , with  $|A|, |B| \leq 2n/3$ , in a natural way, namely as  $A := \{f \in F : f \notin \bigcup_{C \in \mathcal{S}} C \text{ and } p_f \in A(\mathcal{H}_2)\}$  and  $B := \{f \in F : f \notin \bigcup_{C \in \mathcal{S}} C \text{ and } p_f \in B(\mathcal{H}_2)\}$ . The proof that  $\mathcal{S}$  is a valid separator, that is, there are no edges between objects in  $A$  and  $B$ , can be found in the full version [10]. It follows from the fact that for any arc  $(f, f')$  with witness  $q$ , either an ancestor of  $p_f$  or  $p_{f'}$  is in  $V_q$  (and so  $f$  or  $f'$  is in the separator) or  $p_f$  and  $p_{f'}$  (and, hence,  $f$  and  $f'$ ) are in the same component after removing the separator. It remains to prove that  $\mathcal{S}$  has the desired weight.

► **Lemma 2.** *The total weight of the separator  $\mathcal{S}$  satisfies  $\sum_{C \in \mathcal{S}} \log(|C| + 1) = O(\sqrt{n})$ .*

**Proof.** Since  $\mathcal{S}(\mathcal{H}_2)$  contains  $O(\sqrt{n})$  nodes, it suffices to bound the total weight of the cliques added for the gadgets  $G_q$ . Consider a gadget  $G_q$ . Recall that  $V_q$  is the set of all nodes  $v \in \mathcal{T}_q$  that are in  $\mathcal{S}(\mathcal{H}_2)$ . We claim that  $\log(|C_q| + 1) = O(|V_q|)$ , which implies that  $\sum_q \log(|C_q| + 1) = \sum_q O(|V_q|) = O(\sqrt{n})$ , as desired. It remains to prove the claim.

Since  $\mathcal{S}(\mathcal{H}_2)$  is a simple cycle, its intersection with  $G_q$  consists of one or more paths. Each path  $\pi$  enters and exits  $G_q$  at a node in  $N(q)$ . Let  $D_\pi$  denote the set of all descendants of the nodes in  $\pi$ . We will prove that  $\log(|D_\pi| + 1) = O(|\pi|)$ , where  $|\pi|$  denotes the number of nodes of  $\pi$ . This implies the claim since  $\log(|C_q| + 1) \leq \sum_\pi \log(|D_\pi| + 1) = \sum_\pi O(|\pi|) = O(|V_q|)$ .

To prove that  $\log(|D_\pi| + 1) = O(|\pi|)$ , let  $h_{\max}$  be the maximum height of any node in  $\pi$ . Thus  $|\pi| \geq h_{\max}$  by Observation 1(ii). Consider all subtrees of height  $h_{\max}$  in  $\mathcal{T}_q$ . If  $\pi$  visits  $t$  such subtrees, then  $|\pi| \geq t$ . Moreover,  $|D_\pi| \leq 3t \cdot 2^{h_{\max}-1}$  by Observation 1(i). Hence,  $\log(|D_\pi| + 1) \leq \log(3t \cdot 2^{h_{\max}-1} + 1) < h_{\max} + \log(3t) = O(\max(h_{\max}, t)) = O(|\pi|)$ . ◀

By putting everything together we obtain the following theorem.

► **Theorem 3.** *Let  $F$  be a set of  $n$  interior-disjoint regions in the plane. Then the intersection graph  $\mathcal{G}^\times(F)$  has a clique-based balanced separator of weight  $O(\sqrt{n})$ . The separator can be computed in  $O(n)$  time, assuming that the total complexity of the objects in  $F$  is  $O(n)$ .*

### 3 Pseudo-disk graphs

Our clique-based separator construction for a set  $F$  of pseudo-disks uses so-called planar supports, defined as follows. Let  $\mathcal{H}$  be a hypergraph with node set  $Q$  and hyperedge set  $H$ . A graph  $\mathcal{G}_{\text{sup}}$  is a *planar support* [26] for  $\mathcal{H}$  if  $\mathcal{G}_{\text{sup}}$  is a planar graph with node set  $Q$  such that for any hyperedge  $h \in H$  the subgraph of  $\mathcal{G}_{\text{sup}}$  induced by the nodes in  $h$  is connected. In our application we let the node set  $Q$  correspond to a set of points stabbing all pairwise

<sup>5</sup> We tacitly assume that if an object is in multiple cliques in  $\mathcal{S}$ , we remove all but one of its occurrences.

intersections between the pseudo-disks, that is, for each intersecting pair  $f, f' \in F$  there will be a point  $q \in Q$  that lies in  $f \cap f'$ . The goal is to keep the size of  $Q$  small, by capturing all intersecting pairs with few points. The hyperedges are defined by the regions in  $F$ , that is, for every  $f \in F$  there is a hyperedge  $h_f := Q \cap f$ . Let  $\mathcal{H}_Q(F)$  denote the resulting hypergraph.

► **Lemma 4.** *Let  $F$  be a set of  $n$  objects in the plane, let  $Q$  be a set of points stabbing all pairwise intersections in  $F$ , and let  $\mathcal{H}_Q(F)$  denote the hypergraph as defined above. If  $\mathcal{H}_Q(F)$  has a planar support  $\mathcal{G}_{\text{sup}}$  then  $\mathcal{G}^\times(F)$  has a clique-based separator of size  $O(\sqrt{|Q|})$  and weight  $O(\sqrt{|Q|} \log n)$ .*

**Proof.** Let  $\mathcal{S}(\mathcal{G}_{\text{sup}})$  be a separator for  $\mathcal{G}_{\text{sup}}$  of size  $O(\sqrt{|Q|})$ , which exists by the Planar Separator Theorem, and let  $A(\mathcal{G}_{\text{sup}})$  and  $B(\mathcal{G}_{\text{sup}})$  be the corresponding separated parts. To ensure an appropriately balanced separator we use the cost-balanced version of the Planar Separator Theorem, as stated in the previous section. For each object  $f \in F$  we give one point  $q_f \in Q \cap f$  a cost of  $1/n$  and all other points cost 0. We call  $q_f$  the *representative* of  $f$ . (We assume for simplicity that each  $f \in F$  intersects at least one other object  $f' \in F$ , so we can always find a representative. Objects  $f \in F$  not intersecting any other object are singletons in  $\mathcal{G}^\times(F)$  and can be ignored.) For a point  $q \in Q$ , define  $C_q$  to be the clique in  $\mathcal{G}^\times(F)$  consisting of all objects  $f \in F$  that contain  $q$ . Our clique-based separator  $\mathcal{S}$  for  $\mathcal{G}^\times(F)$  is now defined as  $\mathcal{S} := \{C_q : q \in \mathcal{S}(\mathcal{G}_{\text{sup}})\}$ , and the two separated parts are defined as:  $A := \{f \in F : f \notin \mathcal{S} \text{ and } q_f \in A(\mathcal{G}_{\text{sup}})\}$  and  $B := \{f \in F : f \notin \mathcal{S} \text{ and } q_f \in B(\mathcal{G}_{\text{sup}})\}$ . Clearly, the size of  $\mathcal{S}$  is  $O(\sqrt{|Q|})$  and its weight is  $O(\sqrt{|Q|} \log n)$ . Moreover,  $|A|, |B| \leq 2n/3$  because  $\mathcal{S}(\mathcal{G}^*)$  is balanced with respect to the node costs.

We claim there are no arcs in  $\mathcal{G}^\times(F)$  between a node in  $A$  and a node in  $B$ . Suppose for a contradiction that there are intersecting objects  $f, f'$  such that  $f \in A$  and  $f' \in B$ . By definition of  $Q$  there is a point  $q \in Q$  that lies in  $f \cap f'$ . By the planar-support property, the hyperedge  $h_f$  induces a connected subgraph of  $\mathcal{G}_{\text{sup}}$ , so there is a path  $\pi$  that connects  $q$  to the representative  $q_f$  and such that all nodes of  $\pi$  are points in  $f \cap Q$ . No node on the path  $\pi$  can be in  $\mathcal{S}(\mathcal{G}_{\text{sup}})$ , otherwise  $f$  is in a clique that was added to  $\mathcal{S}$ . Similarly, there is a path  $\pi'$  connecting  $q_{f'}$  to  $q$  such that no point on  $\pi'$  is in  $\mathcal{S}(\mathcal{G}_{\text{sup}})$ . But then there is a path from  $q_f$  to  $q_{f'}$  in  $\mathcal{G}_{\text{sup}}$  after the removal of  $\mathcal{S}(\mathcal{G}_{\text{sup}})$ . Hence,  $q_f$  and  $q_{f'}$  are in the same part of the partition, which contradicts that  $f$  and  $f'$  are in different parts.

We conclude that  $\mathcal{S}$  is a clique-based separator with the desired properties. ◀

► **Remark 5.** The witness set  $Q$  in the previous section stabs all pairwise intersections of objects in the map graph, and so  $P \cup Q$  stabs all pairwise intersections as well.  $P \cap Q$  has planar support, so we can get a separator for map graphs using Lemma 4. Its weight would be  $O(\sqrt{n} \log n)$ , however, while in the previous section we managed to get  $O(\sqrt{n})$  weight.

### Polygonal pseudo-disks

We now apply Lemma 4 to obtain a clique-based separator for a set  $F$  of polygonal pseudo-disks. To this end, let  $Q$  be the set of vertices of the pseudo-disks in  $F$ . Observe that whenever two pseudo-disks intersect, one must have a vertex inside the other. Indeed, either one pseudo-disk is entirely inside the other, or an edge  $e$  of  $f$  intersects an edge  $e'$  of  $f'$ . In the latter case, one of the two edges ends inside the other pseudo-disk, otherwise there are three intersections between the boundaries. Furthermore, pseudo-disks have the *non-piercing*

property:  $f \setminus f'$  is connected for any two pseudo-disks  $f, f'$ . Raman and Ray [26] proved<sup>6</sup> that the hypergraph  $\mathcal{H}_Q(F)$  of a set of non-piercing regions has a planar support for any set  $Q$ , so in particular for the set  $Q$  just defined. We can thus apply Lemma 4 to compute a clique-based separator for  $\mathcal{G}^\times(F)$ . The time to compute the separator is dominated by the computation of the planar support, which takes  $O(n^3)$  time [26].

► **Theorem 6.** *Let  $F$  be a set of  $n$  polygonal pseudo-disks in the plane with  $O(n)$  vertices in total. Then the intersection graph  $\mathcal{G}^\times(F)$  has a clique-based balanced separator of size  $O(\sqrt{n})$  and weight  $O(\sqrt{n} \log n)$ , which can be found in  $O(n^3)$  time.*

### Arbitrary pseudo-disks

To construct a clique-based separator using Lemma 4 we need a small point set  $Q$  that stabs all pairwise intersections. Unfortunately, for general pseudo-disks a linear-size set  $Q$  that stabs all intersections need not exist: there is a collection of  $n$  disks such that stabbing all pairwise intersections requires  $\Omega(n^{4/3})$  points. (Such a collection can be derived from a construction with  $n$  lines and  $n$  points with  $\Omega(n^{4/3})$  incidences [23].) Hence, we need some more work before we can apply Lemma 4.

Our separator result for arbitrary pseudo-disks works in a more general setting, namely for sets from a family  $\mathcal{F}$  with linear union complexity. (We say that  $\mathcal{F}$  has union complexity  $U(n)$  if, for any  $n \geq 1$  and any subset  $F \subset \mathcal{F}$  of size  $n$ , the union complexity of  $F$  is  $U(n)$ .) Recall that the union complexity of a family of pseudo-disks is  $O(n)$  [15]. The next theorem states that such sets admit a clique-based separator of sublinear weight. Note that the bound only depends on the number of objects, not on their complexity.

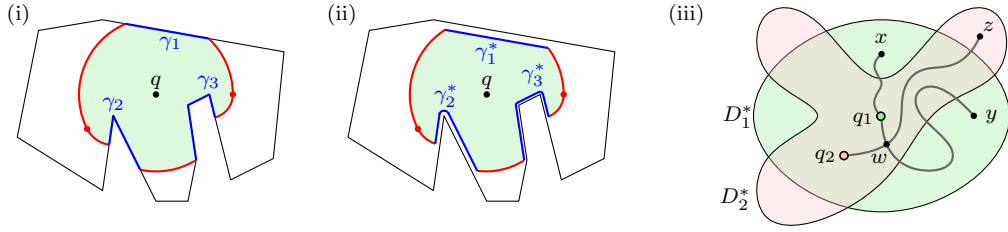
► **Theorem 7.** *Let  $F$  be a set of  $n$  objects from a family  $\mathcal{F}$  of union complexity  $U(n)$ , where  $U(n) \geq n$ . Then  $\mathcal{G}^\times(F)$  has a clique-based separator of size  $O((U(n))^{2/3})$  and weight  $O((U(n))^{2/3} \log n)$ . In particular, if  $F$  is a set of pseudo-disks then  $\mathcal{G}^\times(F)$  has a clique-based separator of size  $O(n^{2/3})$  and weight  $O(n^{2/3} \log n)$ . The separator can be computed in  $O(n^3)$  time, assuming the total complexity of the objects is  $O(n)$ .*

**Proof.** We construct the separator  $\mathcal{S}$  in two steps.

The first step proceeds as follows. For a point  $p$  in the plane, let  $C_p$  denote the set of objects from the (current) set  $F$  containing  $p$ . As long as there is a point  $p$  such that  $|C_p| > n^{1/3}$ , we remove  $C_p$  from  $F$  and put  $C_p$  into  $\mathcal{S}$ ; here  $n$  refers to the size of the initial set  $F$ . Thus the first step adds  $O(n^{2/3})$  cliques to  $\mathcal{S}$  with total weight  $O(n^{2/3} \log n)$ . This step can easily be implemented in  $O(n^3)$  time.

In the second step we have a set  $F^* \subseteq F$  of  $n^*$  objects with ply  $k$ , where  $n^* \leq n$  and  $k \leq n^{1/3}$ . Let  $\mathcal{A}(F^*)$  denote the arrangement induced by  $F^*$ . Since  $F^*$  has ply  $k$ , the Clarkson-Shor technique [7] implies that the complexity of the arrangement  $\mathcal{A}(F^*)$  is  $O(k^2 \cdot U(n^*/k))$ . We can compute this arrangement in  $O(k^2 \cdot U(n^*/k) \log n) = O(n^2 \log n)$  time [9]. Take a point  $q$  in each face of the arrangement, and let  $Q$  be the resulting set of  $O(k^2 \cdot U(n^*/k))$  points. The set  $Q$  stabs all pairwise intersections and the dual graph  $\mathcal{G}^*$  of the arrangement  $\mathcal{A}(F^*)$  is a planar support for the hypergraph  $\mathcal{H}_Q(F)$ . Hence, by Lemma 4 there is a clique-based separator  $\mathcal{S}^*$  for  $\mathcal{G}^\times(F)$  of size  $O\left(k\sqrt{U(n^*/k)}\right)$  and

<sup>6</sup> Raman and Ray assume the sets  $F$  and  $Q$  defining the hypergraph are in general position. Therefore we first slightly perturb the pseudo-disks in  $F$  to get them into general position (while keeping the same intersection graph), then we take  $Q$  to be a point set coinciding with the vertex set of  $F$ , and then we slightly move the points in  $Q$  such that the hypergraph remains the same.



■ **Figure 3** (i) A geodesic disk  $D$  with center  $q$  and radius  $r$ . The set  $\Gamma(D)$  has three pieces,  $\gamma_1, \gamma_2$  and  $\gamma_3$ , shown in blue. (ii) The result of the perturbation. Note that  $|\gamma_1^*| < |\gamma_2^*| < |\gamma_3^*|$  and so  $\varepsilon_{\gamma_1^*} > \varepsilon_{\gamma_2^*} > \varepsilon_{\gamma_3^*}$ . (iii) Illustration for the proof of Theorem 8.

weight  $O\left(k\sqrt{U(n^*/k)}\log n^*\right)$ . Note that  $U(n)$  is a superadditive function [1] which implies that  $U(n/k) \leq U(n)/k$  and therefore  $k\sqrt{U(n^*/k)} \leq \sqrt{k}U(n) \leq (U(n))^{2/3}$ . By adding  $\mathcal{S}^*$  to the set  $\mathcal{S}$  of cliques generated in the first step, we obtain a clique-based separator with the desired properties.  $\blacktriangleleft$

#### 4 Geodesic disks inside a simple polygon

Let  $P$  be a simple polygon. We denote the shortest path (or: *geodesic*) in  $P$  between two points  $p, q \in P$  by  $\pi(p, q)$ ; note that  $\pi(p, q)$  is unique since  $P$  is simple. The *geodesic distance* between  $p$  and  $q$  is defined to be  $\|\pi(p, q)\|$ , where  $\|\pi\|$  denotes the Euclidean length of a path  $\pi$ . For a given point  $q \in P$  and radius  $r > 0$ , we call the region  $D(q, r) := \{p \in P : \|\pi(p, q)\| \leq r\}$  a *geodesic disk*. Let  $\mathcal{D} = \{D_1, \dots, D_n\}$  be a set of geodesic disks in  $P$ . To construct a clique-based separator for  $\mathcal{G}^\times(\mathcal{D})$  we will show that  $\mathcal{D}$  behaves as a set of pseudo-disks so we can apply the result of the previous section.

##### The structure of a geodesic disk

The boundary  $\partial D(q, r)$  of a geodesic disk  $D(q, r)$  consists of circular arcs lying in the interior of  $P$  (centered at  $q$  or at a reflex vertex of  $P$ ) and parts of the edges of  $P$ . We split  $\partial D(q, r)$  into *boundary pieces* at the points where the circular arcs meet  $\partial P$ . This generates two sets of boundary pieces: a set containing the pieces that consist of circular arcs, and a set  $\Gamma(D)$  containing the pieces that consist of parts of edges of  $P$ . An example can be seen in Fig. 3.

A region  $R \subseteq P$  is *geodesically convex* if for any points  $p, q \in R$  we have  $\pi(p, q) \subseteq R$ . Pollack et al. [25] showed that geodesic disks inside a simple polygon are geodesically convex. An immediate consequence is that the intersection of two geodesic disks is connected.

##### Geodesic disks behave as pseudo-disks

Geodesic disks in a simple polygon are not proper pseudo-disks. For example, if  $D_1$  and  $D_2$  are the blue and pink pseudo-disk in the third image in Fig. 1, then  $D_1 \setminus D_2$  has two components, which is not allowed for pseudo-disks. Nevertheless, we will show that  $\mathcal{D}$  behaves as a set of pseudo-disks in the sense that a small perturbation turns them into pseudo-disks, while keeping the intersection graph the same.

As a first step in the perturbation, we increase the radius of each geodesic disk  $D_i \in \mathcal{D}$  by some small  $\varepsilon_i$ . We pick these  $\varepsilon_i$  such that the intersection graph  $\mathcal{G}^\times(\mathcal{D})$  stays the same while all degeneracies disappear. In particular, the boundary pieces of different geodesic disks have different lengths after this perturbation, and no two geodesic disks touch. With a slight abuse of notation, we still denote the resulting set of geodesic disks by  $\mathcal{D}$ .

The second step in the perturbation moves each  $\gamma \in \cup_{i=1}^n \Gamma(D_i)$  into the interior of the polygon over some distance  $\varepsilon_\gamma$ , which is smaller than any of the perturbation distances chosen in the first step. More formally, for each  $\gamma \in \Gamma(D_i)$  we remove all points from  $D_i$  that are at distance less than  $\varepsilon_\gamma$  from  $\gamma$ ; see Fig. 3(ii). To ensure this gives a set of pseudo-disks we choose the perturbation distances  $\varepsilon_\gamma$  according to the reverse order of the Euclidean lengths of the pieces. That is, if  $\|\gamma\| > \|\gamma'\|$  then we pick  $\varepsilon_\gamma < \varepsilon_{\gamma'}$ . The crucial property of this scheme is that whenever  $\gamma_i \subset \gamma_j$  then  $\gamma_i$  is moved more than  $\gamma_j$ .

We denote the perturbed version of  $D_i$  by  $D_i^*$  and define  $\mathcal{D}^* := \{D_i^* : D_i \in \mathcal{D}\}$ . The perturbed versions have the following important property: every connected component of  $D_i^* \setminus D_j^*$  contains a point  $u$  with  $u \in D_i \setminus D_j$ , see [10].

► **Theorem 8.** *Any set  $\mathcal{D}$  of geodesic disks inside a simple polygon  $P$  can be slightly perturbed such that the resulting set  $\mathcal{D}^*$  is a set of pseudo-disks with  $\mathcal{G}^\times(\mathcal{D}) = \mathcal{G}^\times(\mathcal{D}^*)$ .*

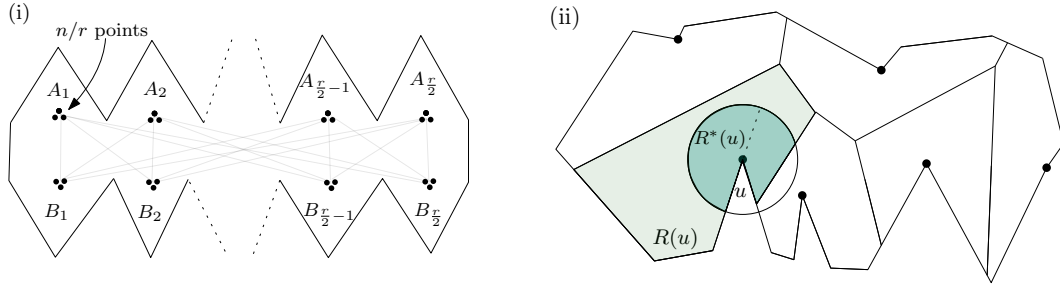
**Proof.** Consider the set  $\mathcal{D}^*$  resulting from the perturbation described above. Suppose for a contradiction that there exist two objects  $D_1^*, D_2^* \in \mathcal{D}^*$  such that  $\partial D_1^*$  and  $\partial D_2^*$  cross four or more times. Recall that the intersection of two geodesic disks is connected. This property is not invalidated by the perturbation. Hence, if  $\partial D_1^*$  and  $\partial D_2^*$  cross four or more times then  $D_1^* \setminus D_2^*$  (and, similarly,  $D_2^* \setminus D_1^*$ ) has two or more components.

For  $i = 1, 2$ , let  $q_i$  and  $r_i$  denote the center and radius of  $D_i$ . Without loss of generality assume that  $r_1 \leq r_2$ . Let  $x$  and  $y$  be points in different components of  $D_1^* \setminus D_2^*$ ; see Fig. 3 (iii). We can pick  $x$  and  $y$  such that  $x, y \in D_1 \setminus D_2$ . By concatenating the geodesics  $\pi(x, q_1)$  and  $\pi(q_1, y)$  we obtain a curve that splits  $D_2^*$  into at least two parts – this is independent of where  $q_1$  lies, or whether  $\pi(x, q_1)$  and  $\pi(q_1, y)$  partially overlap. (Note that these geodesics lie in  $D_1$  but not necessarily in  $D_1^*$ . However, they cannot “go around” a component of  $D_2^* \setminus D_1^*$ , because  $D_1$  cannot fully contain such a component. Hence,  $\pi(x, q_1) \cup \pi(q_1, y)$  must indeed go through  $D_2^*$ .) Not all components of  $D_2^* \setminus D_1^*$  can belong to the same part, otherwise  $x$  and  $y$  would not be in different components of  $D_1^* \setminus D_2^*$ . Take a point  $z \in D_2^* \setminus D_1^*$  that lies in a different part than  $q_2$ , the center of  $D_2$ . Again we can pick  $z$  such that  $z \in D_2 \setminus D_1$ . Then the geodesic  $\pi(q_2, z)$  must cross  $\pi(x, q_1) \cup \pi(q_1, y)$ , say at a point  $w \in \pi(q_1, y)$ . Since  $z \notin D_1$  and  $y \notin D_2$  we must have  $\|\pi(q_1, w) \cup \pi(w, z)\| + \|\pi(q_2, w) \cup \pi(w, y)\| > r_1 + r_2$ . But this gives a contradiction because  $y \in D_1$  and  $z \in D_2$  implies  $\|\pi(q_1, w) \cup \pi(w, y)\| + \|\pi(q_2, w) \cup \pi(w, z)\| \leq r_1 + r_2$ .

It remains to show that  $\mathcal{G}^\times(\mathcal{D}) = \mathcal{G}^\times(\mathcal{D}^*)$ . As mentioned earlier, the increase of the radii in the first step of the perturbation is chosen sufficiently small so that no new intersections are introduced. The second step shrinks the geodesic disks, so no new intersections are introduced in that step either. Finally, the fact that the perturbations in the second step are smaller than in the first step guarantees that no intersections are removed. ◀

Theorem 8 allows us to apply Theorem 7. When doing so, we actually do not need to perturb the geodesic disks. We only use the perturbation to argue that the number of faces in the arrangement defined by  $n$  geodesic disks of ply  $k$  is  $O(nk)$ . Computing the geodesic disks (and then computing the separator) can be done in polynomial time in  $n$  and the number of vertices of  $P$ . We obtain the following result.

► **Corollary 9.** *Let  $\mathcal{D}$  be a set of  $n$  geodesic disks inside a simple polygon with  $m$  vertices. Then  $\mathcal{G}^\times(\mathcal{D})$  has a clique-based separator of size  $O(n^{2/3})$  and weight  $O(n^{2/3} \log n)$ , which can be computed in time polynomial in  $n$  and  $m$ .*



■ **Figure 4** (i) Each cluster  $A_i$  sees any of the clusters  $B_j$  completely and all distances are at most 1, so a separator that splits  $\mathcal{G}_{\text{vis},P}^\times(\bigcup A_i \cup \bigcup B_j)$  into two or more components must fully contain  $\bigcup A_i$  or  $\bigcup B_j$ . Since the clusters  $A_i$  (and similarly  $B_j$ ) do not see each other, such a separator has size at least  $r/2$  and weight at least  $((r/2) \log(n/r))$ . (ii) Splitting  $R^*(u)$  into two convex parts.

## 5 Visibility-restricted unit-disk graphs inside a polygon

Let  $P$  be a polygon, possibly with holes, and let  $Q$  be a set of  $n$  points inside  $P$ . We define  $\mathcal{G}_{\text{vis},P}^\times(Q)$  to be the visibility-restricted unit-disk graph of  $Q$ . The nodes in  $\mathcal{G}_{\text{vis},P}^\times(Q)$  correspond to the points in  $Q$  and there is an edge between two points  $p, q \in Q$  iff  $|pq| \leq 1$  and  $p$  and  $q$  see each other. A vertex of  $P$  is *reflex* if its angle within the polygon is more than 180 degrees; note that for a vertex of a hole we look at the angle within  $P$ , not within the hole. Below we sketch a proof of the following theorem; a detailed proof is in the arXiv version of the paper [10].

► **Theorem 10.** *Let  $Q$  be a set of  $n$  points inside a polygon (possibly with holes) with  $r$  reflex vertices. Then  $\mathcal{G}_{\text{vis},P}^\times(Q)$  admits a clique-based separator of size  $O(\min(n, r) + \sqrt{n})$  and weight  $O(\min(n, r \log(n/r) + \sqrt{n}))$ . The bounds on the size and weight of the separator are tight in the worst case, even for simple polygons.*

### The lower bound

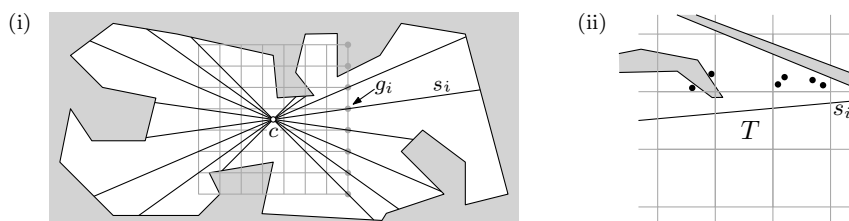
Recall that even for non-visibility restricted unit-disk graphs,  $\Omega(\sqrt{n})$  is a lower bound on the worst-case size of the separator. Hence, to prove the lower bound of Theorem 10 it suffices to give an example where the size and weight are  $\Omega(\min(n, r))$  and  $\Omega(\min(n, r \log(n/r)))$ , respectively. This example is given in Fig. 4(i).

### The upper bound

We now sketch our separator construction for the case where  $P$  is a simple polygon. The extension to polygons with holes can be found in the full version [10].

**Step 1: Handling points that see a nearby reflex vertex.** Let  $V_{\text{ref}}$  be the set of reflex vertices of  $P$ , and let  $Q_1 \subseteq Q$  be the set of points that can see a reflex vertex within distance  $\sqrt{2}$ . Consider the geodesic Voronoi diagram of  $V_{\text{ref}}$  within  $P$ . Let  $R(u)$  be the Voronoi region of vertex  $u \in V_{\text{ref}}$  and define  $R^*(u) := R(u) \cap D(u, \sqrt{2})$ . Note that all points in  $R(u)$  can see  $u$  and that all points in  $Q_1$  are in  $S^*(u)$  for some vertex  $u \in V_{\text{ref}}$ . By extending one of the edges of  $P$  incident to  $u$ , we can split  $R^*(u)$  into two convex parts; see Fig. 4(ii). Since  $R^*(u)$  has diameter  $O(1)$ , this means that  $Q_1 \cap R^*(u)$  can be partitioned into  $O(1)$  cliques. We collect all these cliques into a set  $\mathcal{S}_1$ . Since there are  $r$  reflex vertices,  $\mathcal{S}_1$  consists of  $O(r)$  cliques of total weight is  $O(r \log(n/r))$ .





■ **Figure 5** (i) The grid  $G$  defining the chords  $s_i$ . (ii) The points in the top-left cell see a reflex vertex so they are not in  $Q_2(s_i)$ . The points in the top-right cell can be split into  $O(1)$  cliques.

**Step 2: Handling points that do not see a nearby reflex vertex.** Our separator  $\mathcal{S}$  consists of the cliques in  $\mathcal{S}_1$  plus a set  $\mathcal{S}_2$  of cliques that are found as follows. Let  $Q_2 := Q \setminus Q_1$  be the set of points that do not see a reflex vertex within distance  $\sqrt{2}$ . Let  $c$  be a *centerpoint* for  $Q_2$  inside  $P$ , that is, a point such that any (maximal) chord through  $c$  splits  $P$  into half-polygons containing at most  $2|Q_2|/3$  points from  $Q_2$ . Such a point always exists; see [10]. Let  $G$  be a  $\sqrt{n} \times \sqrt{n}$  grid of unit cells centered at  $c$ . For each of the  $\sqrt{n}$  points  $g_i$  in the rightmost column of the grid (even if  $g_i \notin P$ ), we define a chord  $s_i$  by taking the line  $\ell_i$  through  $c$  and  $g_i$  and then taking the component of  $\ell_i \cap P$  that contains  $c$ ; see Fig. 5(i).

For each chord  $s_i$ , we define  $Q_2(s_i)$  to be the set of points  $q \in Q_2$  such that there is a point  $z \in s_i$  that sees  $q$  with  $|qz| \leq 1/2$ . Note that  $\mathcal{G}_{\text{vis},P}^\times(Q)$  cannot have an arc between a point  $p \in Q_2 \setminus Q_2(s_i)$  above  $s_i$  and a point  $q \in Q_2 \setminus Q_2(s_i)$  below  $s_i$ ; otherwise  $p$  and/or  $q$  see a point on  $s_i$  within distance  $1/2$ , and so at least one of  $p, q$  is in  $Q_2(s_i)$ . Since  $s_i$  is a chord through the centerpoint  $c$ , this means that  $s_i$  induces a balanced separator.

It remains to argue that at least one chord  $s_i$  induces a separator of small weight. We will do this by creating a set  $\mathcal{S}(s_i)$  of cliques for each chord  $s_i$ , and prove that the total weight of these cliques, over all chords  $s_i$ , is  $O(n)$ . Since there are  $\sqrt{n}$  chords, one of them has the desired weight.

First, we put all points from  $Q_2(s_i)$  that lie outside the grid  $G$  into  $\mathcal{S}(s_i)$ , as singletons. By definition of the chords  $s_i$ , a point  $q$  outside the grid  $G$  lies at distance at most  $1/2$  from  $O(1)$  chords. Hence, the total number of singleton cliques over all sets  $\mathcal{S}(s_i)$  is  $O(n)$ .

Next, consider a cell  $T$  of the grid  $G$ . Suppose a point  $q \in Q_2(s_i)$  sees a point  $z \in T \cap s_i$  with  $|qz| \leq 1/2$ . Then  $q$  must lie inside one of the nine grid cells surrounding and including  $T$ . Consider such a cell  $T'$ . The points in  $Q_2(s_i) \cap T'$  that can see a point on  $s_i \cap T$  can be partitioned into  $O(1)$  cliques; we prove this by showing that if two such points do not see each other, then they must see a reflex vertex within distance  $\sqrt{2}$  and, hence, be in  $Q_1$ . We thus create  $O(1)$  cliques for  $T'$  and put them into  $\mathcal{S}(s_i)$ . This adds at most  $O(\log(n_{T'} + 1))$  weight to  $\mathcal{S}(s_i)$ , where  $n_{T'} := |Q_2 \cap T'|$ .

Overall, a cell  $T'$  adds  $O(\log(n_{T'} + 1))$  weight for the chords  $s_i$  that cross the nine cells surrounding it. Since there are  $n$  cells in total, this immediately gives a total weight of  $O(n \log n)$  over all sets  $\mathcal{S}(s_i)$ . A more careful analysis shows that the total weight of the cliques is actually  $O(n)$ . Hence, one of the  $\sqrt{n}$  chords induces a separator of the desired weight. This finishes the sketch of the construction of the clique-based separator for  $\mathcal{G}_{\text{vis},P}^\times(Q)$ .

## 6 Concluding Remarks

We showed how clique-based separators with sub-linear weight can be constructed for various classes of intersection graphs which involve non-fat objects. The main advantage of our approach is that we can solve different problems in the graph classes we study in a uniform manner. There are several natural questions that are left open. Some are listed below.



- **Improving the bound for geodesic disks and adding holes.** Our bound on geodesic disks is directly derived by our result on pseudo-disks. However, geodesic disks are much less general than pseudo-disks (and “closer” to regular disks). Hence, one would expect that the optimal weight is closer to  $O(\sqrt{n})$ . If we allow our polygon to have holes, then our approach for geodesic disks no longer works. Indeed, it is easy to see that even after applying our perturbation scheme, the resulting objects can intersect each other more than two times.
- **Improving the bound for pseudo-disks.** Regarding pseudo-disks, an interesting result [24] states that in every finite family of pseudo-disks in the plane one can find a “small” one, in the sense that it is intersected by only a constant number of disjoint pseudo-disks. This property is also shared by, for instance, convex fat objects. Does this mean that the two graph classes are related in some natural way? If yes, could this connection be exploited to construct separators with better bounds?

---

### References

- 1 B. Aronov, M.T. Berg, de, E. Ezra, and M. Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM Journal on Computing*, 43(2):543–572, 2014. doi:10.1137/120891241.
- 2 Boaz Ben-Moshe, Olaf A. Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. Computing the visibility graph of points within a polygon. In *Proc. 20th ACM Symposium on Computational Geometry*, pages 27–35. ACM, 2004. doi:10.1145/997817.997825.
- 3 Édouard Bonnet and Pawel Rzazewski. Optimality program in segment and string graphs. *Algorithmica*, 81(7):3047–3073, 2019. doi:10.1007/s00453-019-00568-7.
- 4 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003. doi:10.1016/S0196-6774(02)00294-8.
- 5 Zhi-Zhong Chen. Approximation algorithms for independent sets in map graphs. *J. Algorithms*, 41(1):20–40, 2001. doi:10.1006/jagm.2001.1178.
- 6 Zhi-Zhong Chen, Michelangelo Grigni, and Christos H. Papadimitriou. Map graphs. *J. ACM*, 49(2):127–138, 2002. doi:10.1145/506147.506148.
- 7 Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discret. Comput. Geom.*, 4:387–421, 1989. doi:10.1007/BF02187740.
- 8 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for Exponential-Time-Hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM J. Comput.*, 49:1291–1331, 2020. doi:10.1137/20M1320870.
- 9 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications (3rd Edition)*. Springer, 2008. URL: <https://www.worldcat.org/oclc/227584184>.
- 10 Mark de Berg, Sándor Kisfaludi-Bak, Morteza Monemizadeh, and Leonidas Theocharous. Clique-based separators for geometric intersection graphs, 2021. arXiv:2109.09874.
- 11 Hristo Djidjev and Shankar M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34(3):231–243, 1997. doi:10.1007/s002360050082.
- 12 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Decomposition of map graphs with applications. In *Proc. 46th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 60:1–60:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.60.
- 13 Jacob Fox, János Pach, and Csaba D. Tóth. A bipartite strengthening of the crossing lemma. *J. Comb. Theory, Ser. B*, 100(1):23–35, 2010. doi:10.1016/j.jctb.2009.03.005.
- 14 Sariel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM J. Comput.*, 46(6):1712–1744, 2017. doi:10.1137/16M1079336.


- 15 Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discret. Comput. Geom.*, 1:59–70, 1986. doi:10.1007/BF02187683.
- 16 Sándor Kisfaludi-Bak. Hyperbolic intersection graphs and (quasi)-polynomial time. In *Proc. 31st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1621–1638, 2020. doi:10.1137/1.9781611975994.100.
- 17 Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. How does object fatness impact the complexity of packing in  $d$  dimensions? In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPICs*, pages 36:1–36:18, 2019. doi:10.4230/LIPICs.ISAAC.2019.36.
- 18 James R. Lee. Separators in region intersection graphs. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017*, volume 67 of *LIPICs*, pages 1:1–1:8, 2017. doi:10.4230/LIPICs.ITCS.2017.1.
- 19 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1977. doi:doi/10.1137/0136016.
- 20 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. In *Proc. 23rd Annual European Symposium on Algorithms (ESA)*, volume 9294 of *Lecture Notes in Computer Science*, pages 865–877. Springer, 2015. doi:10.1007/978-3-662-48350-3\_72.
- 21 Jirí Matoušek. Near-optimal separators in string graphs. *Comb. Probab. Comput.*, 23(1):135–139, 2014. doi:10.1017/S0963548313000400.
- 22 Gary L. Miller, Shang-Hua Teng, William P. Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997. doi:10.1145/256292.256294.
- 23 János Pach and Micha Sharir. Geometric incidences. In János Pach, editor, *Towards a Theory of Geometric Graphs (Contemporary Mathematics, Vol. 342)*, pages 185–223. Amer. Math. Soc., 2004.
- 24 Rom Pinchasi. A finite family of pseudodiscs must include a “small” pseudodisc. *SIAM Journal on Discrete Mathematics*, 28:1930–1934, October 2014. doi:10.1137/130949750.
- 25 Ricky Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discret. Comput. Geom.*, 4(6):611–626, 1989. doi:10.1007/BF02187751.
- 26 Rajiv Raman and Saurabh Ray. Constructing planar support for non-piercing regions. *Discrete & Computational Geometry*, pages 1–25, 2020.
- 27 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems & applications. In *39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 232–243. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743449.



# Near-Optimal Distance Oracles for Vertex-Labeled Planar Graphs

Jacob Evald

Department of Computer Science, University of Copenhagen, Denmark

Viktor Fredslund-Hansen  

Department of Computer Science, University of Copenhagen, Denmark

Christian Wulff-Nilsen  

Department of Computer Science, University of Copenhagen, Denmark

---

## Abstract

Given an undirected  $n$ -vertex planar graph  $G = (V, E, \omega)$  with non-negative edge weight function  $\omega : E \rightarrow \mathbb{R}$  and given an assigned label to each vertex, a vertex-labeled distance oracle is a data structure which for any query consisting of a vertex  $u$  and a label  $\lambda$  reports the shortest path distance from  $u$  to the nearest vertex with label  $\lambda$ . We show that if there is a distance oracle for undirected  $n$ -vertex planar graphs with non-negative edge weights using  $s(n)$  space and with query time  $q(n)$ , then there is a vertex-labeled distance oracle with  $\tilde{O}(s(n))^1$  space and  $\tilde{O}(q(n))$  query time. Using the state-of-the-art distance oracle of Long and Pettie [12], our construction produces a vertex-labeled distance oracle using  $n^{1+o(1)}$  space and query time  $\tilde{O}(1)$  at one extreme,  $\tilde{O}(n)$  space and  $n^{o(1)}$  query time at the other extreme, as well as such oracles for the full tradeoff between space and query time obtained in their paper. This is the first non-trivial exact vertex-labeled distance oracle for planar graphs and, to our knowledge, for any interesting graph class other than trees.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Data structures design and analysis; Theory of computation  $\rightarrow$  Shortest paths; Mathematics of computing  $\rightarrow$  Graph algorithms

**Keywords and phrases** distance oracle, vertex labels, color distance oracle, planar graph

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.23

**Related Version** *arXiv Version*: <https://arxiv.org/abs/2110.00074>

**Funding** *Jacob Evald*: This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

*Viktor Fredslund-Hansen*: This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

*Christian Wulff-Nilsen*: This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme and by Basic Algorithms Research Copenhagen (BARC).

**Acknowledgements** We are grateful for the thorough and useful feedback provided by the reviewers.

## 1 Introduction

Efficiently answering shortest path distance queries between pairs of vertices in a graph is a fundamental algorithmic problem with a wide range of applications. An algorithm like Dijkstra’s can answer such a query in near-linear time in the size of the graph. If we allow for precomputations, we can break this bound, for instance by simply storing the answers to all possible queries in a look-up table. However, a fast query time should preferably not come at the cost of a large space requirement. A *distance oracle* is a compact data structure that can answer a shortest path distance query in constant or close to constant time.

---

<sup>1</sup> We use  $\tilde{O}$ -notation to suppress  $\text{poly}(\log n)$ -factors.



A lot of research has focused on approximate distance oracles which allow for some approximation in the distances output. This is reasonable since there are graphs for which the trivial look-up table approach is the best possible for exact distances. However, for restricted classes of graphs, it may be possible to obtain exact oracles with a much better tradeoff between space and query time. Indeed, for any planar  $n$ -vertex digraph, there is an exact distance oracle with space close to linear in  $n$  and query time close to constant [7, 2, 12].

A related problem is that of obtaining a vertex-labeled distance oracle. Here, we are given a graph with each vertex assigned a label. A query consists of a pair  $(u, \lambda)$  of a vertex  $u$  and a label  $\lambda$  and the output should be the distance from  $u$  to the nearest vertex with label  $\lambda$ . Each vertex is given only one label but the same label may be assigned to multiple vertices. To give some practical motivation, if the graph represents a road network, a label  $\lambda$  could represent supermarkets and the output of query  $(u, \lambda)$  gives the distance to the nearest supermarket from the location represented by  $u$ .

Note that this is a generalization of the distance oracle problem since vertex-to-vertex distance queries can be answered by a vertex-labeled distance oracle if each vertex is given its own unique label. If  $L$  is the set of labels, a trivial vertex-labeled distance oracle with constant query time is a look-up table that simply stores the answers to all possible queries, requiring space  $O(n|L|)$ . This bound can be as high as quadratic in  $n$ .

Our main result, which we shall state formally later in this section, is that for undirected edge-weighted planar graphs, the vertex-labeled distance oracle problem can be reduced to the more restricted distance oracle problem in the sense that up to  $\log n$ -factors, any space/query time tradeoff for distance oracles also holds for vertex-labeled distance oracles. Hence, the tradeoff from [12] translates to vertex-labeled distance oracles, assuming that the planar graph is undirected. To the best of our knowledge, this is the first non-trivial upper bound for vertex-labeled distance oracles in any interesting graph class other than trees [8, 15]. A strength of our result is that any future progress on distance oracles in undirected planar graphs immediately translates to vertex-labeled distance oracles.

## 1.1 Related work on vertex-labeled distance oracles

Vertex-labeled distance oracles have received considerably more attention in the approximate setting. With  $(1 + \epsilon)$  multiplicative approximation, it is known how to get  $\tilde{O}(n)$  space and  $\tilde{O}(1)$  query time both for undirected [11] and directed planar graphs [13] and it has been shown how oracles with such guarantees can be maintained dynamically under label changes to vertices using  $\tilde{O}(1)$  time per vertex relabel.

For general graphs, vertex-labeled distance oracles with constant approximation have been presented [9, 3, 14] with state of the art being an oracle with  $O(kn|L|^{1/k})$  space,  $4k - 5$  multiplicative approximation, and  $O(\log k)$  query time, for any  $k \in \mathbb{N}$ .

## 1.2 Our contributions

We now state our reduction and its corollary:

► **Theorem 1.** *If there is an exact distance oracle for  $n$ -vertex undirected edge-weighted planar graphs with  $s(n)$  space,  $q(n)$  query time, and  $t(n)$  preprocessing time, then there exists an exact vertex-labeled distance oracle for such graphs with  $s(n) + O(n \log^2 n)$  space, and with  $O(q(n) \log n + \log^3 n)$  query time, and  $t(n) + \text{poly}(n)$  preprocessing time.*

Plugging in the distance oracle of Long and Pettie et al. [12] gives the following corollary which can be seen as a generalization of their result:

► **Corollary 2.** *For  $n$ -vertex undirected edge-weighted planar graphs, there exist exact vertex-labeled distance oracles with the following tradeoffs between space and query time:*

1.  $n^{1+o(1)}$  space and  $\tilde{O}(1)$  query time,
2.  $\tilde{O}(n)$  space and  $n^{o(1)}$  query time.

All oracles have preprocessing time polynomial in  $n$ .

Up to logarithmic factors, the full tradeoff between space and query time in their paper similarly extends to vertex-labeled distance oracles in undirected edge-weighted planar graphs.

The rest of the paper is organized as follows. In Section 2, we introduce basic definitions and notation and present tools from the literature that we will need for our oracle. In Section 3 we state the key lemmas but defer their proofs until later sections, and thus immediately present our reduction by describing how to obtain a vertex-labeled distance oracle given a distance oracle as a black box. In Section 4, we present a point location structure similar to [7] but with some important modifications to improve space in our setting.

## 2 Preliminaries

Let  $G = (V, E, \omega)$  be a graph with edge weight function  $\omega : E \rightarrow \mathbb{R} \cup \{\infty\}$ . We denote by  $V(G) = V$  and  $E(G) = E$  the vertex and edge-set of  $G$ , respectively, and by  $n = |V(G)|$  the number of vertices of  $G$ . A graph  $G'$  is said to be a subgraph of  $G$  if  $V(G') \subseteq V(G)$  and  $E(G') \subseteq E(G)$ . We denote by  $u \rightsquigarrow_G v$  a *shortest path* from  $u$  to  $v$  in  $G$ , by  $\mathbf{d}_G(u, v)$  the weight of  $u \rightsquigarrow_G v$ , and write  $u \rightsquigarrow v = u \rightsquigarrow_G v$  and  $\mathbf{d}(u, v) = \mathbf{d}_G(u, v)$  when  $G$  is clear from context. For a shortest path  $p = u \rightsquigarrow v = (u = p_1), p_2, \dots, (p_k = v)$  we define vertex  $p_i$  to occur *before*  $p_j$  on  $p$  if  $i < j$  and similarly for edges  $p_i p_{i+1}$  and  $p_j p_{j+1}$ . Thus statements such as “the first/last vertex/edge on  $p$  satisfying some property  $P$ ” will always be made w.r.t. this ordering. We also write  $p \circ p'$  to denote the concatenation of paths (or edges)  $p$  and  $p'$ , assuming the last vertex of  $p$  equals the first vertex of  $p'$ . Given  $u, v, v' \in V$ ; we say that  $v$  is *closer* than  $v'$  to  $u$  in  $G$  if  $\mathbf{d}_G(u, v) < \mathbf{d}_G(u, v')$  or  $\mathbf{d}_G(u, v) = \mathbf{d}_G(u, v')$  and  $v < v'$ , assuming some lexicographic ordering on vertices. We denote by  $V(p)$ , respectively  $E(p)$ , the set of vertices, respectively edges, on a path  $p$ .

Assume in the following that  $G$  is undirected.  $G$  is said to be connected, respectively biconnected, if any pair of vertices are connected by at least one, respectively two, vertex-disjoint paths. For a rooted spanning tree  $T$  in  $G$  and for any edge  $e = uv$  not in  $T$ , we define *the fundamental cycle* of  $uv$  w.r.t.  $T$  as the cycle obtained as the concatenation of  $uv$  and the two paths of  $T$  from the root to  $u$  and  $v$ , respectively.

### 2.1 Planar graphs and embeddings

An *embedding* of a planar graph  $G$  assigns to each vertex a point in the plane and to each edge a simple arc such that its endpoints coincide with those of the points assigned to its vertices. A *planar embedding* of  $G$  is an embedding such that no two vertices are assigned the same point and such that no pair of arcs coincide in points other than those corresponding to vertices they share. A graph is said to be *planar* if it admits a planar embedding. When we talk about a planar graph we assume that it is *planar embedded* and hence some implicit, underlying planar embedding of the graph. When it is clear from the context we shall refer interchangeably to a planar graph and its embedding, its edges and arcs and its vertices and points. Thus the term graph can refer to its embedding, an edge to its corresponding arc and a vertex to its corresponding point in the embedding.

### Assumptions about the input

Unless stated otherwise, we shall always assume that  $G$  refers to a graph which is weighted, undirected and planar with some underlying embedding. Furthermore, we shall make the structural assumption that  $G$  is triangulated. Triangulation can be achieved by standard techniques, i.e. adding to each face  $f$  an artificial vertex and artificial edges from the artificial vertex to each vertex of  $V(f)$  with infinite weight. This transformation preserves planarity, shortest paths and ensures that the input graph consists only of simple faces. We also assume that shortest paths in the input graph are unique; this can be ensured for any input graph by either randomly perturbing edge weights or with e.g. the deterministic approach in [6] which gives only an  $O(1)$ -factor overhead in running time. Finally, it will be useful to state the following lemma when talking about separators in a graph with unique shortest paths:

► **Lemma 3.** *Let  $u, v, x, y \in V(G)$ . Then  $u \rightsquigarrow v$  and  $x \rightsquigarrow y$  share at most one edge-maximal subpath.*

**Proof.** Assume that  $x \rightsquigarrow y$  intersects  $u \rightsquigarrow v$  and let  $a$  resp.  $b$  be the first resp. last intersection along  $u \rightsquigarrow v$ . Since  $G$  is undirected, uniqueness of shortest paths implies that  $a \rightsquigarrow b$  is a subpath of  $u \rightsquigarrow v$  shared by  $x \rightsquigarrow y$ . ◀

### Edge orderings, path turns and path intersections

For an edge  $e = uv$  of a planar embedded graph  $H$ , we let  $\langle_e^H$  be the clockwise ordering of edges of  $H$  incident to  $v$  starting at  $e$  (ignoring edge orientations). Hence  $\langle_e^H$  is a strict total order of these edges and  $e$  is the first edge in this order.

For vertices  $u, v \in V(H)$ ,  $x \in V(u \rightsquigarrow v) \setminus \{u, v\}$  and  $y \in V(H) \setminus V(u \rightsquigarrow v)$ , let  $pq$  be the last edge shared by  $u \rightsquigarrow v$  and  $x \rightsquigarrow y$ . Furthermore let  $qz$  resp.  $qz'$  be the edge following  $pq$  in the traversal of  $u \rightsquigarrow v$  and  $x \rightsquigarrow y$ , respectively. We say that  $x \rightsquigarrow y$  *emanates from the left* of  $u \rightsquigarrow v$  if  $qz' \prec_{pq}^H qz$ , and otherwise it *emanates from the right*. We dually say that  $y \rightsquigarrow x$  *intersects  $u \rightsquigarrow v$  from the left (right)* if  $x \rightsquigarrow y$  *emanates from the left (right)*.

Given a face  $f$  of  $H$ , vertices  $u \in V(f)$ , and  $v, v' \in V$ , let  $H'_f$  be a copy of  $H$  with an artificial vertex  $f^*$  embedded in the interior of  $f$  along with an additional edge  $f^*u$ . Define the paths  $p_v = f^*u \circ u \rightsquigarrow_{H'_f} v$  and  $p_{v'} = f^*u \circ u \rightsquigarrow_{H'_f} v'$  and assume that neither path is a prefix of the other. By assumption and Lemma 3,  $p_v$  and  $p_{v'}$  share exactly one edge-maximal subpath  $f^* \rightsquigarrow x$ . We say that  $u \rightsquigarrow_H v$  makes a *left turn* w.r.t  $u \rightsquigarrow_H v'$  from  $f$  if  $x \rightsquigarrow v$  emanates from the left of  $p_v$ , and otherwise it makes a *right turn*; we will omit mention of  $f$  when the context is clear. Note that the notion of a turn is symmetric in the sense that  $u \rightsquigarrow_H v$  makes a left turn w.r.t  $u \rightsquigarrow_H v'$  iff  $u \rightsquigarrow_H v'$  makes a right turn w.r.t  $u \rightsquigarrow_H v$ .

## 2.2 Voronoi Diagrams

The definitions in this subsection will largely be made in a manner identical to those of [7], but are included as they are essential to a point location structure which will be presented in Section 4. Given a planar graph  $G = (V, E, \omega)$ ,  $S \subseteq V$ , the *Voronoi diagram of  $S$  in  $G$* , denoted by  $VD(S)$  in  $G$  is a partition of  $V$  into disjoint sets,  $Vor(u)$ , referred to as *Voronoi cells*, with one such set for each  $u \in S$ . The set  $Vor(u)$  is defined to be  $\{v \in V \mid d(u, v) < d(u', v) \text{ for all } u' \in S \setminus \{u\}\}$ , that is the set of vertices that are closer to  $u$  than any other site in terms of  $d(\cdot, \cdot)$ . We shall simply write  $VD$  when the context is clear.

It will also be useful to work with a dual representation of Voronoi diagrams. Let  $VD_0^*$  be the subgraph of  $G^*$  s.t.  $E(VD_0^*)$  is the subset of edges of  $G^*$  where  $uv^* \in VD_0^*$  iff  $u$  and  $v$  belong to different Voronoi cells in  $VD$ . Let  $VD_1^*$  be the graph obtained by repeatedly



contracting edges of  $VD_0^*$  incident to degree 2 vertices until no such vertex remains<sup>2</sup>. We refer to the vertices of  $VD_1^*$  as *Voronoi vertices*, and each face of the resulting graph  $VD_1^*$  can be thought of as corresponding to some Voronoi cell in the sense that its edges enclose exactly the vertices of some Voronoi cell in the embedding of the primal. We shall restrict ourself to the case in which all vertices of  $S$  lie on a single face  $h$ . In particular,  $h^*$  is a Voronoi vertex, since each site is a vertex on the boundary of  $h$  in the primal. Finally, let  $VD^*$  be the graph obtained by replacing  $h^*$  with multiple copies, one for each edge. We note that since there are  $|S|$  Voronoi sites (and thus faces in  $VD^*$ ), the number of Voronoi vertices in  $VD^*$  is  $O(|S|)$  due to Euler's formula. Furthermore, [7] show that when assuming unique shortest paths and a triangulated input graph,  $VD^*$  is a ternary tree. It follows that the primal face corresponding to a Voronoi vertex  $f^*$  consists of exactly three vertices, each belonging to different Voronoi cells. We refer to the number of sites in a Voronoi diagram as its *complexity*.

Finally, they also note that a *centroid decomposition*,  $T^*$ , can be computed from  $VD^*$  s.t. each node of  $T^*$  corresponds to a Voronoi vertex  $f^*$  and the children of  $f^*$  in  $T^*$  correspond to the subtrees resulting from splitting the tree at  $f^*$ , and s.t. the number of vertices of each child is at most a constant fraction of that of the parent. We remark that  $VD^*(S)$  can be computed by connecting all sites to a super-source and running a single-source shortest paths algorithm, and its centroid decomposition in time proportional to  $|V(VD^*(S))|$ .

### 2.3 Separators and decompositions

In the following, we will outline the graph decomposition framework used by our construction. As part of the preprocessing step, we will recursively partition the input graph using balanced fundamental cycle separators until the resulting graphs are of constant size. We shall associate with the recursive decomposition of  $G$  a binary decomposition tree,  $\mathcal{T}$ , which is a rooted tree whose nodes correspond to the regions of the recursive decomposition of  $G$ . We will refer to nodes and their corresponding regions interchangeably. The root node of  $\mathcal{T}$  corresponds to all of  $G$ . The following lemma states the invariants of the decomposition that will be used in our construction:

► **Lemma 4.** *Let  $G = (V, E, \omega)$  be an undirected, planar embedded, edge-weighted, triangulated graph and let  $T$  be a spanning tree<sup>3</sup> of  $G$ . Then there is an  $\tilde{O}(n)$  time algorithm that returns a binary decomposition tree  $\mathcal{T}$  of  $G$  s.t.*

1. *for any non-leaf node  $G' \in \mathcal{T}$ , its children  $G'_l$ , respectively  $G'_r$ , corresponds to the non-strict interior, respectively non-strict exterior of some fundamental cycle in  $G'$  w.r.t.  $T$ ,*
2. *for any child node, it contains at most a constant fraction of the faces of its parent,*
3. *for any leaf node it contains a constant number of faces of  $G$ ,*
4. *for all nodes at depth  $i$ ,  $\mathcal{T}_i$ ,  $\sum_{G' \in \mathcal{T}_i} |V(G')| = O(n)$*

Properties 1-3 follow from recursively applying a classic linear time algorithm for finding fundamental cycles. Property 4 follows from employing standard techniques that involve contracting degree-two vertices of the separators found at each level of recursion and weighting the resulting edges accordingly. This transformation results in a decomposition where the sum of faces of all regions at any level is preserved. We stress that our construction does not rely on the usual sparse simple cycle separators (of size  $O(\sqrt{n})$ ) but rather fundamental cycle separators of size  $O(n)$ .

<sup>2</sup> Formally, given a degree 2 vertex  $v$  with incident edges  $vw, vv'$ , we replace these edges by  $wv'$ , concatenate their arcs and embed  $wv'$  using this arc in the embedding.

<sup>3</sup> For our purposes, the spanning tree will be a shortest path tree.

### 3 The vertex labeled distance oracle

In this section we describe our reduction which shows our main result. The reduction can be described assuming Lemma 4 and the existence of the point location structure which we will state in the following lemma, the proof of which is deferred to Section 4:

► **Lemma 5.** *Let  $G = (V, E, \omega)$  be an undirected, planar embedded, edge-weighted graph with labeling  $l : V \rightarrow L$  and let  $p$  be a shortest path in  $G$ . There is a data structure  $O_{G,p}$  with  $O(|V| \log |V|)$  space which given  $u \in V$  and  $\lambda \in L$  returns a subset  $C \subset V$  of constant size, s.t. if  $v$  is the vertex with label  $\lambda$  closest to  $u$  and  $v \rightsquigarrow u$  intersects  $p$ , then  $v \in C$ . Each such query takes time at most  $O(\log^2 |V|)$ .*

#### 3.1 Preprocessing

Given the input graph  $G = (V, E, \omega)$ , the preprocessing phase initially computes the decomposition tree,  $\mathcal{T}$ , of Lemma 4. Associated with each non-leaf node  $G' \in \mathcal{T}$  is a fundamental cycle separator of  $ab \in E(G')$  w.r.t. the shortest path tree  $T$  rooted at some  $c \in V(G')$ . For such a  $G'$  we shall refer to  $S_1(G') = c \rightsquigarrow_{G'} a$  and  $S_2(G') = c \rightsquigarrow_{G'} b$ . Thus the fundamental cycle separator is given by  $S_1(G') \circ ab \circ S_2(G')$ . The preprocessing phase proceeds as follows: For all non-leaf nodes  $G' \in \mathcal{T}$ , compute and store data structures  $O_{G',S_1(G')}$  and  $O_{G',S_2(G')}$  of Lemma 5. Finally, a distance oracle  $D$  with  $O(s(|V|))$  space capable of reporting vertex-to-vertex shortest path distances in time  $O(t(|V|))$  is computed for  $G$  and stored alongside the decomposition tree and the point location structures.

#### Space complexity

The decomposition tree  $\mathcal{T}$  can be represented with  $O(|V| \log |V|)$  space and  $D$  with  $O(s(n))$  space. For each node  $G' \in \mathcal{T}$ , we store data structures  $S_1(G')$  and  $S_2(G')$ , so by Lemma 4 and 5, we get

$$\sum_{i=0}^{\infty} \sum_{G' \in \mathcal{T}_i} |V(G')| \log |V(G')| = \sum_{i=0}^{c \log n} O(|V| \log |V|) = O(|V| \log^2 |V|)$$

for a total space complexity of  $O(s(|V|) + |V| \log^2 |V|)$ .

#### 3.2 Query

Let  $G' \in \mathcal{T}$  and consider the query  $\mathbf{d}_{G'}(u, \lambda)$ . If  $G'$  is a leaf node, the query is resolved in time  $O(t(n))$  by querying  $D$  once for each vertex of  $G'$ . If  $G'$  is a non-leaf node, the query is handled as follows: First, data structures  $O_{G',S_1(G')}$  and  $O_{G',S_2(G')}$  are queried with  $u$  and  $\lambda$ , resulting in two “candidate sets”,  $C_1$  and  $C_2$ , one for each query. By Lemma 5,  $C_1 \cup C_2$  contains the nearest vertex with label  $\lambda$  for which  $u \rightsquigarrow_{G'} v'$  intersects either  $S_1$  or  $S_2$  if such a vertex exists. Compute  $d_{G'} = \min \{\mathbf{d}_G(u, c) \mid c \in C_1 \cup C_2\} \cup \{\infty\}$  by querying  $D$  once for each vertex of  $C_1 \cup C_2$ . The query then recursively resolves  $d_{G''} = \mathbf{d}_{G''}(u, \lambda)$  where  $G''$  is a child of  $G'$  in  $\mathcal{T}$  containing  $u$ . Finally, the query returns  $\min \{d_{G'}, d_{G''}\}$ .

#### Correctness

Denote by  $v$  the vertex of  $G$  with label  $\lambda$  nearest to  $u$  in  $G'$ , and consider the case in which  $u \rightsquigarrow_{G'} v$  intersects  $S_1(G')$  or  $S_2(G')$ . In this case,  $v \in C$  by Lemma 5 and  $C \neq \emptyset$ , so

$$d_{G'} = \min \{\mathbf{d}_G(u, c) \mid c \in C\} = \mathbf{d}_G(u, v) = \mathbf{d}_{G'}(u, v) = \mathbf{d}_{G'}(u, \lambda) \leq d_{G''}$$

■ **Algorithm 1** Query procedure for the distance oracle.

---

```

1: procedure QUERY( $u, \lambda, G'$ )
2:   if  $G'$  is a leaf node in  $\mathcal{T}$  then
3:     return  $\min \{d_{G'}(u, v) \mid v \in V(G') \text{ and } l(v) = \lambda\}$ 
4:   else
5:      $C_1 \leftarrow O_{G', S_1(G')}(u, \lambda)$ ;  $C_2 \leftarrow O_{G', S_2(G')}(u, \lambda)$ 
6:      $G'' \leftarrow$  A child of  $G'$  in  $\mathcal{T}$  containing  $u$ 
7:      $d_{G'} \leftarrow \min \{d_G(u, c) \mid c \in C_1 \cup C_2\} \cup \{\infty\}$ 
8:      $d_{G''} \leftarrow$  QUERY( $u, \lambda, G''$ )
9:     return  $\min \{d_{G'}, d_{G''}\}$ 

```

---

with the inequality following from definition of  $v$ . Note that in case  $u$  is a vertex of either  $S_1$  or  $S_2$ , the correct estimate is returned at the current level, but for a simpler description, the recursion proceeds anyways. Otherwise  $u \rightsquigarrow_G v$  intersects neither  $S_1$  and  $S_2$  in which case, the path must be fully contained in the (unique) child node,  $G''$ , of  $G'$  containing  $u$ . In this case, the query reports  $d_{G''} = \mathbf{d}_{G''}(u, \lambda) = \mathbf{d}_{G''}(u, v) \leq d_{G'}$ , showing the correctness.

### Time complexity

At each level of the recursion,  $O_{G', S_1(G')}$  and  $O_{G', S_2(G')}$  are queried in time  $O(\log^2 |V|)$ . Furthermore,  $D$  is queried  $|C| = O(1)$  times in total time  $O(1) \cdot O(t(|V|)) = O(t(|V|))$ . By Lemma 4, the query is recursively resolved on a problem instance which is a constant fraction smaller at each level of recursion, giving rise to the recurrence relation  $T(n) = T(n/a) + O(t(n) + \log^2 n)$ . When  $G'$  is a leaf node, then by Lemma 4,  $G'$  consists of a constant number of faces, described by the base case  $T(n) = O(t(n))$  when  $n \leq b$  for sufficiently small  $b$ . It is easily verified that a solution to the recurrence is bounded by  $O(\log^3 n + t(n) \log n)$ . This shows the main theorem, and the rest of this paper is devoted to proving Lemma 5.

## 4 The point location data structure

Our point-location data-structure uses techniques similar to those of [7] for point location in additively weighted Voronoi diagrams, but with some crucial differences in order to save space.

Both structures rely on being able to determine left/right turns of shortest paths in shortest path trees rooted at sites in  $G$ , but to facilitate this, the data structure of [7] explicitly stores an (augmented) shortest path tree rooted at each site as well as a data structure for answering least common ancestor (LCA) queries. The point location structure thus requires  $\Theta(|S|n)$  space where  $S$  is the number of sites, and since  $S$  may be large as  $\Theta(\sqrt{n})$  (corresponding to the size of a sparse balanced separator in a planar graph), this translates to  $\Theta(n^{3/2})$  space for their problem. This will not work in our case since the number of sites can in fact be as high as  $\Theta(n)$ , leading to a quadratic space bound.

The second issue with applying the techniques from [7] directly to our setting, is that it requires us to store a Voronoi diagram for each label, for each shortest path. Each vertex of the path separator would then be a site in the stored Voronoi diagram but as each separator may be large, i.e.  $\Theta(n)$ , we may use as much as  $\Theta(n|L|)$  space over *all* labels of  $L$  for a single separator. What we need is for the number of sites involved for a label  $\lambda$  to be proportional to the number of vertices with label  $\lambda$ ; this would give a near-linear bound on the number of sites when summing over all  $\lambda \in L$  across all levels of the recursive decomposition. We address these issues in the following sections.

## 4.1 MSSP structure

To compactly represent shortest path trees, our point location structure uses an augmented version of the multiple-source shortest-path (MSSP) data structure of Klein [10]. It cleverly uses the persistence techniques of [5] in conjunction with top trees [1] to obtain an implicit representation of shortest path trees rooted at each site. Top-trees allow for shortest path distance queries and least-common ancestor (LCA) queries in time  $O(\log n)$  per query while using  $O(n \log n)$  space, and can easily be augmented to support turn queries, as we shall see shortly. To be used as a black box, the MSSP structure relies on being initialized from a face of  $G$ . In our construction, we wish to use it for querying left/right turns of paths and distances from vertices residing on shortest paths of fundamental cycle separators, and thus some further preprocessing is required. The guarantees of the augmented MSSP structure used for the point location structure are summarized in the following lemma:

► **Lemma 6.** *Let  $G = (V, E, \omega)$  be an edge-weighted planar graph,  $f$  be a face of  $G$  and let  $T_u$  denote the shortest path tree rooted at  $u$ . Then there exists a data structure  $\text{MSSP}(G, f)$  with  $O(n \log n)$  space which given  $u \in V(f)$  and  $c, v \in V$  supports queries*

1.  $\text{DIST}(u, v)$ : report  $\mathbf{d}_G(u, v)$ ,
2.  $\text{LCA}(u, c, v)$ : report the least common ancestor of  $v, c$  in  $T_u$ ,
3.  $\text{TURN}(u, c, v)$ : report whether  $u \rightsquigarrow_{T_u} c$  makes a left or right turn w.r.t.  $u \rightsquigarrow_{T_u} v$  or if one is a prefix of the other.

*in time  $O(\log n)$  per query. The data structure can be preprocessed in  $O(n \log n)$  time.*

Descriptions of  $\text{DIST}$  and  $\text{LCA}$  are available in [10] and [1], and a description of how to implement  $\text{TURN}$  is provided in Appendix A in terms of the vocabulary and interface specified in [1] for completeness. A top-tree representing any shortest path tree rooted at a vertex on  $f$  can be accessed in time  $O(\log n)$  by using persistence in the MSSP structure. Lemma 6 then readily follows from applying the bound of Lemma 13 in Appendix A.

## 4.2 Label sequences

To address the second issue, we first need to make the following definition and state some of its properties when applied in the context of planar graphs:

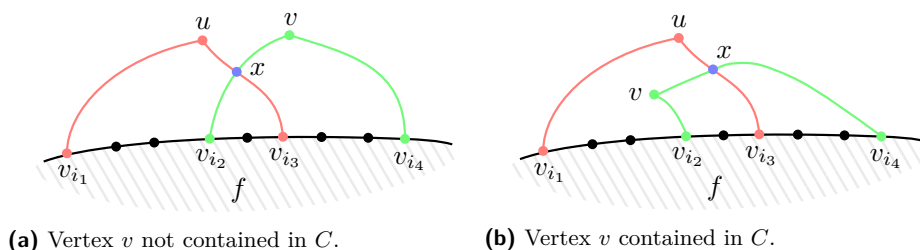
► **Definition 7.** *Let  $G = (V, E)$  be a graph,  $p = p_1, \dots, p_k$  a sequence of vertices and  $S \subseteq V$ . The label-sequence of  $p$  w.r.t.  $S$  is a sequence  $M_{G,S,p} \in S^k$  satisfying  $M_{G,S,p}(i) = \arg \min_{s \in S} \text{dist}_G(s, p_i)$ . The alternation number on  $p$  w.r.t.  $S$  in  $G$  is defined as  $|M_{G,S,p}| = \sum_{i=1}^{k-1} [M_{G,S,p}(i) \neq M_{G,S,p}(i+1)]$ .*

When  $G$ ,  $S$  and  $p$  are clear from the context, we shall simply write  $M$ , and also note that the sequence is well-defined due the tie-breaking scheme chosen in the preliminaries. The alternation number can be thought of as the number of times consecutive vertices on  $p$  change which vertex they are closest to among  $S$  when “moving along”  $p$ .

When  $G$  is an undirected planar graph and  $p$  is a shortest path in  $G$ , it can be observed<sup>4</sup> that  $M$  is essentially a Davenport-Schinzel sequence of order 2, and it immediately follows that the alternation number is “small” in the sense it is proportional to  $S$  while being agnostic towards the length of  $p$  altogether.

---

<sup>4</sup> We thank the anonymous reviewer for this observation which saved a tedious proof.



■ **Figure 1** Illustration of the proof of Lemma 10. The concatenation of  $u \rightsquigarrow v_{i_1}$ ,  $v_{i_1} \rightsquigarrow v_{i_3}$ , and the reverse of  $u \rightsquigarrow v_{i_3}$  forms a cycle  $C$  and  $v \rightsquigarrow v_{i_2}$  intersects  $u \rightsquigarrow v_{i_3}$  in  $x$ . Note that w.l.o.g.  $C$  is not necessarily simple, that  $v \rightsquigarrow v_{i_2}$  may intersect the cycle more than once and that  $v$  may be a vertex of  $C$ .

► **Definition 8** (Davenport-Schinzel [4]). A sequence  $u_1, u_2, \dots, u_k$  over an alphabet  $\Sigma$  on  $n$  symbols is said to be a  $(n, s)$ -Davenport-Schinzel sequence if

1.  $u_i \neq u_{i+1}$  for all  $1 \leq i < k$  and
2. There do not exist  $s + 2$  indices  $1 \leq i_1 < i_2 < \dots < i_{s+2} \leq k$  for which  $u_{i_1} = u_{i_3} = \dots = u_{i_{s+1}} = u$  and  $u_{i_2} = u_{i_4} = \dots = u_{i_{s+2}} = v$  for  $u \neq v \in \Sigma$ .

► **Lemma 9** (Davenport-Schinzel [4]). Let  $U$  be a  $(n, 2)$ -Davenport-Schinzel sequence of length  $m$ . Then  $|U| \leq 2n - 1$ .

For a sequence of  $\mathcal{S} = u_1, u_2, \dots, u_k$  over an alphabet  $\Sigma$ , the *contraction* of  $\mathcal{S}$  is the subsequence obtained from  $\mathcal{S}$  by replacing every maximal substring  $s, s, \dots, s$  of  $\mathcal{S}$  consisting of identical symbols  $s$  by a single occurrence of  $s$ . As an example with  $\Sigma = \{0, 1, 2\}$ , the contraction of  $0, 0, 1, 2, 2, 1, 1, 0, 1, 0, 0, 2$  is  $0, 1, 2, 1, 0, 1, 0, 2$ .

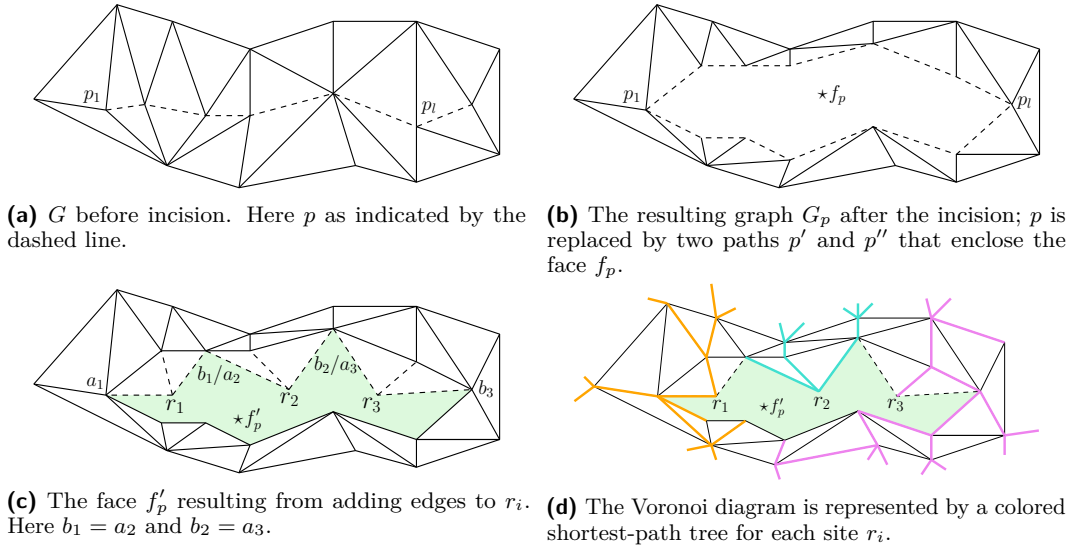
► **Lemma 10.** Let  $G$  be an undirected, weighted planar graph, let  $S \subseteq V$ , and let  $p$  be a shortest path of  $G$  contained in (the boundary of) a face of  $G$ . Then the contraction of  $M$  is a  $(|S|, 2)$ -Davenport-Schinzel sequence.

**Proof.** Define  $v_1, \dots, v_k = p$  and assume for the sake of contradiction that for some  $1 \leq i_1 < i_2 < i_3 < i_4 \leq k$  and  $u, v \in S$  with  $u \neq v$ , it holds that  $u = M(i_1) = M(i_3)$  and  $v = M(i_2) = M(i_4)$ . Then the concatenation of  $u \rightsquigarrow v_{i_1}$ ,  $v_{i_1} \rightsquigarrow v_{i_3}$ , and the reverse of  $u \rightsquigarrow v_{i_3}$  forms a cycle. Thus, either  $v \rightsquigarrow v_{i_2}$  intersects  $u \rightsquigarrow v_{i_1} \cup u \rightsquigarrow v_{i_3}$  or  $v \rightsquigarrow v_{i_4}$  intersects  $u \rightsquigarrow v_{i_1} \cup u \rightsquigarrow v_{i_3}$ ; see Figure 1a and 1b. By symmetry, we only need to consider the former case. If  $v \rightsquigarrow v_{i_2}$  intersects  $u \rightsquigarrow v_{i_3}$  in some vertex  $x$  then  $v \rightsquigarrow x$  has the same weight as  $u \rightsquigarrow x$ . By the “closer than”-relation,  $u = M(i_3) = M(i_2) = v$ , contradicting our assumption that  $u \neq v$ . A similar contradiction is obtained if  $v \rightsquigarrow v_{i_2}$  intersects  $u \rightsquigarrow v_{i_1}$ . ◀

► **Corollary 11.** Let  $G, S$  and  $p$  be as in Lemma 10. Then  $|M_{G,S,p}| = O(|S|)$ .

We remark that  $M$  can be readily computed in polynomial time by adding a super-source connected to each vertex of  $S$  and running an SSSP algorithm. Furthermore  $M$  can be represented with  $O(S)$  space, by storing only the indices for which  $M(i) \neq M(i + 1)$  and  $M(i)$  for each such index.

We will now describe how to achieve  $O(n)$  space for storing Voronoi diagrams for all labels  $\lambda \in L$  at any level of the recursive decomposition. We do so by modifying the preprocessing steps and query scheme of [7] in a manner suitable for application of Lemma 10 and Corollary 11.



■ **Figure 2** Preprocessing steps for the point-location structure.

### 4.3 Preprocessing

Let us briefly recall the statement of Lemma 5; that is, we let  $G$  be an undirected, edge-weighted, planar embedded graph with associated labeling  $l : V \rightarrow L$  and let  $p = p_1 \rightsquigarrow p_k$  be a shortest path in  $G$ . Given a query  $(u, \lambda) \in V \times L$ , we want to identify a small “candidate” set of vertices  $C \subseteq V$  such that if  $v$  is the vertex with label  $\lambda$  closest to  $u$  and  $u \rightsquigarrow v$  intersects  $p$ , then  $v \in C$ .

Here, we first describe how to compute a data structure which provides the guarantees of Lemma 5, but restricts itself to the case only where  $u \rightsquigarrow v$  intersects  $p$  from the left. The description of the data structure for handling paths that intersect  $p$  from the right is symmetric (e.g. by swapping the endpoints of  $p$ ). Lemma 5 thus readily follows from the existence of such structures.

First, a copy,  $G_p$ , of  $G$  is stored and an incision is added along  $p$  in  $G_p$ . This results in a planar embedded graph  $G_p$ , which has exactly one more face than  $G$ . Define by  $p' = p'_1, \dots, p'_l$  and  $p'' = p''_1, \dots, p''_l$  the resulting paths along the incision, where  $p'_1 = p'_l$  and  $p''_1 = p''_l$ . We denote by  $f_p$  the face whose boundary vertices are  $V(p') \cup V(p'')$ . An illustration of this is provided in Figure 2a and 2b.

Next, the MSSP data structure of Lemma 6,  $MSSP(G_p, f_p)$ , is computed and stored as part of the point-location data structure. This structure will be used for the point location query.

#### Centroid decompositions of Voronoi diagrams

The following preprocessing is now done for *each* label  $\lambda \in L$ : First a copy,  $G_p^\lambda$ , of  $G_p$  is made. Next,  $M_{G_p, S_\lambda, p'}$  is computed for  $S_\lambda = \{v \in V \mid l(v) = \lambda\}$ . For convenience we assume that  $M(0) = \text{NIL}$ . The preprocessing phase now consists of modifying  $G_p^\lambda$  before computing the Voronoi diagram and the associated centroid decomposition associated with  $\lambda$  as follows: For  $i \leftarrow 1, \dots, l$ , whenever  $M(i) \neq M(i-1)$ , a new vertex is added to  $G_p^\lambda$  and embedded in  $f_p$  along the curve formed by the deleted arc of the embedding of  $p$ . Denoting by  $r_i$  the most recently added vertex after iteration  $i$ , edge  $p'_i r_i$  with  $\omega(p'_i r_i) = d_G(M(i), p'_i)$  is added to  $G_p^\lambda$

and embedded for all  $i$ . Once again, it is fairly easy to see that  $G_p^\lambda$  is planar embedded. See Figure 2c for an illustration of this. Denote by  $a_i$  and  $b_i$  the endpoints of the first and last edges added incident to  $r_i$  (w.r.t. the order in which they were added). Denote by  $f'_p$  that has  $\{r_i, a_i, b_i \mid 1 \leq i \leq l\} \cup V(p'')$  as its boundary vertices. Now the Voronoi diagram, its dual and subsequently its corresponding centroid decomposition,  $T_{p,\lambda}^*$ , is computed in (the now modified)  $G_p^\lambda$  using  $R = \{r_i \mid 1 \leq i \leq l\}$  as Voronoi sites, see Figure 2d. The intuition is that each site in  $R$  corresponds to a contiguous subsequence  $M(k), \dots, M(l)$  of  $M$  for which  $M(j) = M(j+1) = v$  where  $v$  is the vertex with label  $\lambda$  closest to  $p'_j$  for  $k \leq j < l$ . This implies that the number of sites is proportional to the number of vertices with label  $\lambda$  instead of the length of the original separator  $p$ : By Lemma 11,  $|M| = O(|S_\lambda|)$  and since  $|R| = |M|$  it follows that  $|R| = O(|S_\lambda|)$  bounds the complexity of  $T_{p,\lambda}^*$ , which is stored as part of the data structure. As aforementioned, each centroid  $c \in T_{p,\lambda}^*$  corresponds to some degree three Voronoi vertex,  $f_c^*$ , with vertices,  $\{x_1, x_2, x_3\}$  in the corresponding primal face  $f_c$  s.t. each  $x_j$  belongs to a different Voronoi site  $r_{i_j}$  for  $j \in \{1, 2, 3\}$ . For each such  $j$ , the centroid  $c$  stores a pointer to its corresponding face  $f_c$ , the first vertex  $p_{k_j}$  of  $p'$  on  $r_{i_j} \rightsquigarrow_{G_p'} x_j$  and the weight  $\omega(p_{k_j} r_{i_j})$ .

### Space complexity

The space used for storing the MSSP structure is  $O(|V| \log |V|)$  by Lemma 6. For each centroid, we store a constant amount of data, so the space required for storing the centroid decompositions is

$$\sum_{\lambda \in L} O(1) \cdot |T_{p',\lambda}^*| = \sum_{\lambda \in L} O(|S_\lambda|) = O(V)$$

since  $S_\lambda \cap S_{\lambda'} = \emptyset$  for  $\lambda \neq \lambda' \in L$  as each vertex has exactly one label. The total space used is thus  $O(|V| \log |V|)$ .

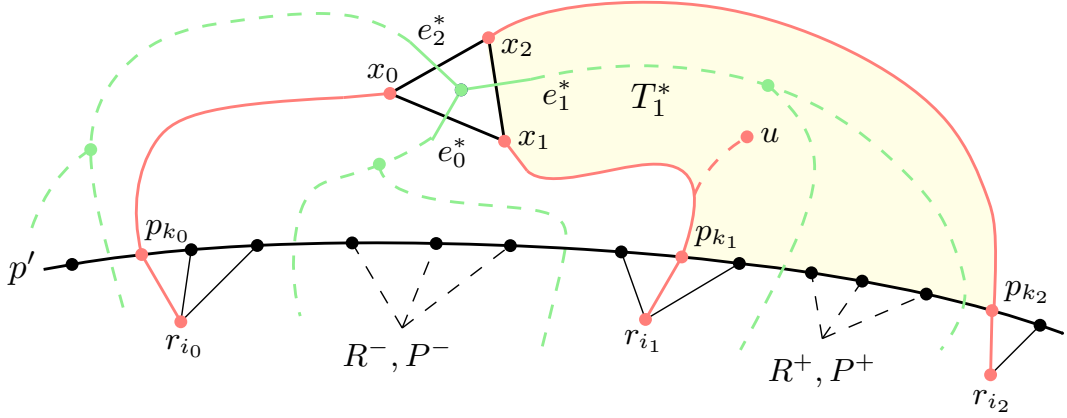
## 4.4 Handling a point location query

We now show how to handle a point location query. Note that in the following, we can assume that the vertices of  $p'$  appear in increasing order of their indices when traversing the boundary of  $f_p$  in a clockwise direction. Recall that given  $u \in V$  and  $\lambda \in L$ , we wish to find a subset  $C \subset V$  of constant size, s.t. if  $v$  is the closest vertex with label  $\lambda$  where  $u \rightsquigarrow v$  intersects  $p$  from the left, then  $v \in C$ . The query works by identifying a subset  $P \subseteq V(p')$  s.t. for some  $p'_k \in P$  it holds that  $M_{G_p, S_\lambda, p'}(k) = v$ . We first show how to identify the subset by recursively querying the centroid decomposition  $T_{p,\lambda}^*$  according to the following lemma, which we note is modified version of the query in [7]:

► **Lemma 12.** *Given a query  $(u, \lambda) \in V \times L$ , consider the centroid decomposition tree  $T_{p,\lambda}^*$  computed from  $G_p^\lambda$  in the preprocessing. Let  $c$  be a centroid  $c \in T_{p,\lambda}^*$  corresponding to some Voronoi vertex,  $f_c^*$ , with associated primal triangle containing vertices  $\{x_0, x_1, x_2\} = V(f_c)$  where  $x_j$  belongs to the Voronoi cell of  $r_{i_j}$  for  $j \in \{0, 1, 2\}$  and  $i_0 < i_1 < i_2$ . Furthermore let  $e_j^*$  be the dual edge incident to  $f_c^*$ , s.t.  $e_j = x_j x_{(j+1) \bmod 3}$ , let  $p_{k_j}$  be the successor of  $r_{i_j}$  on  $r_{i_j} \rightsquigarrow_{G_p^\lambda} x_j$ , let  $P_j = p_{k_j} \rightsquigarrow_{G_p} u$ , and let  $T_j^*$  be the subtree of  $T_{p,\lambda}^*$  attached to  $c$  by  $e_j^*$  for  $j \in \{0, 1, 2\}$ . Finally, let  $j^* = \arg \min_{j \in \{0, 1, 2\}} \{\mathbf{d}_{G_p}(p_{k_j}, u) + \omega(r_{i_j} p_{k_j})\} = \arg \min_{j \in \{0, 1, 2\}} \{\mathbf{d}_{G_p^\lambda}(r_{i_j}, u)\}$ . Then*

1. *If  $p_{k_{j^*}} \rightsquigarrow_{G_p} u$  emanates from the left of  $P_{j^*}$  or  $u \in P_{j^*}$ , then the site closest to  $u$  in  $G_p^\lambda$  belongs to  $R^- = \{r_{(i_{(j^*-1) \bmod 3}}, \dots, r_{i_{j^*}})\}$  and the second vertex on the shortest path from that site to  $u$  in  $G_p^\lambda$  belongs to  $P^- = \{p_{(i_{(j^*-1) \bmod 3}}, \dots, p_{i_{j^*}})\}$ ; furthermore,  $T_{j^*}^*$  is the centroid decomposition tree for  $G_p^\lambda$  when restricted to shortest paths from sites in  $R^-$  through successors in  $P^-$ .*





■ **Figure 3** Illustration of Lemma 12. The dashed green lines represent Voronoi edges in the centroid decomposition and the red lines the primal shortest paths to the primal vertices of the centroid. In this case,  $j^* = 1$ , so  $u$  is contained in primal faces spanned by the subtree  $T_1^*$  contained in the region highlighted in yellow.

2. otherwise, the site closest to  $u$  in  $G_p^\lambda$  belongs to  $R^+ = \{r_{i_{j^*}}, \dots, r_{i_{(j^*+1) \bmod 3}}\}$  and the second vertex on the shortest path from that site to  $u$  belongs to  $P^+ = \{p_{i_{j^*}}, \dots, p_{i_{(j^*+1) \bmod 3}}\}$ ; furthermore,  $T_{(j^*+1) \bmod 3}^*$  is the centroid decomposition tree for  $G_p^\lambda$  when restricted to shortest paths from sites in  $R^+$  through successors in  $P^+$ .

**Proof.** By symmetry, we only consider the first case since the second case occurs when  $p_{k_{j^*}} \rightsquigarrow_{G_p} u$  emanates from the right of  $P_{j^*}$  and the first case also includes the case where  $u \in P_{j^*}$ .

By the choice of  $j^*$ ,  $p_{k_{j^*}} \rightsquigarrow_{G_p} u$  cannot intersect any of the paths  $P_{j'}$  with  $j' \in \{(j^* - 1) \bmod 3, (j^* + 1) \bmod 3\}$  since these two paths are subpaths of shortest paths from sites  $r_{i_{j'}} \neq r_{i_{j^*}}$  in  $G_p^\lambda$  and we assume unique shortest paths. Let  $x$  be the first vertex of  $p_{k_{j^*}} \rightsquigarrow_{G_p} u$  such that either  $x = u$  or the path emanates from the left of  $P_{j^*}$  at  $x$ . The rest of  $p_{k_{j^*}} \rightsquigarrow_{G_p} u$  following  $x$  will not intersect  $P_{j^*}$  again due to uniqueness of shortest paths. Thus  $u$  belongs to the region of the plane enclosed by paths  $P_{(j^*-1) \bmod 3}$ ,  $P_{j^*}$ , edge  $e_{(j^*-1) \bmod 3}$ , and path  $p_{k_{(j^*-1) \bmod 3}}, \dots, p_{k_{j^*}}$ . Note that  $T_{j^*}^*$  is the subtree of  $T_{p', \lambda}^*$  spanning the primal faces contained in this region. Hence,  $T_{j^*}^*$  is the centroid decomposition tree for  $G_p^\lambda$  when restricted to shortest paths from sites in  $R^-$  through successors in  $P^-$ . ◀

For an illustration of Lemma 12, see Figure 3. The Lemma implies a fast recursive point location scheme. On query  $(u, \lambda)$ , obtain centroid  $c$  from  $T_{p', \lambda}^*$ . Since weights of edges from sites have been precomputed,  $MSSP(G_p, f_p)$  is applied to find  $j^*$ .  $MSSP(G_p, f_p)$  is also used to determine if  $p_{k_{j^*}} \rightsquigarrow_{G_p} u$  emanates from the left of  $P_{j^*}$  and hence whether the first or second case of the lemma applies. The point location structure now recurses on a subset of sites and vertices of  $p'$  and on a subtree of  $T_{p', \lambda}^*$ , depending on which case applies.

The recursion stops when reaching a subtree corresponding to a bisector for two sites. The vertices of  $V$  with label  $\lambda$  corresponding to these two sites are reported as the set  $C$ , yielding the desired bound.

### Time complexity

The  $O(\log^2 |V|)$  query time bound of Lemma 5 follows since there are  $O(\log |V|)$  recursion levels and in each step, a constant number of queries to  $MSSP(G_p, f_p)$  are executed, each taking  $O(\log |V|)$  time.

## References

- 1 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. doi:10.1145/1103963.1103966.
- 2 Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 138–151, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316316.
- 3 Shiri Chechik. Improved distance oracles and spanners for vertex-labeled graphs. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, pages 325–336, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- 4 H. Davenport and A. Schinzel. A Combinatorial Problem Connected with Differential Equations. *American Journal of Mathematics*, 87(3):684, July 1965. doi:10.2307/2373068.
- 5 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.
- 6 Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. Holiest minimum-cost paths and flows in surface graphs. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1319–1332. ACM, 2018. doi:10.1145/3188745.3188904.
- 7 Paweł Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2018, pages 515–529, USA, 2018. Society for Industrial and Applied Mathematics.
- 8 Paweł Gawrychowski, Gad M. Landau, Shay Mozes, and Oren Weimann. The nearest colored node in a tree. *Theoretical Computer Science*, 710:66–73, 2018. doi:10.1016/j.tcs.2017.08.021.
- 9 Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. Distance oracles for vertex-labeled graphs. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, pages 490–501, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 10 Philip Klein. Multiple-source shortest paths in planar graphs. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 146–155, January 2005.
- 11 Mingfei Li, Chu Chung Christopher Ma, and Li Ning.  $(1 + \epsilon)$ -distance oracles for vertex-labeled planar graphs. In T.-H. Hubert Chan, Lap Chi Lau, and Luca Trevisan, editors, *Theory and Applications of Models of Computation*, pages 42–51, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- 12 Yaowei Long and Seth Pettie. Planar Distance Oracles with Better Time-Space Tradeoffs. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2517–2537. Society for Industrial and Applied Mathematics, Philadelphia, PA, January 2021. doi:10.1137/1.9781611976465.149.
- 13 Shay Mozes and Eyal E. Skop. Efficient Vertex-Label Distance Oracles for Planar Graphs. *Theory of Computing Systems*, 62(2):419–440, February 2018. doi:10.1007/s00224-017-9827-0.
- 14 Maximilian Probst. On the Complexity of the (Approximate) Nearest Colored Node Problem. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.68.
- 15 Dekel Tsur. Succinct data structures for nearest colored node in a tree. *Information Processing Letters*, 132:6–10, 2018. doi:10.1016/j.ipl.2017.10.001.

## A

 Appendix

A description of how to implement TURN in Lemma 6 is provided here in terms of the terminology and the interface specified in [1]. Readers that are not familiar with the terminology and interface pertaining to top-trees are referred to [1].

► **Lemma 13.** *Let  $G = (V, E, \omega)$  be a weighted planar graph, and let  $\mathcal{T}$  be the root-cluster of a top-tree corresponding to some tree  $T$  in  $G$ . Then in addition to DIST and LCA, we can support TURN queries: For  $u, c, v \in V$ , report whether  $u \rightsquigarrow c$  makes a left or right turn w.r.t.  $u \rightsquigarrow_T v$ , or if one is a prefix of the other in time  $O(\log n)$  per query.*

**Proof.** A description of how to perform LCA queries is found in [1]. Assume that  $u, c, v \in T$  and w.l.o.g. that  $u$  is strictly more rootward in  $T$  than  $v$ . First use  $\mathcal{T}$  to determine the LCA  $c'$  of  $(v, c)$ . If  $c'$  is on both  $u \rightsquigarrow c$  and  $u \rightsquigarrow v$  one path is a prefix of the other. Otherwise invoke EXPOSE( $u, c'$ ) and traverse  $\mathcal{T}$  until a leaf of  $\mathcal{T}$  corresponding to the edge,  $e_v \in E(T)$  is reached, which connects  $c'$  to the subtree containing  $v$  in  $T$ . This can be done in time  $O(\log n)$ . The same is done for  $(u, c')$  and  $(c, c')$ , exposing edges  $e_u, e_c \in T$ . Now, if  $e_c = e_v$  or  $e_c = e_u$ ,  $c$  must be on  $u \rightsquigarrow_T v$ . Otherwise it is easily checked, by maintaining a cyclic order of edges in the adjacency list of  $c'$ , in constant time, whether  $e_c$  emanates to the left or right of the subpath  $e_u e_v$  and hence  $u \rightsquigarrow_T v$ . The total time spent is  $O(\log n)$ . ◀

# A Characterization of Individualization-Refinement Trees

**Markus Anders**

TU Darmstadt, Germany

**Jendrik Brachter**

TU Darmstadt, Germany

**Pascal Schweitzer**

TU Darmstadt, Germany

---

## Abstract

Individualization-Refinement (IR) algorithms form the standard method and currently the only practical method for symmetry computations of graphs and combinatorial objects in general. Through backtracking, on each graph an IR-algorithm implicitly creates an IR-tree whose order is the determining factor of the running time of the algorithm.

We give a precise and constructive characterization which trees are IR-trees. This characterization is applicable both when the tree is regarded as an uncolored object but also when regarded as a colored object where vertex colors stem from a node invariant. We also provide a construction that given a tree produces a corresponding graph whenever possible. This provides a constructive proof that our necessary conditions are also sufficient for the characterization.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Mathematics of computing → Trees

**Keywords and phrases** individualization refinement algorithms, backtracking trees, graph isomorphism

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.24

**Related Version** *Full Version:* <https://arxiv.org/abs/2109.07302> [2]

**Funding** Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (EngageS: grant No. 820148).

## 1 Introduction

The individualization-refinement (IR) framework is a general backtracking technique employed by algorithms solving tasks related to the computation of symmetries of combinatorial objects [16]. These include algorithms computing automorphism groups, isomorphism solvers, canonical labeling tools used for computing normal forms, and to some extent recently also machine learning computations in convolutional neural networks [1,17]. In fact all competitive graph isomorphism/automorphism solvers, specifically NAUTY/TRACES [15,16], BLISS [11,12], SAUCY [8,9], CONAUTO [13,14], and DEJAVU [3,4] fall within the framework. These tools alternate color-refinement techniques (such as the 1-dimensional Weisfeiler-Leman algorithm) with backtracking steps. The latter perform artificial individualization of indistinguishable vertices. This leads to recursive branching and overall to a tree of recursive function calls, the so called IR-tree.

Using clever invariants and heuristics, the tools manage to prune large parts of the IR-tree. Since the non-recursive work is quasi-linear, it has long been known that the number of traversed nodes of the IR-tree is the determining factor in the running time for all the tools (see for example [20, Theorem 9] and [19]). And in fact, the running times of the various tools closely reflect this [4,16]. Indeed, variation in the traversal strategies among the



© Markus Anders, Jendrik Brachter, and Pascal Schweitzer;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 24; pp. 24:1–24:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

tools leads to a different number of traversed nodes which in turn leads to different running times. However, explicit bounds that rigorously show asymptotic advantages of randomized traversals over deterministic ones have only recently been obtained [5]. For this, a specific problem – a search problem in trees with symmetries – is defined. It captures precisely the parameters within which IR-algorithms operate.

While these results are quite general within an abstract model, the bounds proven in [5] apply to the search problem in arbitrary trees with symmetries, independent of whether they originate from actual IR-computations or not. Granted, the vast benchmark library of TRACES [15,16] shows that IR-trees come in an abundance of forms and shapes. However, to date there have been no comprehensive results actually analyzing which trees can arise as an IR-tree.

**Contribution.** In this paper we study which trees are IR-trees. Arising from a branching process, all IR-trees are rooted and all inner vertices have at least 2 children. Such trees are called irreducible (or series reduced). Despite a vast variety of IR-trees arising from benchmark libraries, it turns out that not all irreducible trees are IR-trees. However, we can give a full, constructive characterization of IR-trees.

► **Theorem 1.** *An irreducible tree is an IR-tree if and only if there is no node that has exactly two children of which exactly one is a leaf.*

To prove the theorem we first provide and justify necessary conditions for a tree to be an IR-tree. We then prove that, indeed, these conditions are sufficient by providing graphs on which the execution of an IR-algorithm yields the desired tree. In fact, our proof is constructive, meaning that we obtain an algorithm with the following property. Given a tree  $T$  satisfying the necessary conditions, the algorithm produces a graph whose IR-tree is  $T$ .

As we describe in our definition of IR-trees in Section 2, the trees are naturally associated with a coloring of the vertices. This coloring is a crucial component that is related to the automorphism group structure of the graph. Our characterization also fully describes how color classes may be distributed in a given tree. It turns out that there are several simple restrictions, in particular for vertices that have precisely two children, but apart from that all colorings can be realized and in particular any number of symmetries can be ensured (see Section 4).

Our characterization provides a fundamental argument transferring the analysis of abstract tree traversal strategies performed in [5] to backtracking trees of IR-algorithms on actual instances. Specifically, we may conclude that the abstract trees used for the lower bounds of probabilistic algorithms in [5] indeed appear as IR-trees. However, interestingly, the abstract trees used for the lower bounds of deterministic algorithms (Theorem 13, [5]) are not IR-trees. In fact these trees have nodes with two children, one child that is a leaf and another that is not. This breaks the necessary conditions as laid out by Theorem 1. Fortunately, it also immediately follows from our results that a slight modification can rectify this: by simply replacing the respective leaves with inner nodes that have two attached leaves, the trees become actual IR-trees, due to our characterization. Overall, we therefore prove that the lower bounds of [5] hold true in the IR-paradigm.

**Cell Selectors and Invariants.** Formally, the IR-paradigm allows for different design choices in some of its components. For most of these, competitive practical solvers actually make very similar choices: the refinement is always *color refinement* and solvers commonly choose

as their pruning invariant (essentially) the so-called *quotient graph*. The way in which the actual implementations differ from color refinement and quotient graphs is usually only in minor details and done to achieve practical speed-ups. This only leads to a slightly weaker refinement and invariants in some specific cases. In this paper, we therefore comply with these common design choices.

Many other design choices, such as *how* IR-trees are traversed, have no effect on the characterization of the IR-trees themselves.

There is however one integral design choice where competitive IR-solvers do indeed vary in a way that affects which trees are IR-trees, namely the so-called *cell selectors*. We should emphasize that Theorem 1 only says that for the trees satisfying the necessary conditions there is some cell selector for which the graph is an IR-tree.

However, we can also say something about specific cell selectors. Considering the characterization for a *given* cell selector, there are two possibilities: either, fewer trees turn out to be IR-trees or the same characterization applies. We can use our results to argue that for some cell selectors that are used in practice our necessary conditions are sufficient, while for others they are not (see Section 5 for a discussion).

**Techniques.** Many properties of a graph, e.g. symmetries, are directly tied to properties of its IR-tree. When modeling a graph that is supposed to produce a particular IR-tree, two major difficulties arise, roughly summarized as follows:

1. The effect of color refinement on the graph needs to be kept under control.
2. The shape of the IR-tree may dictate that symmetries must be simultaneously represented in distinct parts of the graph.

We resolve these issues using various gadget constructions specifically crafted for this purpose. We introduce *concealed edges*, which allow us to precisely control the point in time at which the IR-process is able to see a certain set of edges and thus color refinement to take effect (resolving issue (1)). By combining concealed edges with gadgets enforcing particular regular abelian automorphism groups we can synchronize symmetries across multiple branches of the tree (resolving issue (2)).

Here, as the main tool we show the following. As an additional restriction, which stems from the structure of IR-trees, we consider only trees where all leaves can be mapped to the same number of other leaves via symmetries (i.e., under automorphisms all *leaf orbits* have the same size). We show that each such tree  $T$  can be embedded into a graph  $H_T$ , such that  $H_T$  restricts the symmetries of  $T$  in a particular way. Intuitively, we keep just enough symmetries to allow leaves to be mapped to each other whenever this is possible in  $T$ . We thereby effectively couple leaf orbits so that when fixing one leaf, all other leaves are fixed as well. More formally we prove the following theorem.

► **Theorem 2.** *Let  $T$  be a colored tree in which all leaf orbits have the same size. There exists a graph  $H_T$  containing  $T$  as an automorphism invariant induced subgraph so that the action of  $\text{Aut}(H_T)$  is faithful on  $T$  and semiregular on the set of leaves of  $T$ . Moreover,  $\text{Aut}(H_T)$  induces the same orbits on  $T$  as  $\text{Aut}(T)$ .*

Again, we prove the theorem in a constructive manner. All steps can be easily converted into an algorithm that takes as input an admissible (i.e., compatible with our necessary conditions from Section 3) colored tree  $T$  and produces a graph and cell selector with IR-tree  $T$ .

## 2 Individualization-Refinement Trees

Following [16] closely, we introduce the notion of an IR-tree. Algorithms based on the IR-paradigm explore these trees using various traversal strategies to solve graph isomorphism, graph automorphism or canonical labeling problems.

**Colored Graphs.** An undirected, finite graph  $G = (V, E)$  consists of a set of vertices  $V \subseteq \mathbb{N}$  and a set of edges  $E \subseteq V^2$ , where  $E$  is symmetric. Set  $n := |V|$ .

The IR framework relies on coloring the vertices of a graph. A coloring is a surjective map  $\pi: V \rightarrow \{1, \dots, k\}$ . The  $i$ -th *cell* for  $i \in \{1, \dots, k\}$  is  $\pi^{-1}(i) \subseteq V$ . Elements in the same cell are *indistinguishable*. If  $|\pi| = n$ , i.e., whenever each vertex has its own distinct color in  $\pi$ , then  $\pi$  is called *discrete*. A coloring  $\pi$  is *finer* than  $\pi'$  (and  $\pi'$  *coarser* than  $\pi$ ) if  $\pi(v) = \pi(v')$  implies  $\pi'(v) = \pi'(v')$  for all  $v, v' \in V$ . Whenever convenient, we may also view colorings as ordered partitions instead of maps. A colored graph  $(G, \pi)$  consists of a graph and a coloring.

The symmetric group on  $\{1, \dots, n\}$  is denoted  $\text{Sym}(n)$ . An automorphism of a graph  $G$  is a bijective map  $\varphi: V \rightarrow V$  with  $G^\varphi := (\varphi(V), \varphi(E)) = (V, E) = G$ . With  $\text{Aut}(G)$  we denote the automorphism group of  $G$ . For a colored graph  $(G, \pi)$  we require automorphisms to also preserve colors, i.e.,  $\pi(v) = \pi(\varphi(v))$  for all  $v \in V$ . We define the colored automorphism group  $\text{Aut}(G, \pi)$  accordingly.

**Color Refinement and Individualization.** IR-algorithms use a procedure to heuristically refine colorings based on the degree of vertices. The intuition is that if two vertices have different degree, then they can not be mapped to each other by an automorphism. We assign vertices of different degrees distinct colors to indicate this phenomenon. This process is iterated using color degrees: for example, two vertices can only be mapped to each other if they have the same number of neighbors of a particular color  $i$ . Therefore vertices can be distinguished according to the number of neighbors they have in color  $i$ . This gives us a new, refined coloring that (potentially) distinguishes more vertices. This is repeated until the process stabilizes.

The colorings resulting from this process are called equitable colorings. A coloring  $\pi$  is *equitable* if for every pair of (not necessarily distinct) colors  $i, j \in \{1, \dots, k\}$  the number of  $j$ -colored neighbors is the same for all  $i$ -colored vertices. For a colored graph  $(G, \pi)$  there is (up to renaming of colors) a unique coarsest equitable coloring finer than  $\pi$  [16]. We denote this coloring by  $\text{Ref}(G, \pi, \epsilon)$ , where  $\epsilon$  is the empty sequence.

IR-algorithms also use *individualization*. This process artificially forces a vertex into its own cell. We can record which vertices have been individualized in a sequence  $\nu \in V^*$ . We extend the refinement function so that  $\text{Ref}(G, \pi, \nu)$  is the unique coarsest equitable coloring finer than  $\pi$  in which every vertex in  $\nu$  is a singleton with its own artificial color. Specifically, the artificial colors used to individualize  $\nu$  are not interchangeable with colors introduced by the refinement itself and are ordered: the  $i$ -th vertex in  $\nu$  is always colored using the  $i$ -th artificial color.

We require this coloring to be isomorphism invariant (which means that  $\text{Ref}(G, \pi, \nu)(v) = \text{Ref}(G^\varphi, \pi^\varphi, \nu^\varphi)(v^\varphi)$  for  $\varphi \in \text{Sym}(n)$ ). There are efficient *color refinement* algorithms to compute  $\text{Ref}(G, \pi, \nu)$ , for which we refer to [16].

We say two colored graphs  $(G_1, \pi_1)$  and  $(G_2, \pi_2)$  are *distinguishable (by color refinement)*, if with respect to the colorings  $\text{Ref}(G_1, \pi_1, \epsilon)$  and  $\text{Ref}(G_2, \pi_2, \epsilon)$



1. there is a color  $c$  with differently sized cells in  $G_1$  and  $G_2$  (i.e.,  $|\text{Ref}(G_1, \pi_1, \epsilon)^{-1}(c)| \neq |\text{Ref}(G_2, \pi_2, \epsilon)^{-1}(c)|$ ),
2. or there are vertices  $v_1 \in V(G_1)$ ,  $v_2 \in V(G_2)$  of the same color (i.e.,  $\text{Ref}(G_1, \pi_1, \epsilon)(v_1) = \text{Ref}(G_2, \pi_2, \epsilon)(v_2)$ ), such that there is a color  $c$  within which  $v_1$  and  $v_2$  have a differing number of neighbors (i.e.,  $|\{(v_1, w) \in E(G_1) \mid \text{Ref}(G_1, \pi_1, \epsilon)(w) = c\}| \neq |\{(v_2, w) \in E(G_2) \mid \text{Ref}(G_2, \pi_2, \epsilon)(w) = c\}|$ ).

Sequences (or  $t$ -tuples) of vertices  $\nu_1 \in (G_1, \pi_1)^t$  and  $\nu_2 \in (G_2, \pi_2)^t$  are *distinguishable*, if the graphs  $(G_1, \text{Ref}(G_1, \pi_1, \nu_1))$  and  $(G_2, \text{Ref}(G_2, \pi_2, \nu_2))$  are.

**Cell Selector.** In a backtracking fashion, the goal of an IR-algorithm is to reach a discrete coloring using color refinement and individualization. For this, color refinement is first applied. If this does not yield a discrete coloring, individualization is applied, branching over all vertices in one non-singleton cell. The task of the *cell selector* is to isomorphism invariantly pick the non-singleton cell. After individualization, color refinement is applied again and the process continues recursively. Formally, a cell selector is a function  $\text{Sel}: \mathcal{G} \times \Pi \rightarrow 2^V$  (where  $\mathcal{G}$  denotes the set of all graphs and  $\Pi$  denotes the set of all colorings), satisfying:

- Isomorphism invariance, i.e.,  $\text{Sel}(G^\varphi, \pi^\varphi) = \text{Sel}(G, \pi)^\varphi$  for  $\varphi \in \text{Sym}(n)$ .
- If  $\pi$  is discrete then  $\text{Sel}(G, \pi) = \emptyset$ .
- If  $\pi$  is not discrete then  $|\text{Sel}(G, \pi)| > 1$  and  $\text{Sel}(G, \pi)$  is a cell of  $\pi$ .

**IR-Tree.** We describe the IR-tree  $\Gamma_{\text{Sel}}(G, \pi)$  of a colored graph  $(G, \pi)$ , which depends on a chosen cell selector  $\text{Sel}$ . Essentially, IR-Trees simply describe the call-trees stemming from the aforementioned backtracking procedure. Nodes of the search tree are sequences of vertices of  $G$ . The root of  $\Gamma_{\text{Sel}}(G, \pi)$  is the empty sequence  $\epsilon$ . If  $\nu$  is a node in  $\Gamma_{\text{Sel}}(G, \pi)$  and  $C = \text{Sel}(G, \text{Ref}(G, \pi, \nu))$ , then the set of children of  $\nu$  is  $\{\nu.v \mid v \in C\}$ , i.e., all extensions of  $\nu$  by one vertex  $v$  of  $C$ .

By  $\Gamma_{\text{Sel}}(G, \pi, \nu)$  we denote the subtree of  $\Gamma_{\text{Sel}}(G, \pi)$  rooted in  $\nu$ . We omit the index  $\text{Sel}$  when apparent from context.

We recite the following fact on isomorphism invariance of the search tree as given in [16], which follows from the isomorphism invariance of  $\text{Sel}$  and  $\text{Ref}$ :

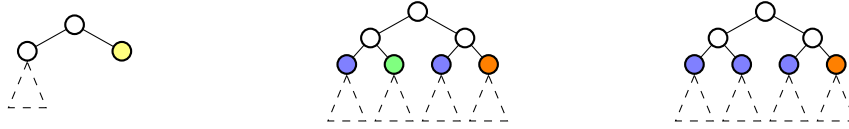
► **Lemma 3.** *If  $\nu$  is a node of  $\Gamma(G, \pi)$  and  $\varphi \in \text{Aut}(G, \pi)$ , then  $\nu^\varphi$  is a node of  $\Gamma(G, \pi)$  and  $\Gamma(G, \pi, \nu)^\varphi = \Gamma(G, \pi, \nu^\varphi)$ .*

**Quotient Graph.** The IR-tree itself can be exponentially large in the order of  $G$  [18]. To decrease its size IR-algorithms use a pruning mechanism. For this a *node invariant* is used. A node invariant is a function  $\text{Inv}: \mathcal{G} \times \Pi \times V^* \rightarrow I$  that assigns to each sequence of nodes of the tree a value in a totally ordered set  $I$ . It satisfies the following.

- Isomorphism invariance, i.e.,  $\text{Inv}(G, \pi, \nu_1) = \text{Inv}(G^\varphi, \pi^\varphi, \nu_1^\varphi)$  for  $\varphi \in \text{Sym}(n)$ .
- If  $|\nu_1| = |\nu_2|$  and  $\text{Inv}(G, \pi, \nu_1) < \text{Inv}(G, \pi, \nu_2)$ , then for all nodes  $\nu'_1 \in \Gamma(G, \pi, \nu_1)$  and  $\nu'_2 \in \Gamma(G, \pi, \nu_2)$  it holds that  $\text{Inv}(G, \pi, \nu'_1) < \text{Inv}(G, \pi, \nu'_2)$ .

The particular way the node invariant can be exploited depends on the problem to be solved. When solving for graph isomorphism, the algorithm may prune all nodes with an invariant differing from an arbitrary node invariant. However, when algorithms want to compute a canonical labeling, they must find a specific canonical node invariant to continue with. However, in the context of the present work these details are not important.

Most IR-algorithms use a specific invariant, the so-called *quotient graph*, which is naturally produced by color refinement.



■ **Figure 1** Forbidden structures in asymmetric binary IR-trees.

For an equitable coloring  $\pi$  of a graph  $G$ , the quotient graph  $Q(G, \pi)$  captures the information of how many neighbors vertices from one cell have in another cell. Quotient graphs are complete directed graphs in which each vertex has a self-loop. They include vertex colors as well as edge colors. The vertex set of  $Q(G, \pi)$  is the set of all colors of  $(G, \pi)$ , i.e.,  $V(Q(G, \pi)) := \pi(V(G))$ . The vertices are colored with the color of the cell they represent in  $G$ . We color the edge  $(c_1, c_2)$  with the number of neighbors a vertex of cell  $c_1$  has in cell  $c_2$  (possibly  $c_1 = c_2$ ). Since  $\pi$  is equitable, all vertices of  $c_1$  have the same number of neighbors in  $c_2$ .

A crucial fact is that graphs are *indistinguishable by color refinement* if and only if their quotient graphs on the coarsest equitable coloring are equal.

We should also remark that quotient graphs are indeed *complete invariants*, yielding the following property.

► **Lemma 4.** *Let  $\nu, \nu'$  be leaves of  $\Gamma(G, \pi)$ . There exists an automorphism  $\varphi \in \text{Aut}(G, \pi)$  with  $\nu = \varphi(\nu')$  if and only if  $Q(G, \text{Ref}(G, \pi, \nu)) = Q(G, \text{Ref}(G, \pi, \nu'))$ .*

Consistent with the colors of trees used in [5], we may also view quotient graphs as a way to color IR-trees themselves, i.e., where we color a node  $\nu$  with  $Q(G, \text{Ref}(G, \pi, \nu))$ .

### 3 Necessary Conditions for IR-Trees

We collect necessary conditions for the structure of IR-trees. Since IR-trees are the result of a branching process, they are naturally irreducible (no node has exactly one child). Also, indistinguishable leaves can be mapped to each other.

► **Lemma 5.** *IR-trees are irreducible.*

► **Lemma 6.** *Let  $l_1, l_2$  be two leaves of an IR-tree  $(T, \pi)$ . If  $l_1$  and  $l_2$  are indistinguishable, there is an automorphism  $\varphi \in \text{Aut}(T, \pi)$  mapping  $l_1$  to  $l_2$ .*

► **Lemma 7** (see e.g. [4]). *A leaf  $l$  can be mapped to exactly  $|\text{Aut}(G, \pi)|$  leaves in  $\Gamma(G, \pi)$  using elements of the automorphism group  $\text{Aut}(G, \pi)$ .*

It follows that all classes of indistinguishable leaves have equal size.

Since in color refinement, partitionings and hence quotient graphs only ever become finer and more expressive, the following properties hold.

► **Lemma 8.** *Let  $n_1, n_2$  be two nodes of an IR-tree where  $n_i$  is on level  $l_i$ .*

1. *If  $l_1 \neq l_2$ , then  $n_1$  and  $n_2$  are distinguishable.*
2. *Consider the two walks starting in the root and ending in  $n_1$  and in  $n_2$ , respectively. If in these walks two nodes on the same level are distinguishable then  $n_1$  and  $n_2$  are distinguishable.*

Some further restrictions apply specifically in the case of cells of size 2.

► **Lemma 9** (Forbidden Binary Structures).

1. If a node  $n$  has two children  $n_1$  and  $n_2$ , then it cannot be that exactly one of the children  $n_1$  or  $n_2$  is a leaf (see Figure 1, left).
2. If  $n_1, n_2$  are any two nodes and  $n_1$  has exactly 2 children then the multiset of colors of the children of  $n_1$  and  $n_2$  are equal or disjoint (Figure 1, middle and right).

**Proof.** Part 1 follows from the fact that individualizing one vertex in a cell of size 2 also individualizes the other vertex of the cell.

For Part 2 we note that individualization of a child of  $n_1$  also individualizes the other child of  $n_1$  and vice versa. This implies that if a child  $c_2$  of  $n_2$  has the same color as some child  $c_1$  of  $n_1$ , then by definition, individualization of  $c_1$  and  $c_2$ , respectively, produces indistinguishable colorings. So in this case there is a one-to-one correspondence between the colors of the children of  $n_1$  and those of  $n_2$ . ◀

It is easy to see that if at any point the cell selector chooses differently sized cells in different branches, the branches subsequently become distinguishable. However, if we assume cell selectors only base their decision on the quotient graph, this restriction applies earlier. More specifically, we call a cell selector *quotient-graph-based*, whenever the result of the cell selector depends only on the quotient graph rather than other aspects of  $G$  and  $\pi$  (i.e., we have  $\text{Sel}(Q(G, \pi))$  rather than  $\text{Sel}(G, \pi)$ ). Then, we have the following.

► **Lemma 10.** *If two nodes  $n$  and  $n'$  in an IR-tree are indistinguishable, then their parents have the same number of children. If additionally the cell selector is quotient-graph-based then  $n$  and  $n'$  also have the same number of children.*

Restricting the cell selector to quotient graphs thus changes whether we can distinguish nodes with a differing number of children *before* or *after* individualizing one more vertex. We may even distinguish cells *before* individualization in both cases, if we include the decision of the cell selector into the invariant itself (i.e., using  $(Q(G, \pi), \text{Sel}(G, \pi))$  instead of  $Q(G, \pi)$ ), which is clearly only more expressive in case the cell selector is *not* quotient-graph-based).

In the following, we assume cell selectors are indeed quotient-graph-based. Since we only require a less powerful cell selector, our construction becomes more general. However, in the construction, we could alternatively drop the additional restriction above with minor adjustments by allowing a more powerful cell selector.

For the remainder of this paper we say that a tree fulfills the *necessary conditions*, if none of the conditions laid out by this section are violated.

## 4 Graph Constructions

Given a colored tree  $(T, \pi)$  which satisfies the necessary conditions, we describe how to construct a graph  $G(T, \pi)$  and a cell selector  $S(T, \pi)$  for which  $(T, \pi)$  is (up to renaming of colors) the IR-tree  $\Gamma_{S(T, \pi)}(G(T, \pi))$ . We make abundant use of gadget constructions, which we describe first. We give a concise description of the construction, details (i.e., the long version) can be found in the full version of the paper [2] (including omitted correctness arguments).

### 4.1 Gadgets

All our gadgets have multiple *input* and *output gates*. Each gate is a pair of vertices that together form their own color class in the gadget. Vertices in the gates are the only vertices of the gadgets connected to other vertices outside the gadget. We say that vertices labeled

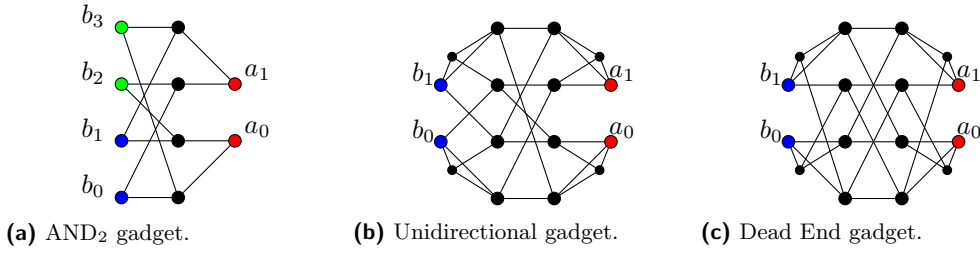


Figure 2 The AND<sub>2</sub> gadget and two variants of directional gadgets.

with  $b_i$  denote the “input”, while  $a_i$  indicates “output”. Gates can be *activated*, by which we mean the process of distinguishing the vertices of the gate pair into distinct color classes, and applying color refinement afterwards. We say activation *discretizes* the gadget if the resulting stable coloring on the gadget vertices is discrete.

Three of the gadgets we present (AND <sub>$i$</sub> , Unidirectional and Dead End gadget) have already been used in other contexts related to color refinement [6, 7, 10].

**AND <sub>$i$</sub>  Gadget [6, 7, 10].** The AND<sub>2</sub> gadget as illustrated in Figure 2a, realizes the logical conjunction of gates with respect to color refinement, and an XOR gadget with respect to automorphisms.

Given  $i > 2$ , we can realize an AND <sub>$i$</sub>  gadget with  $i$  input gates by combining multiple AND<sub>2</sub> gadgets in a tree-like fashion. The AND <sub>$i$</sub>  gadget is constructed by attaching the first and second input gate to an AND<sub>2</sub>, whose output is connected to another AND<sub>2</sub> together with the third input gate, and so on. We use colors to order the input gates, i.e., we color the  $i$ -th input gate with color  $i$ .

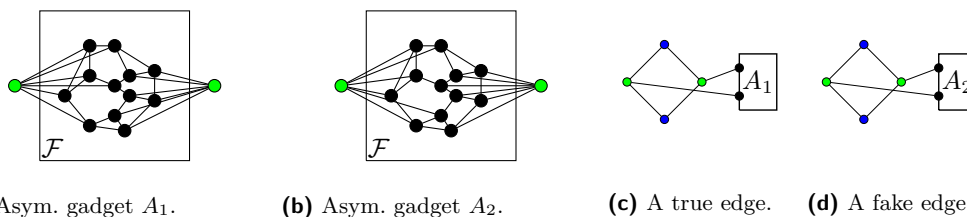
We define the special case of the AND<sub>1</sub> gadget to simply consist of a pair of vertices that functions as the input and output gate at the same time.

► **Lemma 11** ([10]). *The AND <sub>$i$</sub>  gadget admits automorphisms that flip the output gate and either one of the input gates while fixing other input gates. As long as some input gate remains unsplit, the output gate is not split but activating all inputs discretizes the gadget.*

**Unidirectional and Dead End Gadget [6, 10].** Next, we describe gadgets through which gate activation can be propagated or blocked depending on the direction of the gadget. Specifically we construct the unidirectional gadget (Figure 2b) and the dead end gadget (Figure 2c). Note that the two gadgets are indistinguishable from each other by color refinement. The smaller vertices depicted in Figure 2 have been included to guarantee that the gadgets become discrete after the input and output gate has been split and can otherwise be ignored.

► **Lemma 12.** *The unidirectional and dead end gadget are indistinguishable by color refinement. In the unidirectional case, activating the input discretizes the gate but activating the output does not split the input gate. In the dead end case, both input and output have to be activated to discretize the gadget.*

**Asymmetry Gadgets.** Our next gadgets only have one gate (see Figure 3,  $\mathcal{F}$  denotes the Frucht graph). The crucial property of the asymmetry gadgets  $A_1$  and  $A_2$  (Figures 3a and 3b) is for the two gate vertices to be initially indistinguishable by color refinement, but to ensure that individualizing one of the gate vertices leads to a different quotient graph than individualizing the other gate vertex.



■ **Figure 3** Asymmetry gadgets and concealed edges based on asymmetry gadgets.

► **Lemma 13.**  $A_1$  and  $A_2$  are asymmetric and stable under color refinement. Activating the input gate discretizes the gadget and we obtain two non-isomorphic colorings depending on which vertex was individualized. Furthermore,  $A_1 \not\cong A_2$ .

**Concealed Edges.** Lastly, we describe the *concealed edge gadget* that is used to hide edges from color refinement. The gadget has two vertices that represent the endpoints of an edge (the blue vertices in Figure 3c). The idea is that instead of an edge connecting the two vertices, we insert a concealed edge gadget. For this the gadget has a pair consisting of two *inner vertices* (the green vertices in Figure 3c), which are both connected to each input vertex. This pair is then connected to an asymmetry gadget. We define two classes of edges, where one type of edge attaches the gadget  $A_1$  (*true edges*) and the other  $A_2$  (*fake edges*).

The crucial property is that as long as inner vertices of the gadgets are not distinguished, color refinement can not distinguish between true edges and fake edges. However, if we distinguish the inner vertices, true edges can indeed be distinguished from fake edges.

We always employ this gadget within the following design pattern. Whenever we want to connect two sets of vertices  $V_1$  and  $V_2$  with edges  $E \subseteq V_1 \times V_2$  in a concealed manner, we first add a concealed edge gadget between *all* pairs  $(v_1, v_2) \in V_1 \times V_2$ . However, only if  $(v_1, v_2) \in E$ , we use a *true edge*, and whenever  $(v_1, v_2) \notin E$  we use a *fake edge*. Finally, we connect all pairs of inner vertices of the concealed edge gadgets to some construction that is used to *reveal* the edges.

The asymmetry gadget prohibits automorphisms from flipping the concealed edge gadget itself. However, care has to be taken when connecting the inner vertices to other constructions: it is imperative to connect the inner vertices of multiple concealed edge gadgets that are on the, say, left side of the asymmetry gadget, in the same manner. Otherwise, once revealed, edges could possibly be distinguished into even more categories than just fake and true edges.

## 4.2 A construction for asymmetric trees

For our construction, we first restrict ourselves to asymmetric trees, i.e., all leaves have different colors. Building on this, the following section takes symmetries into account. Let  $(T, \pi)$  be an *asymmetric*, colored tree that satisfies the necessary conditions (see Section 3). We describe a graph  $G(T, \pi)$  and cell selector  $S(T, \pi)$  such that  $(T, \pi)$  is (up to renaming of colors) the IR-tree  $\Gamma_{S(T, \pi)}(G(T, \pi))$ .

The goal is to model the graph and cell selector in such a way that there is a one-to-one correspondence between paths in  $T$  and sequences of individualizations in  $G(T, \pi)$ . Such sequences are precisely the paths in the IR-tree  $\Gamma_{S(T, \pi)}(G(T, \pi))$ . To guarantee this correspondence, certain properties of paths in  $T$  must translate into specific properties for their respective sequence of individualizations. In particular, when modeling  $G(T, \pi)$ , we ensure the following.

## 24:10 A Characterization of Individualization-Refinement Trees



■ **Figure 4** Connecting levels of the selector tree. Blue/red edges on the right symbolize true/fake edge gadgets.



■ **Figure 5** Colors of  $T$  are encoded in  $G(T)$  by concealed edges to special *color nodes*.

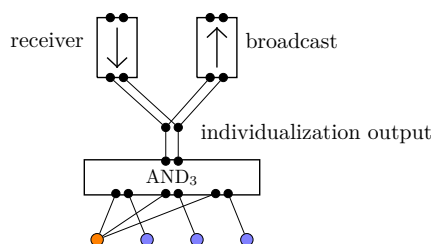
1. Two paths must end in nodes of different color exactly if the corresponding sequences of individualizations result in different quotient graphs.
2. A path must end in a leaf exactly if the corresponding sequence of individualizations (when followed by color refinement) results in a discrete coloring.

**Selector Tree.** We start by describing the part of the graph on which the cell selector operates, i.e., within which cells are chosen. We use a copy of  $T$  itself for this purpose. One of the central difficulties is that color refinement executed on  $T$  may actually result in a coloring that is finer than  $\pi$ .

Hence, the selector tree consists of the nodes of  $T$  but encodes the edges of  $T$  in the selector tree using concealed edges (see Figure 4). This guarantees that our copy of  $T$  is stable under color refinement. At some point, we will need to add another gadget construction to ensure that edges between the levels are actually revealed at the right time. Assuming this for now, the cell selector  $S(T, \pi)$  always chooses as next cell the cell that consists of the children of the node chosen last.

**Colors.** Next, we consider the colors  $\pi$  of  $T$ . Colors indicate whether a sequence of individualizations should lead to differing quotient graphs. We make use of fake edges again to encode this: we encode a one-to-one correspondence between selector tree nodes and their color in  $\pi$  using concealed edges (see Figure 5). We will discuss how edges are revealed further below. Specifically, we will reveal edges incident with node  $n$  at the point in time when node  $n$  is individualized.

**Leaf Detection.** Whenever we individualize a node  $n$  in the selector tree, we want to react to this by revealing a specific set of edges. Therefore, we now add a construction that detects whether a specific node  $n$  in a cell was individualized. Let  $s \geq 2$  be the size of the current cell (in the tree  $T$  the current cell is always the set of children of some node). For each node  $n$  in the cell, consider all  $s - 1$  pairs with other nodes of the cell. We add an  $\text{AND}_{s-1}$  gadget and connect the left node of every input pair to  $n$ , and the other to one of the  $s - 1$  other nodes. An  $\text{AND}_{s-1}$  gadget is not symmetric in its input gates, so in order to keep things symmetrical, we add  $(s - 1)!$  many  $\text{AND}_{s-1}$  gadgets for every possible order of nodes in the input. We connect the output gates of the  $\text{AND}_{s-1}$  gadgets to the *individualization output* of  $n$  (see Figure 6). Crucially, the individualization output is activated (i.e., split) whenever  $n$  is individualized, and not immediately activated if only some other node  $n'$  is individualized.



■ **Figure 6** Leaf detection mechanism for the leftmost node of the cell. If this node is individualized, the  $\text{AND}_3$  gadget is activated. The figure only shows true edges.

We again ensure the construction is stable under color refinement by using concealed edges. We use concealed edges between nodes of level  $i$  and the  $\text{AND}_{s-1}$  of level  $i$  gadgets, using true edges instead of the edges described above.

If a node corresponds to a leaf and its individualization output is activated, we want to propagate discretization to other nodes. We add some control structures for every node  $n$  in the selector tree, namely a *broadcast* and *receiver* gadget as illustrated in Figure 6. We use a dead end gadget instead of a unidirectional as the broadcast gadget whenever  $n$  is not a leaf in  $T$ . Next, we connect the output of the broadcast gadget to the input of *all* receiver gadgets in the graph.

The idea is that if  $n$  is a leaf and individualized, we propagate this split to *all* individualization outputs through the broadcast and receiver gadgets, eventually causing a discretization of the graph, as explained below. If  $n$  is not a leaf, the broadcast gadget is a dead end gadget and no further split occurs.

**Revealing Edges.** We now describe when concealed edges are revealed. Assume we are individualizing a node at level  $i$  of the tree. At this point, we need to reveal the edges in the selector tree connecting level  $i$  to level  $i + 1$ . We use the individualization outputs at level  $i$  to reveal edges of the selector tree to level  $i + 1$ : we connect every individualization output through a unidirectional gadget with the internal nodes of the concealed edges between level  $i$  and level  $i + 1$ .

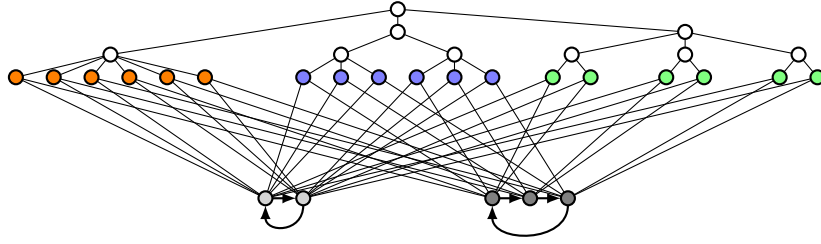
In order to activate the individualization outputs, we also need to reveal edges from level  $i + 1$  nodes to their  $\text{AND}_{s-1}$  gadgets. Hence, we do the same construction as above, connecting the unidirectional gadgets we added on level  $i$  to reveal these edges on level  $i + 1$ . For the first level of the selector tree (children of the root) we do not use concealed edges, such that the level is initially revealed.

Finally, the same technique is also used to reveal colors. The individualization output of node  $n$  reveals the concealed edges between  $n$  and the color nodes. This reveals the color of  $n$  whenever we individualize  $n$ . Due to broadcasting, *all* colors are revealed whenever a leaf is activated, leading to a discrete coloring.

### 4.3 Generating symmetries

We expand our construction so that it can also handle colored trees  $(T, \pi)$  with prescribed symmetries. As such, the graph  $G(T, \pi)$  can also be build from a tree  $(T, \pi)$  that is not necessarily asymmetric. In this case, sequences of individualizations along root-to-leaf paths still produce the desired tree  $(T, \pi)$  as a subtree of  $\Gamma_{S(T, \pi)}(G(T, \pi))$ . However,  $G(T, \pi)$  is supposed to become discrete after the IR-process reaches a leaf, but at this point the selector tree is only split up to orbits prescribed by  $T$ .





■ **Figure 7** Symmetry cycles couple leaf orbits across multiple branches of the selector tree. The illustration omits fake edges. In the construction, cycles do not contain directed edges, but specially colored nodes that indicate direction.

Discretization of orbits is challenging since we need to make sure that the symmetries are not destroyed by the addition of new gadgets. Once leaf orbits have been discretized, discretization propagates through the selector tree and the whole construction becomes discrete.

Overall we need to construct the graph  $H_T$  mentioned in Theorem 2. To construct  $H_T$ , we introduce *symmetry cycles* and *symmetry couplings*. The basic idea is shown in Fig. 7, a detailed explanation can be found in the full version [2]. This in turn defines a new construction  $\tilde{G}(T, \pi)$  by adding a concealed version of  $H_T$  to the selector tree.

**Discretization of Orbits.** We remark that the way we construct  $H_T$  has additional consequences on color refinement: individualization of a root-to-leaf path in  $H_T$  discretizes all symmetry cycles. In terms of  $\tilde{G}(T, \pi)$ , since different leaf orbits have different colors in  $T$ , this means that individualization of a root-to-leaf path now discretizes the set of leaves as well.

**Concealing Symmetry Couplings.** We hide  $H_T$  from color refinement using concealed edges. In the construction of  $H_T$ , we replace edges with true edge gadgets. All other connections between selector tree nodes and symmetry cycles become fake edges. To reveal the edges whenever a leaf is individualized, we connect the inner nodes of the concealed edges to the output of all broadcast gadgets.

## 5 Conclusion and Future Work

We have shown that every tree that meets some simple necessary conditions is an IR-tree. Regarding invariant pruning we should highlight that of course every pruned tree is a subtree of an unpruned tree, so our techniques extend to IR-algorithms with pruning.

Regarding refinement, we use the standard color refinement used by all IR-algorithms. However regarding cell selectors there is no clear standard. In this paper, we did not optimize the construction for any specific cell selector, but rather used the cell selector as part of the construction.

Let us now assume we are given a fixed cell selector. For a particular cell selector, there are two possibilities: either, fewer trees turn out to be IR-trees or the same necessary conditions apply. For the latter, we suspect that for many natural examples the construction of this paper can be adapted. Consider for example the cell selector that always chooses a smallest non-trivial cell. In this case, by adding more concealed structure enforcing specific cell sizes it can be shown that the same necessary conditions are indeed sufficient again.

In contrast to this, consider the cell selector that always chooses a largest non-trivial cell. Here, the degree of the vertices on root-to-leaf walks in a corresponding IR-tree must monotonically decrease. Hence, fewer trees turn out to be IR-trees and the necessary conditions are not sufficient. If interested in specific cell selectors one might therefore want to refine the necessary conditions.

Another interesting direction of research might be to investigate bounds for the order graphs realizing a given tree since this is related to the running time of IR-tools.

---

## References

- 1 Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2112–2118. ijcai.org, 2021. doi:10.24963/ijcai.2021/291.
- 2 Markus Anders, Jendrik Brachter, and Pascal Schweitzer. A characterization of individualization-refinement trees. *CoRR*, abs/2109.07302, 2021. Full version of the paper. arXiv:2109.07302.
- 3 Markus Anders and Pascal Schweitzer. dejavu. URL: <https://www.mathematik.tu-darmstadt.de/dejavu>.
- 4 Markus Anders and Pascal Schweitzer. Engineering a fast probabilistic isomorphism test. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2021, Virtual Conference, January 10-11, 2021*, pages 73–84. SIAM, 2021. doi:10.1137/1.9781611976472.6.
- 5 Markus Anders and Pascal Schweitzer. Search Problems in Trees with Symmetries: Near Optimal Traversal Strategies for Individualization-Refinement Algorithms. In *ICALP 2021*, volume 198 of *LIPICs*, pages 16:1–16:21, 2021. doi:10.4230/LIPICs.ICALP.2021.16.
- 6 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, Sebastian Kuhnert, and Gaurav Rattan. The parameterized complexity of fixing number and vertex individualization in graphs. In *MFCS2016*, volume 58 of *LIPICs*, pages 13:1–13:14, 2016. doi:10.4230/LIPICs.MFCS.2016.13.
- 7 Christoph Berkholtz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. doi:10.1007/s00224-016-9686-0.
- 8 Paul T. Darga, Hadi Katebi, Mark Liffiton, Igor L. Markov, and Karem Sakallah. Saucy3. URL: <http://vlsicad.eecs.umich.edu/BK/SAUCY/>.
- 9 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, Igor L. Markov, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41st Annual Design Automation Conference, DAC '04*, pages 530–534, New York, NY, USA, 2004. ACM. doi:10.1145/996566.996712.
- 10 Martin Grohe. Equivalence in finite-variable logics is complete for polynomial time. In *FOCS '96*, pages 264–273. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548485.
- 11 Tommi Junttila and Petteri Kaski. bliss. URL: <http://www.tcs.hut.fi/Software/bliss/>.
- 12 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.13.
- 13 José Luis López-Presa, Antonio Fernández Anta, and Luis N. Chiroque. conauto2. URL: <https://sites.google.com/site/giconauto/>.
- 14 José Luis López-Presa, Luis Núñez Chiroque, and Antonio Fernández Anta. Novel techniques for automorphism group computation. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, volume 7933 of *LNCS*, pages 296–307. Springer, 2013. doi:10.1007/978-3-642-38527-8\_27.

- 15 Brendan D. McKay and Adolfo Piperno. nauty and Traces. URL: <http://pallini.di.uniroma1.it>.
- 16 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94–112, 2014. doi:10.1016/j.jsc.2013.09.003.
- 17 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI 2019*, pages 4602–4609, 2019. doi:10.1609/aaai.v33i01.33014602.
- 18 Daniel Neuen and Pascal Schweitzer. An exponential lower bound for individualization-refinement algorithms for graph isomorphism. In *STOC 2018*, pages 138–150. ACM, 2018. doi:10.1145/3188745.3188900.
- 19 Adolfo Piperno. Search space contraction in canonical labeling of graphs (preliminary version). *CoRR*, abs/0804.4881, 2008. arXiv:0804.4881.
- 20 Pascal Schweitzer. *Problems of unknown complexity: graph isomorphism and Ramsey theoretic numbers*. Phd. thesis, Universität des Saarlandes, Germany, 2009.

# Truly Subquadratic Exact Distance Oracles with Constant Query Time for Planar Graphs

Viktor Fredslund-Hansen  

Department of Computer Science, University of Copenhagen, Denmark

Shay Mozes  

The Interdisciplinary Center, Herzliya, Israel

Christian Wulff-Nilsen  

Department of Computer Science, University of Copenhagen, Denmark

---

## Abstract

We present a truly subquadratic size distance oracle for reporting, in *constant* time, the exact shortest-path distance between any pair of vertices of an undirected, unweighted planar graph  $G$ . For any  $\varepsilon > 0$ , our distance oracle requires  $O(n^{5/3+\varepsilon})$  space and is capable of answering shortest-path distance queries exactly for any pair of vertices of  $G$  in worst-case time  $O(\log(1/\varepsilon))$ . Previously no truly sub-quadratic size distance oracles with constant query time for answering exact shortest paths distance queries existed.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Data structures design and analysis; Theory of computation  $\rightarrow$  Shortest paths; Mathematics of computing  $\rightarrow$  Graph algorithms

**Keywords and phrases** distance oracle, planar graph, shortest paths, subquadratic

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.25

**Related Version** *Preprint*: <https://arxiv.org/abs/2009.14716>

**Funding** *Shay Mozes*: Partially supported by Israel Science Foundation grant No. 592/17.

*Christian Wulff-Nilsen*: This research is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme and by Basic Algorithms Research Copenhagen (BARC).

## 1 Introduction

Efficiently answering shortest path distance queries between pairs of vertices in a graph is a fundamental algorithmic problem. Given an  $n$ -vertex graph  $G = (V, E)$  a *distance oracle* is a compact data-structure capable of efficiently answering shortest path distance queries between pairs of vertices  $u, v \in V$ . Ideally one would like the data structure to be of linear size and the query time to be constant. However, it is well known that there are graphs for which no distance oracle with  $o(n^2)$  bits of space and  $O(1)$  query time exists. In fact, even resorting to approximation, Pătraşcu and Roditty [21] showed that there are sparse graphs on  $O(n \text{ polylog } n)$  edges for which constant query-time distance oracles with stretch less than 2 must be of size  $\Omega(n^2 \text{ polylog } n)$ , assuming the set intersection conjecture. These impossibility results make it natural to consider the problem in restricted classes of graphs.

In this paper we consider exact distance oracles for planar graphs. Distance oracles for planar graphs are well motivated by important real-world applications, notably in routing, navigation of road and sea maps as well as in the context of computational geometry. To the best of our knowledge there are no non-trivial lower bounds for (static) distance oracles for planar graphs, and thus achieving the “holy grail” of a linear-size distance oracle with constant query time may be possible. Indeed, there has been numerous works over at least three decades developing exact distance oracles for planar graph [10, 2, 3, 8, 11, 19, 20]. However, only in 2017, Cohen-Addad et al. [9] gave the first oracle with truly subquadratic



© Viktor Fredslund-Hansen, Shay Mozes, and Christian Wulff-Nilsen;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 25; pp. 25:1–25:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

space and polylogarithmic query time. Their result was inspired by Cabello’s [4] breakthrough result, who gave the first truly sub-quadratic time algorithm for computing the diameter of planar graphs by a novel use of Voronoi diagrams. The approach of [9] was subsequently improved by [12, 7, 18], who gave an elegant point-location mechanism for Voronoi diagrams in planar graphs, and combined it with a clever recursive scheme to obtain exact distance oracles for directed weighted planar graphs with  $O(n^{1+o(1)})$  space and  $O(\log^{2+o(1)} n)$  query time. We note that even though the oracles of [7, 18] get quite close to optimal, it remains wide open to support exact queries in *constant time* using truly subquadratic space, even in the most basic case of unweighted undirected planar graphs [26, 5, 25].

Allowing approximate answers does help in planar graphs. Many results reporting  $(1 + \varepsilon)$ -approximate distances with various tradeoffs exist, all with (nearly) linear size and polylogarithmic, or even  $O(1/\varepsilon)$  query-time [24, 15, 14, 27]. Gu and Xu [13] presented a size  $O(n \text{ polylog } n)$  distance oracle capable of reporting  $(1 + \varepsilon)$ -approximate distances in time  $O(1)$ . While their query time is a constant independent of  $\varepsilon$ , the preprocessing time and space are nearly linear, but with an exponential dependency on  $(1/\varepsilon)$ . This exponential dependency can be made polynomial, but then the query time increases to  $O(\log(1/\varepsilon))$  [6].

Thus, despite the large body of work on distance oracles for planar graphs, it has remained an open question to determine whether an exact distance oracle with of size  $O(n^{2-\varepsilon})$  with *constant* query-time can be constructed for some constant  $\varepsilon > 0$ .

## 1.1 Our contributions

We answer this question in the affirmative, and discuss our techniques in the following section. Our result is summarized in the following theorem:

► **Theorem 1.** *Let  $G = (V, E)$  be an undirected unweighted  $n$ -vertex planar graph. For any  $\varepsilon > 0$  there exists a data-structure requiring  $O(n^{5/3+\varepsilon})$  space that, for any  $s, t \in V$ , reports the shortest path distance between  $s$  and  $t$  in  $G$  in time  $O(\log(1/\varepsilon))$ .*

We also present another denser subquadratic distance oracle in Section 4, and remark that it can be distributed into a distance labeling scheme with size  $O(n^{3/4})$  per label, such that the distance between any two vertices  $s, t$  can be computed in  $O(1)$  time given just the labels of  $s$  and  $t$ .

## 1.2 Technical overview

The main concept we use to obtain our result is that of a *pattern* capturing distances between a vertex and a cycle. This concept was used by [26] and by [17] (where it was called a “distance tuple”). Consider a vector storing the distances from a vertex  $u$  to the vertices of a cycle  $\beta$  in their cyclic order. The pattern of  $u$  w.r.t.  $\beta$  is simply the discrete derivative of this vector. That is, the vector obtained by taking the difference between every pair of consecutive values. Li and Parter [17] showed that when the input graph is planar, the number of different patterns w.r.t. a face with  $r$  vertices is  $O(r^3)$ , regardless of the size of the graph. We next outline how this observation can be used to break the quadratic space barrier.

Roughly speaking, any planar graph can be decomposed into  $O(n/r)$  subgraphs, called regions, of size  $r$  each, where the boundary of each region (i.e., the vertices that have neighbors outside the region) is a single cycle  $h$ .<sup>1</sup> Applying Li and Parter’s observation in

<sup>1</sup> In fact, a constant number of cycles. To readers familiar with the concept, this is just an  $r$ -division with a few holes, but *without* the important feature that each region has just  $O(\sqrt{r})$  boundary vertices. This is because one cannot triangulate unweighted graphs without changing the distances.

this setting, the number of different patterns for the hole of each region  $R$  is  $O(r^3)$ . Hence, we can represent the distances from any vertex  $s \notin R$  to  $h$  by just storing the distance from  $s$  to an arbitrarily chosen canonical vertex  $v_h$  of  $h$ , and a pointer to the pattern of  $s$  with respect to  $h$ . This requires just a total of  $O(n)$  space for all vertices not in  $R$  plus  $O(r^3 \cdot r)$  for storing all the patterns for  $h$ . Summing over all  $O(n/r)$  regions, the space required is  $O(n^2/r + nr^3)$ .

We then define the notion of distance from a pattern to a vertex (see Definition 6). While this definition is simple, it is somewhat unnatural because the distance from a pattern to a vertex does not necessarily correspond to the length of any specific path in the graph! However, the distance between  $s$  and any vertex  $t \in R$  is just the sum of the distance between  $s$  and the canonical vertex  $v_h$  and the distance from the pattern of  $s$  with respect to  $h$  to  $t$ .

We therefore store the distances from each of the  $O(r^3)$  possible patterns of  $R$  to each vertex of  $R$ . This requires  $O(r^3 \cdot r)$  space per region, so  $O(nr^3)$  space overall. This way we can report the distance between  $s$  and  $t$  in constant time by (i) retrieving the pattern  $p$  of  $s$  w.r.t.  $h$ , and (ii) adding the distance from  $s$  to the canonical vertex  $v_h$  of  $h$  and the distance from the pattern  $p$  to  $t$ . These ideas alone already imply an oracle with space  $\tilde{O}(n^{7/4})$  and constant query time. Combining these ideas with recursion yields the improved space bound of Theorem 1.

### 1.3 Technical differences from previous oracles

As we previously mentioned, breaking the quadratic space barrier for constant query time has remained a long standing open question and can therefore be considered an important and significant result in its own right. We highlight the following difference between the approach taken in our recursive oracle and the approaches used in all existing distance oracles we are aware of. To the best of our knowledge, all existing distance oracles, both exact and approximate, and both for general graphs and planar graphs, recover the distance from  $s$  to  $t$  by identifying a vertex or vertices on some (possibly approximate) shortest path between  $s$  and  $t$ , for which distances have been stored in the preprocessing stage. These vertices are usually referred to as landmarks, portals, hubs, beacons, seeds, or transit nodes (cf. [23]). Our oracle, on the other hand, reports the exact shortest path without identifying vertices on the shortest path from  $s$  to  $t$ . Instead, it “zooms in” on  $t$  by recovering distances to the canonical vertices of a sequence of subgraphs of decreasing size that terminates at a constant size graph containing  $t$ . We emphasize that *none of these canonical vertices necessarily lies on a shortest path* from  $s$  to  $t$ . This property may be viewed as a disadvantage if we also want to report the shortest path, but when reporting multiple edges on long paths, constant query time is no longer relevant. On the other hand, it may be that just reporting the distance is easier than also reporting an internal vertex on a shortest path. Hence, it may be that developing oracles based on this new approach may lead to further advances on the way to linear size distance oracles for planar graphs with constant query time, and in other related problems.

## 2 Preliminaries

Let  $G$  be a graph. We denote by  $V(G)$  and  $E(G)$  the vertex and edge-set of  $G$ , and denote by  $n = |V(G)|$  the number of vertices of  $G$ .

For a subset  $S$  of edges or vertices we denote by  $G[S]$  the subgraph of  $G$  induced on  $S$ . We denote by  $u \rightsquigarrow_H v$  a *shortest path* from  $u$  to  $v$  in the subgraph  $H$ , by  $\mathbf{d}_H(u, v)$  the length of  $u \rightsquigarrow_H v$ , and define  $u \rightsquigarrow v \equiv u \rightsquigarrow_G v$ .

The following definitions will be useful when talking about decompositions of  $G$ . A *region*  $R$  of  $G$  is an edge-induced subgraph of  $G$ , and its *boundary*  $\partial R$  is the set of vertices of  $R$  that are adjacent to some vertex of  $V(G) \setminus V(R)$  in  $G$ . Vertices of  $V(R) \setminus \partial R$  are called *interior* vertices of  $R$ . Observe that for a region  $R$  and for  $u \in R$  and  $v \in V \setminus V(R)$ , any path from  $u$  to  $v$  in  $G$  must intersect  $\partial R$ .

It will be useful to assume some global strict order on a vertex set  $V$  s.t. for any  $U \subseteq V$  there is a minimum vertex  $\min U \in U$  w.r.t this order. We refer to this as the *canonical vertex* of  $U$ .

We assume the reader is familiar with the the basic definitions of planarity and of planar embeddings.

## 2.1 Faces and holes

The edges of a plane graph induce maximal open portion of the plane that do not intersect any edges. A *face* of the graph is the closure of one such portion of the plane. We refer to the edges bounding a face as the *boundary* of that face. Given a face  $f$ ,  $V(f)$  is the set of vertices on the boundary of  $f$ . We denote by  $w(f)$  the *facial walk* of  $f$  which is the sequence of vertices encountered when walking along  $f$  starting at  $\min V(f)$  and going in the clockwise direction. Note that  $f$  may be non-simple, so some vertices may appear multiple times in  $w(f)$ . A *hole*  $h$  in a region  $R$  of a graph  $G$  is a face of  $R$  which is not a face in  $G$ . We say that a vertex  $u \in V(G) \setminus V(R)$  is *inside* hole  $h$  if  $u$  lies in the region of the plane corresponding to the face  $h$  of  $R$ . We denote by  $V^+(h) = \{u \in V(G) \mid u \text{ is inside } h\}$  all the vertices that are inside  $h$ .

## 2.2 Decompositions of unweighted planar graphs

An  $r$ -division is a widely used decomposition of planar graphs into regions with small boundary. We use the  $r$ -divisions with a few holes as studied in [16], which works for triangulated biconnected graphs:

► **Lemma 2** ( *$r$ -division with few holes for triangulated graphs [16]*). *Let  $G$  be a biconnected, triangulated  $n$ -vertex planar embedded graph, and let  $0 < r \leq n$ .  $G$  can be decomposed into  $\Theta(n/r)$  connected regions, each of which with  $O(r)$  vertices and  $O(\sqrt{r})$  boundary vertices. Each region has a constant number of holes. Every boundary vertex lies on some hole, and each hole has  $O(\sqrt{r})$  vertices.*

The fact that the boundaries of regions are small (only  $O(\sqrt{r})$  boundary vertices for a region with  $r$  vertices) is the basis for many efficient algorithms and data structures for planar graphs. Unweighted planar graphs possess additional structure (in comparison to weighted planar graphs), which may also be useful algorithmically. See for example the unit-Monge property in [1], or the limited number of patterns [26, 17], which we use in this work. However, exploiting such additional structure in conjunction with a decomposition into regions with small boundaries has been elusive because of the seemingly technical requirement in Lemma 2 that the graph be triangulated and biconnected.

Any graph can be triangulated and biconnected by adding to each face  $f$  an artificial vertex and infinitely weighted artificial edges from the artificial vertex to each vertex of  $V(f)$ . This transformation preserves planarity and shortest paths, and ensures that the graph consists only of simple faces of size 3. However, the graph is no longer unweighted. We refer to an *artificial* vertex (edge) of  $G$  as a vertex (edge) which was added in the triangulation step, and a *natural* vertex (edge) of  $G$  as a vertex (edge) which is not artificial. In order to exploit the structure of the unweighted input graph we will remove the artificial edges and



vertices after computing the decomposition using Lemma 2. On the one hand the graph is again unweighted. On the other hand, while the number of boundary vertices in each region remains  $O(\sqrt{r})$ , the holes may now contain new non-boundary vertices, and the total size of the holes in each region may be  $\Theta(r)$ . We note, however, that the deletion of artificial edges and vertices does not disconnect regions [16]. We therefore restate the decomposition lemma for unweighted graphs that are not necessarily triangulated or biconnected.

► **Lemma 3** (*r*-division with few holes for non-triangulated graphs). *Let  $G$  be a  $n$ -vertex planar embedded graph  $G$ , and let  $0 < r \leq n$ .  $G$  can be decomposed into  $\Theta(n/r)$  connected regions, each with  $O(r)$  vertices and  $O(\sqrt{r})$  boundary vertices. Each region has a constant number of holes, and each boundary vertex lies on some hole.*

### 2.3 Recursive $r$ -divisions

Our second construction relies on a *recursive  $r$ -division* which is a recursive decomposition of  $G$  into  $r$ -divisions for varying values of  $r$ . Specifically, for a decreasing sequence  $\mathbf{r} = r_1, r_2, \dots$ , where  $n \geq r_1 > r_2 > \dots \geq 1$ , we want  $r_i$ -divisions for all  $i = 1, 2, \dots$ , such that each region in the  $r_i$  division is the union of regions in the  $r_{i+1}$ -division on the next level. We associate with the recursive  $r$ -division a decomposition tree,  $\mathcal{T}_{\mathbf{r}}$ , which is a rooted tree whose nodes correspond to the regions of the recursive decomposition of  $G$ . We will refer to nodes and their corresponding regions interchangeably. The root node corresponds to all of  $G$ . A node  $x$  of  $\mathcal{T}_{\mathbf{r}}$  at depth  $i$  corresponds to a region of the  $r_i$ -division, and its children are the regions of the  $r_{i+1}$ -division whose union is the region corresponding to  $x$ . We denote by  $\mathcal{T}_{\mathbf{r}}^i$  all the nodes at level  $i$ . It was shown in [16] that recursive  $r$ -divisions can be computed efficiently:

► **Lemma 4** (Recursive  $r$ -division). *Given a biconnected, triangulated  $n$ -vertex planar graph  $G$  and an exponentially decreasing sequence  $\mathbf{r} = n \geq r_1, r_2, \dots \geq 1$ , a decomposition tree,  $\mathcal{T}_{\mathbf{r}}$  can be computed in linear time s.t.  $\mathcal{T}_{\mathbf{r}}^i$  corresponds to an  $r_i$ -division of  $G$  with few holes for each  $i$ .*

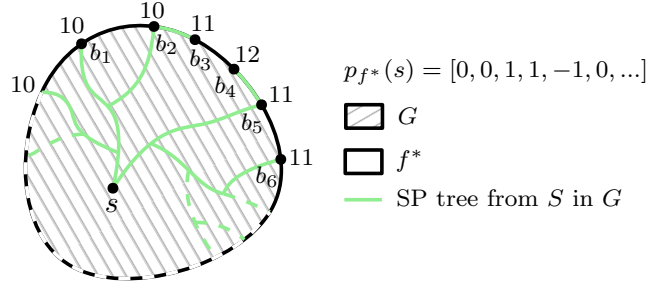
## 3 Patterns

Both [26] and [17] introduce a notion of a “distance tuple” which can be thought of as a vector of shortest-path distances from a vertex to consecutive vertices of some hole. We introduce the following similar notion of a *pattern* (See Figure 1 for an illustration):

► **Definition 5** (Pattern). *Let  $G$  be a graph. Let  $H$  be a subgraph of  $G$ . Let  $u$  be a vertex in  $H$ , and let  $\beta = b_0, b_1, \dots, b_k$  be a path in  $H$ . The pattern of  $u$  (w.r.t.  $\beta$  in  $H$ ) is a vector  $p_{\beta, H}(u) \in \{-1, 0, 1\}^k$  satisfying  $p_{\beta, H}(u)[i] = \mathbf{d}_H(u, b_i) - \mathbf{d}_H(u, b_{i-1})$  for  $1 \leq i \leq k$ . When the path  $\beta$  is the boundary walk  $w(h)$  of a hole  $h$  of a region  $R$ , we write  $p_{h, H}(u)$  instead of  $p_{w(h), H}(u)$ .*

► **Definition 6** (pattern to vertex distance). *Let  $R$  be a region in a graph  $G$ . Let  $h$  be a hole of  $R$ . Let  $b_0, b_1, \dots, b_k$  be the vertices of  $w(h)$  in their cyclic order. Let  $p$  be some pattern w.r.t.  $h$  (i.e.,  $p = p_{h, G}(u)$  for some  $u \in V^+(h)$ ). For a vertex  $v \in R$  we define  $\mathbf{d}_G(p, v)$  the distance between  $p$  and  $v$  to be  $\min_{i=0}^k \left\{ \mathbf{d}_G(b_i, v) + \sum_{j=0}^i p[j] \right\}$ .*

► **Lemma 7**. *Let  $R$  be a region of a graph  $G$ . Let  $h$  be a hole of  $R$ . For every  $u \in V^+(h)$  and every  $v \in R$ ,  $\mathbf{d}_G(u, v) = \mathbf{d}_G(u, b_0) + \mathbf{d}_G(p_{h, G}(u), v)$ .*



■ **Figure 1** Illustration of the pattern of the vertex  $s$  w.r.t.  $f^*$  in an undirected graph. In this case  $f^*$  is the external face of the embedding. The numeric labels indicate the shortest path distances from  $s$  to each  $b_i$  where  $b_i \in V(f^*)$  for  $1 \leq i \leq 6$ .

**Proof.** By definition of pattern and by a telescoping sum, for every  $0 \leq i \leq k$ ,  $\mathbf{d}_G(u, b_i) = \mathbf{d}_G(u, b_0) + \sum_{j=0}^i p[j]$ . Let  $b_\ell$  be any vertex of  $w(h)$  on a shortest  $u$ -to- $v$  path ( $b_\ell$  exists since  $u \in V^+(h)$  and  $v \in R$ ). By choice of  $b_\ell$ ,  $\mathbf{d}_G(u, v) = \mathbf{d}_G(u, b_\ell) + \mathbf{d}_G(b_\ell, v) = \min_{0 \leq i \leq k} \{\mathbf{d}_G(u, b_i) + \mathbf{d}_G(b_i, v)\} = \min_{0 \leq i \leq k} \{\mathbf{d}_G(u, b_0) + \sum_{j=0}^i p[j] + \mathbf{d}_G(b_i, v)\} = \mathbf{d}_G(u, b_0) + \mathbf{d}_G(p, v)$ . ◀

### 3.1 Bounding the number of patterns

As mentioned, in a recent paper Li and Parter [17] achieve improved bounds for diameter computation for planar graphs by showing that in unweighted undirected planar graphs the number of patterns is quite small. More specifically, they show that the VC-dimension of a set corresponding to all patterns is at most 3. By the Sauer-Shelah lemma [22], this implies that the number of distinct patterns w.r.t. a sequence  $S$  of consecutive vertices on a face is in  $O(|S|^3)$ . Their result is stated in the following lemma:

► **Lemma 8** (Pattern compression [17]). *Let  $G' = (V, E)$  be an  $n$ -vertex unweighted undirected planar graph, let  $f$  be a face in  $G'$ , and let  $S$  be a sequence of consecutive vertices on  $f$ . Then the number of distinct patterns w.r.t.  $S$ ,  $|\bigcup_{u \in V} \{p_{S, G'}(u)\}|$ , is bounded by  $O(|S|^3)$ .*

We observe that the bound of Lemma 8 also holds for patterns w.r.t. the entire set of vertices on a hole  $h$  of a region  $R$  even when distances are defined in the entire graph  $G$ .

► **Corollary 9.** *Let  $R$  be a region in an  $n$ -vertex unweighted undirected planar graph  $G$ , and let  $h$  be a hole of  $R$ . Then the number of distinct patterns w.r.t.  $h$ ,  $|\bigcup_{u \in V^+(h)} \{p_{h, G}(u)\}|$ , is bounded by  $O(|h|^3)$ .*

**Proof.** For any  $u \in V^+(h)$  and  $v \in h$ , a shortest  $u$ -to- $v$  path in  $G$  can be decomposed into the concatenation of a shortest  $u$ -to- $v'$  path in  $G \setminus (R \setminus h)$ , for some  $v' \in h$ , and a shortest  $v'$ -to- $v$  path in  $G$ . Note that the former depends on  $u$ , but the latter does not. Hence, for every two vertices  $u, u' \in V^+(h)$ ,  $p_{h, G \setminus (R \setminus h)}(u) = p_{h, G \setminus (R \setminus h)}(u')$  implies  $p_{h, G}(u) = p_{h, G}(u')$ . Hence  $|\bigcup_{u \in V^+(h)} \{p_{h, G}(u)\}| = |\bigcup_{v \in V^+[h]} \{p_{h, G \setminus (R \setminus h)}(v)\}|$ .

The corollary now follows since  $h$  is a hole of  $R$  implies that  $h$  is a face of  $G - (R - h)$ , so by Lemma 8,  $|\bigcup_{v \in V^+[h]} \{p_{h, G - (R - h)}(v)\}| = O(|h|^3)$ . ◀

For the remainder of the paper we only deal with distances in  $G$  and with patterns in  $G$ , so we will omit the subscript  $G$ , and write  $\mathbf{d}(\cdot, \cdot)$  and  $p_h(\cdot)$  instead of  $\mathbf{d}_G(\cdot, \cdot)$  and  $p_{h, G}(\cdot)$ .

## 4 $O(n^{7/4})$ space distance oracle

Before presenting our main result, we describe a simpler construction which yields a distance oracle with a larger space requirement of  $O(n^{7/4})$  and  $O(1)$  query time.

### 4.1 Preprocessing

The preprocessing consists of computing an  $r$ -division  $\mathcal{R}$  of  $G$  with a parameter  $r$  to be determined later. For every vertex  $v$  of  $G$  and every region  $R$  of  $\mathcal{R}$ , we store the hole  $h$  of  $R$  s.t.  $v$  is in  $V^+(h)$ . This requires  $O(n \cdot n/r) = O(n^2/r)$  space.

For every region  $R \in \mathcal{R}$ , for every hole  $h$  of  $R$ , we maintain the  $O(r^3)$  patterns of the vertices in  $V^+(h)$  w.r.t.  $h$  as follows. Let  $k$  denote the size of the boundary walk  $w(h)$  of  $h$ . Let  $v_h$  be the canonical (i.e., first) vertex of  $w(h)$ . We maintain the patterns seen so far in a ternary tree  $\mathcal{A}$  whose edges are labeled by  $\{-1, 0, 1\}$ . The depth of  $\mathcal{A}$  is  $k - 1$ , and the labels along each root-to-leaf path correspond to a unique pattern, which we associate with that leaf. For every vertex  $v \in V^+(h)$ , we compute the pattern  $p_h(v)$  and we make sure that  $p_h(v)$  is represented in the tree  $\mathcal{A}$  by adding the corresponding labeled edges that are not yet present in  $\mathcal{A}$ . After all the vertices in  $V^+(h)$  are handled, the tree  $\mathcal{A}$  has  $O(r^3)$  leaves. For each leaf of  $\mathcal{A}$  with an associated pattern  $p$ , we compute and store (i) the distance from  $p$  to each vertex of  $R$ . Storing (i) requires  $O(r^4)$  time and space for all leaves of  $\mathcal{A}$ , so a total of  $O(n/r \cdot r^4) = O(nr^3)$  space for storing all this information over all regions.

For each vertex  $v \in V^+(h)$  we store (ii) a pointer to (the leaf of  $\mathcal{A}$  that is associated with) the pattern  $p_{h,G}(v)$ , as well as (iii) the distance  $\mathbf{d}(v, v_h)$  between  $v$  and the canonical vertex of  $h$ . The total space required to store all these pointers and distances is  $O(n \cdot n/r) = O(n^2/r)$ .

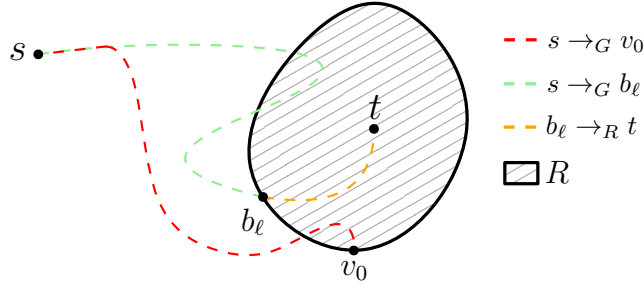
To complete the preprocessing we also store (iv) for each region  $R \in \mathcal{R}$ , the distance  $\mathbf{d}(u, v)$  for all pairs of vertices  $u, v \in R$ . This takes  $O(n/r \cdot r^2) = O(nr)$  additional space, which is dominated by the above terms.

The total space required by the oracle is thus  $O(n^2/r) + O(nr^3)$ . This is minimized for  $r = n^{1/4}$ , resulting in an  $O(n^{7/4})$ -space data structure.

We note that once this information has been computed we no longer need to store the entire tree  $\mathcal{A}$ . Rather, it suffices to store just the list of leaves of  $\mathcal{A}$  and the distances stored with each of them. In particular, we no longer need to remember what is the actual pattern associated with each leaf, we only need to have some identifier for each pattern, and the distances from this pattern to the vertices of the region  $R$ . In the current scheme this has no asymptotic effect on the size of the data structure, since each pattern is of size  $O(r)$ , and we anyway store the  $O(r)$  distances from each pattern to all vertices of  $R$ . However, in the recursive scheme in the next section this observation will become useful.

### 4.2 Handling a query

To answer a query for the distance between vertices  $s$  and  $t$  we proceed as follows. If  $s$  and  $t$  are in the same region, we simply return the distance  $\mathbf{d}(s, t)$  stored in item (iv). Otherwise, let  $R$  be the region containing  $t$ , and let  $h$  be the hole of  $R$  such that  $s \in V^+(h)$ . Let  $v_h$  be the canonical vertex of  $h$ . We return  $\mathbf{d}(s, v_h) + \mathbf{d}(p_{h,G}(s), t)$ . The correctness is immediate from Lemma 7. We note that  $\mathbf{d}(s, v_h)$  is stored in item (iii), a pointer to  $p_{h,G}(s)$  is stored in item (ii), and  $\mathbf{d}(p_{h,G}(s), t)$  is stored in item (i). The query is illustrated in Figure 2.



■ **Figure 2** Illustration of the query in Section 4. By Lemma 7 the query returns  $\mathbf{d}(s, b_0) + \mathbf{d}_R(p, t) = \mathbf{d}(s, b_\ell) + \mathbf{d}_R(b_\ell, t) = \mathbf{d}(s, t)$  where  $b_\ell$  is some boundary vertex of  $R$  on  $s \rightsquigarrow t$ .

### 4.3 Distributed Labels

This oracle can be distributed into a distance labeling scheme of size  $O(n^{3/4})$  per label. Each vertex  $v$  in a region  $R$  stores its distance from each of the patterns of  $R$  (item (i)) using  $O(r^3) = O(n^{3/4})$  space, as well as its part of items (ii) and (iii) in  $O(n/r) = O(n^{3/4})$  space. Finally,  $v$  stores its distance to each other vertex in  $R$  in  $O(r) = O(n^{1/4})$  space. Using the same query algorithm, the distance between any two vertices  $s, t$  can be computed in  $O(1)$  time given just the labels of  $s$  and  $t$ .

## 5 $O(n^{5/3+\epsilon})$ space distance oracle

A bottleneck in the above approach comes from having to store, for each pattern  $p$  of a hole  $h$  of a region  $R$ , the distances from  $p$  to all vertices of  $R$ . Instead, we use a recursive  $r$ -division, in which we store for  $p$ , only the distances to the canonical vertex of a hole  $h'$  of each child region  $R'$  of  $R$  instead of all the vertices in the region. For this information to be useful we also store the pattern induced by  $p$  on the hole  $h'$ , which is defined as follows.

► **Definition 10** (Pattern induced by a pattern). *Let  $R$  be a region in a graph  $G$ . Let  $h$  be a hole of  $R$  and  $p_h$  be a pattern of  $h$  (w.r.t. a vertex or another pattern). Let  $R'$  be a child region of  $R$ . Let  $b_0, b_1, \dots, b_k$  be the vertices of the boundary walk of a hole of  $h'$  of  $R'$ . The pattern induced by  $p_h$  on  $h'$  is the vector  $p_{h'}$  satisfying  $p_{h'}[i] = \mathbf{d}(p_h, b_i) - \mathbf{d}(p_h, b_{i-1})$  for  $1 \leq i \leq k$ .*

► **Lemma 11.** *Consider the settings of Definition 10. If  $p_h = p_h(u)$  for some  $u \in V^+(h)$ , then  $p_{h'} = p_{h'}(u)$ .*

**Proof.** By Lemma 7, for every  $0 \leq i \leq k$ ,  $\mathbf{d}(u, b_i) - \mathbf{d}(u, b_{i-1}) = \mathbf{d}(p_h, b_i) - \mathbf{d}(p_h, b_{i-1})$ . Hence for all  $1 \leq i \leq k$ ,  $p_{h'}[i] = \mathbf{d}(p_h, b_i) - \mathbf{d}(p_h, b_{i-1}) = \mathbf{d}(u, b_i) - \mathbf{d}(u, b_{i-1}) - (\mathbf{d}(u, b_{i-1}) - \mathbf{d}(u, b_i)) = \mathbf{d}(u, b_i) - \mathbf{d}(u, b_{i-1})$ , which is, by definition,  $p_{h'}(u)[i]$ . ◀

### 5.1 Preprocessing

We first compute an  $\mathbf{r} = (r_0, r_1, \dots, r_k, r_{k+1})$ -division of  $G$  for  $\mathbf{r}$  to be determined later, and denote by  $\mathcal{T}_{\mathbf{r}}$  the associated decomposition tree. For convenience, we let  $r_0 = n$ ,  $r_{k+1} = 1$  and define  $C(R) = \{R' \mid R' \text{ is a child of } R \text{ in } \mathcal{T}_{\mathbf{r}}\}$ . In the following we let  $P_h$  denote the set  $\{p_h(u) : u \in V(G)\}$ . We store the following:

1. For each  $u \in V(G)$  we store a list of regions  $R_0 \supset R_1 \supset \dots \supset R_k$  containing  $u$ , where  $R_i \in \mathcal{T}_{\mathbf{r}}^i$ . (Recall that  $\mathcal{T}_{\mathbf{r}}^i$  is the set of all nodes of  $T_{\mathbf{r}}$  at level  $i$ ).

2. For each  $u \in V(G)$ , for each  $0 \leq i \leq k-1$ , for each region  $R \in \mathcal{T}_r^i$  containing  $u$ , for each child region  $R' \subset R$  at level- $(i+1)$ , let  $h$  be the hole of  $R'$  such that  $u \in V^+(h)$ . We associate with the pair  $(u, R')$  (i) a pointer to  $p_h(u)$ , (ii) the canonical vertex  $v_h$ , and (iii) the distance  $\mathbf{d}(u, v_h)$ .
3. For each  $1 \leq i \leq k$ , for each  $R \in \mathcal{T}_r^i$ , for each hole  $h$  in  $R$ , for each  $p \in P_h$  and for each  $R' \in C(R)$ , let  $h'$  be the hole of  $R'$  such that  $v_h \in V^+(h')$ . We associate with the pair  $(p, R')$  (i) a pointer to the pattern  $p_{h'}(p)$  induced by  $p$  on  $h'$ , (ii) the canonical vertex  $v_{h'}$ , and (iii) the distance  $\mathbf{d}(p, v_{h'})$ .

## 5.2 Space analysis

Storing 1 requires space  $O(kn)$ . To bound the space for item 2, we note that the number of regions at level  $i$  to which a vertex  $u$  belongs is bounded by the degree of  $u$ . Since the average vertex degree in a planar graph is at most 6, the average number of regions at level  $i$  to which  $u$  belongs is at most 6. Each such region has  $r_i/r_{i+1}$  subregions at level- $(i+1)$ , so storing 2 requires space  $O(n \sum_{i=0}^{k-1} r_i/r_{i+1}) = O(n^2/r_1) + O(n \sum_{i=1}^{k-1} r_i/r_{i+1})$ . Storing 3 requires space  $O(\sum_{i=1}^k (n/r_i) \cdot r_i^3 \cdot r_i/r_{i+1}) = O(n \sum_{i=1}^k r_i^3/r_{i+1})$ . The total space is thus,  $O(nk + n^2/r_1 + n \sum_{i=1}^k r_i^3/r_{i+1})$ .

## 5.3 Handling a query

Algorithm 1 shows pseudocode describing the query procedure.

■ **Algorithm 1** Query procedure for the  $O(n^{5/3+\varepsilon})$  construction.

---

```

1: procedure QUERY( $s, t$ )
2:    $i \leftarrow$  the largest  $i$  s.t. the region  $R_i$  stored in item 1 for  $t$  contains both  $s$  and  $t$ 
3:    $R_t \leftarrow$  level  $(i+1)$  region stored in item 1 for  $t$ 
4:    $(p, d) \leftarrow$  the tuple associated with  $(s, R_t)$ 
5:    $i \leftarrow i+1$ 
6:   while  $i \leq k$  do
7:      $R'_t \leftarrow$  level  $(i+1)$  subregion of  $R_t$  stored in item 1 for  $t$ 
8:      $(p', d') \leftarrow$  the tuple associated with  $(p, R'_t)$ 
9:      $d \leftarrow d + d'$  ;  $p \leftarrow p'$  ;  $R_t \leftarrow R'_t$  ;  $i \leftarrow i+1$ 
10:  return  $d$ 

```

---

To process a query  $\mathbf{d}(s, t)$  the query procedure first determines the largest value  $i$  for which  $s$  and  $t$  belong to the same region in  $\mathcal{T}_r^i$ . Note that such a region must always exist as the root of  $\mathcal{T}_r$  is all of  $G$ . This level can be found in  $O(k)$  time by traversing  $\mathcal{T}_r$ , starting from a leaf region containing  $s$  and a leaf region containing  $t$ .

Let  $R_t$  be the level- $(i+1)$  region stored for  $t$  in item 1. Note that,  $t \in R_t$ , and, by choice of  $i$ ,  $s \notin R_t$ . Hence,  $s$  is in some hole  $h$  of  $R_t$ . We retrieve the pattern  $p_h(s)$  and the distance  $\mathbf{d}(s, v_h)$  associated with  $(s, R_t)$  in item 2. We then proceed iteratively, “zooming” into increasingly smaller regions containing  $t$ .

We show that the algorithm maintains the invariant that, at the beginning of each iteration, we have a level- $i$  region  $R_t$  containing  $t$ , the variable  $d$  stores  $\mathbf{d}(s, v_h)$ , where  $h$  is the hole of  $R_t$  such that  $s \in V^+(h)$ , and the variable  $p$  stores (a pointer) to the pattern  $p_h(t)$ . Thus, when we reach the singleton region containing  $t$ , the variable  $d$  stores  $\mathbf{d}(s, t)$ .

We have already established that the invariant is maintained just before the loop is entered for the first time. In each iteration of the loop we retrieve  $R'_t$ , a level- $(i+1)$  subregion or  $R_t$  containing  $t$  (available in item 1), and retrieve  $d' \leftarrow \mathbf{d}(p, v_{h'})$  and  $p' \leftarrow p_{h'}(p)$  (associated with the pair  $(p, R'_t)$  in item 3). By Lemma 7,  $d + d' = \mathbf{d}(s, v_h) + \mathbf{d}(p_h(u), v_{h'}) = \mathbf{d}(s, v_{h'})$ . By Lemma 11,  $p' = p_{h'}(t)$ . Hence, after the assignments in Line 9, the invariant is restored.

The time complexity of the query is clearly  $O(k)$ .

#### 5.4 Choosing parameters

Recall that the space requirement is  $O(nk + n^2/r + n \sum_{i=1}^k r_i^3/r_{i+1})$ . Picking each  $r_i$  s.t.  $r_i/r_{i+1} = r_1^\varepsilon$  results in  $r_k = \Theta(1)$  when  $k = \Theta(1/\varepsilon)$ , and in a query time of  $O(1/\varepsilon)$ . Choosing  $r_1 = n^{1/3+\varepsilon}$ , the total space used becomes

$$O\left(n \sum_{i=1}^k r_i^3/r_{i+1}\right) = O(nr_1^2 r_1^\varepsilon) = O(n^{1+2/3+2\varepsilon+\varepsilon/3+\varepsilon^2}) = O(n^{5/3+\varepsilon'})$$

for a suitable choice of  $\varepsilon'$ .

One can decrease the sizes of regions more aggressively to get the query time of  $k = O(\log(1/\varepsilon))$  of Theorem 1. To this end we choose  $\mathbf{r}$  such that  $r_i^3/r_{i+1} = n^{2/3+\varepsilon}$ , and  $r_1 = n^{1/3}$ . Then the space requirement is  $O(n^{5/3} + knn^{2/3+\varepsilon}) = O(kn^{5/3+\varepsilon})$ . It is not hard to verify that one gets  $r_i = O(n^{1/3-\varepsilon \frac{3^i-2^i-1}{2}})$ , so  $r_k = O(1)$  with  $k = O(\log(1/\varepsilon))$ .

As a last remark we note that the smallest interesting choice of  $\varepsilon$  in Theorem 1 is  $\Theta(1/\log n)$ , giving  $O(n^{5/3})$  space and  $O(\log \log n)$  query-time, which is a faster query-time than was previously known for this amount of space [9, 7].

---

#### References

- 1 Amir Abboud, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Near-Optimal Compression for the Planar Graph Metric. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 530–549. Society for Industrial and Applied Mathematics, Philadelphia, PA, January 2018. doi:10.1137/1.9781611975031.35.
- 2 Srinivasa Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel Smid, and Christos D. Zaroliagis. Planar Spanners and Approximate Shortest Path Queries Among Obstacles in the Plane. In Josep Diaz and Maria Serna, editors, *Algorithms – ESA '96*, pages 514–528, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- 3 Sergio Cabello. Many Distances in Planar Graphs. *Algorithmica*, 62(1-2):361–381, February 2012. doi:10.1007/s00453-010-9459-0.
- 4 Sergio Cabello. Subquadratic Algorithms for the Diameter and the Sum of Pairwise Distances in Planar Graphs. *ACM Transactions on Algorithms*, 15(2):1–38, December 2018. doi:10.1145/3218821.
- 5 Timothy M. Chan. All-Pairs Shortest Paths for Unweighted Undirected Graphs in  $o(mn)$  Time. *ACM Trans. Algorithms*, 8(4):34:1–34:17, 2012. doi:10.1145/2344422.2344424.
- 6 Timothy M. Chan and Dimitrios Skrepetos. Faster Approximate Diameter and Distance Oracles in Planar Graphs. *Algorithmica*, 81(8):3075–3098, August 2019. doi:10.1007/s00453-019-00570-z.
- 7 Panagiotis Charalampopoulos, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Almost Optimal Distance Oracles for Planar Graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 138–151, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316316.
- 8 Danny Z. Chen and Jinhui Xu. Shortest Path Queries in Planar Graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC 2000, pages 469–478, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335359.



- 9 Vincent Cohen-Addad, Soren Dahlgaard, and Christian Wulff-Nilsen. Fast and Compact Exact Distance Oracle for Planar Graphs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 962–973. IEEE, October 2017. doi:10.1109/FOCS.2017.93.
- 10 Hristo Djidjev. On-Line Algorithms for Shortest Path Problems on Planar Digraphs. In *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science*, WG 1996, pages 151–165, Berlin, Heidelberg, 1996. Springer-Verlag.
- 11 Jittat Fakcharoenphol and Satish Rao. Planar Graphs, Negative Weight Edges, Shortest Paths, and Near Linear Time. *Journal of Computer and System Sciences*, 72(5):868–889, August 2006. doi:10.1016/j.jcss.2005.05.007.
- 12 Paweł Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better Tradeoffs for Exact Distance Oracles in Planar Graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2018, pages 515–529, USA, 2018. Society for Industrial and Applied Mathematics.
- 13 Qian-Ping Gu and Gengchun Xu. Constant Query Time  $(1 + \epsilon)$ -Approximate Distance Oracle for Planar Graphs. *Theor. Comput. Sci.*, 761:78–88, 2019. doi:10.1016/j.tcs.2018.08.024.
- 14 Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2011. doi:10.1007/978-3-642-22006-7\_12.
- 15 Philip Klein. Preprocessing an Undirected Planar Network to Enable Fast Approximate Distance Queries. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 820–827, USA, 2002. Society for Industrial and Applied Mathematics.
- 16 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing - STOC '13*, page 505, New York, New York, USA, 2013. ACM Press. doi:10.1145/2488608.2488672.
- 17 Jason Li and Merav Parter. Planar Diameter via Metric Compression. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 152–163, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316358.
- 18 Yaowei Long and Seth Pettie. Planar Distance Oracles with Better Time-Space Tradeoffs. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2021, pages 2517–2537. Society for Industrial and Applied Mathematics, 2021. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9781611976465.149>.
- 19 Shay Mozes and Christian Sommer. Exact Distance Oracles for Planar Graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2012, pages 209–222, USA, 2012. Society for Industrial and Applied Mathematics.
- 20 Yahav Nussbaum. Improved Distance Queries in Planar Graphs. In *Proceedings of the 12th International Conference on Algorithms and Data Structures*, WADS 2011, pages 642–653, Berlin, Heidelberg, 2011. Springer-Verlag.
- 21 Mihai Pătraşcu and Liam Roditty. Distance Oracles beyond the Thorup–Zwick Bound. *SIAM Journal on Computing*, 43(1):300–311, January 2014. doi:10.1137/11084128X.
- 22 N Sauer. On the Density of Families of Sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, July 1972. doi:10.1016/0097-3165(72)90019-2.
- 23 Christian Sommer. Shortest-Path Queries in Static Networks. *ACM Computing Surveys*, 46(4):1–31, April 2014. doi:10.1145/2530531.
- 24 Mikkel Thorup. Compact Oracles for Reachability and Approximate Distances in Planar Digraphs. *Journal of the ACM (JACM)*, 51(6):993–1024, November 2004. doi:10.1145/1039488.1039493.



## 25:12 Subquadratic Distance Oracles for Planar Graphs

- 25 Christian Wulff-Nilsen. *Algorithms for Planar Graphs and Graphs in Metric Spaces*. PhD thesis, Department of Computer Science, University of Copenhagen, Denmark, 2010. URL: [https://di.ku.dk/english/research/phd/phd-theses/2010/thesischristianwulff.pdf\\_kopi](https://di.ku.dk/english/research/phd/phd-theses/2010/thesischristianwulff.pdf_kopi).
- 26 Christian Wulff-Nilsen. Constant Time Distance Queries in Planar Unweighted Graphs with Subquadratic Preprocessing Time. *Computational Geometry*, 46(7):831–838, October 2013. doi:10.1016/j.comgeo.2012.01.016.
- 27 Christian Wulff-Nilsen. Approximate Distance Oracles for Planar Graphs with Improved Query Time-Space Tradeoff. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 351–362, USA, 2016. Society for Industrial and Applied Mathematics.

# Interval Edge Coloring of Bipartite Graphs with Small Vertex Degrees

Anna Małafiejska ✉

Independent Researcher, Gdańsk, Poland

Michał Małafiejski<sup>1</sup> ✉

Faculty of Electronics, Telecommunications and Informatics, Gdańsk Tech, Poland

Faculty of Business and Technologies, International Black Sea University, Tbilisi, Georgia

Krzysztof M. Ocetkiewicz ✉

CI TASK, Gdańsk Tech, Poland

Krzysztof Pastuszak ✉

Faculty of Electronics, Telecommunications and Informatics, Gdańsk Tech, Poland

---

## Abstract

An edge coloring of a graph  $G$  is called *interval edge coloring* if for each  $v \in V(G)$  the set of colors on edges incident to  $v$  forms an interval of integers. A graph  $G$  is *interval colorable* if there is an interval coloring of  $G$ . For an interval colorable graph  $G$ , by the *interval chromatic index* of  $G$ , denoted by  $\chi'_i(G)$ , we mean the smallest number  $k$  such that  $G$  is interval colorable with  $k$  colors. A bipartite graph  $G$  is called  $(\alpha, \beta)$ -biregular if each vertex in one part has degree  $\alpha$  and each vertex in the other part has degree  $\beta$ . A graph  $G$  is called  $(\alpha^*, \beta^*)$ -bipartite if  $G$  is a subgraph of an  $(\alpha, \beta)$ -biregular graph and the maximum degree in one part is  $\alpha$  and the maximum degree in the other part is  $\beta$ .

In the paper we study the problem of interval edge colorings of  $(k^*, 2^*)$ -bipartite graphs, for  $k \in \{3, 4, 5\}$ , and of  $(5^*, 3^*)$ -bipartite graphs. We prove that every  $(5^*, 2^*)$ -bipartite graph admits an interval edge coloring using at most 6 colors, which can be found in  $O(n^{3/2})$  time, and we prove that an interval edge 5-coloring of a  $(5^*, 2^*)$ -bipartite graph can be found in  $O(n^{3/2})$  time, if it exists. We show that every  $(4^*, 2^*)$ -bipartite graph admits an interval edge 4-coloring, which can be found in  $O(n)$  time. The two following problems of interval edge coloring are known to be  $\mathcal{NP}$ -complete: 6-coloring of  $(6, 3)$ -biregular graphs (Asratian and Casselgren (2006)) and 5-coloring of  $(5^*, 5^*)$ -bipartite graphs (Giaro (1997)). In the paper we prove  $\mathcal{NP}$ -completeness of 5-coloring of  $(5^*, 3^*)$ -bipartite graphs.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Graph coloring

**Keywords and phrases** interval edge coloring, biregular graphs, coloring algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.26

## 1 Introduction

We use standard definitions and notations of graph theory. In the following, by a graph we mean a nonempty simple graph (i.e., without multiple edges or loops), and by a multigraph we mean a multigraph with possible multiple edges, but without loops.

Let  $G$  be a multigraph with vertex set  $V(G)$  and edge set  $E(G)$ . For some technical reasons, we assume that  $V(G) \cap \mathbb{N} = \emptyset$ . For each vertex  $v \in V(G)$ , by  $N_G(v)$  we mean the set of neighbours of  $v$  in  $G$ , and by  $E_G(v)$  we mean the set of edges incident with  $v$ . The degree of vertex  $v$  in  $G$ , denoted by  $\deg_G(v)$ , is the number  $|E_G(v)|$ . By  $n(G), m(G), \Delta(G)$  and  $\delta(G)$  we denote the number of vertices of  $G$ , the number of edges of  $G$ , the maximum and the minimum degree of a vertex of  $G$ , respectively. By *isolated vertex* we mean a vertex

---

<sup>1</sup> corresponding author



of degree 0, and by *pendant vertex* a vertex of degree 1. The set of all pendant vertices of  $G$  we denote by  $P(G)$ . By  $G[A]$ , where  $A \subset V(G)$ , we denote a subgraph of  $G$  induced by set  $A$ , and by  $G \setminus A$  we mean the graph  $G[V \setminus A]$ . We write  $H \subset G$  if and only if  $H$  is a subgraph of  $G$ , and  $H \sqsubset G$  if and only if  $H$  is an induced subgraph of  $G$ , i.e.,  $H = G[V(H)]$ .

A set of integers  $[a, b] = \{a, a + 1, \dots, b - 1, b\}$ , where  $a, b \in \mathbb{N}$  and  $a \leq b$ , is said to be an *interval* of integers. Let  $X \times Y = \{\{r, s\} : r \in X \wedge s \in Y\}$ .

A bipartite graph  $G$  is called  $(\alpha, \beta)$ -biregular if all vertices in one part of  $G$  have degree  $\alpha$  and all vertices in the other part have degree  $\beta$ . If  $G$  is a subgraph of an  $(\alpha, \beta)$ -biregular graph with the maximum degree in one part  $\alpha$  and the maximum degree in the other part  $\beta$ , it is called an  $(\alpha^*, \beta^*)$ -bipartite graph. If all vertices in the first part of an  $(\alpha^*, \beta^*)$ -bipartite graph have the same degree  $\alpha$ , then it is called an  $(\alpha, \beta^*)$ -bipartite graph. Analogously, we define  $(\alpha^*, \beta)$ -bipartite graphs. If we take the partition  $(A, B)$  of  $V(G)$  of an  $(\alpha^*, \beta^*)$ -bipartite graph  $G$ , we mean that the vertices from set  $A$  are of degree  $\alpha$  or less, and the vertices from set  $B$  are of degree  $\beta$  or less.

### 1.1 Interval coloring and interval $\chi'_i$ -coloring problems

Let  $G$  be a graph. Let  $c: E(G) \rightarrow \mathbb{N}$  be an edge coloring, i.e., a function assigning different colors to adjacent edges. By an *interval edge coloring* we mean an edge coloring  $c$  such that for each  $v \in V(G)$ , the set  $c(E_G(v))$  is an interval of integers. An interval edge coloring  $c$  such that  $c(E(G)) = \{1, 2, \dots, k\}$  is called *interval  $k$ -coloring*. We say that graph  $G$  is *interval colorable* ( *$k$ -colorable*) if there is an interval coloring ( $k$ -coloring) of  $G$ . If  $G$  is interval  $l$ -colorable for some  $l \leq k$ , then we say that  $G$  is interval  $k^*$ -colorable or there is an interval  $k^*$ -coloring of  $G$ . The problem of interval coloring of graphs is the problem of verifying if an arbitrary graph  $G$  is interval colorable. If  $G$  is interval colorable, then by *interval chromatic index* of  $G$ , denoted by  $\chi'_i(G)$ , we mean the smallest number  $k$  such that  $G$  is interval  $k$ -colorable. The problem of interval  $\chi'_i$ -coloring in the class of interval colorable graphs is to find an interval  $\chi'_i(G)$ -coloring for an arbitrary interval colorable graph  $G$ . Let  $\mathcal{A}$  be an interval edge coloring algorithm for some class  $\mathcal{C}$  of interval colorable graphs. We say that  $\mathcal{A}$  is  $k^*$ -algorithm for class  $\mathcal{C}$  if for every graph  $G \in \mathcal{C}$  it gives an interval  $k^*$ -coloring of  $G$ , and we say that  $\mathcal{A}$  is  $(\chi'_i + k)^*$ -algorithm for class  $\mathcal{C}$  if for each graph  $G \in \mathcal{C}$  it gives an interval  $(\chi'_i(G) + k)^*$ -coloring of  $G$  (i.e.,  $\mathcal{A}$  is an additive  $k$ -approximation algorithm for the interval  $\chi'_i$ -coloring problem).

The problem of finding school timetables without idle times for both teachers and classes, which may be modelled by edge colorings of bipartite graphs, probably motivated Asratian and Kamalian to introduce in [2, 3] the concept of interval edge coloring of graphs. The open shop scheduling models with unit time operations, *no wait&idle* criterion, and some special bipartite scheduling graphs were considered in [10, 9], where the authors studied schedules with the minimum makespan which corresponds to the interval  $\chi'_i$ -coloring problem of bipartite scheduling graphs.

In general, not every graph has an interval coloring, since interval colorable graphs are  $\Delta$ -colorable [2]. Moreover, the problem of determining whether a bipartite graph has an interval coloring turned out to be  $\mathcal{NP}$ -complete [19], and the smallest known maximum degree of a bipartite graph without an interval coloring is 11 [16].

Interval colorable graphs, which are known to be interval  $\chi'_i$ -colored in a polynomial time, are regular bipartite graphs (by König theorem) (in  $O(n\Delta \log \Delta)$  time with  $\chi'_i(G) = \Delta(G)$  colors, for a regular graph  $G$ ) [5], trees [14, 15, 11, 3] (in  $O(n)$  time with  $\chi'_i(T) = \Delta(T)$  colors, for a tree  $T$ ) and complete bipartite graphs (in  $O(m)$  time with  $\chi'_i(K_{a,b}) = a + b - \gcd(a, b)$  colors, for a complete bipartite graph  $K_{a,b}$ ) [14, 15, 11, 12]. In [8] the authors constructed an  $O(n)$  time algorithm for interval  $\chi'_i$ -coloring of a grid  $G$  with  $\chi'_i(G) = \Delta(G)$  colors.

In [9] the authors proposed the linear time algorithm for interval coloring of any outerplanar bipartite graph, but the complexity of the interval  $\chi'_i$ -coloring problem of outerplanar bipartite graphs seems to be open. In [10] the authors proposed the linear time algorithm giving an interval coloring of bipartite cacti graph  $G$  with  $\Delta(G) + 1$  colors, which is an  $(\chi'_i + 1)^*$ -algorithm.

In [11] the author proved that any  $(3^*, 3^*)$ -bipartite graph has an interval 4-coloring, which can be constructed in  $O(n)$  time. In [7] the author proved that an interval  $\alpha$ -coloring of  $(\alpha^*, \alpha^*)$ -bipartite graph can be found in  $O(n^{3/2})$  time (if it exists), for  $\alpha \in \{3, 4\}$ .

In [12] the authors proved that if an  $(\alpha, \beta)$ -biregular graph has an interval  $k$ -coloring, then  $k \geq \alpha + \beta - \gcd(\alpha, \beta)$ . In [11] the author proved that every  $(2\alpha, 2)$ -biregular graph admits an interval  $2\alpha$ -coloring (i.e.,  $\chi'_i$ -coloring), for each  $\alpha \geq 1$ , and this construction can be done in  $O(n\Delta \log \Delta)$  time. In [13] the authors proved that every  $(2\alpha + 1, 2)$ -biregular graph admits an interval  $(2\alpha + 2)$ -coloring (i.e.,  $\chi'_i$ -coloring), for every  $\alpha \geq 1$ , and to the best of our knowledge this construction can be done in  $O(n^{3/2}\Delta^2)$  time.

The following problems of interval  $\chi'_i$ -coloring are  $\mathcal{NP}$ -complete: 6-coloring of  $(6, 3)$ -biregular graphs [1] and 5-coloring of  $(5^*, 5^*)$ -bipartite graphs [7].

In [4] the authors proved that every  $(6, 3)$ -biregular graph admits an interval 7-coloring. The problems of interval coloring of  $(4, 3)$ -biregular and  $(5, 3)$ -biregular graphs are still open.

In the paper we solve the interval  $\chi'_i$ -coloring problem for  $(\alpha^*, 2^*)$ -bipartite graphs, for  $\alpha \in \{3, 4, 5\}$ . We show that if  $G$  is a  $(5^*, 2^*)$ -bipartite graph, then  $\chi'_i(G) \leq 6$ , and the interval  $\chi'_i$ -coloring problem for  $(5^*, 2^*)$ -bipartite graphs can be solved in  $O(n^{3/2})$  time. If  $G$  is a  $(4^*, 2^*)$ -bipartite graph, then  $\chi'_i(G) \leq 4$ , and the interval  $\chi'_i$ -coloring problem for  $(4^*, 2^*)$ -bipartite graphs can be solved in  $O(n)$  time. In section 3 we prove  $\mathcal{NP}$ -completeness of the interval 5-coloring of  $(5^*, 3^*)$ -bipartite graphs.

## 1.2 General and interval factor problem

We introduce the *general factor problem* [17, 6] as follows: let  $G$  be a graph and let  $\mathcal{F}: V(G) \rightarrow 2^{\mathbb{N}} \setminus \{\emptyset\}$ , where  $\mathcal{F}(v) \subset \{0, \dots, \deg_G(v)\}$ . Does  $G$  admit an  $\mathcal{F}$ -factor, i.e., a set  $F \subset E(G)$  such that for each vertex  $v \in V(G)$ ,  $|\{e: v \in e \wedge e \in F\}| \in \mathcal{F}(v)$ ? Lóvasz [17] proved that the general factor problem is  $\mathcal{NP}$ -complete for general graphs and mappings taking values  $\{0, 3\}$  for some vertices. In [6] Cornuéjols showed that the general factor problem is  $\mathcal{NP}$ -complete for planar bipartite graphs and mappings taking values  $\{0, 3\}$  for some vertices. A finite set  $A \subset \mathbb{N}$  is said to have a *gap of length*  $k \geq 1$  if  $a, a + k + 1 \in A$  and  $a + 1, \dots, a + k \notin A$ , for some  $a$ . A finite set  $A \subset \mathbb{N}$  has no gaps if and only if it is an interval. Thus, the general factor problem is  $\mathcal{NP}$ -complete for planar bipartite graphs and mappings taking values, i.e., sets, having gaps of length of at least two. In [6] the author proved the conjecture of Lóvasz [17]: there is a polynomial time algorithm for deciding whether a graph  $G$  has an  $\mathcal{F}$ -factor, where the sets  $\mathcal{F}(v)$  have no gap of length of two or more. The complexity of the proposed algorithm is  $O(n^4)$  [6].

If for each  $v \in V(G)$ , set  $\mathcal{F}(v)$  is an interval, then  $\mathcal{F}$ -factor is called an *interval factor*. A special case of interval factors is  $\mathcal{F}$ -factor, where  $\mathcal{F} \equiv \{k\}$ ,  $k \in \mathbb{N}$ , which we denote by  $k$ -factor, e.g., perfect matching is 1-factor.

Let us assume that there is an  $O(\phi(m, n))$  time algorithm ( $\phi(m, n) = \Omega(m + n)$  and  $\phi(m, n) = O(mn^{1/2})$  [18]) solving perfect matching problem in the class of connected bipartite graphs with at most  $m$  edges and  $n$  vertices. Combining the idea of replacing an edge with a complete bipartite graph [6] and the idea of doubling a graph [7] we prove the following theorem.

► **Theorem 1.** *There is an  $O(\phi(m\Delta, m))$  time algorithm solving the interval factor problem in the class of connected bipartite graphs with at most  $m$  edges and the degree bounded by  $\Delta$ .*

**Proof.** Let  $G$  be a connected bipartite graph and let  $\mathcal{F}: V(G) \rightarrow 2^{\mathbb{N}} \setminus \{\emptyset\}$ , such that  $\mathcal{F}(v) = [a_v, b_v]$ ,  $0 \leq a_v \leq b_v \leq \deg_G(v)$ . We construct a bipartite graph  $H$  with  $\Delta(H) \leq \Delta(G) + 1$ ,  $n(H) \leq 8m(G)$  and  $m(H) \leq 4m(G)(\Delta(G) + 1) + n(G)\Delta(G)$  such that there is 1-factor in  $H$  if and only if there is  $\mathcal{F}$ -factor in  $G$ .

For each vertex  $v \in V(G)$ , we denote  $d_v = \deg_G(v)$ ,  $p_v = b_v - a_v$  and  $q_v = d_v - a_v$ . Obviously,  $b_v \leq d_v$  and  $p_v \leq q_v$ . In the first step we construct graph  $H_1$  from graph  $G$ , by replacing each vertex  $v \in V(G)$  with the complete bipartite graph isomorphic to  $K_{d_v, q_v}$  with parts  $A_v^1 = \{v_u^1: u \in N_G(v)\}$  and  $B_v^1 = \{v_1^1, \dots, v_{q_v}^1\}$ , and by replacing each edge  $\{u, v\} \in E(G)$  with an edge  $\{u_v^1, v_u^1\}$ . In the second step, we take the resultant graph  $H_1$  and its isomorphic copy  $H_2$ , where  $v_*^1$  and  $v_*^2$  are corresponding vertices (under isomorphism), and, for each  $v \in V(G)$ , we add edges  $\{v_1^1, v_1^2\}, \dots, \{v_{p_v}^1, v_{p_v}^2\}$ .

Let  $V(H) = V(H_1) \cup V(H_2)$  and  $E(H) = E(H_1) \cup E(H_2) \cup E^*$ , where  $V(H_i) = \bigcup_{v \in V(G)} A_v^i \cup B_v^i$ ,  $E(H_i) = \bigcup_{v \in V(G)} A_v^i \times B_v^i \cup F^i$ , and  $F^i = \bigcup_{\{u, v\} \in E(G)} \{\{u_v^i, v_u^i\}\}$ , for  $i \in \{1, 2\}$ , and  $E^* = \bigcup_{v \in V(G)} \bigcup_{j \in \{1, \dots, p_v\}} \{\{v_j^1, v_j^2\}\}$ .

We prove that there is an  $\mathcal{F}$ -factor in  $G$  iff there is a 1-factor in  $H$ .

( $\Rightarrow$ ) Let  $F$  be  $\mathcal{F}$ -factor in  $G$ .

Let us define  $F_v = \{u \in N_G(v): \{v, u\} \in F\}$ ,  $\hat{F}_v = N_G(v) \setminus F_v$  and  $f_v = |F_v|$ , for each  $v \in V(G)$ . Since  $F$  is  $\mathcal{F}$ -factor, then for each  $v \in V(G)$ ,  $a_v \leq f_v \leq b_v$ . Hence,  $d_v - f_v \leq q_v$  and  $0 \leq f_v - a_v \leq p_v$ . Let  $v \in V(G)$ . Let  $F_v^i = \{v_u^i \in V(H_i): u \in F_v\}$  and  $\hat{F}_v^i = A_v^i \setminus F_v^i$ , for  $i \in \{1, 2\}$ . Observe that  $|\hat{F}_v^i| = d_v - f_v$ , and  $v_u^i \in F_v^i \iff u_v^i \in F_u^i$ . Since  $H_i[A_v^i \cup B_v^i] \simeq K_{d_v, q_v}$ , there is a 1-factor in  $H_i[\hat{F}_v^i \cup \{v_{f_v - a_v + 1}^i, \dots, v_{d_v - a_v}^i\}]$ , denote it by  $\hat{E}_v^i$ . Thus,  $Q = \bigcup_{\{v, u\} \in F} \{\{v_u^1, u_v^1\}, \{v_u^2, u_v^2\}\} \cup \bigcup_{v \in V(G)} \hat{E}_v^1 \cup \hat{E}_v^2 \cup E_v^{12}$ , where  $E_v^{12} = \{\{v_1^1, v_1^2\}, \dots, \{v_{f_v - a_v}^1, v_{f_v - a_v}^2\}\}$ , is a 1-factor in  $H$ . This construction can be done in  $O(m(H))$  time.

From definition of  $H$ ,  $\Delta(H) \leq \Delta(G) + 1$ ,  $n(H) = 2n(H_1) \leq 2 \sum_{v \in V(G)} 2d_v = 8m(G)$ , and  $m(H) \leq 4m(G) + \sum_{v \in V(G)} (2d_v(d_v - a_v) + b_v - a_v) \leq 4m(G) + \sum_{v \in V(G)} (2d_v^2 + d_v) \leq 4m(G) + \Delta(G) \sum_{v \in V(G)} (2d_v + 1) \leq 4m(G)(\Delta(G) + 1) + n(G)\Delta(G)$ . Since  $G$  is bipartite,  $H_1$  and  $H_2$  are bipartite, and hence  $H$  is bipartite. Since there is an  $O(\phi(m(H), n(H)))$  time algorithm finding 1-factor in  $H$ , we get the thesis.

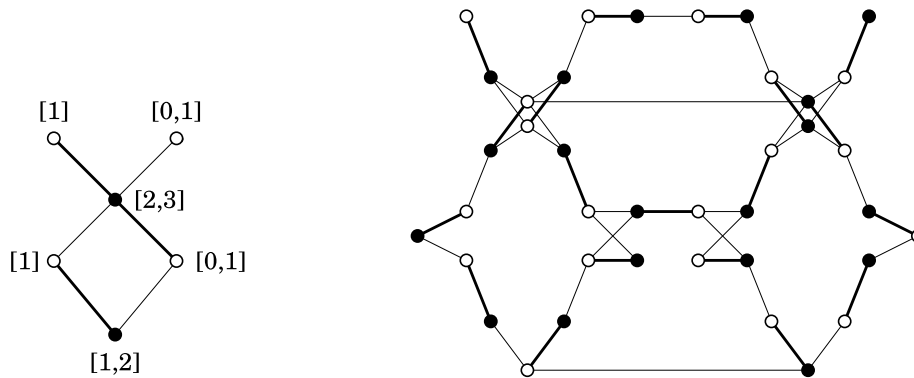
( $\Leftarrow$ ) Let  $Q$  be a 1-factor in  $H$ .

Let  $v \in V(G)$  and  $i \in \{1, 2\}$ . Since  $|\{\{v_1^1, v_1^2\}, \dots, \{v_{p_v}^1, v_{p_v}^2\}\} \cap Q| \leq p_v$ ,  $q_v - p_v \leq |(A_v^i \times B_v^i) \cap Q| \leq q_v$ . Hence,  $a_v \leq |\{u \in N_G(v): \{v_u^i, u_v^i\} \in Q\}| \leq b_v$ . Thus,  $F = \{\{v, u\} \in E(G): \{v_u^1, u_v^1\} \in Q\}$  is an  $\mathcal{F}$ -factor in  $G$ . This construction can be done in  $O(m(H))$  time. ◀

In Fig. 1 there is a graph  $G$  with an interval factor and in Fig. 2 the constructed graph  $H$  with a 1-factor corresponding to the interval factor of  $G$ . White and black vertices form the partition of a bipartite graph.

Since the problem of 1-factor in the class of bipartite graphs with bounded degrees can be solved in  $O(n^{3/2})$  time [18], we have the following corollary.

► **Corollary 2.** *There is an  $O(n^{3/2})$  time algorithm solving the interval factor problem in the class of bipartite graphs with bounded degree.*



■ **Figure 1** A bipartite graph  $G$  with an interval factor (bold edges). ■ **Figure 2** A bipartite graph  $H$  with a 1-factor (bold edges) corresponding to the interval factor of  $G$ .

## 2 Interval $\chi'_i$ -coloring problem of $(\alpha^*, 2^*)$ -bipartite graphs

In this section we construct polynomial time algorithms for the interval  $\chi'_i$ -coloring problem for  $(\alpha^*, 2^*)$ -bipartite graphs, for  $\alpha \in \{3, 4, 5\}$ , and give some other minor results.

### 2.1 Introductory properties

► **Observation 3.** *Let  $G$  be an interval colorable graph, and let  $H$  be an induced subgraph of  $G$  such that for each  $v \in V(H)$ ,  $\deg_H(v) = \deg_G(v)$  or  $\deg_H(v) = 1$ . Then, for any interval edge coloring  $c$  of  $G$ , the coloring  $c' = c|_{E(H)}$  is an interval edge coloring of  $H$ .*

Let  $G$  be an  $(\alpha^*, 2^*)$ -bipartite graph with partition  $(X, Y)$  of  $V(G)$  and let  $P_2(G) = P(G) \cap Y$ . Let  $G'$  be the graph obtained from  $G$  by adding and joining the unique vertex  $x_v$  to each pendant vertex  $v \in Y$ . Formally,  $V(G') = V(G) \cup \{x_v : v \in P_2(G)\}$  and  $E(G') = E(G) \cup \{\{x_v, v\} : v \in P_2(G)\}$ . In the following, this operation (transformation) we denote by  $G \rightarrow_p G'$ . Since the extension of an interval  $k$ -coloring of  $G$  to an interval  $k$ -coloring of  $G'$  is trivial, by Observation 3 we get the properties.

► **Proposition 4.** *Let  $\alpha \in \mathbb{N}, \alpha \geq 1$ . Let  $G$  be an interval colorable  $(\alpha^*, 2^*)$ -bipartite graph and let  $G'$  be the graph obtained by the operation  $G \rightarrow_p G'$ . Then,  $G'$  is an interval colorable  $(\alpha^*, 2)$ -bipartite graph and  $\chi'_i(G) = \chi'_i(G')$ .*

Observe that for each  $(\alpha^*, 2^*)$ -bipartite graph  $G$  we have  $m(G) \leq 2n(G)$ .

► **Proposition 5.** *Let  $k, \alpha \in \mathbb{N}, \alpha \geq 1$ . If there is an  $O(\phi(n))$  time  $k^*$ -algorithm for  $(\alpha^*, 2)$ -bipartite graphs with at most  $n$  vertices, then there is an  $O(\phi(n))$  time  $k^*$ -algorithm for  $(\alpha^*, 2^*)$ -bipartite graphs with at most  $n$  vertices.*

Let  $G$  be an  $(\alpha, 2^*)$ -bipartite graph with the partition  $(X, Y)$  and let  $G'$  be an isomorphic copy of  $G$ . Let us denote by  $v'$  the image of  $v$  under isomorphism. Let  $P_2(G) = P(G) \cap Y$ ,  $q = |P_2(G)|$ , and for each  $v \in P_2(G)$ , let  $p_G(v)$  be the only one neighbour of  $v$  in  $G$ . Let  $W = \{w_v : v \in P_2(G)\}$ , such that  $W \cap (V(G) \cup V(G')) = \emptyset$  and  $|W| = q$ . Let  $H$  be a graph defined as follows:  $V(H) = (V(G) \cup V(G')) \setminus (P_2(G) \cup P_2(G')) \cup W$ , and  $E(H) = E(G \setminus P_2(G)) \cup E(G' \setminus P_2(G')) \cup \bigcup_{v \in P_2(G)} \{\{w_v, p_G(v)\}, \{w_v, p_{G'}(v')\}\}$ . In the following, this operation (transformation) is denoted by  $G \rightarrow_d H$ . Note that  $G \simeq G_W = H[(V(G) \setminus P_2(G)) \cup W]$ , hence  $G_W \simeq G'_W = H[(V(G') \setminus P_2(G')) \cup W]$ . Thus, if  $G_W$  is

an interval colorable graph, then for any interval  $k$ -coloring  $c$  of  $G_W$ , we can extend the coloring  $c$  to the coloring of the whole graph  $H$  by defining a coloring  $c'$  of  $G'_W$  as follows:  $c'(e') = c(e) + 1$ , where  $e$  and  $e'$  are isomorphic edges. Thus, by Observation 3 we get the following properties.

► **Proposition 6.** *Let  $\alpha \in \mathbb{N}, \alpha \geq 1$ . Let  $G$  be an interval colorable  $(\alpha, 2^*)$ -bipartite graph and let  $H$  be the graph obtained by the operation  $G \rightarrow_d H$ . Thus,  $H$  is an interval colorable  $(\alpha, 2)$ -biregular graph and  $\chi'_i(G) \leq \chi'_i(H) \leq \chi'_i(G) + 1$ .*

► **Proposition 7.** *Let  $k, \alpha \in \mathbb{N}, \alpha \geq 1$ . If there is an  $O(\phi(n))$  time  $k^*$ -algorithm for  $(\alpha, 2)$ -biregular graphs with at most  $n$  vertices, then there is an  $O(\phi(n))$  time  $k^*$ -algorithm for  $(\alpha, 2^*)$ -bipartite graphs with at most  $n$  vertices.*

Let us recall that  $\chi'_i(G) = 2\alpha$  if  $G$  is an  $(2\alpha, 2)$ -biregular graph [11], and  $\chi'_i(G) = 2\alpha + 2$  if  $G$  is an  $(2\alpha + 1, 2)$ -biregular graph [13].

► **Corollary 8.** *Let  $\alpha \in \mathbb{N}, \alpha \geq 1$ . If there is an  $O(\phi(n))$  time  $\chi'_i$ -algorithm for  $(2\alpha, 2)$ -biregular graphs with at most  $n$  vertices, then there is an  $O(\phi(n))$  time  $\chi'_i$ -algorithm for  $(2\alpha, 2^*)$ -bipartite graphs with at most  $n$  vertices, and for every  $(2\alpha, 2^*)$ -bipartite graph  $G$ ,  $\chi'_i(G) = 2\alpha$ .*

► **Corollary 9.** *Let  $\alpha \in \mathbb{N}$ . If there is an  $O(\phi(n))$  time  $\chi'_i$ -algorithm for  $(2\alpha + 1, 2)$ -biregular graphs with at most  $n$  vertices, then there is an  $O(\phi(n))$  time  $(\chi'_i + 1)^*$ -algorithm for  $(2\alpha + 1, 2^*)$ -bipartite graphs with at most  $n$  vertices, and for every  $(2\alpha + 1, 2^*)$ -bipartite graph  $G$ ,  $\chi'_i(G) \leq 2\alpha + 2$ .*

## 2.2 Operations on multigraphs and pom-graphs

Let  $H$  be a multigraph. Since  $H$  may have multiple edges incident with the same two vertices, we introduce the notation  $e_i(x, y)$  or  $e(x, y, i)$ , where  $i$  is an identifier, to distinguish two or more edges incident with  $x$  and  $y$ , e.g.,  $e_1(x, y)$  and  $e_2(x, y)$ . Let  $u, v \in V(H)$  such that there is no edge in  $E(H)$  incident with  $u$  and  $v$ . We say that a multigraph  $H'$  is obtained from  $H$  by *contracting* vertices  $v$  and  $u$  if vertices  $v, u$  are replaced by a new vertex  $w(u, v)$  and each edge  $e_i(x, t) \in E(H)$ , where  $x \in \{u, v\}, t \in V(H)$ , is replaced with  $e_i(w(u, v), t)$  (we say further that  $e_i(x, t)$  and  $e_i(w(u, v), t)$  are corresponding edges). Formally,  $H' = ((V(H) \setminus \{v, u\}) \cup \{w(u, v)\}, E(H \setminus \{u, v\}) \cup \{e_i(w(u, v), t) : x \in \{u, v\} \wedge t \in V(H) \wedge e_i(x, t) \in E(H)\})$ .

In the following by a multidigraph we mean a multidigraph without loops. Let  $D$  be a multidigraph. If  $a = (x, y)$  is an arc, then  $y$  is said to be the *head* and  $x$  the *tail* of the arc  $a$ . Since  $D$  may have multiple arcs with the same head and the same tail, we introduce the notation  $a_i(x, y)$  or  $a(x, y, i)$ , to distinguish two or more arcs with the same head and tail, e.g.,  $a_1(x, y)$  and  $a_2(x, y)$ . By  $\text{indeg}_D(v)$  we mean the number of arcs with the head at  $v$ , and by  $\text{outdeg}_D(v)$  we mean the number of arcs with the tail at  $v$  in  $D$ . We say that  $v \in V(D)$  is a pendant vertex in  $D$  if and only if  $\text{indeg}_D(v) + \text{outdeg}_D(v) = 1$ .

Let  $G$  be an  $(\alpha^*, 2)$ -bipartite graph with partition  $(X, Y)$  of  $V(G)$ ,  $\alpha \in \mathbb{N}, \alpha \geq 1$ . Let  $H$  be the multigraph obtained from  $G$  by replacing each two edges  $\{u, v\}$  and  $\{w, v\}$ , where  $v \in Y$ , with one new edge  $e_v(u, w)$  joining  $u$  and  $w$  (we allow multiple edges between  $u$  and  $w$ ). Formally,  $V(H) = X$ , and  $E(H) = \{e_v(u, w) : v \in Y\}$ . In the following, the multigraph  $H$  is said to be the *contraction multigraph* of  $G$ , which we denote by  $G \rightarrow_{cn} H$ . Let  $c$  be an interval  $k$ -coloring of  $G$ . We replace each edge  $e_v(u, w) \in E(H)$  with the arc  $a_v^c$ , where  $a_v^c$  has the tail at  $u$ , if  $c(\{u, v\})$  is an odd number, otherwise  $a_v^c$  has the tail at  $w$ . Since



$c$  is an interval edge coloring and  $\deg_G(v) = 2$ , then only one of  $c(\{u, v\})$  and  $c(\{w, v\})$  is an odd number. Formally,  $D_c(G)$  is the directed multigraph (multidigraph) with vertex set  $V(D_c(G)) = V(H)$  and arc set  $A(D_c(G)) = \{a_v^c : v \in Y\}$ .

By a *partially oriented multigraph* or a *pom-graph*  $P = (V, E \cup A)$  we mean the union of a multigraph  $G = (V, E)$  and a directed multigraph  $D = (V, A)$  on the same vertex set  $V$ , which we denote by  $P = G \cup D$ . In the following, by  $\text{Gr}(P)$  we mean the multigraph  $G$ , by  $\text{Di}(P)$  we mean the multidigraph  $D$ , by  $E(P)$  we mean  $E(G)$ , and by  $A(P)$  we mean  $A(D)$ . The underlying multigraph of a *pom-graph*  $P$ , denoted by  $\text{Un}(P)$ , is the multigraph obtained from  $P$  by replacing each arc  $a_i(x, y) \in A(P)$  with a new edge  $e_i(x, y)$ . Formally,  $\text{Un}(P) = (V(P), E(P) \cup E_A(P))$ , where  $E_A(P) = \{e_i(x, y) : a_i(x, y) \in A(P)\}$ . For each  $e \in E(\text{Un}(P))$ , by  $o(e)$  we mean  $e$ , if  $e \in E(P)$ , or  $a_i(x, y)$ , if  $e \in E_A(P)$  and  $e = e_i(x, y)$ . Let  $H = \text{Un}(P)$  and let  $v \in V(P)$ . By  $EA_P(v)$  we mean  $\{o(e) : e \in E_H(v)\}$ . By  $\deg_P(v)$  we mean  $\deg_H(v)$ , and hence  $\Delta(P) = \Delta(H)$ . Let  $B \subset V(P)$ , by  $P[B]$  we denote a *pom-graph*  $G[B] \cup D[B]$ , and by  $P \setminus B$  we mean the *pom-graph*  $P[V \setminus B]$ . We say that two vertices  $u, v \in V(P)$  are *neighbours* in  $P$  if and only if  $E(\text{Un}(P[\{u, v\}]))$  is a non-empty set.

Let  $P$  be a *pom-graph* and let  $a \leq b \leq \Delta(P)$ ,  $a, b \in \mathbb{N}$ . By  $V_{a,b}(P)$  we denote the set  $\{v \in V(P) : \deg_P(v) \in [a, b]\}$  and by  $G_{a,b}(P)$  we mean  $\text{Gr}(P[V_{a,b}(P)])$ . If  $a = b$ , then we write  $G_a(P)$  instead of  $G_{a,a}(P)$ .

Let  $P'$  be the multigraph obtained from *pom-graph*  $P$  by adding new vertices on each edge and each arc. If we add vertex  $w_e$  on an edge  $e = e_i(x, y)$ , then we add two new edges  $e_i(w_e, x)$  and  $e_i(w_e, y)$ . If we add vertex  $w_a$  on an arc  $a = a_i(x, y)$ , then we add an edge  $e_i(x, w_a)$  and an arc  $a_i(w_a, y)$ . Formally,  $V(P') = V(P) \cup \{w_e : e \in E(P)\} \cup \{w_a : a \in A(P)\}$ ,  $E(P') = \{e_i(w_e, x), e_i(w_e, y) : e \in E(P) \wedge e = e_i(x, y)\} \cup \{e_i(x, w_a) : a \in A(P) \wedge a = a_i(x, y)\}$ ,  $A(P') = \{a_i(w_a, y) : a \in A(P) \wedge a = a_i(x, y)\}$ . In the following, the *pom-graph*  $P'$  is said to be the *subdivision pom-graph* of *pom-graph*  $P$ , which we denote by  $P \rightarrow_{sd} P'$ .

### 2.3 Interval $\chi'_i$ -coloring problem of $(\alpha^*, 2^*)$ -bipartite graphs for $\alpha \in \{3, 4\}$

► **Theorem 10.** *Let  $G$  be a  $(3^*, 2^*)$ -bipartite graph. Then,  $\chi'_i(G) = 3$  if and only if each connected component of  $G$  contains at most one cycle. The construction of interval 3-coloring can be done in linear time.*

**Proof.** Let  $G$  be a  $(3^*, 2^*)$ -bipartite graph with partition  $(X, Y)$  of  $V(G)$ , and let  $P_2(G) = P(G) \cap Y$ . By definition  $\Delta(G) = 3$ .

( $\Rightarrow$ ) Let us assume that  $\chi'_i(G) = 3$ , and let  $G$  contain at least one cycle.

Let  $G'$  be the graph obtained from  $G$  by the operation  $G \rightarrow_p G'$ . By Proposition 4,  $G'$  is  $(3^*, 2)$ -bipartite graph and  $G'$  is interval 3-colorable if and only if  $G$  is interval 3-colorable.

Let  $c$  be an interval 3-coloring of  $G'$ , and let  $D = D_c(G')$ . Obviously,  $\text{indeg}_D(v) \leq 1$ , for each  $v \in V(D)$ . Let  $D'$  be a digraph obtained from  $D$  by successively removing pendant vertices. Hence, for each  $v \in V(D')$ ,  $2 \leq \text{indeg}_{D'}(v) + \text{outdeg}_{D'}(v) \leq 3$ , and  $\text{indeg}_{D'}(v) \leq 1$ . Thus,  $\text{outdeg}_{D'}(v) \geq 1$ . Since  $\sum_{v \in V(D')} \text{indeg}_{D'}(v) = \sum_{v \in V(D')} \text{outdeg}_{D'}(v)$ , each component of  $D'$  is a directed cycle. Thus, each component of  $G$  contains at most one cycle.

( $\Leftarrow$ ) Let us assume that each connected component of  $G$  has at most one cycle. First, we color each cycle with colors 1 and 2, alternately. Next, for each vertex  $v$  of degree 3 that belongs to a colored cycle, color edge  $\{v, u\}$  with 3, where  $u$  does not belong to the colored cycle. In the last step, color the remaining trees in a greedy way using 3 colors, preserving intervals at vertices. Thus, we get an interval 3-coloring of  $G$  in linear time. ◀

► **Theorem 11.** Let  $G$  be a  $(4^*, 2)$ -bipartite graph. Then,  $\chi'_i(G) = 4$ . The construction of an interval 4-coloring of  $G$  can be done in linear time.

**Proof.** Let  $G$  be a  $(4^*, 2)$ -bipartite graph. The construction proceeds in two crucial stages: first, we construct a *pom*-graph  $P$  from graph  $G$ , then we use the structure of  $P$  to build the interval 4-coloring of graph  $G$ . In the first stage we apply to  $G$  the sequence of transformations  $G \rightarrow_{cn} P_0 \dashrightarrow_1 P_1 \dashrightarrow_2 P_2 \rightarrow_{sd} P$ , where  $P_0$  is the contraction multigraph of  $G$ ,  $P_1$  and  $P_2$  are some *pom*-graphs, and  $P$  is the subdivision *pom*-graph of  $P_2$ . In the second stage we start from an edge&arc 4-coloring of  $P$ , preserve this coloring on the underlying graph  $G^* = \text{Un}(P)$ , and in the final step we transform the edge colored graph  $G^*$  to an interval edge colored initial graph  $G$ , by contracting vertices that come from vertices splitted in the transformations  $\dashrightarrow_1$  or  $\dashrightarrow_2$ .

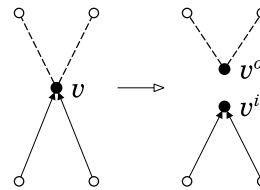
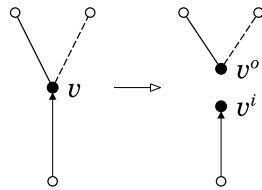
$(P_0 \dashrightarrow_1 P_1)$  Initially, let  $H' = P_0, D' = (V(P_0), \emptyset)$ , and let  $P' = H' \cup D'$  be a *pom*-graph. We proceed with the following successive steps in a loop until there is no cycle in  $G_{3,4}(P')$ .

1. (*find a cycle*) Let  $C$  be a subgraph of  $G_{3,4}(P')$ , which is a cycle. Let  $V(C) = \{v_1, \dots, v_k\}$  and let  $E(C) = \{e(v_1, v_2, i_1), \dots, e(v_{k-1}, v_k, i_{k-1}), e(v_k, v_1, i_k)\}$ . Note that  $C$  may have two vertices.
2. (*orient the cycle*) Remove  $E(C)$  from  $E(P')$  and add  $k$  arcs  $a(v_1, v_2, i_1), \dots, a(v_{k-1}, v_k, i_{k-1}), a(v_k, v_1, i_k)$  to  $A(P')$ .
3. (*split vertices of the cycle*) For each  $v \in V(C)$  such that  $\deg_{P'}(v) = 3$ , or  $\deg_{P'}(v) = 4$  and  $\text{indeg}_{D'}(v) = 2$ , *split*  $v$  into vertices  $v^i$  and  $v^o$  as shown in Fig. 3 and 4. Note that in this case each dashed line is an arc with the tail at  $v$  or  $v^o$ . Formally, add new vertices  $v^i$  and  $v^o$  to  $V(P')$ , for each arc  $a_d(u, v)$  add new arc  $a_d(u, v^i)$ , for each arc  $a_d(v, x)$  add new arc  $a_d(v^o, x)$ , and for each edge  $e_d(v, x)$  add new edge  $e_d(v^o, x)$ , and remove  $v$  from  $V(P')$ .

Knowing that there is no cycle in  $G_{3,4}(P')$ , let  $P_1 = P'$ .

▷ **Claim 12.** For each  $v \in V(P_1)$ ,

- if  $\deg_{P_1}(v) = 1$ , then  $v$  is incident with an edge or an arc with the head at  $v$ ,
- if  $\deg_{P_1}(v) = 2$ , then  $v$  is incident with two edges or an edge and an arc with the tail at  $v$ , or two arcs with tails at  $v$ , or two arcs with heads at  $v$ ,
- if  $\deg_{P_1}(v) = 3$ , then  $v$  is incident with three edges,
- if  $\deg_{P_1}(v) = 4$ , then  $v$  is incident with four edges or two edges and two arcs, one with the tail at  $v$  and one with the head at  $v$ .



■ **Figure 3** Splitting vertex  $v$  of  $\deg_{P'}(v) = 3$ . ■ **Figure 4** Splitting vertex  $v$  of  $\deg_{P'}(v) = 4$ .

$(P_1 \dashrightarrow_2 P_2)$  Let  $P_1 = H_1 \cup D_1$ , where  $H_1$  is a multigraph and  $D_1$  is a multidigraph. Initially, let  $P' = P_1, H' = H_1$  and  $D' = D_1$ . We proceed with the following successive steps in a loop until there is no path in  $G_{3,4}(P')$ . Note that if  $V(G_{3,4}(P')) \neq \emptyset$ , then there is a path in  $G_{3,4}(P')$  with at least one vertex, and by Claim 12,  $\deg_{H'}(v) \geq 2$ , for each  $v \in V(G_{3,4}(P'))$ .

1. (*find a maximal path*) Let  $T$  be a subgraph of  $G_{3,4}(P')$ , which is a maximal path. Let  $V(T) = \{v_1, \dots, v_k\}$  and let  $E(T) = \{e(v_1, v_2, i_1), \dots, e(v_{k-1}, v_k, i_{k-1})\}$ . Note that  $T$  may have one vertex.
2. (*orient the path*) Since  $T$  is a maximal path in  $G_{3,4}(P')$  and  $\deg_{H'}(v) \geq 2$ , for each  $v \in V(G_{3,4}(P'))$ , where  $H' = \text{Gr}(P')$ , there is a vertex  $v \in V(P')$  with  $\deg_{P'}(v) \leq 2$  such that  $e_d(v, v_1) \in E(P')$ . Remove  $E(T) \cup \{e_d(v, v_1)\}$  from  $E(P')$  and add  $k$  arcs  $a(v, v_1, i_1), \dots, a(v_{k-1}, v_k, i_{k-1})$  to  $A(P')$ .
3. (*split vertices of the path*) For each  $v \in V(T)$  such that  $\deg_{P'}(v) = 3$ , or  $\deg_{P'}(v) = 4$  and  $\text{indeg}_{D'}(v) = 2$ , *split*  $v$  into vertices  $v^i$  and  $v^o$  as shown in Fig. 3 and 4. Note that in this case each dashed line may be an edge or an arc with the tail at  $v$  or  $v^o$ . Formally, add new vertices  $v^i$  and  $v^o$  to  $V(P')$ , for each arc  $a_d(u, v)$  add new arc  $a_d(u, v^i)$ , for each arc  $a_d(v, x)$  add new arc  $a_d(v^o, x)$ , and for each edge  $e_d(v, x)$  add new edge  $e_d(v^o, x)$ , and remove  $v$  from  $V(P')$ .

Knowing that there is no path in  $G_{3,4}(P')$ , let  $P_2 = P'$  and let  $P_2 = H_2 \cup D_2$ , where  $H_2$  is a multigraph and  $D_2$  is a multidigraph.

▷ **Claim 13.** For each  $v \in V(P_2)$ ,  $\deg_{P_2}(v) \leq 2$ . If  $\deg_{P_2}(v) = 2$  and there is an arc with the head at  $v$ , then  $v$  is incident with two arcs with heads at  $v$ .

*Proof.* If  $v \in V(P_2)$  is a vertex such that  $v = w^o$  or  $v = w^i$ , for some  $w \in V(P_0)$ , then  $\deg_{P_2}(v) \leq 2$ . Let  $v \in V(P_1)$  and  $\deg_{P_1}(v) > 2$ . Then,  $v \in V(G_{3,4}(P_1))$ , and by Claim 12, there is  $u \in V(P_1)$  such that  $e_d(u, v) \in E(P_1)$ . Thus,  $v \in V(T)$  for some path  $T$  while applying step (1) in the transformation  $P_1 \dashrightarrow P_2$ . After orienting the path  $T$  in step (2), vertex  $v$  is splitted in the next step (3) or its degree is 4 and there are two edges incident with  $v$ . In the latter case,  $v \in V(T')$  for some other path  $T'$  while applying step (1). Thus, after orienting the path  $T'$  in step (2),  $v$  is splitted in the successive step (3).

Let  $\deg_{P_2}(v) = 2$ , for some  $v \in V(P_2)$ , and let us assume that there is an arc with the head at  $v$ . If  $v \in V(P_0)$ , then there is no arc with the head at  $v$  in *pom*-graph  $P_2$ . Thus,  $v = w^i$  for some splitted vertex  $w$ , and  $v$  is incident with two arcs with heads at  $v$ . ◁

Let  $P$  be the *pom*-graph obtained by the transformation  $P_2 \rightarrow_{sd} P$ , and let  $H = \text{Gr}(P)$  and  $G^* = \text{Un}(P)$ . Obviously,  $H$  and  $G^*$  are simple graphs, and by Claim 13,  $\Delta(P) \leq 2$ .

▷ **Claim 14.** Let  $T \sqsubset H$  be a maximal path. Let  $v, u \in V(T)$  such that  $\deg_H(v) = \deg_H(u) = 1$ . If there are arcs  $(v, x), (u, y) \in A(P)$ , then  $|E(T)|$  is even.

▷ **Claim 15.** The graph  $G^*$  is bipartite and each component of  $G^*$  is a path of length of at least 2 or a cycle of length at least 4.

We define  $c': E(P) \cup A(P) \rightarrow \{1, 2, 3, 4\}$  separately for each  $P' \subset P$  such that  $P^*$  is a connected component of  $G^*$ , where  $P^* = \text{Un}(P')$ . By Claim 15,  $P^*$  is a path or a cycle. Let  $V(P') = \{v_1, \dots, v_l\}$  and let  $k = |E(P') \cup A(P')|$ . If  $P^*$  is a cycle, let us denote  $v_{k+1} = v_1$  and  $v_{k+2} = v_2$ . Let us assume that for each  $i \in \{1, \dots, k\}$ ,  $v_i$  and  $v_{i+1}$  are neighbours in  $P'$ . For each  $i \in \{1, \dots, k\}$ , let  $o_i = o(\{v_i, v_{i+1}\})$ , where  $\{v_i, v_{i+1}\} \in E(P^*)$ .

First, color arcs in  $A(P')$ . For each  $i \in \{1, \dots, k\}$ , if  $o_i$  is an arc with the head at  $v_i$ , then let  $c'(o_i) = 1$ , and if  $o_i$  is an arc with head at  $v_{i+1}$ , then let  $c'(o_i) = 4$ . Next, we color edges in  $E(P')$ . If  $o_1$  is an edge, then let  $c'(o_1) = 3$ . For each  $i \in \{1, \dots, k-1\}$  (if  $P^*$  is a cycle, also for  $i = k$ ), if  $o_i$  is an arc and  $o_{i+1}$  is an edge, then by Claim 13,  $o_i$  has the head at  $v_i$ . Since  $c'(o_i) = 1$ , let  $c'(o_{i+1}) = 2$ . We extend  $c'$  to the rest of uncolored edges of  $E(P')$ , coloring them with colors 2 and 3 such that no two adjacent edges have the same color.

## 26:10 Interval Edge Coloring of Bipartite Graphs with Small Vertex Degrees

Observe that if for some  $i \in \{1, \dots, k\}$ ,  $o_i$  is an edge and  $o_{i+1}$  is an arc with the tail at  $v_{i+1}$ , then  $c'(o_{i+1}) = 4$ , and by Claim 14,  $c'(o_i) = 3$ . If  $o_i$  and  $o_{i+1}$  are arcs, then by Claim 13,  $o_i$  and  $o_{i+1}$  have heads at  $v_{i+1}$ , hence  $c'(o_i) = 4$  and  $c'(o_{i+1}) = 1$ .

Let us define  $c^*: E(G^*) \rightarrow \{1, 2, 3, 4\}$  as follows:  $c^*(e) = c'(o(e))$ , where  $o(e) \in E(P) \cup A(P)$ . By the above construction of  $c'$ ,  $c^*$  is an edge 4-coloring.

Now, we define  $c: E(G) \rightarrow \{1, 2, 3, 4\}$ . We contract all the pairs of vertices from  $V(G^*)$  that come from vertices splitted in the transformations  $--\rightarrow_1$  or  $--\rightarrow_2$  and we preserve the colors on the corresponding edges, and thus we get the initial graph  $G$  that is edge colored. Formally, for each  $v^i, v^o \in V(G^*)$ , where  $v \in V(P_0)$  (see Fig. 3 and 4), we contract vertices  $v^i, v^o$  and we preserve the colors on the corresponding edges  $\{v^*, x\}$  and  $\{v, x\}$ , i.e.,  $c(\{v, x\}) = c^*(\{v^*, x\})$ , where  $v^*$  is  $v^i$  or  $v^o$ . Since  $c^*(E_{G^*}(v^i))$  is equal to  $\{1\}$ ,  $\{4\}$  or  $\{1, 4\}$  and  $c^*(E_{G^*}(v^o)) = \{2, 3\}$ , we get  $c(E_G(v)) = \{1, 2, 3\}$  or  $c(E_G(v)) = \{2, 3, 4\}$ , or  $c(E_G(v)) = \{1, 2, 3, 4\}$ . Thus,  $c$  is an interval 4-coloring of  $G$ .

Since the transformation of  $G$  into  $P_2$  can be done in linear time, and the coloring of  $P$  can be done in linear time, the final construction of the coloring of  $G$  can be done in linear time.  $\blacktriangleleft$

By Theorem 11, and Propositions 4 and 5 we get the following theorem.

► **Theorem 16.** *Let  $G$  be a  $(4^*, 2^*)$ -bipartite graph. Then,  $\chi'_i(G) = 4$ . The construction of an interval 4-coloring of  $G$  can be done in linear time.*

### 2.4 Interval $\chi'_i$ -coloring problem of $(5^*, 2^*)$ -bipartite graphs

Let  $G$  be a  $(5^*, 2^*)$ -bipartite graph. Let  $\mathcal{F}_5: V(G) \rightarrow 2^{\mathbb{N}} \setminus \{\emptyset\}$  be defined as follows: if  $\deg_G(v) = 2i + 1$ , for  $i \in \{0, 1\}$ , then let  $\mathcal{F}_5(v) = \{i, i + 1\}$ , if  $\deg_G(v) = 2i$ , for  $i \in \{1, 2\}$ , then let  $\mathcal{F}_5(v) = \{i\}$ , and if  $\deg_G(v) = 5$ , then let  $\mathcal{F}_5(v) = \{2\}$ .

Let  $G$  be a  $(5^*, 2^*)$ -bipartite graph. If  $c$  is an interval 5-coloring of  $G$ , then  $F = \{e \in E(G): c(e) \in \{2, 4\}\}$  is an  $\mathcal{F}_5$ -factor of  $G$ .

► **Theorem 17.** *Let  $G$  be a  $(5^*, 2^*)$ -bipartite graph with  $n$  vertices. Then,  $\chi'_i(G) = 5$  if and only if  $G$  admits an  $\mathcal{F}_5$ -factor. The construction of an interval 5-coloring can be done in  $O(n^{3/2})$  time.*

► **Theorem 18.** *Let  $G$  be a  $(5^*, 2)$ -bipartite graph. Then,  $5 \leq \chi'_i(G) \leq 6$  and the construction of an interval 6\*-coloring of  $G$  can be done in  $O(n^{3/2})$  time.*

By Theorems 17 and 18, and by Propositions 4 and 5 we get

► **Theorem 19.** *Let  $G$  be a  $(5^*, 2^*)$ -bipartite graph. Then,  $\chi'_i(G) \leq 6$  and the construction of an interval  $\chi'_i$ -coloring of  $G$  can be done in  $O(n^{3/2})$  time.*

## 3 $\mathcal{NP}$ -completeness results

► **Theorem 20.** *The problem of interval 5-coloring of  $(5^*, 3^*)$ -bipartite graphs is  $\mathcal{NP}$ -complete.*

The Table 1 contains the state-of-art and our results presented in this paper, and some open problems for further research.

■ **Table 1** The complexity of the algorithms for the interval  $\chi'_i$ -coloring problem.

Interval edge $\chi'_i$ -coloring problem for $(\alpha^*, \beta^*)$ -bipartite or $(\alpha, \beta)$ -biregular graphs			
Graphs	$\chi'_i$	Complexity	Comments
$(\alpha^*, 1^*)$	$k$	$O(n)$	stars
$(2^*, 2^*)$	2	$O(n)$	paths and cycles
$(3^*, 2^*)$	3 or 4	$O(n)$	Thm. 10 ( $\chi'_i = 3$ ), [11] ( $\chi'_i \leq 4$ )
$(4^*, 2^*)$	4	$O(n)$	Thm. 16
$(5^*, 2^*)$	5 or 6	$O(n^{3/2})$	Thm. 17 ( $\chi'_i = 5$ ), Thm. 19 ( $\chi'_i \leq 6$ )
$(6^*, 2^*)$	?	?	<i>interval coloring problem</i> is open
$(3^*, 3^*)$	3 or 4	$O(n^{3/2})$ or $O(n)$	[7] ( $\chi'_i = 3$ ) [11] ( $\chi'_i \leq 4$ )
$(4, 3)$	?	?	<i>interval coloring problem</i> is open
$(5, 3)$	?	?	<i>interval coloring problem</i> is open
$(6, 3)$	$\leq 7$	$O(n)$	[4] $((\chi'_i + 1)^*$ -algorithm)
$(5^*, 3^*)$	5	$\mathcal{NP}$ -complete	Thm. 20
$(6, 3)$	6	$\mathcal{NP}$ -complete	[1]
$(2\alpha, 2)$	$2\alpha$	$O(n\Delta \log \Delta)$	[11]
$(2\alpha + 1, 2)$	$2\alpha + 2$	$O(n^{3/2}\Delta^2)$	[13] (compl. of 2-factor by Thm. 1)
$(2\alpha, 2^*)$	$2\alpha$	$O(n\Delta \log \Delta)$	[11] and Cor. 8
$(2\alpha + 1, 2^*)$	$\leq 2\alpha + 2$	$O(n^{3/2}\Delta^2)$	[13] and Cor. 9 $((\chi'_i + 1)^*$ -algorithm)

## References

- 1 A.S. Asratian and C.J. Casselgren. On interval edge colorings of  $(\alpha, \beta)$ -biregular bipartite graphs. *Discret. Math.*, 307:1951–1956, 2006. doi:10.1016/j.disc.2006.11.001.
- 2 A.S. Asratian and R.R. Kamalian. Interval coloring of the edges of a multigraph (in Russian). *Appl. Math.*, 5:25–34, 1987.
- 3 A.S. Asratian and R.R. Kamalian. Investigation of interval edge-colorings of graphs. *J. Combin. Theory Ser. B*, 62:34–43, 1994. doi:10.1006/jctb.1994.1053.
- 4 C.J. Casselgren and B. Toft. On Interval Edge Colorings of Biregular Bipartite Graphs with Small Vertex Degrees. *J. Graph Theory*, 80:83–97, 2015. doi:10.1002/jgt.21841.
- 5 R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in  $O(E \log D)$  time. *Combinatorica*, 21:5–12, 2001. doi:10.1007/s004930170002.
- 6 G.Cornuéjols. General factors of graphs. *J. Combin. Theory Ser. B*, 45:185–198, 1988. doi:10.1016/0095-8956(88)90068-8.
- 7 K. Giaro. The complexity of consecutive  $\Delta$ -coloring of bipartite graphs: 4 is easy, 5 is hard. *Ars Combin.*, 47:287–298, 1997.
- 8 K. Giaro and M. Kubale. Consecutive edge-colorings of complete and incomplete Cartesian products of graphs. *Congr. Numer.*, 128:143–149, 1997.
- 9 K. Giaro and M. Kubale. Compact scheduling of zero–one time operations in multi-stage systems. *Discret. Appl. Math.*, 145:95–103, 2004. doi:10.1016/j.dam.2003.09.010.
- 10 K. Giaro, M. Kubale, and M. Malafiejski. Compact Scheduling In Open Shop with Zero-One Time Operations. *INFOR: Information Systems and Operational Research*, 37:37–47, 1999.
- 11 H.M. Hansen. Skemalægning med henblik på minimering af ventetid (in Danish). *M.Sc. Thesis*, University of Odense, 1992.
- 12 D. Hanson and C.O.M. Loten. A lower bound for Interval colouring bi-regular bipartite graphs. *Bull. ICA*, 18:69–74, 1996.
- 13 D. Hanson, C.O.M. Loten, and B. Toft. On interval coloring of biregular bipartite graphs. *Ars Combin.*, 50:23–32, 1998.
- 14 R.R. Kamalian. Interval colorings of complete bipartite graphs and trees (in Russian). *Comp. Cen. of Acad. Sci. of Armenian SSR (Preprint)*, 1989.

## 26:12 Interval Edge Coloring of Bipartite Graphs with Small Vertex Degrees

- 15 R.R. Kamalian. Interval edge-colorings of graphs. *Ph.D. Thesis, Novosibirsk State University*, 1990.
- 16 H.H. Khachatryan and P. Petrosyan. Interval Non-edge-Colorable Bipartite Graphs and Multigraphs. *J. Graph Theory*, 76:200–216, 2014. doi:10.1002/jgt.21759.
- 17 L. Lovász. The factorization of graphs. II. *Acta Math. Acad. Sci. Hungar.*, 23:223–246, 1972. doi:10.1007/BF01889919.
- 18 S. Micali and V.V. Vazirani. An  $O(\sqrt{|V|} \cdot |E|)$  algorithm for finding maximum matching in general graphs. *Proc. of 21st FOCS*, pages 17–27, 1980. doi:10.1109/SFCS.1980.12.
- 19 S.V. Sevastjanov. Interval colorability of the edges of a bipartite graph (in Russian). *Metody Diskretnogo Analiza*, 50:61–72, 1990.

# Selected Neighbor Degree Forest Realization

Amotz Bar-Noy ✉

City University of New York (CUNY), NY, USA

David Peleg ✉ 

Weizmann Institute of Science, Rehovot, Israel

Dror Rawitz ✉

Bar Ilan University, Ramat-Gan, Israel

Elad Yehezkel ✉

Weizmann Institute of Science, Rehovot, Israel

---

## Abstract

The classical degree realization problem is defined as follows: Given a sequence  $\bar{d} = (d_1, \dots, d_n)$  of positive integers, construct an  $n$ -vertex graph in which each vertex  $u_i$  has degree  $d_i$  (or decide that no such graph exists). In this article, we present and study the related *selected neighbor degree realization* problem, which requires that each vertex  $u_i$  of  $G$  has a *neighbor* of degree  $d_i$ . We solve the problem when  $G$  is required to be acyclic (i.e., a forest), and present a sufficient and necessary condition for a given sequence to be realizable.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms

**Keywords and phrases** network realization, graph algorithms, lower bound

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.27

**Funding** Supported in part by a US-Israel BSF grant (2018043).

## 1 Introduction

**Background and motivation.** Different properties of a given network can be described using “profiles” of the network. As a classical example, the *degree profile* (or *degree sequence*) of an  $n$ -vertex graph  $G$  is the sequence  $\text{DEG}(G) = (d_1, \dots, d_n)$ , where  $d_i = \deg(u_i)$  is the degree of the vertex  $u_i$ .

The extensively studied *degree realization* problem concerns the situation where given a sequence of positive integers  $\bar{d} = (d_1, \dots, d_n)$ , we are asked whether there exists a graph whose degree sequence conforms to  $\bar{d}$ . (If so, then the sequence  $\bar{d}$  is called *graphic*.) Erdős and Gallai [15] gave a necessary and sufficient condition for deciding if a given degree profile is realizable (also implying a  $\Theta(n)$  time decision algorithm), and Havel and Hakimi [18, 19] gave a  $\Theta(\sum_i d_i)$  time algorithm that given a degree profile  $\bar{d}$  computes a realizing graph, or proves that the profile is not realizable. The problem is known to be particularly simple when the realizing graph is required to be acyclic, in which case the necessary and sufficient condition for realizability is simply that  $\sum_i d_i = 2(n - k)$ , where  $k \in \{1, \dots, n\}$  (see [16] for a short analysis for trees). Many extensions and variations of the degree realization problem were studied in the past, cf. [1, 11, 20, 24, 26, 31, 32, 34, 35]. Interesting applications in the context of social networks are studied in [9, 13, 21].

Other aspects of the graph structures may be described using other types of profiles. We focus on profiles that capture aspects of the *vertex neighborhoods* in the given graph. One reason for our interest in neighbor degrees is that in the context of social networks, it is often informative to observe not only the individual degree of each vertex, but also the degrees of nearby vertices, since obtaining a more complete picture of the degree distribution in a given neighborhood may reveal useful information regarding the interrelationships among vertices, and their relative standing in their immediate society.



© Amotz Bar-Noy, David Peleg, Dror Rawitz, and Elad Yehezkel;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 27; pp. 27:1–27:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Let  $N[u]$  denote the closed (inclusive) neighborhood of the vertex  $u$  in  $G$ , namely  $N[u] = \{v \mid (v, u) \in E\} \cup \{u\}$ . Clearly, the profile  $N(G) = \langle N[u_1], \dots, N[u_n] \rangle$  tells us everything we may need to know. However, this profile is costly, or “heavy”, in the sense that storing it requires as much memory as storing the entire graph. Instead, one is often interested in studying “lighter” profiles, storing only a small amount of information per vertex.

For a graph  $G(V, E)$ ,  $V = \{u_1, \dots, u_n\}$ , let  $\text{ctr} : V \rightarrow V$  be a *neighbor selection* function, such that  $\text{ctr}(u) \in N[u]$  for each vertex  $u$ . We refer to the vertex  $\text{ctr}(u)$  as  $u$ 's *selected neighbor* or *center*. For each vertex  $u$ , denote the degree of  $u$ 's selected neighbor by  $\text{snd}(u) = \deg(\text{ctr}(u))$ . Then the sequence  $\text{SND}(G, \text{ctr}) = (\text{snd}(u_1), \dots, \text{snd}(u_n))$  is referred to as the *selected neighbor degree (SND) profile* of the pair  $(G, \text{ctr})$ .

Interesting special cases of the selected neighbor degree profile SND arise when the neighbor selection function  $\text{ctr}$  targets some specific neighbor for each vertex. The ordinary degree profile is obtained by using  $\text{ctr}^{\text{self}}(u) = u$ . Picking the function  $\text{ctr}^{\text{max}}$ , which selects for every vertex  $u$  its neighbor of maximum degree, yields

$$\text{maxnd}(u) = \deg(\text{ctr}^{\text{max}}(u)) = \max\{\deg(w) \mid w \in N[u]\},$$

which gives the *maximum neighbor degree* profile  $\text{MaxND}(G) = (\text{maxnd}(u_1), \dots, \text{maxnd}(u_n))$ . The functions  $\text{ctr}^{\text{min}}$  and  $\text{minnd}$  and the *minimum neighbor degree* profile  $\text{MinND}(G)$  can be defined analogously. Note that the profile  $\text{MaxND}(G)$  (resp.,  $\text{MinND}(G)$ ) is independent of the choice of  $\text{ctr}^{\text{max}}$  (resp.,  $\text{ctr}^{\text{min}}$ ). This is not the case for general SND profiles.

Recently, we studied the realization problem for the  $\text{MinND}$  and  $\text{MaxND}$  profiles. In particular, the *minimum neighbor degree realization* problem, where given a sequence  $\bar{d}$  of  $n$  integers one must decide if there is a graph  $G$  such that  $\text{MinND}(G) = \bar{d}$  and construct such a graph (if exists), was studied in [3]. A complete characterization was given for realization by *forests* (i.e., acyclic graphs), but the problem over general graphs was left open. Surprisingly, when studying the realizability of the *maximum neighbor degree* profile  $\text{MaxND}$  [6], the picture was reversed: we were able to give a complete characterization for the realization problem of maximum neighbor degrees on general graphs, but on forests the problem appears to be harder, and was left open.

It is therefore natural to investigate the problem's behavior when, instead of  $\text{MaxND}$  and  $\text{MinND}$ , we look at general *selected neighbor degree* profiles. The current study addresses this question. We resolve the realizability of SND profiles by forests, although surprisingly even this more relaxed variant turned out to be subtle and considerably more difficult than anticipated initially. Formally, we study the *selected neighbor degree Forest realization (SNDF) problem*. Consider a given  $n$ -integer *SNDF-specification*  $\bar{d}$ . We say that  $\bar{d}$  is a *forest-realizable SNDF-profile* if there exists an  $n$ -vertex forest  $F$ , and a neighbor selection function  $\text{ctr}$ , whose SNDF-profile satisfies  $\text{SND}(F, \text{ctr}) = \bar{d}$ .

**Our Contribution.** We study an optimization version of the SNDF problem. As mentioned earlier, not every SNDF-specification  $\bar{d}$  is realizable. To cope with unrealizable profiles, we define a measure for the deviation of a given profile  $\bar{d}$  from realizability in Sect. 2, where we also introduce the basic elements of the problem, as well as some preliminary notions used in our solution. In Sect. 3 we introduce the framework, give a high-level overview of the general approach and present basic tools for handling SNDF profiles. In Section 4 we present a tight lower bound on the deviation of SNDF-profiles. This lower bound lays the foundation for our algorithm. We also outline our construction algorithm for the problem, which is optimal in the sense that when  $\bar{d}$  is realizable by a forest, the resulting construction  $(F, \text{ctr})$

is a realization of  $\bar{d}$ , and when it is not realizable, the resulting  $(F, \text{ctr})$  has the minimum possible deviation (matching the lower bound). Both the lower bound and the algorithm are rather involved, hence most of the details are omitted for lack of space and can be found in [36]. As a byproduct, our analysis also yields necessary and sufficient conditions for a specification  $\bar{d}$  to be realizable by a forest, as well as a fast algorithm for deciding realizability, thus providing a complete solution for the SNDF problem. Finally, our algorithm can be easily modified into one that minimizes the number of centers.

**Related Work.** Over the years, various extensions of the degree realization problem were studied, cf. [1, 34]. Many studies have addressed related questions such as finding all the (non-isomorphic) graphs that realize a given degree sequence, counting all the (non-isomorphic) realizing graphs of a given degree sequence, sampling a random realization for a given degree sequence as uniformly as possible, or determining the conditions under which a given degree sequence defines a unique realizing graph (a.k.a. the *graph reconstruction* problem), cf. [11, 15, 18, 19, 20, 24, 26, 31, 32, 35]. Interesting applications in the context of social networks are studied in [9, 21, 13]. The somewhat related *shotgun assembly* problem [22] studies graph specifications consisting of a description of the  $r$ -neighborhood (up to radius  $r$ ) of each vertex  $i$ . Realization questions of a similar nature were studied for *other* applications, where given *some* type of information profile specifying the desired vertex properties (concerning distances, connectivity, centrality, or any other property of significance), one may ask whether there exists a graph conforming to the specified profile (see, e.g., [2, 4, 5, 7, 8, 12, 14, 10, 17, 23, 25, 27, 28, 29, 30, 33, 37]). The selected neighbor degree realization problem belongs to this class of problems.

## 2 Preliminaries

Let  $F = (V, E)$  be a forest. For a vertex set  $U \subseteq V$ , let  $N[U] = \bigcup_{u \in U} N[u]$  be the *closed neighborhood* of  $U$ . Let  $\text{ctr} : V \rightarrow V$  be a *neighbor* function on  $F$ 's vertices such that  $\text{ctr}(u) \in N[u]$  for each  $u \in V$ . For every  $u \in V$ , define

$$\text{snd}(u) = \text{snd}_{(F, \text{ctr})}(u) = \deg_F(\text{ctr}(u)) .$$

When  $F$  and  $\text{ctr}$  are clear from the context, we omit them and write  $\text{snd}(u)$ . We refer to  $\text{snd}(u)$  as the *snd value* of  $u$ . The *SND profile* of  $(F, \text{ctr})$  is the sequence

$$\text{SND}(F, \text{ctr}) = (\text{snd}(u))_{u \in V} .$$

It is convenient to represent an SNDF profile in a condensed form as a list of non-negative integers  $(k_i^{n_i})_{i=1}^{\ell}$ , meaning that each value  $k_i$  appears in the list  $n_i$  times and the list contains  $\ell$  distinct values, i.e. there are  $n_i$  vertices that have *snd* value  $k_i$ . We assume  $n - 1 \geq k_1 > \dots > k_\ell \geq 0$ . Overall  $n = \sum_{i=1}^{\ell} n_i$ .

We are interested in the following SNDF realization problem. A given sequence  $\bar{d} = (k_i^{n_i})_{i=1}^{\ell}$  is viewed as an *SNDF profile*. It is realizable if there exists a pair,  $(F, \text{ctr})$ , where  $F$  is a forest, such that  $\text{SND}(F, \text{ctr}) = \bar{d}$ . We call  $(F, \text{ctr})$  an *SNDF realization* of  $\bar{d}$ . (Note that  $\bar{d}$  may or may not be realizable.) The problem concerns finding a realizing  $(F, \text{ctr})$  for a given profile  $\bar{d}$ , if exists. Observe that the *snd* value 0 can only be realized by singleton vertices, independently of the rest of the profile. So hereafter assume that  $k_\ell \geq 1$ .

**Star formations.** When  $k_i + 1$  vertices are required by the profile to have a neighbor of degree  $k_i$ , the requirement can be easily satisfied in a self-sufficient manner by a star composed of a root  $v$  and  $k_i$  leaves, all pointing at the root (i.e., with  $\text{ctr}(u) = v$ ). Likewise,

## 27:4 Selected Neighbor Degree Forest Realization

if the profile contains  $k_i^{n_i}$  where  $n_i = c \cdot (k_i + 1)$  for some integer  $c$ , then this component of the profile can be realized on its own, using  $c$  stars of size  $k_i$  (where by the *size* of a star we refer to the number of leaves). This motivates the following alternative representation for a profile  $\bar{d}$ , using two sequences  $(c_i)_{i=1}^\ell$  and  $(\rho_i)_{i=1}^\ell$  such that

$$n_i = c_i(k_i + 1) - \rho_i \quad \text{and} \quad 0 \leq \rho_i \leq k_i, \quad (1)$$

where  $\rho_i = c_i(k_i + 1) - n_i$  is the  $i$ th *residue*. We refer to the tuple  $(c_i, \rho_i)_{i=1}^\ell$  as the *star formation* of  $\bar{d}$ . Note that

$$n = \sum_{i=1}^{\ell} n_i = \sum_{i=1}^{\ell} (c_i(k_i + 1) - \rho_i).$$

As mentioned before, not every profile  $\bar{d}$  is realizable. We therefore seek approximate solutions to the SNDF realization problem.

**Upper realizations and deviation.** Let  $F = (V, E)$  be a forest and let  $\text{ctr} : V \rightarrow V$ . We say that  $(F, \text{ctr})$  is an *upper realization* (or U-realization) of the profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  if the SNDF profile of  $(F, \text{ctr})$  is of the form  $\text{SND}(F, \text{ctr}) = \bar{d}' = (k_i^{n'_i})_{i=1}^\ell$  (so in particular  $\text{snd}(u) \in \{k_1, \dots, k_\ell\}$  for every  $u$ ) and  $n'_i \geq n_i$  for every  $i$ . Denote  $n(F) = \sum_{i=1}^{\ell} n'_i$ . Denote by  $cc(F)$  the number of connected components in  $F$ . Define the *deviation* of  $(F, \text{ctr})$  from the profile  $\bar{d}$  as

$$\text{Dev}(\bar{d}, (F, \text{ctr})) = \sum_{i=1}^{\ell} (n'_i - n_i) = n(F) - n.$$

**The trivial U-realization.** Observe that there is a straightforward way for constructing a U-realization to a given profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$ . Define  $(c_i)_{i=1}^\ell$  and  $(\rho_i)_{i=1}^\ell$  as above. For each  $1 \leq i \leq \ell$  create  $c_i$  stars of size  $k_i$ , and for each leaf  $v$  in a star with center  $u$  define  $\text{ctr}(v) = u$  and  $\text{ctr}(u) = u$ . The resulting forest  $\tilde{F}$  contains  $cc(\tilde{F}) = \sum_{i=1}^{\ell} c_i$  connected components and  $\sum_{i=1}^{\ell} c_i(k_i + 1)$  vertices. We refer to this construction as the *trivial* construction and denote it by  $(\tilde{F}(\bar{d}), \tilde{\text{ctr}}(\bar{d}))$ . Note that

$$\text{Dev}(\bar{d}, (\tilde{F}, \tilde{\text{ctr}})) = \sum_{i=1}^{\ell} (c_i(k_i + 1) - n_i) = \sum_{i=1}^{\ell} \rho_i. \quad (2)$$

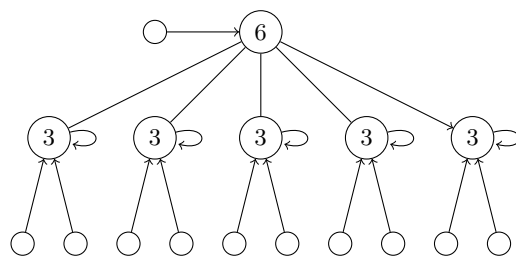
While this realization may in some cases be near-optimal, our goal is to construct a U-realization  $(F, \text{ctr})$  whose number of vertices,  $n(F)$ , is as close as possible to  $n$ , the specified number of vertices. Define the *realizable size* of a given profile  $\bar{d}$  as the minimal size of any U-realization for it,

$$n^*(\bar{d}) = \min\{n(F) \mid \exists \text{ctr} : (F, \text{ctr}) \text{ is a U-realization for } \bar{d}\}.$$

Define the deviation of a given profile  $\bar{d}$  as the minimum deviation over all of its U-realizations,

$$\text{Dev}(\bar{d}) = n^*(\bar{d}) - n = \min\{\text{Dev}(\bar{d}, (F, \text{ctr})) \mid (F, \text{ctr}) \text{ is a U-realization for } \bar{d}\}. \quad (3)$$

In this way, we can redefine realizability of SNDF profiles as follows. A given SNDF profile  $\bar{d}$  is realizable if and only if  $n^*(\bar{d}) = n$ , or alternatively,  $\text{Dev}(\bar{d}) = 0$ . Denote by  $(F^*, \text{ctr}^*)$  an optimal U-realization of  $\bar{d}$  (note that  $(F^*, \text{ctr}^*)$  is not necessarily unique), namely, such that  $n(F^*) = n^*(\bar{d})$ , or,  $\text{Dev}(\bar{d}, (F^*, \text{ctr}^*)) = \text{Dev}(\bar{d})$ . Our goal is to find such a realization.



■ **Figure 1** Realization of  $\bar{d}^1 = (6^1, 3^{16})$  using  $\bar{s} = (1, 5)$ . The centers are marked by their degrees.

**Centers and members.** The sets of  $k_i$ -centers and  $k_i$ -members in upper realization  $(F, \text{ctr})$  for a profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  are

$$C_i(F, \text{ctr}) = \{u \in F \mid \deg(u) = k_i, \exists v \in N[u] \text{ s.t. } \text{ctr}(v) = u\},$$

$$M_i(F, \text{ctr}) = \{u \in F \mid \text{snd}(u) = k_i\}.$$

(We may write simply  $C_i$  and  $M_i$  when clear from context.) Clearly,  $C_i \cap C_j = \emptyset$  and  $M_i \cap M_j = \emptyset$  for every  $i \neq j$ , and  $\bigcup_i M_i = V$ , i.e., the sets  $M_i$  form a partition of the vertices of  $F$ . Set  $\sigma_i(F, \text{ctr}) = |C_i(F, \text{ctr})|$ , and let

$$C(F, \text{ctr}) = \bigcup_{i=1}^\ell C_i(F, \text{ctr})$$

and

$$\bar{\sigma}(F, \text{ctr}) = (\sigma_1(F, \text{ctr}), \dots, \sigma_\ell(F, \text{ctr})).$$

**Center specification sequences (CSS).** Clearly, any U-realization  $(F, \text{ctr})$  for  $\bar{d}$  must use at least  $c_i$  centers on layer  $i$ , so  $\sigma_i \geq c_i$  for every  $i$  (see also Lemma 5(1)). Unfortunately, for some profiles, using exactly  $c_i$  centers for every  $i$  yields a suboptimal upper realization (with deviation greater than  $Dev(\bar{d})$ ). A key component of the problem is thus to decide, given  $\bar{d}$ , on the right number of centers for each layer. These numbers are represented as a *center specification sequence* (or *CSS*)  $\bar{s} = (s_1, \dots, s_\ell)$ , which must satisfy  $s_i \geq c_i$  for every  $i$ . A U-realization  $(F, \text{ctr})$  *conforms* to  $(\bar{d}, \bar{s})$  if the number of centers on each layer is as specified by  $\bar{s}$ , i.e.,  $\bar{\sigma}(F, \text{ctr}) = \bar{s}$ . In our construction, we first select a CSS  $\bar{s}$ , and then look for a conforming U-realization  $(F, \text{ctr})$  for it.

For example, the profile  $\bar{d}^1 = (6^1, 3^{16})$  requires at least  $\bar{c} = (1, 4)$  centers, but any U-realization with this many centers has deviation at least 1 (as follows from Thm. 8), whereas using the CSS  $\bar{s} = (1, 5)$  yields the optimal  $Dev(\bar{d}^1) = 0$ , as demonstrated in the Figure 1.

Intuitively, in a conforming U-realization  $(F, \text{ctr})$ , each  $x \in C_i$  acts as the center of a star of degree  $k_i$ , potentially allowing its “clients”  $y \in M_i$  to have  $\text{snd}(y) = k_i$  by setting  $\text{ctr}(y) = x$ . Each center is also a member, possibly for a different  $i$ . Also,  $\text{ctr}(u) \in C_i \cap N[u]$  for every  $i \in [\ell]$  and  $u \in M_i$ , so  $M_i \subseteq N[C_i]$  and  $|M_i| \leq s_i(k_i + 1)$ . Define the *residue* of  $k_i$  w.r.t. a CSS  $\bar{s}$  as

$$\rho_i^s = s_i(k_i + 1) - n_i. \tag{4}$$

Note that Eq. (4) for  $\bar{s} = \sigma(F, \text{ctr}) = (s_1, \dots, s_\ell)$  and  $\bar{\rho}^s = \bar{\rho}^\sigma(F, \text{ctr}) = (\rho_1^s, \dots, \rho_\ell^s)$  is analogous to Eq. (1) for our star formation  $(\bar{c}, \bar{\rho})$ , since

$$n_i = s_i(k_i + 1) - \rho_i^s = c_i(k_i + 1) - \rho_i, \tag{5}$$

which is related to our definition of star formation, since each  $k_i$ -center  $u$  has  $|N[u]| = k_i + 1$ . We break the forest  $F$  into (possibly overlapping) stars centered around the  $k_i$ -centers in  $C_i$ .

The following two assumptions on U-realizations  $(F, \text{ctr})$  are used hereafter without loss of generality.

**Member independence.** In  $(F, \text{ctr})$ , the non-centers are independent, namely,  $F$  contains no edge  $(u, v)$  between any  $u, v \notin C$ . (Such edges can always be removed without changing the profile of  $F$ , and our constructions never use them.)

**No cross-pointing.** In  $(F, \text{ctr})$ , there is no *cross-pointing*, i.e., there are no two centers  $u$  and  $v$  such that  $\text{ctr}(u) = v$  and  $\text{ctr}(v) = u$ . (If cross-pointing occurs, one can change the two ctr values to  $\text{ctr}(u) = u$  and  $\text{ctr}(v) = v$  without changing the profile. Again, our constructions never use cross-pointing.)

### 3 Framework and basic tools

In this section, we describe the basic framework and the tools we use for realizing SNDF profiles.

#### Handling leaf centers

We first show how to handle the cases where  $k_\ell = 1$ . Consequently, in the rest of this article, we consider only SNDF profiles  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  where  $k_\ell \geq 2$ .

Trivial profiles of the form  $\bar{d} = (1^n)$  for  $n \geq 2$  can be realized by a star  $(F, \text{ctr})$  with  $n - 1$  leaves, such that  $\text{ctr}(u) = u$  for each leaf  $u$ , and for the center,  $v$ ,  $\text{ctr}(v)$  is defined to be one of the leaves.

It remains to handle profiles  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  where  $\ell \geq 2$  and  $k_\ell = 1$ . This is done by the following approach. Given such a profile  $\bar{d}$ , denote the *truncated profile* (without  $k_\ell$ ) by  $\bar{d}' = (k_i^{n_i})_{i=1}^{\ell-1}$ . We show that the deviation of  $\bar{d}$  is  $\text{Dev}(\bar{d}) = \max\{\text{Dev}(\bar{d}') - n_\ell, 0\}$ , and moreover, there is a polynomial time algorithm that given an optimal U-realization (of a special type, referred to as a *leaf-covered* U-realization) for  $\bar{d}'$ , transforms it into an optimal U-realization for  $\bar{d}$ . Hence the problem is reduced to finding optimal leaf-covered realizations for truncated profiles (with  $k_\ell \geq 2$ ).

#### Leaf-covered U-realizations

Let  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  be an SNDF profile with U-realization  $(F, \text{ctr})$ . Denote  $e_i = |M_i| - n_i$ , namely, the number of excess vertices with snd value  $k_i$ . Let  $L = \{u \in V(F) \mid \deg(u) = 1\}$  be the set of leaves of  $F$ . We say that  $(F, \text{ctr})$  is a *leaf-covered* U-realization for  $\bar{d}$  if  $|L \cap M_i| \geq e_i$  for every  $i$ .

Intuitively, a leaf-covered U-realization is easy to work with, since we may think of all the excess vertices as being among the leaves.

The reduction and its analysis are deferred to the full paper (see [36]). We get:

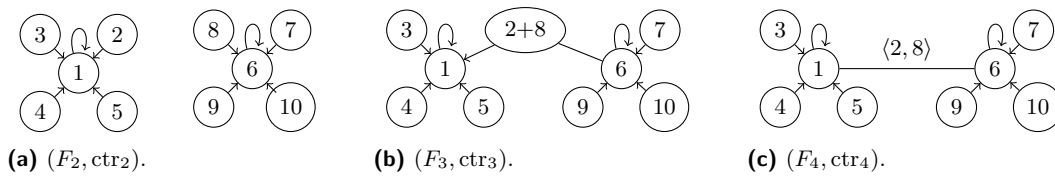
► **Proposition 1.** *Let  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  be a profile with  $\ell \geq 2$  and  $k_\ell = 1$ . Let  $(F', \text{ctr}')$  be an optimal leaf-covered U-realization for  $\bar{d}' = (k_i^{n_i})_{i=1}^{\ell-1}$ , namely,  $\text{Dev}(\bar{d}', (F', \text{ctr}')) = \text{Dev}(\bar{d}')$ . Then  $(F', \text{ctr}')$  can be converted, in polynomial time, into an optimal U-realization  $(F, \text{ctr})$  for  $\bar{d}$ , with  $\text{Dev}(\bar{d}, (F, \text{ctr})) = \text{Dev}(\bar{d})$ .*

By Prop. 1, it suffices to focus on finding optimal leaf-covered U-realizations for profiles without degree 1. Our main result is the following.

► **Theorem 2.** *Let  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  be an SNDF profile such that  $k_\ell \geq 2$ . There exists an algorithm that constructs an optimal U-realization  $(F^*, \text{ctr}^*)$  for  $\bar{d}$ , namely,  $\text{Dev}(\bar{d}, (F^*, \text{ctr}^*)) = \text{Dev}(\bar{d})$ . In addition,  $(F^*, \text{ctr}^*)$  is leaf-covered. The run-time of the algorithm is  $O(n^*(\bar{d}))$ , which is optimal.*

**Overview of the general approach**

Consider a profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  and let  $(F, \text{ctr})$  be some U-realization of  $\bar{d}$  with center classes  $C_1, \dots, C_\ell$  and  $C = \bigcup_i C_i$ . Consider a collection of stars  $\{S_u\}_{u \in C}$ , where  $S_u$  is centered at  $u$  and contains all of  $u$ 's neighbors in  $F$  and their respective edges. Note that  $S_u$  and  $S_v$  are not necessarily disjoint, but they can share at most two vertices, namely,  $|N[u] \cap N[v]| \leq 2$ , since otherwise, there is a cycle in  $F$ . The trivial U-realization for  $\bar{d}$ ,  $(\tilde{F}, \tilde{\text{ctr}})$  is wasteful, since it employs  $\sum_{i=1}^\ell c_i$  pairwise disjoint stars, resulting in  $\sum_{i=1}^\ell c_i(k_i + 1)$  vertices. To improve it, we construct a realization by starting from disjoint stars and then forcing them to share vertices by performing certain merge operations. The key operation performed by our algorithm involves merging stars. Specifically, the algorithm employs two operations, referred to as *head-merges* and *leaf-merges*.



■ **Figure 2** The forests  $F_2$ ,  $F_3$  and  $F_4$ .

To illustrate these operations, consider the example profile  $\bar{d}^2 = (4^{10})$ . As  $10 = 2 \cdot (4 + 1)$ , an exact realization of this profile is obtained by the trivial U-realization composed of the following pair  $(F_2, \text{ctr}_2)$ , consisting of two stars of size 4 (see Figure 2a). The directional edges represent the value of the ctr pointer of a specific vertex.

A *leaf-merge* of two stars fuses together two leaves, one of each star, into a single vertex, thus creating a single tree. Consider, for example, the profile  $\bar{d}^3 = (4^9)$ . As  $9 = 2 \cdot (4 + 1) - 1$ , the trivial U-realization composed of two stars of size 4, has one excess vertex (namely, a deviation of 1). To overcome this problem, we can leaf-merge vertices 2 and 8 of the forest  $F_2$  into a single vertex denoted  $2 + 8$ , yielding the following pair  $(F_3, \text{ctr}_3)$  (see Figure 2b).

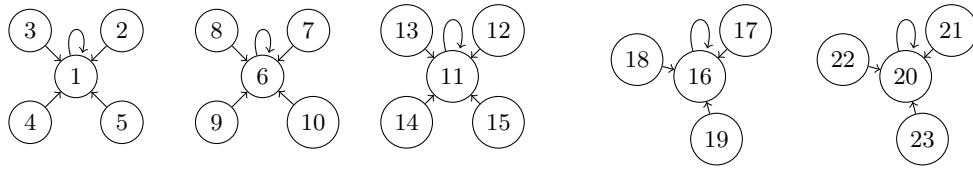
A *head-merge* of two stars is obtained by discarding one leaf of each star and connecting the star roots by a new edge. For example, to realize the profile  $\bar{d}^4 = (4^8)$ , whose trivial U-realization has a deviation of *two* excess vertices, we can head-merge the two stars of forest  $F_2$  by completely removing vertices 2 and 8 and connecting the star heads by a new edge (marked by  $\langle 2, 8 \rangle$  in the figure), yielding the following pair  $(F_4, \text{ctr}_4)$  (see Figure 2c).

Generally, satisfying part  $k_i^{n_i}$  of the profile requires using stars with  $k_i$  leaves, but this will satisfy the profile only if  $n_i$  is a multiple of  $k_i + 1$ . For other  $n_i$  values, using  $c_i = \lceil \frac{n_i}{k_i + 1} \rceil$  stars yields more vertices than needed. We therefore use head and leaf merges to get rid of the excess vertices.

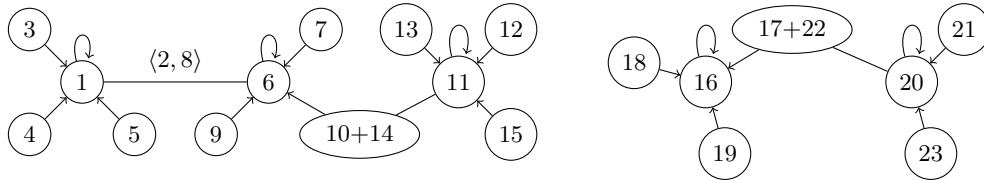
For example, consider the profile  $\bar{d}^5 = (4^{12}, 3^7)$ . As  $12 = 3(4 + 1) - 3$  and  $7 = 2(3 + 1) - 1$ , we start by creating three size 4 stars and two size 3 stars, yielding the pair  $(F_5, \text{ctr}_5)$  shown in Figure 3.

The forest  $F_5$  has three excess vertices with **snd** value 4 and one excess vertex with **snd** value 3. To correct it, we apply a head-merge operation and a leaf-merge operation on the size 4 stars, and a leaf-merge operation on the size 3 stars. This creates the following desired forest  $(F'_5, \text{ctr}'_5)$  depicted in Figure 4, which satisfies  $\bar{d}$ .

27:8 Selected Neighbor Degree Forest Realization

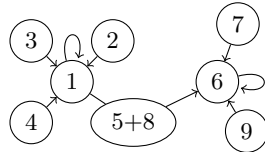


■ Figure 3 ( $F_5, \text{ctr}_5$ ).



■ Figure 4 ( $F'_5, \text{ctr}'_5$ ).

Our algorithm also exploits the fact that the value of  $\text{ctr}$  can be set so as to change the  $\text{snd}$  value of a vertex, by applying head or leaf merge operation to two stars of *different* sizes. To illustrate this point, let us consider the following example. The profile  $\bar{d}^6 = (4^4, 3^4)$  has one excess vertex with  $\text{snd}$  value 4. This profile is realizable in this way by starting from stars of size 3 and 4, and applying a leaf merge, to get the forest  $(F_6, \text{ctr}_6)$  shown in Figure 5.



■ Figure 5 ( $F_6, \text{ctr}_6$ ).

The analysis of our algorithm, and its proof of optimality, are based on the crucial observation that the *only* way to merge two individual stars is by head or leaf merges, namely, discarding one or two vertices, since trying to modify two stars by fusing together three or more vertices will create a cycle. This implies that certain profiles cannot be satisfied. For example, consider the profile  $\bar{d}^7 = (4^7)$ . Again, we must start with a forest  $F_2$ . However, there is no way to discard three vertices from  $F_2$  and end up with a satisfying forest of 7 vertices. Indeed, based on Lemma 5 one can show that  $\text{Dev}(\bar{d}^7) \geq 1$ .

**The  $\mathcal{LM}$  operation and Procedure *Connect***

We now present Procedure  $\mathcal{LM}$ , which reduces the deviation (and the number of connected components in the forest) by 1, by performing a single leaf merge. Formally, given a profile  $\bar{d} = (k_i^{n_i})_{i=1}^{\ell}$  and a U-realization  $(F, \text{ctr})$  of  $\bar{d}$  with  $\text{Dev}(\bar{d}, (F, \text{ctr})) > 0$ , such that  $F$  has  $q = \text{cc}(F) \geq 2$  connected components, the operation constructs a U-realization  $(F', \text{ctr}')$  for  $\bar{d}$  with  $\text{Dev}(\bar{d}, (F', \text{ctr}')) = \text{Dev}(\bar{d}, (F, \text{ctr})) - 1$ , such that  $F'$  has  $\text{cc}(F') = q - 1$  connected components.

While the leaf merge operation is straightforward, applying it to an arbitrary forest raises some subtle points, as discussed next. Let  $i \in [\ell]$  such that  $|M_i(F, \text{ctr})| > n_i$  namely, the  $\text{snd}$  value  $k_i$  appears in  $(F, \text{ctr})$  more than  $n_i$  times. Let  $u \in V(F)$  be a  $k_i$ -center in  $F$ . The procedure aims to remove one of its  $k_i$ -members from  $F$  (possibly,  $u$  itself) in order to reduce



the number of appearances of the  $\text{snd}$  value  $k_i$  in  $F$  (hence reducing the deviation of  $(F, \text{ctr})$  from  $\bar{d}$ ). However, this cannot always be done directly, since removing vertices from  $F$  results in an undesired change in the degree of their neighbors, which might affect the  $\text{snd}$  values of other vertices in  $F$ . Hence, removing vertices from  $F$  needs to be done carefully. The method details, as well as formal code and analysis, are deferred to the full paper (see [36]).

► **Lemma 3.** *Given  $\bar{d}$  and  $(F, \text{ctr})$  such that  $cc(F) \geq 2$  and  $Dev(\bar{d}, (F, \text{ctr})) \geq 1$ ,  $\mathcal{LM}$  returns a U-realization  $(F', \text{ctr}')$  for  $\bar{d}$  such that  $cc(F') = cc(F) - 1$  and  $Dev(\bar{d}, (F', \text{ctr}')) = Dev(\bar{d}, (F, \text{ctr})) - 1$ .*

We now present a simple construction algorithm named *Connect*, based on performing only leaf merges. Given a profile  $\bar{d}$  and a U-realization  $(F, \text{ctr})$ , The algorithm produces a new U-realization  $(F', \text{ctr}')$  by repeatedly invoking  $\mathcal{LM}$  and applying leaf merges as long as possible. It halts when either the forest becomes connected, or the deviation becomes zero, yielding a proper realization.

Note that if the constructed  $(F', \text{ctr}')$  is a U-realization of  $\bar{d}$ , then  $Dev(\bar{d}, (F', \text{ctr}')) = 0$ . Otherwise,  $\mathcal{LM}$  was invoked exactly  $cc(F) - 1$  times, each decreasing the deviation by 1. We thus have the following.

► **Lemma 4.** *Let  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  be a profile with a U-realization  $(F, \text{ctr})$ . Also, let  $(F', \text{ctr}')$  be the construction returned by Algorithm *Connect*. Then*

$$Dev(\bar{d}, (F', \text{ctr}')) = \max\{Dev(\bar{d}, (F, \text{ctr})) - cc(F) + 1, 0\}.$$

**A basic lower bound on  $Dev(\bar{d})$ .** We next establish preliminary lower bounds on  $Dev(\bar{d})$  and on the number of connected components in specific types of U-realizations.

For a forest  $F$  and a vertex subset  $U \subseteq V(F)$ , denote by  $F[U]$  the induced forest on  $U$ 's vertices, i.e.,  $V(F[U]) = U$  and  $E(F[U]) = E(F) \cap (U \times U)$ . For an SNDF profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  with a U-realization  $(F, \text{ctr})$ , our first lower bound expression involves  $\bar{\rho}$  and  $\rho^\sigma$  as defined in Sect. 2, and  $cc(F[C])$ , the number of connected components in the *centers forest*  $F[C]$  induced by the centers:

$$LB_1^{dev} = LB_1^{dev}(\bar{d}, F, \text{ctr}) = \max \left\{ \sum_{i=1}^{\ell} \rho_i^\sigma - 2 \sum_{i=1}^{\ell} \sigma_i + cc(F[C]) + 1, 0 \right\}. \quad (6)$$

The intuition for the lower bound is as follows. To minimize the deviation, we use head and leaf merges, each of which reduces  $cc(F)$ , the number of connected components, by 1. Hence the total number of both head and leaf merges that can be performed on some U-realization  $(F, \text{ctr})$  is bounded by  $cc(F) - 1$ . However, a head-merge reduces the deviation by 2, whereas a leaf-merge reduces the deviation by 1. Since both of these merges “cost” the same (in the sense that they both decrease  $cc(F)$  by 1), it is desirable to use as many head-merges as possible.

Now consider a given U-realization  $(\tilde{F}, \tilde{\text{ctr}})$  consisting of  $s_i$  stars with  $k_i$  leaves for each  $i \in [\ell]$ , for some predetermined CSS  $\bar{s}$ . This U-realization has deviation  $Dev(\bar{d}, (\tilde{F}, \tilde{\text{ctr}})) = \sum_{i=1}^{\ell} \rho_i^s$  and the number of connected components in  $\tilde{F}$  is  $cc(\tilde{F}) = \sum_{i=1}^{\ell} s_i$ . In the best case, one can perform only head-merges to construct the final  $(F, \text{ctr})$ . Each head-merge reduces the deviation by 2 and  $cc(F)$  by 1, therefore

$$Dev(\bar{d}, (F, \text{ctr})) = \sum_{i=1}^{\ell} \rho_i^s - 2 \sum_{i=1}^{\ell} s_i + 2.$$

## 27:10 Selected Neighbor Degree Forest Realization

However, in many profiles it is not possible to use only head-merges. So assume we used only  $x$  head-merges, and removed  $2x$  excess vertices. Since the total number of merges is bounded by  $cc(F) - 1 = \sum_{i=1}^{\ell} s_i - 1$ , it is now possible to perform up to  $\sum_{i=1}^{\ell} s_i - x - 1$  leaf-merges, removing  $\sum_{i=1}^{\ell} s_i - x - 1$  additional excess vertices, totalling to  $\sum_{i=1}^{\ell} s_i + x - 1$  excess vertices being removed. Note that each head-merge adds an edge  $(u, v)$  for some  $u, v \in C$ , so we have  $x = |E(F[C])|$ , where  $E(F[C])$  is the set of edges in the centers forest  $F[C]$ . Since  $F[C]$  is a forest,

$$|E(F[C])| = |V(F[C])| - cc(F[C]) = \sum_{i=1}^{\ell} s_i - cc(F[C]),$$

and therefore  $x = \sum_{i=1}^{\ell} s_i - cc(F[C])$ . In summary, at most

$$\sum_{i=1}^{\ell} s_i + x - 1 = 2 \cdot \sum_{i=1}^{\ell} s_i - cc(F[C]) - 1$$

out of the  $\sum_{i=1}^{\ell} \rho_i^s$  initial excess vertices were removed.

This lower bound idea is formalized in the following lemma. (Proofs are deferred to the full paper, see [36].)

► **Lemma 5.** *For every profile  $\bar{d}$  and U-realization  $(F, ctr)$ :*

1.  $\sigma_i(F, ctr) \geq c_i$ , for every  $i$ ,
2.  $Dev(\bar{d}, (F, ctr)) \geq LB_1^{dev}$ .

### Reducing the error via head merges

We now present the main idea used later to construct an optimal U-realization  $(F, ctr)$  for a profile  $\bar{d}$ , namely, s.t.  $Dev(\bar{d}, (F, ctr)) = Dev(\bar{d})$ . Our construction has two stages. In the first, we select a CSS  $\bar{s} = (s_1, \dots, s_{\ell})$  specifying the number of  $k_i$ -centers for every  $i$ . The selection ensures  $\bar{s}$  is a CSS and has minimum deviation. The second stage builds a realization  $(F, ctr)$  that conforms to  $(\bar{d}, \bar{s})$ . A key observation, formalized by combining Lemmas 5(2) and 6, is that while  $(\bar{d}, \bar{s})$  has many different conforming realizations  $(F, ctr)$ , with different deviations, their deviations directly depend on  $cc(F[C])$ , the number of connected components in the centers forest  $F[C]$  (where  $C = C(F, ctr)$  is the set of centers). Recall that a head merge is performed by taking two centers  $u, v \in C$ , removing one neighbor with degree 1 from each, and connecting  $u, v$  by an edge, thus it decreases the deviation by 2 while decreasing  $cc(F[C])$  by 1. This means that in order to construct an optimal U-realization (with minimal  $cc(F[C])$ ) that conforms to  $(\bar{d}, \bar{s})$  for a CSS  $\bar{s}$ , we need to perform the maximal number of head merges. Suppose the resulting U-realization  $(F, ctr)$  allows no more head-merges. By Lemma 5(2),  $Dev(\bar{d}, (F, ctr)) \geq LB_1^{dev}(\bar{d}, F, ctr)$ . This bound can be matched by transforming  $(F, ctr)$  using leaf-merges.

► **Lemma 6.** *Consider a profile  $\bar{d} = (k_i^{n_i})_{i=1}^{\ell}$  with a U-realization  $(F, ctr)$ . There exists a U-realization  $(F', ctr')$  with deviation  $Dev(\bar{d}, (F', ctr')) = LB_1^{dev}(\bar{d}, F, ctr)$ .*

We conclude the discussion by stating that, given a profile  $\bar{d}$ , constructing an optimal U-realization for  $\bar{d}$  boils down to

1. Choosing the “right” CSS  $\bar{s}$ .
2. Constructing a U-realization  $(F, ctr)$  that conforms to  $(\bar{d}, \bar{s})$ , such that  $cc(F[C])$  is minimal among all other U-realizations with this property.

To formalize this workplan, we make the following definitions. Let  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  be a profile with star formation  $(c_i, \rho_i)_{i=1}^\ell$ , and let  $\bar{s}$  be a CSS for  $\bar{d}$ . Define the minimum number of connected components in a centers forest  $F[C]$  for  $F$  that admits a conforming realization for  $\bar{d}$  as

$$cc^*(\bar{d}, \bar{s}) = \min \{cc(F[C]) \mid \exists \text{ctr} : (F, \text{ctr}) \text{ conforms to } (\bar{d}, \bar{s})\}, \quad (7)$$

and define the expression obtained from Eq. (6) by replacing  $cc(F[C])$  with  $cc^*(\bar{d}, \bar{s})$  as

$$LB_3^{dev}(\bar{d}, \bar{s}) = \max \left\{ \sum_{i=1}^{\ell} \rho_i^s - 2 \sum_{i=1}^{\ell} s_i + cc^*(\bar{d}, \bar{s}) + 1, 0 \right\} \quad (8)$$

A CSS  $\bar{s}^*$  is *optimal* for  $\bar{d}$  if  $\bar{s}^*$  minimizes  $LB_3^{dev}(\bar{d}, \bar{s}^*)$  over all CSS's for  $\bar{d}$ . With the above definition, we state the following lemma.

► **Lemma 7.**  $Dev(\bar{d}) = LB_3^{dev}(\bar{d}, \bar{s}^*) = \min \{LB_3^{dev}(\bar{d}, \bar{s}) \mid \bar{s} \text{ is a CSS for } \bar{d}\}$ .

Hereafter, we focus on constructing, for a given profile  $\bar{d}$  and any  $\bar{s}$ , a leaf-covered U-realization  $(F, \text{ctr})$  conforming to  $(\bar{d}, \bar{s})$ . This U-realization has a minimal number of connected components in the centers forest, namely,  $cc(F[C]) = cc^*(\bar{d}, \bar{s})$ . In addition, in Section 4 we show how to find an optimal CSS  $\bar{s}^*$  for  $\bar{d}$ . Combining these results, Theorem 2 follows.

### Layer classification

To construct an optimal solution and analyze the structure of a profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$ , we classify each of its  $\ell$  layers according to the values  $s_i$  and the residues  $\rho_i^s$ ; later, the algorithm and analysis treat each class differently.

Consider a profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  with a U-realization  $(F, \text{ctr})$ . Recall that the centers forest  $F[C]$  is the induced forest of  $F$  on  $C$ . Note that for each  $u \in V(F)$ ,  $\text{ctr}(u) \in C$ . For  $\bar{d}$ ,  $(F, \text{ctr})$ , a residue sequence  $(\rho_i^s)_{i=1}^\ell$  and  $I \subseteq [\ell]$ , define the partial residual deviation of  $I$  as  $D_I = \sum_{i \in I} \rho_i^s$ .

For a profile  $\bar{d}$  and a CSS  $\bar{s}$ , define the following four sets of layers, referred to as *very good*, *good*, *bad*, and *very bad* layers.

$$\begin{aligned} VG &= \{i \mid \rho_i^s \leq s_i - 1\}, \\ G &= \{i \mid s_i \leq \rho_i^s \leq 2s_i - 1\}, \\ B &= \{i \mid 2s_i \leq \rho_i^s \leq 3s_i - 1\}, \\ VB &= \{i \mid \rho_i^s \geq 3s_i\}. \end{aligned}$$

As intuition for the terminology, note that a “very good” layer  $i \in VG$  can take care of its deviation on its own (namely, by merging its own stars into a single tree) using leaf-merges alone. A good layer  $i \in G$  can also take care of its deviation on its own, but it must apply some head-merges. In contrast, bad layers (in  $B \cup VB$ ) require the help of other layers in order to reduce their deviation.

In particular, our SNDF problem is easy for *benign profiles*, namely, profiles in which all layers are good or very good w.r.t.  $\bar{c}$ , as well as for profiles in which each layer is either very good for  $\bar{c}$  or  $(c_i = 1$  and  $\rho_i = 2)$ . (See the full paper or [36].) Hereafter, we consider only profiles that are not benign.

**The completion forest**

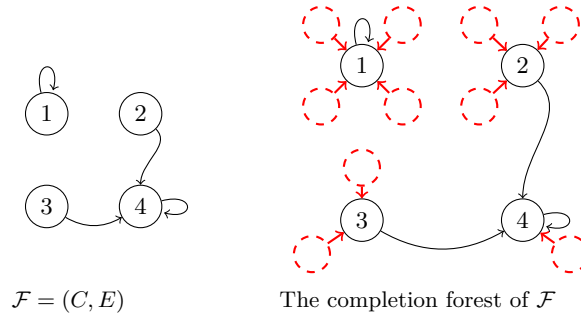
As described earlier, given a profile  $\bar{d}$  and a CSS  $\bar{s}$ , it is sufficient to construct a U-realization  $(F, \text{ctr})$  conforming to  $(\bar{d}, \bar{s})$ , such that the number of connected components in the centers forest  $cc(F[C])$  is minimal. The completion forest serves as a basic tool in our algorithm for constructing a U-realization for  $\bar{d}$ , where we first construct the centers forest and then add the non-centers. This tool allows us to focus only on the centers forest for some U-realization  $(F, \text{ctr})$ , and use its structure to deduce lower and upper bounds on properties of  $F$ .

Consider a profile  $\bar{d} = (k_i^{n_i})_{i=1}^\ell$  with star formation  $(c_i, \rho_i)_{i=1}^\ell$ , a forest  $\mathcal{F} = (C, E)$  and a neighbor function  $\text{ctr} : C \rightarrow C$  where  $\text{ctr}(u) \in N[u]$  for every  $u \in C$ . (Note that  $(\mathcal{F}, \text{ctr})$  is not a U-realization of  $\bar{d}$ .) Let  $\bar{C} = (C_1, \dots, C_\ell)$  be a partition of  $C$ . The tuple  $(\bar{d}, \mathcal{F}, \text{ctr}, \bar{C})$  is *legal* if  $\deg(u) \leq k_i$  for every  $u \in C_i$ .

The *completion forest for a legal*  $(\bar{d}, \mathcal{F}, \text{ctr}, \bar{C})$  is a pair  $(F, \text{ctr})$  constructed by the following Procedure *CompForest*. For every  $i$  and every  $u \in C_i$ , the procedure adds  $k_i - \deg(u)$  new vertices to the forest, connects them to  $u$  thus increasing its degree to  $k_i$  and making it a proper  $k_i$ -center. For each vertex  $v$  that was connected to  $u$ , the procedure sets  $\text{ctr}(v) = u$ , thus making it a  $k_i$ -member of its center.

Note that the tuple  $(\bar{d}, \mathcal{F}, \text{ctr}, \bar{C})$  is legal if and only if its completion forest is well defined, since it has to satisfy  $\deg(u) \leq k_i$  for each  $u \in C_i$ . Also note that the completion forest  $(F, \text{ctr})$  for  $(\bar{d}, \mathcal{F}, \text{ctr}, \bar{C})$  is not necessarily a U-realization of  $\bar{d}$ , since the number of  $k_i$ -members in  $(F, \text{ctr})$  might happen to be smaller than  $n_i$ . For convenience, we refer to  $(F, \text{ctr})$  as the completion forest for  $\mathcal{F}$ .

For example, consider the profile  $\bar{d} = (4^8, 3^6)$ , where  $k_1 = 4, k_2 = 3$ . Let  $(\mathcal{F} = (C, E), \text{ctr})$ , where the center set  $C = \{1, 2, 3, 4\}$  is partitioned into  $C_1 = \{1, 2\}$  and  $C_2 = \{3, 4\}$ , the sets of  $k_1$  and  $k_2$  centers respectively. The completion forest of the above tuple is defined by “completing” the neighborhoods of  $C$  centers according to  $\bar{d}$  and the layers of  $C$ . See Figure 6.



■ **Figure 6** Completion forest example.

**4 Lower bound and algorithm**

**Tight lower bound for SNDF profiles**

Finally, we derive a tight lower bound on the minimum deviation of a given SNDF profile. To do that, we first lower bound  $cc^*(\bar{d}, \bar{s})$ , defined in Eq. (7), and then combine it with our lower bound on  $LB_3^{dev}(\bar{d}, \bar{s})$  from Lemma 7. Our bound on  $cc^*(\bar{d}, \bar{s})$  depends only on the CSS  $\bar{s}$  and the profile  $\bar{d}$ . Our method of proving the bound is by considering an arbitrary U-realization  $(F, \text{ctr})$ , that conforms to  $(\bar{d}, \bar{s})$ , and proving a lower bound on  $cc(F[C])$ , which

imply a lower bound on  $cc^*(\bar{d}, \bar{s})$ . This is done by decomposing the centers forest  $F[C]$  in a special manner, and then reconstructing  $F[C]$  while keeping track of the number of its connected components. As mentioned earlier, we may assume that the profile  $\bar{d}$  is not benign.

The entire derivation of the lower bound is deferred to the full paper (see [36]). Letting

$$LB_4^{dev}(\bar{d}, \bar{s}) = \max \left\{ 0, 2 + \sum_{i=1}^{\ell} \rho_i^s - 2 \sum_{i=1}^{\ell} s_i \right. \\ \left. + \max \left\{ \sum_{i \in VG} s_i - \left( D_{VG} + \sum_{i \in GUB} \lfloor (\rho_i^s - s_i)/2 \rfloor + \sum_{i \in VB} s_i \right), 0 \right\} \right\}$$

we get the following.

► **Theorem 8.** *Let  $\bar{d} = (k_i^{n_i})_{i=1}^{\ell}$  be an SNDF-profile and let  $\bar{s}'$  be a CSS for  $\bar{d}$ . Assume that  $\bar{s}'$  minimizes  $LB_4^{dev}$  over all choices of CSS  $\bar{s}$  (Note that  $\bar{s}'$  does not necessarily minimize  $LB_3^{dev}$ ). Then  $Dev(\bar{d}) \geq LB_4^{dev}(\bar{d}, \bar{s}')$ .*

### Optimal algorithm

While our algorithm is based on the components introduced above, its actual operation is rather involved, hence its description is deferred to the full paper (see [36]) due to space constraints. It provides an optimal explicit construction of SNDF-realizations, which for a given profile  $\bar{d}$  and sequence  $\bar{s}$  yields a realization with deviation at most  $LB_4^{dev}(\bar{d}, \bar{s})$ , and also show how to select a sequence  $\bar{s}'$  that minimizes  $LB_4^{dev}$ . Combining the above with Theorem 8 yields an explicit construction for optimal realizations. In the full paper (see [36]) we also show a more efficient solution for the decision version of SNDF, namely, decide if a given SNDF-profile  $\bar{d}$  is realizable (i.e.,  $Dev(\bar{d}) = 0$ ).

## 5 Discussion

In this paper we introduced the selected neighbor degree realization problem and solved it when the graph is required to be acyclic. We presented a necessary and sufficient condition for realizability. In addition, we provided an algorithm that given a specification computes an upper realization with minimum deviation from the given specification. In particular, if the specification is realizable, the algorithm computes a realization.

A natural open question is to solve the realization problem on general graphs. One may also consider the problem on other graph families, such as trees or bipartite graphs. (Note that the realization problem is easy in regular graphs.) Finally, another interesting direction for future study is to consider variants of the realization problem on directed graphs.

---

### References

- 1 M. Aigner and E. Triesch. Realizability and uniqueness in graphs. *Discrete Math.*, 136:3–20, 1994.
- 2 Takao Asano. Graphical degree sequence problems with connectivity requirements. In *4th ISAAC*, volume 762 of *LNCS*, pages 38–47, 1993.
- 3 A. Bar-Noy, K. Choudhary, A. Cohen, D. Peleg, and D. Rawitz. Minimum neighboring degree realization in graphs and trees. In *28th ESA*, volume 173 of *LIPICs*, pages 10:1–10:15, 2020.
- 4 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Realizability of graph specifications: Characterizations and algorithms. In *25th International Colloquium on Structural Information and Communication Complexity*, volume 11085 of *LNCS*, pages 3–13, 2018.

- 5 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph profile realizations and applications to social networks. In *13th WALCOM*, volume 11355 of *LNCS*, pages 3–14, 2019.
- 6 Amotz Bar-Noy, Keerti Choudhary, David Peleg, and Dror Rawitz. Graph realizations: Maximum degree in vertex neighborhoods. In *17th SWAT*, volume 162 of *LIPICs*, pages 10:1–10:17, 2020.
- 7 Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Composed degree-distance realizations of graphs. In *32nd IWOCA*, volume 12757 of *LNCS*, pages 63–77, 2021.
- 8 Vladimir Batagelj, Tomaz Pisanski, and J. M. S. Simões-Pereira. An algorithm for tree-realizability of distance matrices. *Int. J. Comput. Math.*, 34(3-4):171–176, 1990.
- 9 Joseph K. Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, 6(4):489–522, 2011.
- 10 W. Chou and H. Frank. Survivable communication networks and the terminal capacity matrix. *IEEE Trans. Circuit Theory*, CT-17:192–197, 1970.
- 11 S.A. Choudum. A simple proof of the Erdős-Gallai theorem on graph sequences. *Bulletin of the Australian Mathematical Society*, 33:67–70, 1986.
- 12 Fan R. K. Chung, Mark W. Garrett, Ronald L. Graham, and David Shallcross. Distance realization problems with applications to internet tomography. *J. Comput. Syst. Sci.*, 63(3):432–448, 2001.
- 13 Brian Cloteaux. Fast sequential creation of random realizations of degree sequences. *Internet Mathematics*, 12(3):205–219, 2016.
- 14 Joseph C. Culberson and Piotr Rudnicki. A fast algorithm for constructing trees from distance matrices. *Inf. Process. Lett.*, 30(4):215–220, 1989.
- 15 P. Erdős and T. Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.
- 16 G. Gupta, P. Joshi, and A. Tripathi. Graphic sequences of trees and a problem of frobenius. *Czechoslovak Math. J.*, 57:49–52, 2007.
- 17 S. Louis Hakimi and S. S. Yau. Distance matrix of a graph and its realizability. *Quart. Appl. Math.*, 22:305–317, 1965.
- 18 S.L. Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph –I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.
- 19 V. Havel. A remark on the existence of finite graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.
- 20 P.J. Kelly. A congruence theorem for trees. *Pacific J. Math.*, 7:961–968, 1957.
- 21 Milena Mihail and Nisheeth Vishnoi. On generating graphs with prescribed degree sequences for complex network modeling applications. *3rd Workshop on Approximation and Randomization Algorithms in Communication Networks*, 2002.
- 22 Elchanan Mossel and Nathan Ross. Shotgun assembly of labeled graphs. *CoRR*, abs/1504.07682, 2015. [arXiv:1504.07682](https://arxiv.org/abs/1504.07682).
- 23 Juhani Nieminen. Realizing the distance matrix of a graph. *J. Inf. Process. Cybern.*, 12(1/2):29–31, 1976.
- 24 P.V. O’Neil. Ulam’s conjecture and graph reconstructions. *Amer. Math. Monthly*, 77:35–43, 1970.
- 25 A. N. Patrinos and S. L. Hakimi. The distance matrix of a graph and its tree realizability. *Quarterly of Applied Math.*, 255, 1972.
- 26 G. Sierksma and H. Hoogeveen. Seven criteria for integer sequences being graphic. *J. Graph Theory*, 15:223–231, 1991.
- 27 J. M. S. Simões-Pereira. A note on the tree realizability of a distance matrix. *J. Combinat. Theory B*, 6:303–310, 1969.
- 28 J. M. S. Simões-Pereira. A note on distance matrices with unicyclic graph realizations. *Discrete Mathematics*, 65:277–287, 1987.

- 29 J. M. S. Simões-Pereira. An algorithm and its role in the study of optimal graph realizations of distance matrices. *Discrete Mathematics*, 79(3):299–312, 1990.
- 30 Hiroshi Tamura, Masakazu Sengoku, Shoji Shinoda, and Takeo Abe. Realization of a network from the upper and lower bounds of the distances (or capacities) between vertices. In *IEEE ISCAS*, pages 2545–2548. IEEE, 1993.
- 31 A. Tripathi and H. Tyagi. A simple criterion on degree sequences of graphs. *Discrete Applied Math.*, 156:3513–3517, 2008.
- 32 S.M. Ulam. *A collection of mathematical problems*. Wiley, 1960.
- 33 Sacha C. Varone. A constructive algorithm for realizing a distance matrix. *Eur. J. Oper. Res.*, 174(1):102–111, 2006.
- 34 D.L. Wang and D.J. Kleitman. On the existence of  $n$ -connected graphs with prescribed degrees ( $n > 2$ ). *Networks*, 3:225–239, 1973.
- 35 N.C. Wormald. Models of random regular graphs. *Surveys in combinatorics*, 267:239–298, 1999.
- 36 Elad Yehezkel. Selected neighbor degree tree realization. Master’s thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, 2020.
- 37 K. A. Zaretskii. Constructing a tree on the basis of a set of distances between the hanging vertices. *Uspekhi Mat. Nauk*, 20:90–92, 1965.






# MAX CUT in Weighted Random Intersection Graphs and Discrepancy of Sparse Random Set Systems

Sotiris Nikolettseas  

Computer Engineering & Informatics Department, University of Patras, Greece  
Computer Technology Institute, Patras, Greece

Christoforos Raptopoulos<sup>1</sup>  

Computer Engineering & Informatics Department, University of Patras, Greece

Paul Spirakis  

Department of Computer Science, University of Liverpool, UK  
Computer Engineering & Informatics Department, University of Patras, Greece

---

## Abstract

Let  $V$  be a set of  $n$  vertices,  $\mathcal{M}$  a set of  $m$  labels, and let  $\mathbf{R}$  be an  $m \times n$  matrix of independent Bernoulli random variables with probability of success  $p$ ; columns of  $\mathbf{R}$  are incidence vectors of label sets assigned to vertices. A random instance  $G(V, E, \mathbf{R}^T \mathbf{R})$  of the weighted random intersection graph model is constructed by drawing an edge with weight equal to the number of common labels (namely  $[\mathbf{R}^T \mathbf{R}]_{v,u}$ ) between any two vertices  $u, v$  for which this weight is strictly larger than 0. In this paper we study the average case analysis of WEIGHTED MAX CUT, assuming the input is a weighted random intersection graph, i.e. given  $G(V, E, \mathbf{R}^T \mathbf{R})$  we wish to find a partition of  $V$  into two sets so that the total weight of the edges having exactly one endpoint in each set is maximized.

In particular, we initially prove that the weight of a maximum cut of  $G(V, E, \mathbf{R}^T \mathbf{R})$  is concentrated around its expected value, and then show that, when the number of labels is much smaller than the number of vertices (in particular,  $m = n^\alpha, \alpha < 1$ ), a random partition of the vertices achieves asymptotically optimal cut weight with high probability. Furthermore, in the case  $n = m$  and constant average degree (i.e.  $p = \frac{\Theta(1)}{n}$ ), we show that with high probability, a majority type randomized algorithm outputs a cut with weight that is larger than the weight of a random cut by a multiplicative constant strictly larger than 1. Then, we formally prove a connection between the computational problem of finding a (weighted) maximum cut in  $G(V, E, \mathbf{R}^T \mathbf{R})$  and the problem of finding a 2-coloring that achieves minimum discrepancy for a set system  $\Sigma$  with incidence matrix  $\mathbf{R}$  (i.e. minimum imbalance over all sets in  $\Sigma$ ). We exploit this connection by proposing a (weak) bipartization algorithm for the case  $m = n, p = \frac{\Theta(1)}{n}$  that, when it terminates, its output can be used to find a 2-coloring with minimum discrepancy in a set system with incidence matrix  $\mathbf{R}$ . In fact, with high probability, the latter 2-coloring corresponds to a bipartition with maximum cut-weight in  $G(V, E, \mathbf{R}^T \mathbf{R})$ . Finally, we prove that our (weak) bipartization algorithm terminates in polynomial time, with high probability, at least when  $p = \frac{c}{n}, c < 1$ .

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Random graphs

**Keywords and phrases** Random Intersection Graphs, Maximum Cut, Discrepancy

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.28

**Related Version** *Full Version*: <https://arxiv.org/abs/2009.01567> [17]

**Funding** *Christoforos Raptopoulos*: Supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “2nd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers” (Project Number:704).

---

<sup>1</sup> Corresponding author



*Paul Spirakis:* Supported by NeST initiative of the School of EEE and CS at the U. of Liverpool and by the EPSRC grant EP/P02002X/1

## 1 Introduction

Given an undirected graph  $G(V, E)$ , the MAX CUT problem asks for a partition of the vertices of  $G$  into two sets, such that the number of edges with exactly one endpoint in each set of the partition is maximized. This problem can be naturally generalized for weighted (undirected) graphs. A weighted graph is denoted by  $G(V, E, \mathbf{W})$ , where  $V$  is the set of vertices,  $E$  is the set of edges and  $\mathbf{W}$  is a weight matrix, which specifies a weight  $\mathbf{W}_{i,j} = w_{i,j}$ , for each pair of vertices  $i, j$ . In particular, we assume that  $\mathbf{W}_{i,j} = 0$ , for each edge  $\{i, j\} \notin E$ .

► **Definition 1** (WEIGHTED MAX CUT). *Given a weighted graph  $G(V, E, \mathbf{W})$ , find a partition of  $V$  into two (disjoint) subsets  $A, B$ , so as to maximize the cumulative weight of the edges of  $G$  having one endpoint in  $A$  and the other in  $B$ .*

WEIGHTED MAX CUT is fundamental in theoretical computer science and is relevant in various graph layout and embedding problems [10]. Furthermore, it also has many practical applications, including infrastructure cost and circuit layout optimization in network and VLSI design [20], minimizing the Hamiltonian of a spin glass model in statistical physics [3], and data clustering [19]. In the worst case MAX CUT (and also WEIGHTED MAX CUT) is APX-hard, meaning that there is no polynomial-time approximation scheme that finds a solution that is arbitrarily close to the optimum, unless  $P = NP$  [18].

The average case analysis of MAX CUT, namely the case where the input graph is chosen at random from a probabilistic space of graphs, is also of considerable interest and is further motivated by the desire to justify and understand why various graph partitioning heuristics work well in practical applications. In most research works the input graphs are drawn from the Erdős-Rényi random graphs model  $\mathcal{G}_{n,m}$ , i.e. random instances are drawn equiprobably from the set of simple undirected graphs on  $n$  vertices and  $m$  edges, where  $m$  is a linear function of  $n$  (see also [13, 7] for the average case analysis of MAX CUT and its generalizations with respect to other random graph models). One of the earliest results in this area is that MAX CUT undergoes a phase transition on  $\mathcal{G}_{n,\gamma n}$  at  $\gamma = \frac{1}{2}$  [8], in that the difference between the number of edges of the graph and the Max-Cut size is  $O(1)$ , for  $\gamma < \frac{1}{2}$ , while it is  $\Omega(n)$ , when  $\gamma > \frac{1}{2}$ . For large values of  $\gamma$ , it was proved in [4] that the maximum cut size of  $\mathcal{G}_{n,\gamma n}$  normalized by the number of vertices  $n$  reaches an absolute limit in probability as  $n \rightarrow \infty$ , but it was not until recently that the latter limit was established and expressed analytically in [9], using the interpolation method; in particular, it was shown to be asymptotically equal to  $(\frac{\gamma}{2} + P_* \sqrt{\frac{\gamma}{2}})n$ , where  $P_* \approx 0.7632$ . We note however that these results are existential, and thus do not lead to an efficient approximation scheme for finding a tight approximation of the maximum cut with large enough probability when the input graph is drawn from  $\mathcal{G}_{n,\gamma n}$ . An efficient approximation scheme in this case was designed in [8], and it was proved that, with high probability, this scheme constructs a cut with at least  $(\frac{\gamma}{2} + 0.37613\sqrt{\gamma})n = (1 + 0.75226\frac{1}{\sqrt{\gamma}})\frac{\gamma}{2}n$  edges, noting that  $\frac{\gamma}{2}n$  is the size of a random cut (in which each vertex is placed independently and equiprobably in one of the two sets of the partition). Whether there exists an efficient approximation scheme that can close the gap between the approximation guarantee of [8] and the limit of [9] remains an open problem.

In this paper, we study the average case analysis of WEIGHTED MAX CUT when input graphs are drawn from the generalization of another well-established model of random graphs, namely the *weighted random intersection graphs model* (the unweighted version of the model

was initially defined in [15]). In this model, edges are formed through the intersection of label sets assigned to each vertex and edge weights are equal to the number of common labels between edgepoints.

► **Definition 2** (Weighted random intersection graph). *Consider a universe  $\mathcal{M} = \{1, 2, \dots, m\}$  of labels and a set of  $n$  vertices  $V$ . We define the  $m \times n$  representation matrix  $\mathbf{R}$  whose entries are independent Bernoulli random variables with probability of success  $p$ . For  $\ell \in \mathcal{M}$  and  $v \in V$ , we say that vertex  $v$  has chosen label  $\ell$  iff  $\mathbf{R}_{\ell,v} = 1$ . Furthermore, we draw an edge with weight  $[\mathbf{R}^T \mathbf{R}]_{v,u}$  between any two vertices  $u, v$  for which this weight is strictly larger than 0. The weighted graph  $G = (V, E, \mathbf{R}^T \mathbf{R})$  is then a random instance of the weighted random intersection graphs model  $\overline{\mathcal{G}}_{n,m,p}$ .*

Random intersection graphs are relevant to and capture quite nicely social networking; vertices are the individual actors and labels correspond to specific types of interdependency. Other applications include oblivious resource sharing in a (general) distributed setting, efficient and secure communication in sensor networks [20], interactions of mobile agents traversing the web etc. (see e.g. the survey papers [6, 16] for further motivation and recent research related to random intersection graphs). In all these settings, weighted random intersection graphs, in particular, also capture the strength of connections between actors (e.g. in a social network, individuals having several characteristics in common have more intimate relationships than those that share only a few common characteristics). One of the most celebrated results in this area is equivalence (measured in terms of total variation distance) of random intersection graphs and Erdős-Rényi random graphs when the number of labels satisfies  $m = n^\alpha, \alpha > 6$  [12]. This bound on the number of labels was improved in [23], by showing equivalence of sharp threshold functions among the two models for  $\alpha \geq 3$ . Similarity of the two models has been proved even for smaller values of  $\alpha$  (e.g. for any  $\alpha > 1$ ) in the form of various translation results (see e.g. Theorem 1 in [22]), suggesting that some algorithmic ideas developed for Erdős-Rényi random graphs also work for random intersection graphs (and also weighted random intersection graphs).

In view of this, in the present paper we study the average case analysis of WEIGHTED MAX CUT under the weighted random intersection graphs model, for the range  $m = n^\alpha, \alpha \leq 1$  for two main reasons: First, the average case analysis of MAX CUT has not been considered in the literature so far when the input is drawn from the random intersection graphs model, and thus the asymptotic behaviour of the maximum cut remains unknown especially for the range of values where random intersection graphs and Erdős-Rényi random graphs differ the most. Furthermore, studying a model where we can implicitly control its intersection number (indeed  $m$  is an obvious upper bound on the number of cliques that can cover all edges of the graph) may help understand algorithmic bottlenecks for finding maximum cuts in Erdős-Rényi random graphs.

Second, we note that the representation matrix  $\mathbf{R}$  of a weighted random intersection graph can be used to define a random set system  $\Sigma$  consisting of  $m$  sets  $\Sigma = \{L_1, \dots, L_m\}$ , where  $L_\ell$  is the set of vertices that have chosen label  $\ell$ ; we say that  $\mathbf{R}$  is the *incidence matrix* of  $\Sigma$ . Therefore, there is a natural connection between WEIGHTED MAX CUT and the DISCREPANCY of such random set systems, which we formalize in this paper. In particular, given a set system  $\Sigma$  with incidence matrix  $\mathbf{R}$ , its *discrepancy* is defined as  $\text{disc}(\Sigma) = \min_{\mathbf{x} \in \{\pm 1\}^n} \max_{L \in \Sigma} \left| \sum_{v \in L} x_v \right| = \|\mathbf{R}\mathbf{x}\|_\infty$ , i.e. it is the minimum imbalance of all sets in  $\Sigma$  over all 2-colorings  $\mathbf{x}$ . Recent work on the discrepancy of random rectangular matrices defined as above [1] has shown that, when the number of labels (sets)  $m$  satisfies  $n \geq 0.73m \log m$ , the discrepancy of  $\Sigma$  is at most 1 with high probability. The proof of the main result in [1] is based on a conditional second moment method combined with Stein's

method of exchangeable pairs, and improves upon a Fourier analytic result of [14], and also upon previous results in [11, 21]. The design of an efficient algorithm that can find a 2-coloring having discrepancy  $O(1)$  in this range still remains an open problem. Approximation algorithms for a similar model for random set systems were designed and analyzed in [2]; however, the algorithmic ideas there do not apply in our case.

## 1.1 Our Contribution

In this paper, we introduce the model of weighted random intersection graphs and we study the average case analysis of WEIGHTED MAX CUT through the prism of DISCREPANCY of random set systems. We formalize the connection between these two combinatorial problems for the case of arbitrary weighted intersection graphs in Corollary 4. We prove that, given a weighted intersection graph  $G = (V, E, \mathbf{R}^T \mathbf{R})$  with representation matrix  $\mathbf{R}$ , and a set system with incidence matrix  $\mathbf{R}$ , such that  $\text{disc}(\Sigma) \leq 1$ , a 2-coloring has maximum cut weight in  $G$  if and only if it achieves minimum discrepancy in  $\Sigma$ . In particular, Corollary 4 applies in the range of values considered in [1] (i.e.  $n \geq 0.73m \log m$ ), and thus any algorithm that finds a maximum cut in  $G(V, E, \mathbf{R}^T \mathbf{R})$  with large enough probability can also be used to find a 2-coloring with minimum discrepancy in a set system  $\Sigma$  with incidence matrix  $\mathbf{R}$ , with the same probability of success.

We then consider weighted random intersection graphs in the case  $m = n^\alpha, \alpha \leq 1$ , and we prove that the maximum cut weight of a random instance  $G(V, E, \mathbf{R}^T \mathbf{R})$  of  $\bar{\mathcal{G}}_{n,m,p}$  concentrates around its expected value (see Theorem 5). In particular, with high probability (whp, i.e. with probability tending to 1 as  $n \rightarrow \infty$ ) over the choices of  $\mathbf{R}$ ,  $\text{Max-Cut}(G) \sim \mathbb{E}_{\mathbf{R}}[\text{Max-Cut}(G)]$ , where  $\mathbb{E}_{\mathbf{R}}$  denotes expectation with respect to  $\mathbf{R}$ . The proof is based on the Efron-Stein inequality for upper bounding the variance of the maximum cut. As a consequence of our concentration result, we prove in Theorem 6 that, in the case  $\alpha < 1$ , a random 2-coloring (i.e. bipartition)  $\mathbf{x}^{(rand)}$  in which each vertex chooses its color independently and equiprobably, has cut weight asymptotically equal to  $\text{Max-Cut}(G)$ , with high probability over the choices of  $\mathbf{x}^{(rand)}$  and  $\mathbf{R}$ .

The latter result on random cuts allows us to focus the analysis of our randomized algorithms of Section 4 on the case  $m = n$  (i.e.  $\alpha = 1$ ), and  $p = \frac{c}{n}$ , for some constant  $c$  (see also the discussion at the end of Subsection 3.1), where the assumptions of Theorem 6 do not hold. It is worth noting that, in this range of values, the expected weight of a fixed edge in a weighted random intersection graph is equal to  $mp^2 = \Theta(1/n)$ , and thus we hope that our work here will serve as an intermediate step towards understanding when algorithmic bottlenecks for MAX CUT appear in sparse random graphs (especially Erdős-Rényi random graphs) with respect to the intersection number. In particular, we analyze a Majority Cut Algorithm 1 that extends the algorithmic idea of [8] to weighted intersection graphs as follows: vertices are colored sequentially (each color  $+1$  or  $-1$  corresponding to a different set in the partition of the vertices), and the  $t$ -th vertex is colored opposite to the sign of  $\sum_{i \in [t-1]} [\mathbf{R}^T \mathbf{R}]_{i,t} x_i$ , namely the total available weight of its incident edges, taking into account colors of adjacent vertices. Our average case analysis of the Majority Cut Algorithm shows that, when  $m = n$  and  $p = \frac{c}{n}$ , for large constant  $c$ , with high probability over the choices of  $\mathbf{R}$ , the expected weight of the constructed cut is at least  $1 + \beta$  times larger than the expected weight of a random cut, for some constant  $\beta = \beta(c) \geq \sqrt{\frac{16}{27\pi c^3}} - o(1)$ . The fact that the lower bound on beta is inversely proportional to  $c^{3/2}$  was to be expected, because, as  $p$  increases, the approximation of the maximum cut that we get from the weight of a random cut improves (see also the discussion at the end of Subsection 3.1).

In Subsection 4.2 we propose a framework for finding maximum cuts in weighted random intersection graphs for  $m = n$  and  $p = \frac{c}{n}$ , for constant  $c$ , by exploiting the connection between WEIGHTED MAX CUT and the problem of discrepancy minimization in random set systems. In particular, we design a Weak Bipartization Algorithm 2, that takes as input an intersection graph with representation matrix  $\mathbf{R}$  and outputs a subgraph that is “almost” bipartite. In fact, the input intersection graph is treated as a multigraph composed by overlapping cliques formed by the label sets  $L_\ell = \{v : \mathbf{R}_{\ell,v} = 1\}, \ell \in \mathcal{M}$ . The algorithm attempts to destroy all odd cycles of the input (except from odd cycles that are formed by labels with only two vertices) by replacing each clique induced by some label set  $L_\ell$  by a random maximal matching. In Theorem 11 we prove that, with high probability over the choices of  $\mathbf{R}$ , if the Weak Bipartization Algorithm terminates, then its output can be used to construct a 2-coloring that has minimum discrepancy in a set system with incidence matrix  $\mathbf{R}$ , which also gives a maximum cut in  $G(V, E, \mathbf{R}^T \mathbf{R})$ . It is worth noting that this does not follow from Corollary 4, because a random set system with incidence matrix  $\mathbf{R}$  has discrepancy larger than 1 with (at least) constant probability when  $m = n$  and  $p = \frac{c}{n}$ . Our proof relies on a structural property of closed 0-strong vertex-label sequences (loosely defined as closed walks of edges formed by distinct labels) in the weighted random intersection graph  $G(V, E, \mathbf{R}^T \mathbf{R})$  (Lemma 8). Finally, in Theorem 12, we prove that our Weak Bipartization Algorithm terminates in polynomial time, with high probability, if the constant  $c$  is strictly less than 1. Therefore, there is a polynomial time algorithm for finding weighted maximum cuts, with high probability, when the input is drawn from  $\bar{\mathcal{G}}_{n,n,\frac{c}{n}}$ , with  $c < 1$ . We believe that this part of our work may also be of interest regarding the design of efficient algorithms for finding minimum discrepancy colorings in random set systems.

Due to lack of space, some proofs have been omitted; the full proofs of this paper can be found in our technical report [17].

## 2 Notation and preliminary results

We denote weighted undirected graphs by  $G(V, E, \mathbf{W})$ ; in particular,  $V = V(G)$  (resp.  $E = E(G)$ ) is the set of vertices (resp. set of edges) and  $\mathbf{W} = \mathbf{W}(G)$  is the weight matrix, i.e.  $\mathbf{W}_{i,j} = w_{i,j}$  is the weight of (undirected) edge  $\{i, j\} \in E$ . We allow  $\mathbf{W}$  to have non-zero diagonal entries, as these do not affect cut weights. We also denote the number of vertices by  $n$ , and we use the notation  $[n] = \{1, 2, \dots, n\}$ . We also use this notation to define parts of matrices, for example  $\mathbf{W}_{[n],1}$  denotes the first column of the weight matrix.

A bipartition of the sets of vertices is a partition of  $V$  into two sets  $A, B$  such that  $A \cap B = \emptyset$  and  $A \cup B = V$ . Bipartitions correspond to 2-colorings, which we denote by vectors  $\mathbf{x}$  such that  $x_i = +1$  if  $i \in A$  and  $x_i = -1$  if  $i \in B$ .

Given a weighted graph  $G(V, E, \mathbf{W})$ , we denote by  $\text{Cut}(G, \mathbf{x})$  the weight of a cut defined by a bipartition  $\mathbf{x}$ , namely  $\text{Cut}(G, \mathbf{x}) = \sum_{\{i,j\} \in E: i \in A, j \in B} w_{i,j} = \frac{1}{4} \sum_{\{i,j\} \in E} w_{i,j} (x_i - x_j)^2$ . The maximum cut of  $G$  is  $\text{Max-Cut}(G) = \max_{\mathbf{x} \in \{-1, +1\}^n} \text{Cut}(G, \mathbf{x})$ .

For a weighted random intersection graph  $G(V, E, \mathbf{R}^T \mathbf{R})$  with representation matrix  $\mathbf{R}$ , we denote by  $S_v$  the set of labels chosen by vertex  $v \in V$ , i.e.  $S_v = \{\ell : \mathbf{R}_{\ell,v} = 1\}$ . Furthermore, we denote by  $L_\ell$  the set of vertices having chosen label  $\ell$ , i.e.  $L_\ell = \{v : \mathbf{R}_{\ell,v} = 1\}$ . Using this notation, the weight of an edge  $\{v, u\} \in E$  is  $|S_v \cap S_u|$ ; notice also that this is equal to 0 when  $\{v, u\} \notin E$ . We also note here that we may also think of a weighted random intersection graph as a simple weighted graph where, for any pair of vertices  $v, u$ , there are  $|S_v \cap S_u|$  simple edges between them.

A set system  $\Sigma$  defined on a set  $V$  is a family of sets  $\Sigma = \{L_1, L_2, \dots, L_m\}$ , where  $L_\ell \subseteq V, \ell \in [m]$ . The incidence matrix of  $\Sigma$  is an  $m \times n$  matrix  $\mathbf{R} = \mathbf{R}(\Sigma)$ , where for any  $\ell \in [m], v \in [n]$ ,  $\mathbf{R}_{\ell,v} = 1$  if  $v \in S_\ell$  and 0 otherwise. The discrepancy of  $\Sigma$  with respect to a 2-coloring  $\mathbf{x}$  of the vertices in  $V$  is  $\text{disc}(\Sigma, \mathbf{x}) = \max_{\ell \in [m]} |\sum_{v \in V} \mathbf{R}_{\ell,v} x_v| = \|\mathbf{R}\mathbf{x}\|_\infty$ . The discrepancy of  $\Sigma$  is  $\text{disc}(\Sigma) = \min_{\mathbf{x} \in \{-1, +1\}^n} \text{disc}(\Sigma, \mathbf{x})$ .

It is well-known that the cut size of a bipartition of the set of vertices of a graph  $G(V, E)$  into sets  $A$  and  $B$  is given by  $\frac{1}{4} \sum_{\{i,j\} \in E} (x_i - x_j)^2$ , where  $x_i = +1$  if  $i \in A$  and  $x_i = -1$  if  $i \in B$ . This can be naturally generalized for multigraphs and also for weighted graphs. In particular, the Max-Cut size of a weighted graph  $G(V, E, \mathbf{W})$  is given by

$$\text{Max-Cut}(G) = \max_{\mathbf{x} \in \{-1, +1\}^n} \frac{1}{4} \sum_{\{i,j\} \in E} \mathbf{W}_{i,j} (x_i - x_j)^2. \quad (1)$$

In particular, we get the following Corollary (refer to the full version of our paper [17] for the proof):

► **Corollary 3.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a weighted intersection graph with representation matrix  $\mathbf{R}$ . Then, for any  $\mathbf{x} \in \{-1, +1\}^n$ ,*

$$\text{Cut}(G, \mathbf{x}) = \frac{1}{4} \left( \sum_{i,j \in [n]^2} [\mathbf{R}^T \mathbf{R}]_{i,j} - \|\mathbf{R}\mathbf{x}\|^2 \right) \quad (2)$$

and so

$$\text{Max-Cut}(G) = \frac{1}{4} \left( \sum_{i,j \in [n]^2} [\mathbf{R}^T \mathbf{R}]_{i,j} - \min_{\mathbf{x} \in \{-1, +1\}^n} \|\mathbf{R}\mathbf{x}\|^2 \right), \quad (3)$$

where  $\|\cdot\|$  denotes the 2-norm. In particular, the expectation of the size of a random cut, where each entry of  $\mathbf{x}$  is independently and equiprobably either  $+1$  or  $-1$  is equal to  $\mathbb{E}_{\mathbf{x}} [\text{Cut}(G, \mathbf{x})] = \frac{1}{4} \sum_{i \neq j, i,j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i,j}$ , where  $\mathbb{E}_{\mathbf{x}}$  denotes expectation with respect to  $\mathbf{x}$ .

Since  $\sum_{i,j \in [n]^2} [\mathbf{R}^T \mathbf{R}]_{i,j}$  is fixed for any given representation matrix  $\mathbf{R}$ , the above Corollary implies that, to find a bipartition of the vertex set  $V$  that corresponds to a maximum cut, we need to find an  $n$ -dimensional vector in  $\arg \min_{\mathbf{x} \in \{-1, +1\}^n} \|\mathbf{R}\mathbf{x}\|^2$ . We thus get the following (refer to the full version of our paper [17] for the proof):

► **Corollary 4.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a weighted intersection graph with representation matrix  $\mathbf{R}$  and  $\Sigma$  a set system with incidence matrix  $\mathbf{R}$ . If  $\text{disc}(\Sigma) \leq 1$ , then  $\mathbf{x}^* \in \arg \min_{\mathbf{x} \in \{-1, +1\}^n} \|\mathbf{R}\mathbf{x}\|^2$  if and only if  $\mathbf{x}^* \in \arg \min_{\mathbf{x} \in \{-1, +1\}^n} \text{disc}(\Sigma, \mathbf{x})$ . In particular, if the minimum discrepancy of  $\Sigma$  is at most 1, a bipartition corresponds to a maximum cut iff it achieves minimum discrepancy.*

Notice that above result is not necessarily true when  $\text{disc}(\Sigma) > 1$ , since the minimum of  $\|\mathbf{R}\mathbf{x}\|$  could be achieved by 2-colorings with larger discrepancy than the optimal.

## 2.1 Range of values for $p$

Concerning the success probability  $p$ , we note that, when  $p = o\left(\sqrt{\frac{1}{nm}}\right)$ , direct application of the results of [5] suggest that  $G(V, E, \mathbf{R}^T \mathbf{R})$  is chordal with high probability, but in fact the same proofs reveal that a stronger property holds, namely that there is no closed vertex-label



sequence (refer to the precise definition in Subsection 4.2) having distinct labels. Therefore, in this case, finding a bipartition with maximum cut weight is straightforward: indeed, one way to construct a maximum cut is to run our Weak Bipartization Algorithm 2 from Subsection 4.2, and then to apply Theorem 11 (noting that Weak Bipartization termination condition trivially holds, since the set  $\mathcal{C}_{\text{odd}}(G^{(b)})$  defined in Subsection 4.2 is empty). Furthermore, even though we consider weighted graphs, we will also assume that  $mp^2 = O(1)$ , noting that, otherwise,  $G(V, E, \mathbf{R}^T \mathbf{R})$  will be almost complete with high probability (indeed, the unconditional edge existence probability is  $1 - (1 - p^2)^m$ , which tends to 1 for  $mp^2 = \omega(1)$ ). In particular, we will assume that  $C_1 \sqrt{\frac{1}{nm}} \leq p \leq C_2 \frac{1}{\sqrt{m}}$ , for arbitrary positive constants  $C_1, C_2$ ;  $C_1$  can be as small as possible, and  $C_2$  can be as large as possible, provided  $C_2 \frac{1}{\sqrt{m}} \leq 1$ . We note that, when  $p$  is asymptotically equal to the upper bound  $C_2 \frac{1}{\sqrt{m}}$ , there is no constant weight upper bound that holds with high probability, whereas, when  $p$  is asymptotically equal to the lower bound  $C_1 \sqrt{\frac{1}{nm}}$ , all weights in the graph are bounded by a small constant with high probability. Our results in Section 3 assume this range of values for  $p$ , and thus graph instances may contain edges with large (but constant) weights. On the other hand, in the analysis of our randomized algorithms in section 4, we assume  $n = m$  and  $p = \Theta\left(\frac{1}{n}\right)$ ; this range of values gives sparse graph instances (even though the distribution is different from sparse Erdős-Rényi random graphs).

### 3 Concentration of Max-Cut

In this section we prove that the size of the maximum cut in a weighted random intersection graph concentrates around its expected value. We note however, that the following Theorem does not provide an explicit formula for the expected value of the maximum cut.

► **Theorem 5.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\bar{\mathcal{G}}_{n,m,p}$  model with  $m = n^\alpha, \alpha \leq 1$ , and  $C_1 \sqrt{\frac{1}{nm}} \leq p \leq 1$ , for arbitrary positive constant  $C_1$ , and let  $\mathbf{R}$  be its representation matrix. Then  $\text{Max-Cut}(G) \sim \mathbb{E}_{\mathbf{R}}[\text{Max-Cut}(G)]$  with high probability, where  $\mathbb{E}_{\mathbf{R}}$  denotes expectation with respect to  $\mathbf{R}$ , i.e.  $\text{Max-Cut}(G)$  concentrates around its expected value.*

**Proof.** Let  $G = G(V, E, \mathbf{R}^T \mathbf{R})$  be a weighted random intersection graph, and let  $\mathbf{D}$  denote the (random) diagonal matrix containing all diagonal elements of  $\mathbf{R}^T \mathbf{R}$ . In particular, equation (3) of Corollary 3 can be written as

$$\text{Max-Cut}(G) = \frac{1}{4} \left( \sum_{i \neq j, i, j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i,j} - \min_{\mathbf{x} \in \{-1, +1\}^n} \mathbf{x}^T (\mathbf{R}^T \mathbf{R} - \mathbf{D}) \mathbf{x} \right).$$

Furthermore, for any given  $\mathbf{R}$ , notice that, if we select each element of  $\mathbf{x}$  independently and equiprobably from  $\{-1, +1\}$ , then  $\mathbb{E}_{\mathbf{x}}[\mathbf{x}^T (\mathbf{R}^T \mathbf{R} - \mathbf{D}) \mathbf{x}] = 0$ , where  $\mathbb{E}_{\mathbf{x}}$  denotes expectation with respect to  $\mathbf{x}$ . Therefore, by the probabilistic method,  $\min_{\mathbf{x} \in \{-1, +1\}^n} \mathbf{x}^T (\mathbf{R}^T \mathbf{R} - \mathbf{D}) \mathbf{x} \leq 0$ , implying the following bound:

$$\frac{1}{4} \sum_{i \neq j, i, j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i,j} \leq \text{Max-Cut}(G) \leq \frac{1}{2} \sum_{i \neq j, i, j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i,j}, \quad (4)$$

where the second inequality follows trivially by observing that  $\frac{1}{2} \sum_{i \neq j, i, j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i,j}$  equals the sum of the weights of all edges.

By linearity,  $\mathbb{E}_{\mathbf{R}} \left[ \sum_{i \neq j, i, j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i, j} \right] = \mathbb{E}_{\mathbf{R}} \left[ \sum_{i \neq j, i, j \in [n]} \sum_{\ell \in [m]} \mathbf{R}_{\ell, i} \mathbf{R}_{\ell, j} \right] = n(n-1)mp^2 = \Theta(n^2 mp^2)$ , which goes to infinity as  $n \rightarrow \infty$ , because  $np = \Omega\left(\sqrt{\frac{n}{m}}\right) = \Omega(1)$  in the range of parameters that we consider. In particular, by (4), we have

$$\mathbb{E}_{\mathbf{R}}[\text{Max-Cut}(G)] = \Theta(n^2 mp^2). \quad (5)$$

By Chebyshev's inequality, for any  $\epsilon > 0$ , we have

$$\Pr\left(|\text{Max-Cut}(G) - \mathbb{E}_{\mathbf{R}}[\text{Max-Cut}(G)]| \geq \epsilon n^2 mp^2\right) \leq \frac{\text{Var}_{\mathbf{R}}(\text{Max-Cut}(G))}{\epsilon^2 n^4 m^2 p^4}, \quad (6)$$

where  $\text{Var}_{\mathbf{R}}$  denotes variance with respect to  $\mathbf{R}$ . To bound the variance on the right hand side of the above inequality, we use the Efron-Stein inequality. In particular, we write  $\text{Max-Cut}(G) := f(\mathbf{R})$ , i.e. we view  $\text{Max-Cut}(G)$  as a function of the label choices. For  $\ell \in [m], i \in [n]$ , we also write  $\mathbf{R}^{(\ell, i)}$  for the matrix  $\mathbf{R}$  where entry  $(\ell, i)$  has been replaced by an independent, identically distributed (i.i.d.) copy of  $\mathbf{R}_{\ell, i}$ , which we denote by  $\mathbf{R}'_{\ell, i}$ . By the Efron-Stein inequality, we have

$$\text{Var}_{\mathbf{R}}(\text{Max-Cut}(G)) \leq \frac{1}{2} \sum_{\ell \in [m], i \in [n]} \mathbb{E} \left[ \left( f(\mathbf{R}) - f(\mathbf{R}^{(\ell, i)}) \right)^2 \right]. \quad (7)$$

Notice now that, given all entries of  $\mathbf{R}$  except  $\mathbf{R}_{\ell, i}$ , the probability that  $f(\mathbf{R})$  is different from  $f(\mathbf{R}^{(\ell, i)})$  is at most  $\Pr(\mathbf{R}_{\ell, i} \neq \mathbf{R}'_{\ell, i}) = 2p(1-p)$ . Furthermore, if  $L_{\ell} \setminus \{i\}$  is the set of vertices different from  $i$  which have selected  $\ell$ , we then have that  $(f(\mathbf{R}) - f(\mathbf{R}^{(\ell, i)}))^2 \leq |L_{\ell} \setminus \{i\}|^2$ , because the intersection graph with representation matrix  $\mathbf{R}$  differs by at most  $|L_{\ell} \setminus \{i\}|$  edges from the intersection graph with representation matrix  $\mathbf{R}^{(\ell, i)}$ . Also note that, by definition,  $|L_{\ell} \setminus \{i\}|$  follows the Binomial distribution  $\mathcal{B}(n-1, p)$ . In particular,  $\mathbb{E}[|L_{\ell} \setminus \{i\}|^2] = (n-1)p(np - 2p + 1)$ , implying  $\mathbb{E} \left[ \left( f(\mathbf{R}) - f(\mathbf{R}^{(\ell, i)}) \right)^2 \right] \leq 2p(1-p)(n-1)p(np - 2p + 1)$ , for any fixed  $\ell \in [m], i \in [n]$ .

Putting this all together, (7) becomes

$$\begin{aligned} \text{Var}_{\mathbf{R}}(\text{Max-Cut}(G)) &\leq \frac{1}{2} \sum_{\ell \in [m], i \in [n]} 2p(1-p)(n-1)p(np - 2p + 1) \\ &= nmp(1-p)(n-1)p(np - 2p + 1) = O(n^3 mp^3), \end{aligned} \quad (8)$$

Therefore, by (6), we get

$$\Pr\left(|\text{Max-Cut}(G) - \mathbb{E}_{\mathbf{R}}[\text{Max-Cut}(G)]| \geq \epsilon n^2 mp^2\right) \leq \frac{O(n^3 mp^3)}{\epsilon^2 n^4 m^2 p^4} = O\left(\frac{1}{\epsilon^2 nmp}\right),$$

which goes to 0 in the range of values that we consider. Together with (5), the above bound proves that  $\text{Max-Cut}(G)$  is concentrated around its expected value.  $\blacktriangleleft$

### 3.1 Max-Cut for small number of labels

Using Theorem 5, we can now show that, in the case  $m = n^{\alpha}$ ,  $\alpha < 1$ , and  $p = O\left(\frac{1}{\sqrt{m}}\right)$ , a random cut has asymptotically the same weight as  $\text{Max-Cut}(G)$ , where  $G = G(V, E, \mathbf{R}^T \mathbf{R})$  is a random instance of  $\mathcal{G}_{n, m, p}$ . In particular, let  $\mathbf{x}^{(rand)}$  be constructed as follows: for each  $i \in [n]$ , set  $x_i^{(rand)} = -1$  independently with probability  $\frac{1}{2}$ , and  $x_i^{(rand)} = +1$  otherwise.

The proof details of the following Theorem can be found in the full version of our paper [17]. In view of equation (3), the main idea is to prove that, with high probability over random  $\mathbf{x}$  and  $\mathbf{R}$ ,  $\|\mathbf{R}\mathbf{x}\|^2$  is asymptotically smaller than the expectation of the weight of the cut defined by  $\mathbf{x}^{(rand)}$ , in which case the theorem follows by concentration of  $\text{Max-Cut}(G)$  around its expected value (Theorem 5), and straightforward bounds on  $\text{Max-Cut}(G)$ .

► **Theorem 6.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\bar{\mathcal{G}}_{n,m,p}$  model with  $m = n^\alpha$ ,  $\alpha < 1$ , and  $C_1 \sqrt{\frac{1}{nm}} \leq p \leq C_2 \frac{1}{\sqrt{m}}$ , for arbitrary positive constants  $C_1, C_2$ , and let  $\mathbf{R}$  be its representation matrix. Then the cut weight of the random 2-coloring  $\mathbf{x}^{(rand)}$  satisfies  $\text{Cut}(G, \mathbf{x}^{(rand)}) = (1 - o(1))\text{Max-Cut}(G)$  with high probability over the choices of  $\mathbf{x}^{(rand)}$ ,  $\mathbf{R}$ .*

We note that the same analysis also holds when  $n = m$  and  $p$  is sufficiently large (e.g.  $p = \omega(\frac{\ln n}{n})$ ); more details can be found in our technical report [17]. In view of this, in the following sections we will only assume  $m = n$  (i.e.  $\alpha = 1$ ) and also  $p = \frac{c}{n}$ , for some positive constant  $c$ . Besides avoiding complicated formulae for  $p$ , the reason behind this assumption is that, in this range of values, the expected weight of a fixed edge in  $G(V, E, \mathbf{R}^T \mathbf{R})$  is equal to  $mp^2 = \Theta(1/n)$ , and thus we hope that our work will serve as an intermediate step towards understanding algorithmic bottlenecks for finding maximum cuts in Erdős-Rényi random graphs  $G_{n,c/n}$  with respect to their intersection number.

## 4 Algorithmic results (randomized algorithms)

### 4.1 The Majority Cut Algorithm

In the following algorithm, the 2-coloring representing the bipartition of a cut is constructed as follows: initially, a small constant fraction  $\epsilon$  of vertices are randomly placed in the two partitions, and then in each subsequent step, one of the remaining vertices is placed in the partition that maximizes the weight of incident edges with endpoints in the opposite partition.

■ **Algorithm 1** Majority Cut.

---

**Input:**  $G(V, E, \mathbf{R}^T \mathbf{R})$  and its representation matrix  $\mathbf{R} \in \{0, 1\}^{m \times n}$   
**Output:** Large cut 2-coloring  $\mathbf{x} \in \{-1, +1\}^n$

- 1 Let  $v_1, \dots, v_n$  an arbitrary ordering of vertices;
- 2 **for**  $t = 1$  **to**  $\epsilon n$  **do**
- 3   └ Set  $x_t$  to either  $-1$  or  $+1$  independently with equal probability;
- 4 **for**  $t = \epsilon n + 1$  **to**  $n$  **do**
- 5   └ **if**  $\sum_{i \in [t-1]} [\mathbf{R}^T \mathbf{R}]_{i,t} x_i \geq 0$  **then**
- 6     └  $x_t = -1$ ;
- 7     **else**
- 8     └  $x_t = +1$ ;
- 9 **return**  $\mathbf{x}$ ;

---

Clearly the Majority Algorithm runs in polynomial time in  $n, m$ . Furthermore, the following Theorem provides a lower bound on the expected weight of the cut constructed by the algorithm in the case  $m = n$ ,  $p = \frac{c}{n}$ , for large constant  $c$ , and  $\epsilon \rightarrow 0$ . The full proof details can be found in our technical report [17].

► **Theorem 7.** Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\bar{\mathcal{G}}_{n,m,p}$  model, with  $m = n$ , and  $p = \frac{c}{n}$ , for large positive constant  $c$ , and let  $\mathbf{R}$  be its representation matrix. Then, with high probability over the choices of  $\mathbf{R}$ , the majority algorithm constructs a cut with expected weight at least  $(1 + \beta) \frac{1}{4} \mathbb{E} \left[ \sum_{i \neq j, i, j \in [n]} [\mathbf{R}^T \mathbf{R}]_{i,j} \right]$ , where  $\beta = \beta(c) \geq \sqrt{\frac{16}{27\pi c^3}} - o(1)$  is a constant, i.e. at least  $1 + \beta$  times larger than the expected weight of a random cut.

**Proof sketch.** Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\bar{\mathcal{G}}_{n,m,p}$  model, with  $m = n$ , and  $p = \frac{c}{n}$ , for some large enough constant  $c$ . For  $t \in [n]$ , let  $M_t$  denote the constructed cut size just after the consideration of a vertex  $v_t$ , for some  $t \geq \epsilon n + 1$ . By equation (3) for  $n = t$ , and since the values  $x_1, \dots, x_{t-1}$  are already decided in previous steps, we have  $M_t = \frac{1}{4} \left( \sum_{i,j \in [t]^2} [\mathbf{R}^T \mathbf{R}]_{i,j} - \min_{x_t \in \{-1, +1\}} \|\mathbf{R}_{[m],[t]} \mathbf{x}_{[t]}\|^2 \right)$ , and after careful calculation we get the recurrence

$$M_t = M_{t-1} + \frac{1}{2} \sum_{i \in [t-1]} [\mathbf{R}^T \mathbf{R}]_{i,t} + \frac{1}{2} |Z_t|,$$

where  $Z_t = Z_t(\mathbf{x}, \mathbf{R}) = \sum_{i \in [t-1]} [\mathbf{R}^T \mathbf{R}]_{i,t} x_i = \sum_{\ell \in [m]} \mathbf{R}_{\ell,t} \sum_{i \in [t-1]} \mathbf{R}_{\ell,i} x_i$ . Observe that, in the latter recursive equation, the term  $\frac{1}{2} \sum_{i \in [t-1]} [\mathbf{R}^T \mathbf{R}]_{i,t}$  corresponds to the expected increment of the constructed cut if the  $t$ -vertex chose its color uniformly at random. Therefore, lower bounding the expectation of  $\frac{1}{2} |Z_t|$  will tell us how much better the Majority Algorithm does when considering the  $t$ -th vertex.

Towards this end, we note that, given  $\mathbf{x}_{[t-1]} = \{x_i, i \in [t-1]\}$ , and  $\mathbf{R}_{[m],[t-1]} = \{\mathbf{R}_{\ell,i}, \ell \in [m], i \in [t-1]\}$ ,  $Z_t$  is the sum of  $m$  independent random variables, since the Bernoulli random variables  $\mathbf{R}_{\ell,t}, \ell \in [m]$ , are independent, for any given  $t$  (note that the conditioning is essential for independence, otherwise the inner sums in the definition of  $Z_t$  would also depend on the  $x_i$ 's, which are not random when  $i$  is large). By using a domination argument, we can then prove that

$$\mathbb{E}[|Z_t| | \mathbf{x}_{[t-1]}, \mathbf{R}_{[m],[t-1]}] \geq \text{MD}(Z_t^B),$$

where  $Z_t^B$  is a certain Binomial random variable (formally defined in the full proof), and  $\text{MD}(\cdot)$  is the mean absolute difference of (two independent copies of)  $Z_t^B$ , namely  $\text{MD}(Z_t^B) = \mathbb{E}[|Z_t^B - Z_t'^B|]$ . Even though we are aware of no simple closed formula for  $\text{MD}(Z_t^B)$ , we resort to Gaussian approximation of  $Z_t^B - Z_t'^B$  through the Berry-Esseen Theorem, ultimately showing that  $|Z_t^B - Z_t'^B|$  follows approximately the *folded normal distribution*. In particular, we show that  $\text{MD}(Z_t^B) \geq \sqrt{\frac{c(t-1)}{3\pi n}} - o(1)$ , and since the right hand side is independent of  $\mathbf{x}_{[t-1]}, \mathbf{R}_{[m],[t-1]}$ , we get the same lower bound on the expectation of  $|Z_t|$ , namely,  $\mathbb{E}[|Z_t|] \geq \sqrt{\frac{c(t-1)}{3\pi n}} - o(1)$ . Summing over all  $t \geq \epsilon n + 1$ , we get

$$\sum_{t \geq \epsilon n + 1} \mathbb{E}[|Z_t|] \geq \sqrt{\frac{c}{3\pi}} \left( \frac{2}{3} - \epsilon^{3/2} \right) n - o(n),$$

and the result follows by noting that the expected weight of a random cut is equal to  $\frac{1}{4} n(n-1) m p^2 = \frac{c^2}{4} n + o(n)$ , and taking  $\epsilon \rightarrow 0$ . ◀

## 4.2 Intersection graph (weak) bipartization

Notice that we can view a weighted intersection graph  $G(V, E, \mathbf{R}^T \mathbf{R})$  as a multigraph, composed by  $m$  (possibly) overlapping cliques corresponding to the sets of vertices having chosen a certain label, namely  $L_\ell = \{v : \mathbf{R}_{\ell,v}\}, \ell \in [m]$ . In particular, let  $K^{(\ell)}$  denote the

clique induced by label  $\ell$ . Then  $G = \cup_{\ell \in [m]}^+ K^{(\ell)}$ , where  $\cup^+$  denotes union that keeps multiple edges. In this section, we present an algorithm that takes as input an intersection graph  $G$  given as a union of overlapping cliques and outputs a subgraph that is “almost” bipartite.

To facilitate the presentation of our algorithm, we first give some useful definitions. A *closed vertex-label sequence* is a sequence of alternating vertices and labels starting and ending at the same vertex, namely  $\sigma := v_1, \ell_1, v_2, \ell_2, \dots, v_k, \ell_k, v_{k+1} = v_1$ , where the size of the sequence  $k = |\sigma|$  is the number of its labels,  $v_i \in V$ ,  $\ell_i \in \mathcal{M}$ , and  $\{v_i, v_{i+1}\} \subseteq L_{\ell_i}$ , for all  $i \in [k]$  (i.e.  $v_i$  is connected to  $v_{i+1}$  in the intersection graph). We will also say that label  $\ell$  is *strong* if  $|L_{\ell}| \geq 3$ , otherwise it is *weak*. For a given closed vertex-label sequence  $\sigma$ , and any integer  $\lambda \in [|\sigma|]$ , we will say that  $\sigma$  is  $\lambda$ -*strong* if  $|L_{\ell_i}| \geq 3$ , for  $\lambda$  indices  $i \in [|\sigma|]$ . The structural Lemma below is useful for our analysis (refer to the full version of our paper [17] for the proof).<sup>2</sup>

► **Lemma 8.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\overline{\mathcal{G}}_{n,m,p}$  model, with  $m = n$ , and  $p = \frac{c}{n}$ , for some constant  $c > 0$ . With high probability over the choices of  $\mathbf{R}$ , 0-strong closed vertex-label sequences in  $G$  do not have labels in common.*

The following definition is essential for the presentation of our algorithm.

► **Definition 9.** *Given a weighted intersection graph  $G = G(V, E, \mathbf{R}^T \mathbf{R})$  and a subgraph  $G^{(b)} \subseteq G$ , let  $\mathcal{C}_{odd}(G^{(b)})$  be the set of odd length closed vertex-label sequences  $\sigma := v_1, \ell_1, v_2, \ell_2, \dots, v_k, \ell_k, v_{k+1} = v_1$  that additionally satisfy the following:*

- (a)  $\sigma$  has distinct vertices (except the first and the last) and distinct labels.
- (b)  $v_i$  is connected to  $v_{i+1}$  in  $G^{(b)}$ , for all  $i \in [|\sigma|]$ .
- (c)  $\sigma$  is  $\lambda$ -strong, for some  $\lambda > 0$ .

Algorithm 2 initially replaces each clique  $K^{(\ell)}$  by a random maximal matching  $M^{(\ell)}$ , and thus gets a subgraph  $G^{(b)} \subseteq G$ . If  $\mathcal{C}_{odd}(G^{(b)})$  is not empty, then the algorithm selects  $\sigma \in \mathcal{C}_{odd}(G^{(b)})$  and a strong label  $\ell \in \sigma$ , and then replaces  $M^{(\ell)}$  in  $G^{(b)}$  by a new random matching of  $K^{(\ell)}$ . The algorithm repeats until all odd cycles are destroyed (or runs forever trying to do so).

■ **Algorithm 2** Intersection Graph Weak Bipartization.

---

**Input:** Weighted intersection graph  $G = \cup_{\ell \in [m]}^+ K^{(\ell)}$

**Output:** A subgraph of  $G^{(b)}$  that has only 0-strong odd cycles

- 1 **for** each  $\ell \in [m]$  **do**
  - 2     Let  $M^{(\ell)}$  be a random maximal matching of  $K^{(\ell)}$ ;
  - 3 Set  $G^{(b)} = \cup_{\ell \in [m]}^+ M^{(\ell)}$  ;
  - 4 **while**  $\mathcal{C}_{odd}(G^{(b)}) \neq \emptyset$  **do**
  - 5     Let  $\sigma \in \mathcal{C}_{odd}(G^{(b)})$  and  $\ell$  a label in  $\sigma$  with  $|L_{\ell}| \geq 3$ ;
  - 6     Replace the part of  $G^{(b)}$  corresponding to  $\ell$  by a new random maximal matching  $M^{(\ell)}$ ;
  - 7 **return**  $G^{(b)}$ ;
- 

The following results are the main technical tools that justify the use of the Weak Bipartization Algorithm for WEIGHTED MAX CUT. The proof details for Lemma 10 can be found in our technical report [17].

<sup>2</sup> We conjecture that the structural property of Lemma 8 also holds if we replace 0-strong with  $\lambda$ -strong, for any constant  $\lambda$ , but this stronger version is not necessary for our analysis.

► **Lemma 10.** *If  $\mathcal{C}_{\text{odd}}(G^{(b)})$  is empty, then  $G^{(b)}$  may only have 0-strong odd cycles.*

► **Theorem 11.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\overline{\mathcal{G}}_{n,m,p}$  model, with  $n = m$  and  $p = \frac{c}{n}$ , where  $c > 0$  is a constant, and let  $\mathbf{R}$  be its representation matrix. Let also  $\Sigma$  be a set system with incidence matrix  $\mathbf{R}$ . With high probability over the choices of  $\mathbf{R}$ , if Algorithm 2 for weak bipartization terminates on input  $G$ , its output can be used to construct a 2-coloring  $\mathbf{x}^{(\text{disc})} \in \arg \min_{\mathbf{x} \in \{\pm 1\}^n} \text{disc}(\Sigma, \mathbf{x})$ , which also gives a maximum cut in  $G$ , i.e.  $\mathbf{x}^{(\text{disc})} \in \arg \max_{\mathbf{x} \in \{\pm 1\}^n} \text{Cut}(G, \mathbf{x})$ .*

**Proof.** By construction, the output of Algorithm 2, namely  $G^{(b)}$ , has only 0-strong odd cycles. Furthermore, by Lemma 8 these cycles correspond to vertex-label sequences that are label-disjoint. Let  $H$  denote the subgraph of  $G^{(b)}$  in which we have destroyed all 0-strong odd cycles by deleting a single (arbitrary) edge  $e_C$  from each 0-strong odd cycle  $C$  (keeping all other edges intact), and notice that  $e_C$  corresponds to a weak label. In particular,  $H$  is a bipartite multi-graph and thus its vertices can be partitioned into two independent sets  $A, B$  constructed as follows: In each connected component of  $H$ , start with an arbitrary vertex  $v$  and include in  $A$  (resp. in  $B$ ) the set of vertices reachable from  $v$  that are at an even (resp. odd) distance from  $v$ . Since  $H$  is bipartite, it does not have odd cycles, and thus this construction is well-defined, i.e. no vertex can be placed in both  $A$  and  $B$ .

We now define  $\mathbf{x}^{(\text{disc})}$  by setting  $x_i^{(\text{disc})} = +1$  if  $i \in A$  and  $x_i^{(\text{disc})} = -1$  if  $i \in B$ . Let  $\mathcal{M}_0$  denote the set of weak labels corresponding to the edges removed from  $G^{(b)}$  in the construction of  $H$ . We first note that, for each  $\ell_C \in \mathcal{M}_0$  corresponding to the removal of an edge  $e_C$ , we have  $\left| \sum_{i \in L_{\ell_C}} x_i^{(\text{disc})} \right| = 2$ . Indeed, since  $e_C$  belongs to an odd cycle in  $G^{(b)}$ , its endpoints are at even distance in  $H$ , which means that either they both belong to  $A$  or they both belong to  $B$ . Therefore, their corresponding entries of  $\mathbf{x}^{(\text{disc})}$  have the same sign, and so (taking into account that the endpoints of  $e_C$  are the only vertices in  $L_{\ell_C}$ ), we have  $\left| \sum_{i \in L_{\ell_C}} x_i^{(\text{disc})} \right| = 2$ . Second, we show that, for all the other labels  $\ell \in [m] \setminus \mathcal{M}_0$ ,  $\left| \sum_{i \in L_{\ell}} x_i^{(\text{disc})} \right|$  will be equal to 1 if  $|L_{\ell}|$  is odd and 0 otherwise. For any label  $\ell \in [m] \setminus \mathcal{M}_0$ , let  $M^{(\ell)}$  denote the part of  $G^{(b)}$  corresponding to a maximal matching of  $K^{(\ell)}$ , and note that all edges of  $M^{(\ell)}$  are contained in  $H$ . Since  $H$  is bipartite, no edge in  $M^{(\ell)}$  can have both its endpoints in either  $A$  or  $B$ . Therefore, by construction, the contribution of entries of  $\mathbf{x}^{(\text{disc})}$  corresponding to endpoints of edges in  $M^{(\ell)}$  to the sum  $\sum_{i \in L_{\ell}} x_i^{(\text{disc})}$  is 0. In particular, if  $|L_{\ell}|$  is even, then  $M^{(\ell)}$  is a perfect matching and  $\left| \sum_{i \in L_{\ell}} x_i^{(\text{disc})} \right| = 0$ , otherwise (i.e. if  $|L_{\ell}|$  is odd) there is a single vertex not matched in  $M^{(\ell)}$  and  $\left| \sum_{i \in L_{\ell}} x_i^{(\text{disc})} \right| = 1$ .

To complete the proof of the theorem, we need to show that  $\text{Cut}(G, \mathbf{x}^{(\text{disc})})$  is maximum. By Corollary 3, this is equivalent to proving that  $\|\mathbf{R}\mathbf{x}^{(\text{disc})}\| \leq \|\mathbf{R}\mathbf{x}\|$  for all  $\mathbf{x} \in \{-1, +1\}^n$ . Suppose that there is some  $\mathbf{x}^{(\text{min})} \in \{-1, +1\}^n$  such that  $\|\mathbf{R}\mathbf{x}^{(\text{disc})}\| > \|\mathbf{R}\mathbf{x}^{(\text{min})}\|$ . As mentioned above, for all  $\ell \in [m] \setminus \mathcal{M}_0$ , we have  $[\mathbf{R}\mathbf{x}^{(\text{disc})}]_{\ell} \leq 1$ , and so  $[\mathbf{R}\mathbf{x}^{(\text{disc})}]_{\ell} \leq [\mathbf{R}\mathbf{x}^{(\text{min})}]_{\ell}$ . Therefore, the only labels where  $\mathbf{x}^{(\text{min})}$  could do better are those corresponding to edges  $e_C$  that are removed from  $G^{(b)}$  in the construction of  $H$ , i.e.  $\ell_C \in \mathcal{M}_0$ , for which we have  $[\mathbf{R}\mathbf{x}^{(\text{disc})}]_{\ell_C} = 2$ . However, any such edge  $e_C$  belongs to an odd cycle  $C$ , and thus any 2-coloring of the vertices of  $C$  will force at least one of the 0-strong labels corresponding to edges of  $C$  to be monochromatic. Taking into account the fact that, by Lemma 8, with high probability over the choices of  $\mathbf{R}$ , all 0-strong odd cycles correspond to vertex-label sequences that are label-disjoint, we conclude that  $\|\mathbf{R}\mathbf{x}^{(\text{disc})}\| \leq \|\mathbf{R}\mathbf{x}^{(\text{min})}\|$ , which completes the proof. ◀

The fact that Theorem 11 is not an immediate consequence of Corollary 4 follows from the observation that a random set system with incidence matrix  $\mathbf{R}$  has discrepancy larger than 1 with (at least) constant probability when  $m = n$  and  $p = \frac{c}{n}$ . Indeed, by a straightforward counting argument, we can see that the expected number of 0-strong odd cycles is at least constant. Furthermore, in any 2-coloring of the vertices at least one of the weak labels forming edges in a 0-strong odd cycle will be monochromatic. Therefore, with at least constant probability, for any  $\mathbf{x} \in \{-1, +1\}^n$ , there exists a weak label  $\ell$ , such that  $x_i x_j = 1$ , for both  $i, j \in L_\ell$ , implying that  $\text{disc}(L_\ell) = 2$ .

We close this section by a result indicating that the conditional statement of Theorem 11 is not void, namely there is a range of values for  $c$  where the Weak Bipartization Algorithm terminates in polynomial time.

► **Theorem 12.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\overline{\mathcal{G}}_{n,m,p}$  model, with  $n = m$  and  $p = \frac{c}{n}$ , where  $0 < c < 1$  is a constant, and let  $\mathbf{R}$  be its representation matrix. With high probability over the choices of  $\mathbf{R}$ , Algorithm 2 for weak bipartization terminates on input  $G$  in  $O\left((n + \sum_{\ell \in [m]} |L_\ell|) \cdot \log n\right)$  polynomial time.*

The proof of the above theorem uses the following structural Lemma regarding the expected number of closed vertex label sequences.

► **Lemma 13.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\overline{\mathcal{G}}_{n,m,p}$  model. Let also  $C_k$  denote the number of distinct closed vertex-label sequences of size  $k$  in  $G$ . Then*

$$\mathbb{E}[C_k] = \frac{1}{k} \frac{n!}{(n-k)!} \frac{m!}{(m-k)!} p^{2k}. \quad (9)$$

In particular, when  $m = n \rightarrow \infty$ ,  $p = \frac{c}{n}$ ,  $c > 0$ , and  $k \geq 3$ , we have  $\mathbb{E}[C_k] \leq \frac{c}{2\pi} c^{2k}$ .

**Proof.** Notice that there are  $\frac{1}{k} \frac{n!}{(n-k)!}$  ways to arrange  $k$  out of  $n$  vertices in a cycle. Furthermore, in each such arrangement, there are  $\frac{m!}{(m-k)!}$  ways to place  $k$  out of  $m$  labels so that there is exactly one label between each pair of vertices. Since labels in any given arrangement must be selected by both its adjacent vertices, (9) follows by linearity of expectation.

Setting  $m = n$  and  $p = \frac{c}{n}$ , and using the inequalities  $\sqrt{2\pi} n^{n+\frac{1}{2}} e^{-n} \leq n! \leq e n^{n+\frac{1}{2}} e^{-n}$ ,

$$\begin{aligned} \mathbb{E}[C_k] &= \frac{1}{k} \left( \frac{n!}{(n-k)!} \right)^2 \left( \frac{c}{n} \right)^{2k} \\ &\leq \frac{1}{k} \frac{e^2 n^{2n+1} e^{-2n}}{2\pi (n-k)^{2n-2k+1} e^{2k-2n}} \left( \frac{c}{n} \right)^{2k} = \frac{1}{k} \frac{e^2}{2\pi} \left( \frac{n}{n-k} \right)^{2n-2k+1} \left( \frac{c}{e} \right)^{2k} \\ &\leq \frac{e^2}{2\pi} \frac{n}{k(n-k)} e^{\frac{k}{n-k}(2n-2k)} \left( \frac{c}{e} \right)^{2k} = \frac{e^2}{2\pi} \frac{n}{k(n-k)} c^{2k}. \end{aligned}$$

When  $n$  goes to  $\infty$  and  $k \geq 3$ , then the above is at most  $\frac{c}{2\pi} c^{2k}$  as needed. ◀

We are now ready for the proof of the Theorem.

**Proof of Theorem 12.** We will prove that, when  $m = n \rightarrow \infty$ ,  $p = \frac{c}{n}$ ,  $c < 1$ , and  $k \geq 3$ , with high probability, there are no closed vertex-label sequences that have labels in common. To this end, recalling Definition 9 for  $\mathcal{C}_{\text{odd}}(G^{(b)})$ , we provide upper bounds on the following events:  $A \stackrel{\text{def}}{=} \{\exists k \geq \log n : C_k \geq 1\}$ ,  $B \stackrel{\text{def}}{=} \{|\mathcal{C}_{\text{odd}}(G^{(b)})| \geq \log n\}$  and  $C \stackrel{\text{def}}{=} \{\exists \sigma \neq \sigma' \in \mathcal{C}_{\text{odd}}(G^{(b)}) : \exists \ell \in \sigma, \ell \in \sigma'\}$ .

By the union bound, Markov's inequality and Lemma 13, we get that, whp all closed vertex-label sequences have less than  $\log n$  labels:



$$\Pr(A) \leq \sum_{k \geq \log n} \mathbb{E}[C_k] \leq \sum_{k \geq \log n} \frac{e}{2\pi} c^{2k} = \frac{e}{2\pi} \frac{c^{2 \log n}}{1 - c^2} = O(c^{2 \log n}) = o(1), \quad (10)$$

where the last equality follows since  $c < 1$  is a constant. Furthermore, by Markov's inequality and Lemma 13, and noting that any closed vertex-label sequence in  $\mathcal{C}_{\text{odd}}(G^{(b)})$  must have at least  $k \geq 3$  labels, we get that, whp there less than  $\log n$  closed vertex-label sequences in  $\mathcal{C}_{\text{odd}}(G^{(b)})$ :

$$\Pr(B) \leq \frac{1}{\log n} \sum_{k \geq 3} \mathbb{E}[C_k] \leq \frac{1}{\log n} \sum_{k \geq 3} \frac{e}{2\pi} c^{2k} = \frac{1}{\log n} \frac{e}{2\pi} \frac{c^6}{1 - c^2} = O\left(\frac{1}{\log n}\right). \quad (11)$$

To bound  $\Pr(C)$ , fix a closed vertex-label sequence  $\sigma$ , and let  $|\sigma| \geq 3$  be the number of its labels. Notice that, the probability that there is another closed vertex-label sequence that has labels in common with  $\sigma$  implies the existence of a vertex-label sequence  $\check{\sigma}$  that starts with either a vertex or a label from  $\sigma$ , ends with either a vertex or a label from  $\sigma$ , and has at least one label or at least one vertex that does not belong to  $\sigma$ . Let  $|\check{\sigma}|$  denote the number of labels of  $\check{\sigma}$  that do not belong to  $\sigma$ . Then the number of different vertex-label sequences  $\check{\sigma}$  that start and end in labels from  $\sigma$  is at most  $|\sigma|^2 n^{|\check{\sigma}|+1} m^{|\check{\sigma}|}$ ; indeed  $\check{\sigma}$  in this case has  $|\check{\sigma}|$  labels and  $|\check{\sigma}| + 1$  vertices that do not belong to  $\sigma$ . Therefore, by independence, each such sequence  $\check{\sigma}$  has probability  $p^{2|\check{\sigma}|+2}$  to appear. Similarly, the number of different vertex-label sequences  $\check{\sigma}$  that start and end in vertices from  $\sigma$  is at most  $|\sigma|^2 n^{|\check{\sigma}|-1} m^{|\check{\sigma}|}$  and each one has probability  $p^{2|\check{\sigma}|}$  to appear. Finally, the number of different vertex-label sequences  $\check{\sigma}$  that start in a vertex from  $\sigma$  and end in a label from  $\sigma$  (notice that this also covers the case where  $\check{\sigma}$  starts in a label from  $\sigma$  and ends in a vertex from  $\sigma$ ) is at most  $|\sigma|^2 n^{|\check{\sigma}|} m^{|\check{\sigma}|}$  and each one has probability  $p^{2|\check{\sigma}|+1}$  to appear. Overall, for a given sequence  $\sigma$ , the expected number of sequences  $\check{\sigma}$  described above that additionally satisfies  $|\check{\sigma}| < \log n$ , is at most

$$\sum_{k=0}^{\log n - 1} |\sigma|^2 n^{k+1} m^k p^{2k+2} + \sum_{k=1}^{\log n - 1} |\sigma|^2 n^{k-1} m^k p^{2k} + \sum_{k=1}^{\log n - 1} |\sigma|^2 n^k m^k p^{2k+1} \leq c |\sigma|^2 \frac{\log n}{n}, \quad (12)$$

where in the last inequality we used the fact that  $m = n, p = \frac{c}{n}$  and  $c < 1$ . Since the existence of a sequence  $\check{\sigma}$  for  $\sigma$  that additionally satisfies  $|\check{\sigma}| \geq \log n$  implies event  $A$ , and on other hand the existence of more than  $\log n$  different sequences  $\sigma \in |\mathcal{C}_{\text{odd}}(G^{(b)})|$  implies event  $B$ , by Markov's inequality and (12), we get

$$\Pr(C) \leq \Pr(A) + \Pr(B) + c \frac{(\log n)^4}{n} = O(c^{2 \log n}) + O\left(\frac{1}{\log n}\right) + O\left(\frac{(\log n)^4}{n}\right) = O\left(\frac{1}{\log n}\right).$$

We have thus proved that, with high probability over the choices of  $\mathbf{R}$ , closed vertex-label sequences in  $\mathcal{C}_{\text{odd}}(G^{(b)})$  are label disjoint, as needed.

In view of this, the proof of the Theorem follows by noting that, since closed vertex label sequences in  $\mathcal{C}_{\text{odd}}(G^{(b)})$  are label disjoint, steps 5 and 6 within the while loop of the Weak Bipartization Algorithm will be executed exactly once for each sequence in  $\mathcal{C}_{\text{odd}}(G^{(b)})$ , where  $G^{(b)}$  is defined in step 3 of the algorithm; indeed, once a closed vertex label sequence  $\sigma \in \mathcal{C}_{\text{odd}}(G^{(b)})$  is destroyed in step 6, no new closed vertex label sequence is created. In fact, once  $\sigma$  is destroyed we can remove the corresponding labels and edges from  $G^{(b)}$ , as these will no longer belong to other closed vertex label sequences. Furthermore, to find a closed vertex label sequences in  $\mathcal{C}_{\text{odd}}(G^{(b)})$ , it suffices to find an odd cycle in  $G^{(b)}$ , which can be done by running DFS, requiring  $O(n + \sum_{\ell \in [m]} |L_\ell|)$  time, because  $G^{(b)}$  has at most

$\sum_{\ell \in [m]} |L_\ell|$  edges. Finally, by (11), we have  $|\mathcal{C}_{\text{odd}}(G^{(b)})| < \log n$  with high probability, and so the running time of the Weak Bipartization Algorithm is  $O((n + \sum_{\ell \in [m]} |L_\ell|) \log n)$ , which concludes the proof of Theorem 12.  $\blacktriangleleft$

## 5 Discussion and some open problems

In this paper, we introduced the model of weighted random intersection graphs and we studied the average case analysis of WEIGHTED MAX CUT through the prism of discrepancy of random set systems. In particular, in the first part of the paper, we proved concentration of the weight of a maximum cut of  $G(V, E, \mathbf{R}^T \mathbf{R})$  around its expected value, and we used it to show that, with high probability, the weight of a random cut is asymptotically equal to the maximum cut weight of the input graph, when  $m = n^\alpha$ ,  $\alpha < 1$ . On the other hand, in the case where the number of labels is equal to the number of vertices (i.e.  $m = n$ ), we proved that a majority algorithm gives a cut with weight that is larger than the weight of a random cut by at least a constant factor, when  $p = \frac{c}{n}$  and  $c$  is large.

In the second part of the paper, we highlighted a connection between WEIGHTED MAX CUT of sparse weighted random intersection graphs and DISCREPANCY of sparse random set systems, formalized through our Weak Bipartization Algorithm and its analysis. We demonstrated how our proposed framework can be used to find optimal solutions for these problems, with high probability, in special cases of sparse inputs ( $m = n, p = \frac{c}{n}, c < 1$ ).

One of the main problems left open in our work concerns the termination of our Weak Bipartization Algorithm for large values of  $c$ . We conjecture the following:

► **Conjecture 14.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\overline{\mathcal{G}}_{n,m,p}$  model, with  $m = n$ , and  $p = \frac{c}{n}$ , for some constant  $c \geq 1$ . With high probability over the choices of  $\mathbf{R}$ , on input  $G$ , Algorithm 2 for weak bipartization terminates in polynomial time.*

We also leave the problem of determining whether Algorithm 2 terminates in polynomial time, in the case  $m = n$  and  $p = \omega(1/n)$ , as an open question for future research.

Towards strengthening the connection between WEIGHTED MAX CUT under the  $\overline{\mathcal{G}}_{n,m,p}$  model, and DISCREPANCY in random set systems, we conjecture the following:

► **Conjecture 15.** *Let  $G(V, E, \mathbf{R}^T \mathbf{R})$  be a random instance of the  $\overline{\mathcal{G}}_{n,m,p}$  model, with  $m = n^\alpha$ ,  $\alpha \leq 1$  and  $mp^2 = O(1)$ , and let  $\mathbf{R}$  be its representation matrix. Let also  $\Sigma$  be a set system with incidence matrix  $\mathbf{R}$ . Then, with high probability over the choices of  $\mathbf{R}$ , there exists  $\mathbf{x}^{\text{disc}} \in \arg \min_{\mathbf{x} \in \{-1, +1\}^n} \text{disc}(\Sigma, \mathbf{x})$ , such that  $\text{Cut}(G, \mathbf{x}^{\text{disc}})$  is asymptotically equal to  $\text{Max-Cut}(G)$ .*

---

## References

- 1 D. Altschuler and J. Niles-Weed. The discrepancy of random rectangular matrices. CoRR abs/2101.04036, 2021. [arXiv:2101.04036](#).
- 2 N. Bansal and R. Meka. On the discrepancy of random low degree set systems. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 2557–2564, 2019.
- 3 F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.
- 4 M. Bayati, D. Gamarnik, and P. Tetel. Combinatorial approach to the interpolation method and scaling limits in sparse random graphs. *Ann. Probab.*, 41:4080–4115, 2013.

- 5 M. Behrisch, A. Taraz, and M. Ueckerdt. Coloring random intersection graphs and complex networks. *SIAM J. Discret. Math.*, 23(1):288–299, 2009.
- 6 M. Bloznelis, E. Godehardt, J. Jaworski, V. Kurauskas, and K. Rybarczyk. Recent progress in complex network analysis: Models of random intersection graphs. In *Studies in Classification, Data Analysis, and Knowledge Organization*, pages 69–78. Springer, 2015.
- 7 A. Coja-Oghlan, C. Moore, and V. Sanwalani. Max k-cut and approximating the chromatic number of random graphs. *Random Struct. Algorithms*, 28(3):289–322, 2006.
- 8 D. Coppersmith, D. Gamarnik, M. Hajiaghayi, and G. Sorkin. Random maxsat, random maxcut, and their phase transitions. *Rand. Struct. Alg.*, 24(4):502–545, 2004.
- 9 A. Dembo, A. Montanari, and S. Sen. Extremal cuts of sparse random graphs. *The Annals of Probability*, 45(2):1190–1217, 2017.
- 10 J. Díaz, J. Petit, and M. Serna. A survey on graph layout problems. *ACM Comput. Surveys*, 34:313–356, 2002.
- 11 E. Ezra and S. Lovett. On the beck-fiala conjecture for random set systems. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization - Algorithms and Techniques (APPROX-RANDOM)*, pages 29:1–29:10, 2016.
- 12 J. Fill, E. Sheinerman, and K. Singer-Cohen. Random intersection graphs when  $m = \omega(n)$ : an equivalence theorem relating the evolution of the  $g(n, m, p)$  and  $g(n, p)$  models. *Random Struct. Algorithms*, 16(2):156–176, 2000.
- 13 D. Gamarnik and Q. Li. On the max-cut of sparse random graphs. *Random Struct. Algorithms*, 52(2):219–262, 2018.
- 14 R. Hoberg and T. Rothvoss. A fourier-analytic approach for the discrepancy of random set systems. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2547–2556, 2019.
- 15 M. Karoński, E. Scheinerman, and K. Singer-Cohen. On random intersection graphs: the subgraph problem. *Combinatorics, Probability and Computing*, 8:131–159, 1999.
- 16 S. Nikolettseas, C. Raptopoulos, and P. Spirakis. Efficient approximation algorithms in random intersection graphs. In *Handbook of Approximation Algorithms and Metaheuristics*, volume 2. Chapman and Hall/CRC, 2018.
- 17 S. Nikolettseas, C. Raptopoulos, and P. Spirakis. Max cut in weighted random intersection graphs and discrepancy of sparse random set systems, 2021. [arXiv:2009.01567v2](https://arxiv.org/abs/2009.01567v2).
- 18 C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Computer and System Sciences*, 43(3):425–440, 1991.
- 19 J. Poland and T. Zeugmann. Clustering pairwise distances with missing data: Maximum cuts versus normalized cuts. *Lecture Notes in Comput. Sci.*, 4265:197–208, 2006.
- 20 S. Poljak and Z. Tuza. Maximum cuts and largest bipartite subgraphs. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 20:181–244, 1995. American Mathematical Society, Providence, R.I.
- 21 A. Potukuchi. Discrepancy in random hypergraph models. CoRR abs/1811.01491, 2018. [arXiv:1811.01491](https://arxiv.org/abs/1811.01491).
- 22 C. Raptopoulos and P. Spirakis. Simple and efficient greedy algorithms for hamilton cycles in random intersection graphs. In *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC)*, pages 493–504, 2005.
- 23 K. Rybarczyk. Equivalence of a random intersection graph and  $g(n, p)$ . *Random Structures and Algorithms*, 38(1-2):205–234, 2011.

# The Impact of Geometry on Monochrome Regions in the Flip Schelling Process

Thomas Bläsius  

Karlsruhe Institute of Technology, Germany

Tobias Friedrich   

Hasso Plattner Institute, University of Potsdam, Germany

Martin S. Krejca   

Sorbonne University, CNRS, LIP6, Paris, France

Louise Molitor   

Hasso Plattner Institute, University of Potsdam, Germany

---

## Abstract

---

Schelling’s classical segregation model gives a coherent explanation for the wide-spread phenomenon of residential segregation. We introduce an agent-based saturated open-city variant, the Flip Schelling Process (FSP), in which agents, placed on a graph, have one out of two types and, based on the predominant type in their neighborhood, decide whether to change their types; similar to a new agent arriving as soon as another agent leaves the vertex.

We investigate the probability that an edge  $\{u, v\}$  is monochrome, i.e., that both vertices  $u$  and  $v$  have the same type in the FSP, and we provide a general framework for analyzing the influence of the underlying graph topology on residential segregation. In particular, for two adjacent vertices, we show that a highly decisive common neighborhood, i.e., a common neighborhood where the absolute value of the difference between the number of vertices with different types is high, supports segregation and, moreover, that large common neighborhoods are more decisive.

As an application, we study the expected behavior of the FSP on two common random graph models with and without geometry: (1) For random geometric graphs, we show that the existence of an edge  $\{u, v\}$  makes a highly decisive common neighborhood for  $u$  and  $v$  more likely. Based on this, we prove the existence of a constant  $c > 0$  such that the expected fraction of monochrome edges after the FSP is at least  $1/2 + c$ . (2) For Erdős–Rényi graphs we show that large common neighborhoods are unlikely and that the expected fraction of monochrome edges after the FSP is at most  $1/2 + o(1)$ . Our results indicate that the cluster structure of the underlying graph has a significant impact on the obtained segregation strength.

**2012 ACM Subject Classification** Theory of computation → Network formation; Theory of computation → Random network models

**Keywords and phrases** Agent-based Model, Schelling Segregation, Spin System

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.29

**Related Version** *Full Version:* <https://arxiv.org/pdf/2102.09856.pdf> [12]

**Funding** *Martin S. Krejca:* This work was supported by the Paris Île-de-France Region.

**Acknowledgements** We want to thank Thomas Sauerwald for the discussions on random walks.

## 1 Introduction

Residential segregation is a well-known sociological phenomenon [49] where different groups of people tend to separate into largely homogeneous neighborhoods. Studies, e.g., [18], show that individual preferences are the driving force behind present residential patterns and bear much to the explanatory weight. Local choices therefore lead to a global phenomenon [47]. A simple model for analyzing residential segregation was introduced by Schelling [46, 47] in



© Thomas Bläsius, Tobias Friedrich, Martin S. Krejca, and Louise Molitor;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 29; pp. 29:1–29:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the 1970s. In his model, two types of agents, placed on a grid, act according to the following threshold behavior, with  $\tau \in (0, 1)$  as the *intolerance threshold*: agents are *content* with their current position on the grid if at least a  $\tau$ -fraction of neighbors is of their own type. Otherwise they are *discontent* and want to move, either via swapping with another random discontent agent or via jumping to a vacant position. Schelling demonstrated via simulations that, starting from a uniform random distribution, the described process drifts towards strong segregation, even if agents are tolerant and agree to live in mixed neighborhoods, i.e., if  $\tau \leq \frac{1}{2}$ . Many empirical studies have been conducted to investigate the influence of various parameters on the obtained segregation, see [8, 9, 25, 41, 45]. On the theoretical side, Schelling's model started recently gaining traction within the algorithmic game theory and artificial intelligence communities [1, 11, 16, 17, 21, 22, 33], with focus on core game theoretic questions, where agents strategically select locations. Henry et al. [31] described a simple model of graph clustering motivated by Schelling where they showed that segregated graphs always emerge. Variants of the random Schelling segregation process were analyzed by a line of work that showed that residential segregation occurs with high probability [5, 7, 10, 13, 32, 51].

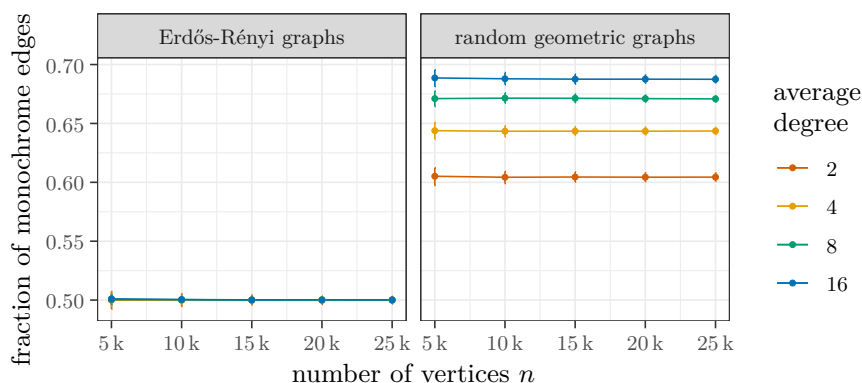
We initiate the study of an agent-based model, called the *Flip Schelling Process (FSP)*, which can be understood as the Schelling model in a *saturated open city*. In contrast to *closed cities* [7, 13, 32, 51], which require fixed populations, open cities [4, 5, 10, 27] allow resident to move away. In saturated city models, also known as voter models [20, 35, 36], vertices are not allowed to be unoccupied, hence, a new agent enters as soon as one agent vacates a vertex. In general, in voter models, two types of agents are placed on a graph. Agents examine their neighbors and, if a certain threshold is of another type, they change their types. Also in this model, segregation is visible. There is a line of work, mainly in physics, that studies the voting dynamics on several types of graphs [3, 14, 37, 43, 50]. Related to voter models, Granovetter [30] proposed another threshold model treating binary decisions and spurred a number of research, which studied and motivated variants of the model, see [2, 34, 38, 44].

In the FSP, agents have binary types. An agent is content if the fraction of agents in its neighborhood with the same type is larger than  $\frac{1}{2}$ . Otherwise, if the fraction is smaller than  $\frac{1}{2}$ , an agent is discontent and is willing to flip its type to become content. If the fraction of same type agents in its neighborhood is exactly  $\frac{1}{2}$ , an agent flips its type with probability  $\frac{1}{2}$ . Starting from an initial configuration where the type of each agent is chosen uniformly at random, we investigate a simultaneous-move, one-shot process and bound the number of monochrome edges, which is a popular measurement for segregation strength [19, 26].

Close to our model is the work by Omidvar and Franceschetti [39, 40], who initiated an analysis of the size of monochrome regions in the so called *Schelling Spin Systems*. Agents of two different types are placed on a grid [39] and a geometric graph [40], respectively. Then independent and identical Poisson clocks are assigned to all agents and, every time a clock rings, the state of the corresponding agent is flipped if and only if the agent is discontent w.r.t. a certain intolerance threshold  $\tau$  regarding the neighborhood size. The model corresponds to the Ising model with zero temperature with Glauber dynamics [15, 48].

The commonly used underlying topology for modeling the residential areas are (toroidal) grid graphs [11, 32, 39], regular graphs [11, 17, 21], paths [11, 33], cycles [4, 6, 7, 13, 51] and trees [1, 11, 22, 33]. Considering the influence of the given topology that models the residential area regarding, e.g., the existence of stable states and convergence behavior leads to phenomena like non-existence of stable states [21, 22], non-convergence to stable states [11, 17, 21], and high-mixing times in corresponding Markov chains [10, 28].

To avoid such undesirable characteristics, we suggest to investigate *random geometric graphs* [42], like in [40]. Random geometric graphs demonstrate, in contrast to other random graphs without geometry, such as *Erdős-Rényi graphs* [23, 29], community structures, i.e.,



■ **Figure 1** The fraction of monochrome edges after the Flip Schelling Process (FSP) in Erdős–Rényi graphs and random geometric graphs for different graph sizes (number of vertices  $n$ ) and different expected average degrees. Each data point shows the average over 1000 generated graphs with one simulation of the FSP per graph. The error bars show the interquartile range, i.e., 50% of the measurements lie between the top and bottom end of the error bar.

densely connected clusters of vertices. An effect observed by simulating the FSP is that the fraction of monochrome edges is significantly higher in random geometric graphs compared to Erdős–Rényi graphs, where the fraction stays almost stable around  $\frac{1}{2}$ , cf. Fig 1.

We set out for rigorously proving this phenomenon. In particular, we prove for random geometric graphs that there exists a constant  $c$  such that, given an edge  $\{u, v\}$ , the probability that  $\{u, v\}$  is monochrome is lower-bounded by  $\frac{1}{2} + c$ , cf. Theorem 6. In contrast, we show for Erdős–Rényi graphs that segregation is not likely to occur and that the probability that  $\{u, v\}$  is monochrome is upper-bounded by  $\frac{1}{2} + o(1)$ , cf. Theorem 17.

We introduce a general framework to deepen the understanding of the influence of the underlying topology on residential segregation. To this end, we first show that a highly decisive common neighborhood supports segregation, cf. Section 3.1. In particular, we provide a lower bound on the probability that an edge  $\{u, v\}$  is monochrome based on the probability that the difference between the majority and the minority regarding both types in the common neighborhood, i.e., the number of agents which are adjacent to  $u$  and  $v$ , is larger than their exclusive neighborhoods, i.e., the number of agents which are adjacent to either  $u$  or  $v$ . Next, we show that large sets of agents are more decisive, cf. Section 3.2. This implies that a large common neighborhood, compared to the exclusive neighborhood, is likely to be more decisive, i.e., makes it more likely that the absolute value of the difference between the number of different types in the common neighborhood is larger than in the exclusive ones. These considerations hold for arbitrary graphs. Hence, we reduce the question concerning a lower bound for the fraction of monochrome edges in the FSP to the probability that, given  $\{u, v\}$ , the common neighborhood is larger than the exclusive neighborhoods of  $u$  and  $v$ , respectively.

For random geometric graphs, we prove that a large geometric region, i.e., the intersecting region that is formed by intersecting disks, leads to a large vertex set, cf. Section 3.3, and that random geometric graphs have enough edges that have sufficiently large intersecting regions, cf. Section 3.4, such that segregation is likely to occur. In contrast, for Erdős–Rényi graphs, we show that the common neighborhood between two vertices  $u$  and  $v$  is with high probability empty and therefore segregation is not likely to occur, cf. Section 4.

Overall, we shed light on the influence of the structure of the underlying graph and discovered the significant impact of the community structure as an important factor on the obtained segregation strength. We reveal for random geometric graphs that already after one round a provable tendency is apparent and a strong segregation occurs.

## 2 Model and Preliminaries

Let  $G = (V, E)$  be an unweighted and undirected graph, with vertex set  $V$  and edge set  $E$ . For any vertex  $v \in V$ , we denote the *neighborhood* of  $v$  in  $G$  by  $N_v = \{u \in V : \{u, v\} \in E\}$  and the degree of  $v$  in  $G$  by  $\delta_v = |N_v|$ . We consider *random geometric graphs* and *Erdős–Rényi graphs* with a total of  $n \in \mathbf{N}^+$  vertices and an *expected average degree*  $\bar{\delta} > 0$ .

For a given  $r \in \mathbf{R}^+$ , a random geometric graph  $G \sim \mathcal{G}(n, r)$  is obtained by distributing  $n$  vertices uniformly at random in some geometric ground space and connecting vertices  $u$  and  $v$  if and only if  $\text{dist}(u, v) \leq r$ . We use a two-dimensional toroidal Euclidean space with total area 1 as ground space. More formally, each vertex  $v$  is assigned to a point  $(v_1, v_2) \in [0, 1]^2$  and the distance between  $u = (u_1, u_2)$  and  $v$  is  $\text{dist}(u, v) = \sqrt{|u_1 - v_1|_0^2 + |u_2 - v_2|_0^2}$  for  $|u_i - v_i|_0 = \min\{|u_i - v_i|, 1 - |u_i - v_i|\}$ . We note that using a torus instead of, e.g., a unit square, has the advantage that we do not have to consider edge cases, for vertices that are close to the boundary. In fact, a disk of radius  $r$  around any point has the same area  $\pi r^2$ . Since we consider a ground space with total area 1,  $r \leq \frac{1}{\sqrt{\pi}}$ . As every vertex  $v$  is connected to all vertices in the disk of radius  $r$  around it, its expected average degree is  $\bar{\delta} = (n-1)\pi r^2$ .

For a given  $p \in [0, 1]$ , let  $\mathcal{G}(n, p)$  denote an Erdős–Rényi graph. Each edge  $\{u, v\}$  is included with probability  $p$ , independently from every other edge. It holds that  $\bar{\delta} = (n-1)p$ .

Consider two different vertices  $u$  and  $v$ . Let  $N_{u \cap v} := |N_u \cap N_v|$  be the number of vertices in the *common neighborhood*, let  $N_{u \setminus v} := |N_u \setminus N_v|$  be the number of vertices in the *exclusive neighborhood* of  $u$ , and let  $N_{v \setminus u} := |N_v \setminus N_u|$  be the number of vertices in the exclusive neighborhood of  $v$ . Furthermore, with  $N_{\overline{u \cup v}} := |V \setminus (N_u \cup N_v)|$ , we denote the number of vertices that are neither adjacent to  $u$  nor to  $v$ .

Let  $G$  be a graph where each vertex represents an agent of type  $t^+$  or  $t^-$ . The type of each agent is chosen independently and uniformly at random. An edge  $\{u, v\}$  is *monochrome* if and only if  $u$  and  $v$  are of the same type. The *Flip Schelling Process* (FSP) is defined as follows: an agent  $v$  whose type is aligned with the type of more than  $\delta_v/2$  of its neighbors keeps its type. If more than  $\delta_v/2$  neighbors have a different type, then agent  $v$  changes its type. In case of a tie, i.e., if exactly  $\delta_v/2$  neighbors have a different type, then  $v$  changes its type with probability  $\frac{1}{2}$ . FSP is a simultaneous-move, one-shot process, i.e., all agents make their decision at the same time and, moreover, only once.

For  $x, y \in \mathbf{N}$ , we define  $[x..y] = [x, y] \cap \mathbf{N}$  and for  $x \in \mathbf{N}^+$ , we define  $[x] = [1..x]$ .

### 2.1 Useful Technial Lemmas

In this section, we state several lemmas that we will use in order to prove our results in the next sections.

► **Lemma 1.** *Let  $X \sim \text{Bin}(n, p)$  and  $Y \sim \text{Bin}(n, q)$  with  $p \geq q$  be independent. Then  $\Pr[X \geq Y] \geq \frac{1}{2}$ .*

**Proof.** Let  $Y_1, \dots, Y_n$  be the individual Bernoulli trials for  $Y$ , i.e.,  $Y = \sum_{i \in [n]} Y_i$ . Define new random variables  $Y'_1, \dots, Y'_n$  such that  $Y_i = 1$  implies  $Y'_i = 1$  and if  $Y_i = 0$ , then  $Y'_i = 1$  with probability  $(p - q)/(1 - q)$  and  $Y'_i = 0$  otherwise. Note that for each individual  $Y'_i$ , we have  $Y'_i = 1$  with probability  $p$ , i.e.,  $Y' = \sum_{i \in [n]} Y'_i \sim \text{Bin}(n, p)$ . Moreover, as  $Y' \geq Y$  for every outcome, we have  $\Pr[X \geq Y] \geq \Pr[X \geq Y']$ . It remains to show that  $\Pr[X \geq Y'] \geq \frac{1}{2}$ .

As  $X$  and  $Y'$  are equally distributed, we have  $\Pr[X \geq Y'] = \Pr[X \leq Y']$ . Moreover, as one of the two inequalities holds in any event, we get  $\Pr[X \geq Y'] + \Pr[X \leq Y'] \geq 1$ , and thus equivalently  $2\Pr[X \geq Y'] \geq 1$ , which proves the claim. ◀



► **Lemma 2.** *Let  $n \in \mathbf{N}^+$ ,  $p \in [0, 1)$ , and let  $X \sim \text{Bin}(n, p)$ . Then, for all  $i \in [0..n]$ , it holds that  $\Pr[X = i] \leq \Pr[X = \lfloor p(n+1) \rfloor]$ .*

**Proof.** We interpret the distribution of  $X$  as a finite series and consider the sign of the differences of two neighboring terms. A maximum of the distribution of  $X$  is located at the position at which the difference switches from positive to negative. To this end, let  $b: [0, n-1] \rightarrow [-1, 1]$  be defined such that, for all  $i \in [0, n-1] \cap \mathbf{N}$ , it holds that

$$b(d) = \binom{n}{d+1} p^{d+1} (1-p)^{n-d-1} - \binom{n}{d} p^d (1-p)^{n-d}.$$

We are interested in the sign of  $b$ . In more detail, for any  $d \in [0, n-2] \cap \mathbf{N}$ , if  $\text{sgn}(b(d)) \geq 0$  and  $\text{sgn}(b(d+1)) \leq 0$ , then  $d+1$  is a local maximum. If the sign is always negative, then there is a global maximum in the distribution of  $X$  at position 0.

In order to determine the sign of  $b$ , for all  $i \in [0..n-1]$ , we rewrite

$$\begin{aligned} b(i) &= \frac{n!}{i!(n-i-1)!} p^i (1-p)^{n-i-1} \frac{p}{i+1} - \frac{n!}{i!(n-i-1)!} p^d (1-p)^{n-i-1} \frac{1-p}{n-i} \\ &= \frac{n!}{i!(n-i-1)!} p^i (1-p)^{n-i-1} \left( \frac{p}{i+1} - \frac{1-p}{n-i} \right). \end{aligned}$$

Since the term  $n! p^i (1-p)^{n-i-1}$  is always non-negative, the sign of  $b(i)$  is determined by the sign of  $p/(i+1) - (1-p)/(n-i)$ .

Solving for  $i$ , we get that

$$\frac{p}{i+1} - \frac{1-p}{n-i} \geq 0 \Leftrightarrow i \leq p(n+1) - 1.$$

Note that  $p(n+1) - 1$  may not be integer. Further note that the distribution of  $X$  is unimodal, as the sign of  $b$  changes at most once. Thus, each local maximum is also a global maximum. As discussed above, the largest value  $d \in [0, n-2] \cap \mathbf{N}$  such that  $\text{sgn}(b(d)) \geq 0$  and  $\text{sgn}(b(d+1)) \leq 0$  then results in a global maximum at position  $d+1$ . Since  $d$  needs to be integer, the largest value that satisfies this constraint is  $\lfloor p(n+1) - 1 \rfloor$ . If the sign of  $b$  is always negative ( $p \leq 1/(n+1)$ ), then the distribution of  $X$  has a global maximum at 0, which is also satisfied by  $\lfloor p(n+1) - 1 \rfloor + 1$ , which concludes the proof. ◀

► **Theorem 3** (Stirling's Formula [24, page 54]). *For all  $n \in \mathbf{N}^+$ , it holds that*

$$\sqrt{2\pi n}^{n+1/2} e^{-n} \cdot e^{(12n+1)^{-1}} < n! < \sqrt{2\pi n}^{n+1/2} e^{-n} \cdot e^{(12n)^{-1}}.$$

► **Corollary 4.** *For all  $n \geq 2$  with  $n \in \mathbf{N}$ , it holds that*

$$n! > \sqrt{2\pi n}^{n+1/2} e^{-n} \quad \text{and} \tag{1}$$

$$n! < e n^{n+1/2} e^{-n}. \tag{2}$$

**Proof.** For both inequalities, we aim at using Theorem 3.

Equation (1): Note that  $e^{(12n+1)^{-1}} > 1$ , since  $\frac{1}{12n+1} > 0$ . Hence,

$$\sqrt{2\pi n}^{n+1/2} e^{-n} < \sqrt{2\pi n}^{n+1/2} e^{-n} \cdot e^{(12n+1)^{-1}}.$$

Equation (2): We prove this case by showing that

$$\sqrt{2\pi} e^{(12n)^{-1}} < e. \tag{3}$$

## 29:6 The Flip Schelling Process on Random Graphs

Note, that  $e^{(12n)^{-1}}$  is strictly decreasing. Hence, we only have to check whether Equation (3) holds for  $n = 2$ .

$$\sqrt{2\pi}e^{(12n)^{-1}} \leq \sqrt{2\pi}e^{\frac{1}{24}} < 2.7 < e. \quad \blacktriangleleft$$

► **Lemma 5.** *Let  $A$ ,  $B$ , and  $C$  be random variables such that  $\Pr[A > C \wedge B > C] > 0$  and  $\Pr[A > C \wedge B \leq C] > 0$ . Then  $\Pr[A > B \wedge A > C] \geq \Pr[A > B] \cdot \Pr[A > C]$ .*

**Proof.** Using the definition of conditional probability, we obtain

$$\Pr[A > B \wedge A > C] = \Pr[A > B \mid A > C] \cdot \Pr[A > C].$$

Hence, we are left with bounding  $\Pr[A > B \mid A > C] \geq \Pr[A > B]$ . Partitioning the sample space into the two events  $B > C$  and  $B \leq C$  and using the law of total probability, we obtain

$$\begin{aligned} \Pr[A > B \mid A > C] &= \Pr[B > C \mid A > C] \cdot \Pr[A > B \mid A > C \wedge B > C] \\ &\quad + \Pr[B \leq C \mid A > C] \cdot \Pr[A > B \mid A > C \wedge B \leq C]. \end{aligned}$$

Note that the condition  $A > C \wedge B \leq C$  already implies  $A > B$  and thus the last probability equals to 1. Moreover, using the definition of conditional probability, we obtain

$$\begin{aligned} \Pr[A > B \mid A > C] &= \Pr[B > C \mid A > C] \cdot \frac{\Pr[A > B \wedge A > C \wedge B > C]}{\Pr[A > C \wedge B > C]} \\ &\quad + \Pr[B \leq C \mid A > C]. \end{aligned}$$

Using that  $\Pr[B > C \mid A > C] \geq \Pr[A > C \wedge B > C]$ , that  $A > B \wedge B > C$  already implies  $A > C$ , that  $\Pr[B \leq C \mid A > C] \geq \Pr[A > B \wedge B \leq C]$ , and finally the law of total probability, we obtain

$$\begin{aligned} \Pr[A > B \mid A > C] &\geq \Pr[A > B \wedge A > C \wedge B > C] + \Pr[B \leq C \mid A > C] \\ &= \Pr[A > B \wedge B > C] + \Pr[B \leq C \mid A > C] \\ &\geq \Pr[A > B \wedge B > C] + \Pr[A > B \wedge B \leq C] \\ &= \Pr[A > B]. \quad \blacktriangleleft \end{aligned}$$

### 3 Monochrome Edges in Geometric Random Graphs

In this section, we prove the following main theorem.

► **Theorem 6.** *Let  $G \sim \mathcal{G}(n, r)$  be a random geometric graph with expected average degree  $\bar{\delta} = o(\sqrt{n})$ . The expected fraction of monochrome edges after the FSP is at least*

$$\frac{1}{2} + \frac{9}{800} \cdot \left( \frac{1}{2} - \frac{1}{\sqrt{2\pi \lfloor \bar{\delta}/2 \rfloor}} \right)^2 \cdot \left( 1 - e^{-\bar{\delta}/2} \left( 1 + \frac{\bar{\delta}}{2} \right) \right) \cdot (1 - o(1)).$$

Note that the bound in Theorem 6 is bounded away from  $\frac{1}{2}$  for all  $\bar{\delta} \geq 2$ . Moreover, the two factors depending on  $\bar{\delta}$  go to  $\frac{1}{2}$  and 1, respectively, for a growing  $\bar{\delta}$ .

Given an edge  $\{u, v\}$ , we prove the above lower bound on the probability that  $\{u, v\}$  is monochrome in the following four steps.

1. For a vertex set, we introduce the concept of *decisiveness* that measures how much the majority is ahead of the minority in the FSP. With this, we give a lower bound on the probability that  $\{u, v\}$  is monochrome based on the probability that the common neighborhood of  $u$  and  $v$  is more decisive than their exclusive neighborhoods.

2. We show that large neighborhoods are likely to be more decisive than small neighborhoods. To this end, we give bounds on the likelihood that two similar random walks behave differently. This step reduces the question of whether the common neighborhood is more decisive than the exclusive neighborhoods to whether the former is larger than the latter.
3. Turning to geometric random graphs, we show that the common neighborhood is sufficiently likely to be larger than the exclusive neighborhoods if the geometric region corresponding to the former is sufficiently large. We do this by first showing that the actual distribution of the neighborhood sizes is well approximated by independent binomial random variables. Then, we give the desired bounds for these random variables.
4. We show that the existence of the edge  $\{u, v\}$  in the geometric random graph makes it sufficiently likely that the geometric region hosting the common neighborhood of  $u$  and  $v$  is sufficiently large.

### 3.1 Monochrome Edges via Decisive Neighborhoods

Let  $\{u, v\}$  be an edge of a given graph. To formally define the above mentioned decisiveness, let  $N_{u \cap v}^+$  and  $N_{u \cap v}^-$  be the number of vertices in the common neighborhood of  $u$  and  $v$  that are occupied by agents of type  $t^+$  and  $t^-$ , respectively. Then  $D_{u \cap v} := |N_{u \cap v}^+ - N_{u \cap v}^-|$  is the *decisiveness* of the common neighborhood of  $u$  and  $v$ . Analogously, we define  $D_{u \setminus v}$  and  $D_{v \setminus u}$  for the exclusive neighborhoods of  $u$  and  $v$ , respectively.

The following theorem bounds the probability for  $\{u, v\}$  to be monochrome based on the probability that the common neighborhood is more decisive than each of the exclusive ones.

► **Theorem 7.** *In the FSP, let  $\{u, v\} \in E$  be an edge and let  $D$  be the event  $\{D_{u \cap v} > D_{u \setminus v} \wedge D_{u \cap v} > D_{v \setminus u}\}$ . Then  $\{u, v\}$  is monochrome with probability at least  $1/2 + \Pr[D]/2$ .*

**Proof.** If  $D$  occurs, then the types of  $u$  and  $v$  after the FSP coincide with the predominant type before the FSP in the shared neighborhood. Thus,  $\{u, v\}$  is monochrome.

Otherwise, assuming  $\bar{D}$ , w.l.o.g., let  $D_{u \cap v} \leq D_{u \setminus v}$  and assume further the type of  $v$  has already been determined. If  $D_{u \cap v} = D_{u \setminus v}$ , then  $u$  chooses a type uniformly at random, which coincides with the type of  $v$  with probability  $\frac{1}{2}$ . Otherwise,  $D_{u \cap v} < D_{u \setminus v}$  and thus  $u$  takes the type that is predominant in  $u$ 's exclusive neighborhood, which is  $t^+$  and  $t^-$  with probability  $\frac{1}{2}$ , each. Moreover, this is independent from the type of  $v$  as  $v$ 's neighborhood is disjoint to  $u$ 's exclusive neighborhood.

Thus, for the event  $M$  that  $\{u, v\}$  is monochrome, we get  $\Pr[M | D] = 1$  and  $\Pr[M | \bar{D}] = \frac{1}{2}$ . Hence, we get  $\Pr[M] > \Pr[D] + \frac{1}{2}(1 - \Pr[D]) = \frac{1}{2} + \Pr[D]/2$ . ◀

### 3.2 Large Neighborhoods are More Decisive

The goal of this section is to reduce the question of how decisive a neighborhood is to the question of how large it is. To be more precise, assume we have a set of vertices of size  $a$  and give each vertex the type  $t^+$  and  $t^-$ , respectively, each with probability  $\frac{1}{2}$ . Let  $A_i$  for  $i \in [a]$  be the random variable that takes the value  $+1$  and  $-1$  if the  $i$ -th vertex in this set has type  $t^+$  and  $t^-$ , respectively. Then, for  $A = \sum_{i \in [a]} A_i$ , the decisiveness of the vertex set is  $|A|$ . In the following, we study the decisiveness  $|A|$  depending on the size  $a$  of the set. Note that this can be viewed as a random walk on the integer line: Starting at 0, in each step, it moves one unit either to the left or to the right with equal probabilities. We are interested in the distance from 0 after  $a$  steps.

Assume for the vertices  $u$  and  $v$  that we know that  $b$  vertices lie in the common neighborhood and  $a$  vertices lie in the exclusive neighborhood of  $u$ . Moreover, let  $A$  and  $B$  be the positions of the above random walk after  $a$  and  $b$  steps, respectively. Then the event

## 29:8 The Flip Schelling Process on Random Graphs

$D_{u \cap v} > D_{u \setminus v}$  is equivalent to  $|B| > |A|$ . Motivated by this, we study the probability of  $|B| > |A|$ , assuming  $b \geq a$ . The core difficulty here comes from the fact that we require  $|B|$  to be strictly larger than  $|A|$ . Also note that  $a + b$  corresponds to the degree of  $u$  in the graph. Thus, we have to study the random walks also for small numbers of  $a$  and  $b$ . We note that all results in this section are independent from the specific application to the FSP, and thus might be of independent interest.

Before we give a lower bound on the probability that  $|B| > |A|$ , we need the following technical lemma. It states that doing more steps in the random walk only makes it more likely to deviate further from the starting position.

► **Lemma 8.** *For  $i \in [a]$  and  $j \in [b]$  with  $0 \leq a \leq b$ , let  $A_i$  and  $B_j$  be independent random variables that are  $-1$  and  $1$  each with probability  $\frac{1}{2}$ . Let  $A = \sum_{i \in [a]} A_i$  and  $B = \sum_{j \in [b]} B_j$ . Then  $\Pr[|A| < |B|] \geq \Pr[|A| > |B|]$ .*

**Proof.** Let  $\Delta_k$  be the event that  $|B| - |A| = k$ . First note that

$$\Pr[|A| < |B|] = \sum_{k \in [b]} \Pr[\Delta_k] \quad \text{and} \quad \Pr[|A| > |B|] = \sum_{k \in [a]} \Pr[\Delta_{-k}].$$

To prove the statement of the lemma, it thus suffices to prove the following claim.

▷ **Claim 9.** For  $k \geq 0$ ,  $\Pr[\Delta_k] \geq \Pr[\Delta_{-k}]$ .

We prove this claim via induction on  $b - a$ . For the base case  $a = b$ ,  $A$  and  $B$  are equally distributed and thus Claim 9 clearly holds.

For the induction step, let  $B^+$  be the random variable that takes the values  $B + 1$  and  $B - 1$  with probability  $\frac{1}{2}$  each. Note that  $B^+$  represents the same type of random walk as  $A$  and  $B$  but with  $b + 1$  steps. Moreover  $B^+$  is coupled with  $B$  to make the same decisions in the first  $b$  steps. Let  $\Delta_k^+$  be the event that  $|B^+| - |A| = k$ . It remains to show that Claim 9 holds for these  $\Delta_k^+$ . For this, first note that the claim trivially holds for  $k = 0$ . For  $k \geq 1$ , we can use the definition of  $\Delta_k^+$  and the induction hypothesis to obtain

$$\begin{aligned} \Pr[\Delta_k^+] &= \frac{\Pr[\Delta_{k-1}]}{2} + \frac{\Pr[\Delta_{k+1}]}{2} \\ &\geq \frac{\Pr[\Delta_{-k+1}]}{2} + \frac{\Pr[\Delta_{-k-1}]}{2} = \Pr[\Delta_{-k}^+]. \end{aligned} \quad \blacktriangleleft$$

Using Lemma 8, we now prove the following general bound for the probability that  $|A| < |B|$ , depending on certain probabilities for binomially distributed variables.

► **Lemma 10.** *For  $i \in [a]$  and  $j \in [b]$  with  $0 \leq a \leq b$ , let  $A_i$  and  $B_j$  be independent random variables that are  $-1$  and  $1$  each with probability  $\frac{1}{2}$ . Let  $A = \sum_{i \in [a]} A_i$  and  $B = \sum_{j \in [b]} B_j$ . Moreover, let  $X \sim \text{Bin}(a, \frac{1}{2})$ ,  $Y \sim \text{Bin}(b, \frac{1}{2})$ , and  $Z \sim \text{Bin}(a + b, \frac{1}{2})$ . Then*

$$\Pr[|A| < |B|] \geq \frac{1}{2} - \Pr\left[Z = \frac{a+b}{2}\right] + \frac{\Pr[X = \frac{a}{2}] \cdot \Pr[Y = \frac{b}{2}]}{2}.$$

**Proof.** Using that  $\Pr[|A| < |B|] \geq \Pr[|A| > |B|]$  (see Lemma 8), we obtain

$$\begin{aligned} &\Pr[|A| < |B|] + \Pr[|A| > |B|] + \Pr[|A| = |B|] = 1 \\ \Rightarrow &2\Pr[|A| < |B|] + \Pr[|A| = |B|] \geq 1 \\ \Leftrightarrow &\Pr[|A| < |B|] \geq \frac{1}{2} - \frac{\Pr[|A| = |B|]}{2}. \end{aligned} \quad (4)$$

Thus, it remains to give an upper bound for  $\Pr[|A| = |B|]$ .

Using the inclusion–exclusion principle and the fact that  $B$  is symmetric around 0, i.e.,  $\Pr[B = x] = \Pr[B = -x]$  for any  $x$ , we obtain

$$\begin{aligned} \Pr[|A| = |B|] &= \Pr[A = B \vee A = -B] \\ &= \Pr[A = B] + \Pr[A = -B] - \Pr[A = B = 0] \\ &= 2\Pr[A = -B] - \Pr[A = B = 0]. \end{aligned} \tag{5}$$

We estimate  $\Pr[A = -B]$  and  $\Pr[A = B = 0]$  using bounds for binomially distributed variables. To this end, define new random variables  $X_i = \frac{A_i+1}{2}$  for  $i \in [a]$  and let  $X = \sum_{i \in [a]} X_i$ . Note that the  $X_i$  are independent and take values 0 and 1, each with probability  $\frac{1}{2}$ . Thus,  $X \sim \text{Bin}(a, \frac{1}{2})$ . Moreover,  $A = 2X - a$ . Analogously, we define  $Y$  with  $Y \sim \text{Bin}(b, \frac{1}{2})$  and  $B = 2Y - b$ . Note that  $X$  and  $Y$  are independent and thus  $Z = X + Y \sim \text{Bin}(a + b, \frac{1}{2})$ . With this, we get

$$\begin{aligned} \Pr[A = -B] &= \Pr[2X - a = -2Y + b] = \Pr\left[Z = \frac{a+b}{2}\right], \text{ and} \\ \Pr[A = B = 0] &= \Pr[A = 0] \cdot \Pr[B = 0] = \Pr\left[X = \frac{a}{2}\right] \cdot \Pr\left[Y = \frac{b}{2}\right]. \end{aligned}$$

This, together with Equations (4) and (5) yield the claim. ◀

The bound in Lemma 10 becomes worse for smaller values of  $a$  and  $b$ . Considering this worst case, we obtain the following specific bound.

► **Theorem 11.** *For  $i \in [a]$  and  $j \in [b]$  with  $0 \leq a \leq b$ , let  $A_i$  and  $B_j$  be independent random variables that are  $-1$  and  $1$  each with probability  $\frac{1}{2}$ . Let  $A = \sum_{i \in [a]} A_i$  and  $B = \sum_{j \in [b]} B_j$ . If  $a = b = 0$  or  $a = b = 1$ , then  $\Pr[|A| < |B|] = 0$ . Otherwise  $\Pr[|A| < |B|] \geq \frac{3}{16}$ .*

**Proof.** Clearly, if  $a = b = 0$ , then  $A = B = 0$  and thus  $\Pr[|A| < |B|] = 0$ . Similarly, if  $a = b = 1$ , then  $|A| = |B| = 1$  and thus  $\Pr[|A| < |B|] = 0$ . For the remainder, assume that neither  $a = b = 0$  nor  $a = b = 1$ , and let  $X, Y$ , and  $Z$  be defined as in Lemma 10, i.e.,  $X \sim \text{Bin}(a, \frac{1}{2})$ ,  $Y \sim \text{Bin}(b, \frac{1}{2})$ , and  $Z \sim \text{Bin}(a + b, \frac{1}{2})$ .

If  $a + b$  is odd, then  $\Pr[Z = \frac{a+b}{2}] = 0$ . Thus, by Lemma 10, we have  $\Pr[|A| < |B|] \geq \frac{1}{2}$ . If  $a + b$  is even and  $a + b \geq 6$ , then

$$\Pr\left[Z = \frac{a+b}{2}\right] = \binom{a+b}{\frac{a+b}{2}} \left(\frac{1}{2}\right)^{a+b} \leq \binom{6}{3} \left(\frac{1}{2}\right)^6 = \frac{5}{16}.$$

Hence, by Lemma 10, we have  $\Pr[|A| < |B|] \geq \frac{1}{2} - \frac{5}{16} = \frac{3}{16}$ .

If  $a + b < 6$  (and  $a + b$  is even), there are four cases:  $a = 0, b = 2$ ;  $a = 0, b = 4$ ;  $a = 1, b = 3$ ;  $a = 2, b = 2$ . If  $a = 0$  and  $b = 2$ , then  $A = 0$  with probability 1 and  $|B| = 2$  with probability  $\frac{1}{2}$ . Thus,  $\Pr[|A| < |B|] = \frac{1}{2}$ . If  $a = 0$  and  $b = 4$ , then  $|A| < |B|$  unless  $B = 0$ . As  $\Pr[B = 0] = \binom{4}{2} \cdot (\frac{1}{2})^4 = \frac{3}{8}$ , we get  $\Pr[|A| < |B|] = 1 - \frac{3}{8} = \frac{5}{8}$ . If  $a = 1$  and  $b = 3$ , then  $|A| = 1$  with probability 1 and  $|B| = 3$  with probability  $\frac{1}{4}$  (either  $B_1 = B_2 = B_3 = 1$  or  $B_1 = B_2 = B_3 = -1$ ). Thus,  $\Pr[|A| < |B|] = \frac{1}{4}$ . If  $a = b = 2$ , then  $|A| = 0$  with probability  $\frac{1}{2}$  and  $|B| = 2$  with probability  $\frac{1}{2}$ . Thus  $\Pr[|A| < |B|] = \frac{1}{4}$ .

We note that the bound of  $\Pr[|A| < |B|] = \frac{3}{16}$  is tight for  $a = b = 3$ . ◀

### 3.3 Large Common Regions Yield Large Common Neighborhoods

To be able to apply Theorem 11 to an edge  $\{u, v\}$ , we need to make sure that the size of their common neighborhood (corresponding to  $b$  in the theorem) is at least the size of the exclusive neighborhoods (corresponding to  $a$  in the theorem). In the following, we give bounds for the probability that this happens. Note that this is the first time we actually take the graph into account. Thus, all above considerations hold for arbitrary graphs.

Recall that we consider random geometric graphs  $\mathcal{G}(n, r)$  and  $u$  and  $v$  are each connected to all vertices that lie within a disk of radius  $r$  around them. As  $u$  and  $v$  are adjacent, their disks intersect, which separates the ground space into four regions; cf. Fig 2a. Let  $R_{u \cap v}$  be the intersection of the two disks. Let  $R_{u \setminus v}$  be the set of points that lie in the disk of  $u$  but not in the disk of  $v$ , and analogously, let  $R_{v \setminus u}$  be the disk of  $v$  minus the disk of  $u$ . Finally, let  $R_{\overline{u \cup v}}$  be the set of points outside both disks. Then, each of the  $n - 2$  remaining vertices ends up in exactly one of these regions with a probability equal to the corresponding measure. Let  $\mu(\cdot)$  be the area of the respective region and  $p = \mu(R_{u \cap v})$  and  $q = \mu(R_{u \setminus v}) = \mu(R_{v \setminus u})$  be the probabilities for a vertex to lie in the common and exclusive regions, respectively. The probability for  $R_{\overline{u \cup v}}$  is then  $1 - p - 2q$ .

We are now interested in the sizes  $N_{u \cap v}$ ,  $N_{u \setminus v}$ , and  $N_{v \setminus u}$  of the common and the exclusive neighborhoods, respectively. As each of the  $n - 2$  remaining vertices ends up in  $N_{u \cap v}$  with probability  $p$ , we have  $N_{u \cap v} \sim \text{Bin}(n - 2, p)$ . For  $N_{u \setminus v}$  and  $N_{v \setminus u}$ , we already know that  $v$  is a neighbor of  $u$  and vice versa. Thus,  $(N_{u \setminus v} - 1) \sim \text{Bin}(n - 2, q)$  and  $(N_{v \setminus u} - 1) \sim \text{Bin}(n - 2, q)$ . Moreover, the three random variables are not independent, as each vertex lies in only exactly one of the four neighborhoods, i.e.,  $N_{u \cap v}$ ,  $(N_{u \setminus v} - 1)$ ,  $(N_{v \setminus u} - 1)$ , and the number of vertices in neither neighborhood together follow a multinomial distribution  $\text{Multi}(n - 2, \mathbf{p})$  with  $\mathbf{p} = (p, q, q, 1 - p - 2q)$ .

The following lemma shows that these dependencies are small if  $p$  and  $q$  are sufficiently small. This lets us assume that  $N_{u \cap v}$ ,  $(N_{u \setminus v} - 1)$ ,  $(N_{v \setminus u} - 1)$  are independent random variables following binomial distributions if the expected average degree is not too large.

► **Lemma 12.** *Let  $X = (X_1, X_2, X_3, X_4) \sim \text{Multi}(n, \mathbf{p})$  with  $\mathbf{p} = (p, q, q, 1 - p - 2q)$ . Then there exist independent random variables  $Y_1 \sim \text{Bin}(n, p)$ ,  $Y_2 \sim \text{Bin}(n, q)$ , and  $Y_3 \sim \text{Bin}(n, q)$  such that  $\Pr[(X_1, X_2, X_3) = (Y_1, Y_2, Y_3)] \geq 1 - 3n \cdot \max(p, q)^2$ .*

**Proof.** Let  $Y_1 \sim \text{Bin}(n, p)$ , and  $Y_2, Y_3 \sim \text{Bin}(n, q)$  be independent random variables. We define the event  $B$  to hold, if each of the  $n$  individual trials increments at most one of the random variables  $Y_1, Y_2$ , or  $Y_3$ . More formally, for  $i \in [3]$  and  $j \in [n]$ , let  $Y_{i,j}$  be the individual Bernoulli trials of  $Y_i$ , i.e.,  $Y_i = \sum_{j \in [n]} Y_{i,j}$ . For  $j \in [n]$ , we define the event  $B_j$  to be  $Y_{1,j} + Y_{2,j} + Y_{3,j} \leq 1$ , and the event  $B = \bigcap_{j \in [n]} B_j$ .

Based on this, we now define the random variables  $X_1, X_2, X_3$ , and  $X_4$  as follows. If  $B$  holds, we set  $X_i = Y_i$  for  $i \in [3]$  and  $X_4 = n - X_1 - X_2 - X_3$ . Otherwise, if  $\overline{B}$ , we draw  $X = (X_1, X_2, X_3, X_4) \sim \text{Multi}(n, \mathbf{p})$  independently from  $Y_1, Y_2$ , and  $Y_3$  with  $\mathbf{p} = (p, q, q, 1 - p - 2q)$ . Note that  $X$  clearly follows  $\text{Multi}(n, \mathbf{p})$  if  $\overline{B}$ . Moreover, conditioned on  $B$ , each individual trial increments exactly one of the variables  $X_1, X_2, X_3$ , or  $X_4$  with probabilities  $p, q, q$ , and  $1 - p - 2q$ , respectively, i.e.,  $X \sim \text{Multi}(n, \mathbf{p})$ .

Thus, we end up with  $X \sim \text{Multi}(n, \mathbf{p})$ . Additionally, we have three independent random variables  $Y_1 \sim \text{Bin}(n, p)$ , and  $Y_2, Y_3 \sim \text{Bin}(n, q)$  with  $(X_1, X_2, X_3) = (Y_1, Y_2, Y_3)$  if  $B$  holds. Thus, to prove the lemma, it remains to show that  $\Pr[B] \geq 1 - 3n \max(p, q)^2$ . For  $j \in [n]$ , the probability that the  $j$ th trial goes wrong is

$$\begin{aligned} \Pr [\overline{B}_j] &= 1 - ((1-p)(1-q)^2) - (p(1-q)^2) - 2(q(1-p)(1-q)) \\ &= 2pq - 2pq^2 + q^2 \leq 2pq + q^2 \leq 3 \cdot \max(p, q)^2. \end{aligned}$$

Using the union bound it follows that  $\Pr [\overline{B}] \leq \sum_{j \in [n]} \Pr [\overline{B}_j] \leq 3n \cdot \max(p, q)^2$ . ◀

As mentioned before, we are interested in the event  $N_{u \cap v} \geq N_{u \setminus v}$  (and likewise  $N_{u \cap v} \geq N_{v \setminus u}$ ), in order to apply Theorem 11. Moreover, due to Lemma 12, we know that  $N_{u \cap v}$  and  $(N_{u \setminus v} - 1)$  almost behave like independent random variables that follow  $\text{Bin}(n-2, p)$  and  $\text{Bin}(n-2, q)$ , respectively. The following lemma helps to bound the probability for  $N_{u \cap v} \geq N_{u \setminus v}$ . Note that it gives a bound for the probability of achieving strict inequality (instead of just  $\geq$ ), which accounts for the fact that  $(N_{u \setminus v} - 1)$  and not  $N_{u \setminus v}$  itself follows a binomial distribution.

► **Lemma 13.** *Let  $n \in \mathbf{N}$  with  $n \geq 2$ , and let  $p \geq q > 0$ . Further, let  $X \sim \text{Bin}(n, p)$  and  $Y \sim \text{Bin}(n, q)$  be independent, let  $d = \lfloor p(n+1) \rfloor$ , and assume  $d = o(\sqrt{n})$ , then  $\Pr [X > Y] \geq (\frac{1}{2} - 1/\sqrt{2\pi d})(1 - o(1))$ .*

**Proof.** By Lemma 1, we get  $\Pr [X \geq Y] \geq \frac{1}{2}$ , and we bound

$$\Pr [X > Y] = \Pr [X \geq Y] - \Pr [X = Y] \geq \frac{1}{2} - \Pr [X = Y],$$

leaving us to bound  $\Pr [X = Y]$  from above. By independence of  $X$  and  $Y$ , we get

$$\Pr [X = Y] = \sum_{i \in [n]} \Pr [X = i] \cdot \Pr [Y = i]. \tag{6}$$

Note that, by Lemma 2, for all  $i \in [0..n]$ , it holds that  $\Pr [X = i] \leq \Pr [X = d]$ . Assume that we have a bound  $B$  such that  $\Pr [X = d] \leq B$ . Substituting this into Equation (6) yields

$$\Pr [X = Y] \leq B \sum_{i \in [n]} \Pr [Y = i] = B,$$

resulting in  $\Pr [X > Y] \geq \frac{1}{2} - B$ . Thus, we now derive such a bound for  $B$  and apply the inequality that for all  $x \in \mathbf{R}$ , it holds that  $1 + x \leq e^x$ , as well as Equation (1). We get

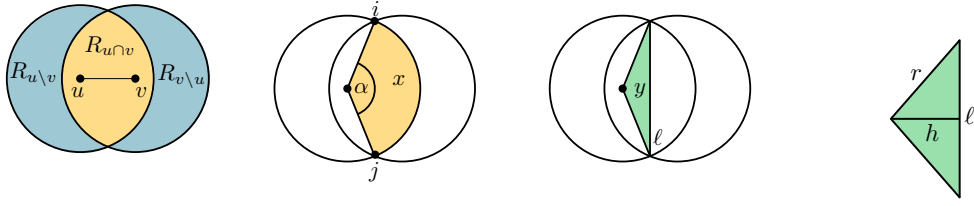
$$\begin{aligned} \binom{n}{d} p^d (1-p)^{n-d} &\leq \frac{n^d}{d!} \left(\frac{d}{n}\right)^d \left(1 - \frac{d}{n}\right)^n \left(1 - \frac{d}{n}\right)^{-d} \\ &\leq \frac{d^d}{d!} e^{-d} \left(1 - \frac{d}{n}\right)^{-d} \\ &\leq \frac{d^d}{\sqrt{2\pi d}^{d+1/2} e^{-d}} e^{-d} \left(1 - \frac{d}{n}\right)^{-d} \\ &= \frac{1}{\sqrt{2\pi d}} \frac{1}{(1 - d/n)^d}. \end{aligned} \tag{7}$$

By Bernoulli's inequality, we bound  $(1 - d/n)^d \geq 1 - d^2/n = 1 - o(1)$  by the assumption  $d = o(\sqrt{n})$ . Substituting this back into Equation (7) concludes the proof. ◀

Finally, in order to apply Theorem 11, we have to make sure not to end up in the special case where  $a = b \leq 1$ , i.e., we have to make sure that the common neighborhood includes at least two vertices. The probability for this to happen is given by the following lemma.



## 29:12 The Flip Schelling Process on Random Graphs



(a) The geometric regions corresponding to the common and exclusive neighborhoods, respectively, with yellow illustrating  $R_{u \cap v}$  and blue illustrating  $R_{u \setminus v}$  and  $R_{v \setminus u}$ .

(b) Let  $\alpha$  be the central angle determined by the intersection points  $i$  and  $j$ , and let  $x$  be the corresponding circular sector (illustrated in yellow).

(c) Let  $y$  be a triangle in the intersection (illustrated in green) determined by the radical axis  $\ell$  and the central angle  $\alpha$ , cf. Fig 2b.

(d) The height  $h$  divides the area  $\mu(y)$  (illustrated in green) of the triangle  $y$ , cf. Fig 2c, into two subareas of equal size, since adjacent and opposite legs have the same length  $r$ .

■ **Figure 2** The neighborhood of two adjacent vertices  $u$  and  $v$  in a random geometric graph.

► **Lemma 14.** Let  $X \sim \text{Bin}(n, p)$  and let  $c = np \in o(n)$ . Then it holds that  $\Pr[X > 1] \geq (1 - e^{-c}(1 + c))(1 - o(1))$ .

**Proof.** As  $X > 1$  holds if and only if  $X \neq 0$  and  $X \neq 1$ , we get

$$\Pr[X > 1] = 1 - \Pr[X = 0] - \Pr[X = 1] = 1 - (1 - p)^n - n \cdot p \cdot (1 - p)^{n-1}.$$

Using that for all  $x \in \mathbf{R}$  it holds that  $1 - x \leq e^{-x}$ , we get

$$\begin{aligned} \Pr[X > 1] &\geq 1 - e^{-pn} - n \cdot p \cdot e^{-p(n-1)} \\ &= 1 - e^{-c} - c \cdot e^{c/n} \cdot e^{-c} \\ &= 1 - e^{-c} \left(1 + c \cdot e^{c/n}\right). \end{aligned}$$

As  $e^{c/n}$  goes to 1 for  $n \rightarrow \infty$ , we get the claimed bound. ◀

### 3.4 Many Edges Have Large Common Regions

In Section 3.3, we derived a lower bound on the probability that  $N_{u \cap v} \geq N_{u \setminus v}$  provided that the probability for a vertex to end up in the shared region  $R_{u \cap v}$  is sufficiently large compared to  $R_{u \setminus v}$ . In the following, we estimate the measures of these regions depending on the distance between  $u$  and  $v$ . Then, we give a lower bound on the probability that  $\mu(R_{u \cap v}) \geq \mu(R_{u \setminus v})$ .

► **Lemma 15.** Let  $G \sim \mathcal{G}(n, r)$  be a random geometric graph with expected average degree  $\bar{\delta}$ , let  $\{u, v\} \in E$  be an edge, and let  $\tau := \frac{\text{dist}(u, v)}{r}$ . Then,

$$\mu(R_{u \cap v}) = \frac{\bar{\delta}}{(n-1)\pi} \left( 2 \arccos\left(\frac{\tau}{2}\right) - \sin\left(2 \arccos\left(\frac{\tau}{2}\right)\right) \right) \text{ and} \quad (8)$$

$$\mu(R_{u \setminus v}) = \mu(R_{v \setminus u}) = \frac{\bar{\delta}}{n-1} - \mu(R_{u \cap v}). \quad (9)$$

**Proof.** We start with proving Equation (8). Let  $i$  and  $j$  be the two intersection points of the disks of  $u$  and  $v$ , let  $\alpha$  be the central angle enclosed by  $i$  and  $j$ , and let  $x$  be the corresponding circular sector, cf. Fig 2b. Moreover, let the triangle  $y$  be a subarea of  $x$  determined by  $\alpha$

and the radical axis  $\ell$ , cf. Fig 2c. Let  $h$  denote the height of the triangle  $y$ , cf. Fig 2d. For our calculations, we restrict the length of  $\ell$  by the intersection points  $i$  and  $j$ . Since we consider the intersection between disks and thus  $\ell$  divides the area  $\mu(R_{u \cap v})$  into two subareas of equal sizes, it holds that  $\mu(R_{u \cap v}) = 2(\mu(x) - \mu(y))$ . Considering the two areas  $\mu(x)$  and  $\mu(y)$ , it holds that

$$\mu(x) = \frac{\alpha}{2}r^2 \quad \text{and} \quad \mu(y) = h \cdot \frac{\ell}{2} = \cos\left(\frac{\alpha}{2}\right)r \cdot \sin\left(\frac{\alpha}{2}\right)r = \frac{\sin(\alpha)}{2}r^2. \quad (10)$$

For the central angle  $\alpha$  we know  $\cos(\alpha/2) = h/r = \tau/2$  and therefore  $\alpha = 2 \arccos(\frac{\tau}{2})$ . Together with Equation (10), we obtain

$$\mu(R_{u \cap v}) = 2(\mu(x) - \mu(y)) = 2 \left( \frac{2 \arccos(\frac{\tau}{2})}{2}r^2 - \frac{\sin(2 \arccos(\frac{\tau}{2}))}{2}r^2 \right). \quad (11)$$

The area of a general circle is equal to  $\pi r^2$ . Since we consider a ground space with total area 1, the area of one disk in the random geometric graph equals  $\frac{\delta}{n-1}$ , i.e.,  $r^2 = \frac{\delta}{(n-1)\pi}$ . Together with Equation (11), we obtain Equation (8).

Equation (9): We get the claimed equality by noting that  $\mu(R_{u \cap v}) + \mu(R_{u \setminus v}) = \pi r^2$ . ◀

► **Lemma 16.** *Let  $G \sim \mathcal{G}(n, r)$  be a random geometric graph, and let  $\{u, v\} \in E$  be an edge. Then  $\Pr[\mu(R_{u \cap v}) \geq \mu(R_{u \setminus v})] \geq (\frac{4}{5})^2$ .*

**Proof.** Let  $\tau = \frac{\text{dist}(u,v)}{r}$ . By Lemma 15 with  $\mu(R_{u \cap v}) \geq \mu(R_{v \setminus u})$ , we get

$$\left( 2 \arccos\left(\frac{\tau}{2}\right) - \sin\left(2 \arccos\left(\frac{\tau}{2}\right)\right) \right) \geq \frac{\pi}{2},$$

which is true for  $\tau \geq \frac{4}{5}$ . The area of a disk of radius  $\frac{4}{5}r$  is  $(\pi(\frac{4}{5}r)^2) / (\pi r^2) = (\frac{4}{5})^2$  times the area of a disk of radius  $r$ . Hence, the fraction of edges with distance at most  $\frac{4}{5}r$  is at least  $(\frac{4}{5})^2$ , concluding the proof. ◀

### 3.5 Proof of Theorem 6

By Theorem 7, the probability that a random edge  $\{u, v\}$  is monochrome is at least  $\frac{1}{2} + \Pr[D]/2$ , where  $D$  is the event that the common neighborhood of  $u$  and  $v$  is more decisive than each exclusive neighborhood. It remains to bound  $\Pr[D]$ .

**Existence of an edge yields a large shared region.** Let  $R$  be the event that  $\mu(R_{u \cap v}) \geq \mu(R_{u \setminus v})$ . Note that this also implies  $\mu(R_{u \cap v}) \geq \mu(R_{v \setminus u})$  as  $\mu(R_{u \setminus v}) = \mu(R_{v \setminus u})$ . Due to the law of total probability, we have

$$\Pr[D] \geq \Pr[R] \cdot \Pr[D | R].$$

Due to Lemma 16, we have  $\Pr[R] \geq (\frac{4}{5})^2$ . By conditioning on  $R$  in the following, we can assume that  $\mu(R_{u \cap v}) \geq \frac{\bar{\delta}}{2n} \geq \mu(R_{u \setminus v}) = \mu(R_{v \setminus u})$ , where  $\bar{\delta}$  is the expected average degree.

**Neighborhood sizes are roughly binomially distributed.** The next step is to go from the size of the regions to the number of vertices in these regions. Each of the remaining  $n' = n - 2$  vertices is sampled independently to lie in one of the regions  $R_{u \cap v}$ ,  $R_{u \setminus v}$ ,  $R_{v \setminus u}$ , or  $R_{\overline{u \cup v}}$ . Denote the resulting numbers of vertices with  $X_1$ ,  $X_2$ ,  $X_3$ , and  $X_4$ , respectively. Then  $(X_1, X_2, X_3, X_4)$  follows a multinomial distribution with parameter  $\mathbf{p} = (p, q, q, 1 - p - 2q)$

## 29:14 The Flip Schelling Process on Random Graphs

for  $p = \mu(R_{u \cap v})$  and  $q = \mu(R_{u \setminus v}) = \mu(R_{v \setminus u})$ . Note that  $N_{u \cap v} = X_1$ ,  $N_{u \setminus v} = X_2 + 1$ , and  $N_{v \setminus u} = X_3 + 1$  holds for the sizes of the common and exclusive neighborhoods, where the  $+1$  comes from the fact that  $v$  is always a neighbor of  $u$  and vice versa.

We apply Lemma 12 to obtain independent binomially distributed random variables  $Y_1$ ,  $Y_2$ , and  $Y_3$  that are likely to coincide with  $X_1 = N_{u \cap v}$ ,  $X_2 = N_{u \setminus v} - 1$ , and  $X_3 = N_{v \setminus u} - 1$ , respectively. Let  $B$  denote the event that  $(N_{u \cap v}, N_{u \setminus v} - 1, N_{v \setminus u} - 1) = (Y_1, Y_2, Y_3)$ . Again, using the law of total probabilities and due to the fact that  $R$  and  $B$  are independent, we get

$$\Pr[D | R] \geq \Pr[B | R] \cdot \Pr[D | R \cap B] = \Pr[B] \cdot \Pr[D | R \cap B].$$

Note that  $p, q \leq \frac{\bar{\delta}}{n}$  for the expected average degree  $\bar{\delta}$ . Thus, Lemma 12 implies that  $\Pr[B] \geq \left(1 - 3\bar{\delta}^2/n\right)$ . Conditioning on  $B$  makes it correct to assume that  $N_{u \cap v} \sim \text{Bin}(n', p)$ ,  $(N_{u \setminus v} - 1) \sim \text{Bin}(n', q)$ ,  $(N_{v \setminus u} - 1) \sim \text{Bin}(n', q)$  are independently distributed. Additionally conditioning on  $R$  gives us  $p \geq \frac{\bar{\delta}}{2n} \geq q$ .

**A large shared region yields a large shared neighborhood.** In the next step, we consider an event that makes sure that the number  $N_{u \cap v}$  of vertices in the shared neighborhood is sufficiently large. Let  $N_1$ ,  $N_2$ , and  $N_3$  be the events that  $N_{u \cap v} \geq N_{u \setminus v}$ ,  $N_{u \cap v} \geq N_{v \setminus u}$ , and  $N_{u \cap v} > 1$ , respectively. Let  $N$  be the intersection of  $N_1$ ,  $N_2$ , and  $N_3$ . We obtain

$$\begin{aligned} \Pr[D | R \cap B] &\geq \Pr[N | R \cap B] \cdot \Pr[D | R \cap B \cap N] \\ &\geq \Pr[N_1 | R \cap B] \cdot \Pr[N_2 | R \cap B] \cdot \Pr[N_3 | R \cap B] \cdot \Pr[D | R \cap B \cap N], \end{aligned}$$

where the last step follows from Lemma 5 as the inequalities in  $N_1$ ,  $N_2$ , and  $N_3$  all go in the same direction. Note that  $N_{u \cap v} \geq N_{u \setminus v}$  is equivalent to  $N_{u \cap v} > N_{u \setminus v} - 1$ . Due to the condition on  $B$ ,  $N_{u \cap v}$  and  $N_{u \setminus v} - 1$  are independent random variables following  $\text{Bin}(n', p)$  and  $\text{Bin}(n', q)$ , respectively, with  $p \geq q$  due to the condition on  $R$ . Thus, we can apply Lemma 13, to obtain

$$\Pr[N_1 | R \cap B] = \Pr[N_2 | R \cap B] \geq \frac{1}{2} - \frac{1}{\sqrt{2\pi[\bar{\delta}/2](1 - o(1))}},$$

and Lemma 14 gives the bound

$$\Pr[N_3 | R \cap B] \geq 1 - e^{-\bar{\delta}/2} \left(1 + \frac{\bar{\delta}}{2} \cdot (1 + o(1))\right).$$

Note that both of these probabilities are bounded away from 0 for  $\bar{\delta} \geq 2$ . Conditioning on  $N$  lets us assume that the shared neighborhood of  $u$  and  $v$  contains at least two vertices and that it is at least as big as each of the exclusive neighborhoods.

**A large shared neighborhood yields high decisiveness.** The last step is to actually bound the remaining probability  $\Pr[D | R \cap B \cap N]$ . Note that, once we know the number of vertices in the shared and exclusive neighborhoods, the decisiveness no longer depends on  $R$  or  $B$ , i.e., we can bound  $\Pr[D | N]$  instead. For this, let  $D_1$  and  $D_2$  be the events that  $D_{u \cap v} > D_{u \setminus v}$  and  $D_{u \cap v} > D_{v \setminus u}$ , respectively. Note that  $D$  is their intersection. Moreover, due to Lemma 5, we have  $\Pr[D | N] \geq \Pr[D_1 | N] \cdot \Pr[D_2 | N]$ . To bound  $\Pr[D_1 | N] = \Pr[D_2 | N]$ , we use Theorem 11. Note that the  $b$  and  $a$  in Theorem 11 correspond to  $N_{u \cap v}$  and  $N_{u \setminus v} + 1$  (the  $+1$  coming from the fact that  $N_{u \setminus v}$  does not count the vertex  $v$ ). Moreover conditioning on  $N$  implies that  $a \leq b$  and  $b > 1$ . Thus, Theorem 11 implies  $\Pr[D_1 | N] \geq \frac{3}{16}$ .

**Conclusion.** The above arguments give us that the fraction of monochrome edges is

$$\frac{1}{2} + \frac{\Pr[D]}{2} \geq \frac{1}{2} + \frac{1}{2} \cdot \underbrace{\Pr[R]}_{\geq (\frac{4}{5})^2} \cdot \underbrace{\Pr[B]}_{1-o(1)} \cdot \underbrace{(\Pr[N_1 | R \cap B])^2}_{\geq \frac{1}{2} - \frac{1}{\sqrt{2\pi \lfloor \bar{\delta}/2 \rfloor}}} \cdot \underbrace{\Pr[N_3 | R \cap B]}_{\geq 1 - e^{-\bar{\delta}/2} (1 + \frac{\bar{\delta}}{2})} \cdot \underbrace{(\Pr[D_1 | N])^2}_{\geq \frac{3}{16}},$$

where we omitted the  $o(1)$  terms for  $\Pr[N_1 | R \cap B]$  and  $\Pr[N_3 | R \cap B]$ , as they are already covered by the  $1 + o(1)$  coming from  $\Pr[B]$ . This yields the bound stated in Theorem 6:

$$\frac{1}{2} + \frac{9}{800} \cdot \left( \frac{1}{2} - \frac{1}{\sqrt{2\pi \lfloor \bar{\delta}/2 \rfloor}} \right)^2 \cdot \left( 1 - e^{-\bar{\delta}/2} \left( 1 + \frac{\bar{\delta}}{2} \right) \right) \cdot (1 - o(1)).$$

#### 4 Monochrome Edges in Erdős–Rényi Graphs

In the following, we are interested in the probability that an edge  $\{u, v\}$  is monochrome after the FSP on Erdős–Rényi graphs. In contrast to geometric random graphs, we prove an upper bound. To this end, we show that it is likely that the common neighborhood is empty and therefore  $u$  and  $v$  choose their types to be the predominant type in their exclusive neighborhood, which is  $t^+$  and  $t^-$  with probability  $\frac{1}{2}$ , each.

► **Theorem 17.** *Let  $G \sim \mathcal{G}(n, p)$  be an Erdős–Rényi graph with expected average degree  $\bar{\delta} = o(\sqrt{n})$ . The expected fraction of monochrome edges after the FSP is at most  $\frac{1}{2} + o(1)$ .*

**Proof.** Given an edge  $\{u, v\}$ , let  $M$  be the event that  $\{u, v\}$  is monochrome. We first split  $M$  into disjoint sets with respect to the size of the common neighborhood and apply the law of total probability and get

$$\begin{aligned} \Pr[M] &= \Pr[M | N_{u \cap v} = 0] \cdot \Pr[N_{u \cap v} = 0] + \Pr[M | N_{u \cap v} > 0] \cdot \Pr[N_{u \cap v} > 0] \\ &\leq \Pr[M | N_{u \cap v} = 0] \cdot 1 + 1 \cdot \Pr[N_{u \cap v} > 0]. \end{aligned}$$

We bound each of the summands separately. For estimating  $\Pr[M | N_{u \cap v} = 0]$ , we note that the types of  $u$  and  $v$  are determined by the predominant type in disjoint vertex sets. By definition of the FSP this implies that the probability of a monochrome edge is equal to  $\frac{1}{2}$ .

We are left with bounding  $\Pr[N_{u \cap v} > 0]$ . Note that  $N_{u \cap v} \sim \text{Bin}(n, p^2)$ . Thus, by Bernoulli's inequality we get  $\Pr[N_{u \cap v} > 0] = 1 - \Pr[N_{u \cap v} = 0] = 1 - (1 - p^2)^n \leq np^2$ . Noting that  $np^2 = o(1)$  holds due to our assumption on  $\bar{\delta}$ , concludes the proof. ◀

---

#### References

- 1 Aishwarya Agarwal, Edith Elkind, Jiarui Gan, and Alexandros A. Voudouris. Swap stability in Schelling games on graphs. In *AAAI'20*, pages 1758–1765, 2020. URL: <https://aaai.org/ojs/index.php/AAAI/article/view/5541>.
- 2 Jennifer Badham, Frank Kee, and Ruth F. Hunter. Network structure influence on simulated network interventions for behaviour change. *Social Networks*, 64:55–62, 2021.
- 3 Paul Balister, Béla Bollobás, J. Robert Johnson, and Mark Walters. Random majority percolation. *Random Structures & Algorithms*, 36(3):315–340, 2010.
- 4 George Barmpalias, Richard Elwes, and Andrew Lewis-Pye. Tipping points in 1-dimensional Schelling models with switching agents. *Journal of Statistical Physics*, 158(4):1572–9613, 2015.
- 5 George Barmpalias, Richard Elwes, and Andrew Lewis-Pye. Unperturbed Schelling segregation in two or three dimensions. *Journal of Statistical Physics*, 164(6):1460–1487, 2016.

- 6 George Barmpalias, Richard Elwes, and Andrew Lewis-Pye. Minority population in the one-dimensional Schelling model of segregation. *Journal of Statistical Physics*, 173(5):1572–9613, 2018.
- 7 George Barmpalias, Richard Elwes, and Andy Lewis-Pye. Digital morphogenesis via Schelling segregation. In *FOCS'14*, pages 156–165, 2014.
- 8 Stephen Benard and Robb Willer. A wealth and status-based model of residential segregation. *Journal of Mathematical Sociology*, 31(2):149–174, 2007.
- 9 Itzhak Benenson, Erez Hatna, and Ehud Or. From Schelling to spatially explicit modeling of urban ethnic and economic residential dynamics. *Sociological Methods and Research*, 37(4):463–497, 2009.
- 10 Prateek Bhakta, Sarah Miracle, and Dana Randall. Clustering and mixing times for segregation models on  $\mathbb{Z}^2$ . In *SODA'14*, pages 327–340, 2014.
- 11 Davide Bilò, Vittorio Bilò, Pascal Lenzner, and Louise Molitor. Topological influence and locality in swap Schelling games. In *MFCS'20*, pages 15:1–15:15, 2020.
- 12 Thomas Bläsius, Tobias Friedrich, Martin S. Krejca, and Louise Molitor. The flip Schelling process on random geometric and Erdős-Rényi graphs, 2021. [arXiv:2102.09856](https://arxiv.org/abs/2102.09856).
- 13 Christina Brandt, Nicole Immorlica, Gautam Kamath, and Robert Kleinberg. An analysis of one-dimensional Schelling segregation. In *STOC'12*, pages 789–804, 2012.
- 14 Paulo R. A. Campos, Viviane M. de Oliveira, and F. G. Brady Moreira. Small-world effects in the majority-vote model. *Physical Review E*, 67:026104, February 2003.
- 15 Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Review of Modern Physics*, 81:591–646, 2009.
- 16 Hau Chan, Mohammad T. Irfan, and Cuong Viet Than. Schelling models with localized social influence: A game-theoretic framework. In *AAMAS'20*, pages 240–248, 2020.
- 17 Ankit Chauhan, Pascal Lenzner, and Louise Molitor. Schelling segregation with strategic agents. In *SAGT'18*, pages 137–149. Springer, 2018.
- 18 William A. V. Clark. Residential segregation in american cities: A review and interpretation. *Population Research and Policy Review*, 5(2):95–127, 1986.
- 19 Vasco Cortez and Sergio Rica. Dynamics of the Schelling social segregation model in networks. *Procedia Computer Science*, 61:60–65, 2015.
- 20 Richard Durrett and Jeffrey E. Steif. Fixation results for threshold voter systems. *Annals of Probability*, 21(1):232–247, 1993.
- 21 Hagen Echezell, Tobias Friedrich, Pascal Lenzner, Louise Molitor, Marcus Pappik, Friedrich Schöne, Fabian Sommer, and David Stangl. Convergence and hardness of strategic Schelling segregation. In *WINE'19*, pages 156–170, 2019.
- 22 Edith Elkind, Jiarui Gan, Atushi Igarashi, Warut Suksompong, and Alexandros A. Voudouris. Schelling games on graphs. In *IJCAI'19*, pages 266–272, 2019.
- 23 Paul Erdős and Alfréd Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.
- 24 William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, 3rd edition, 1968.
- 25 Mark A Fossett. Simseg—a computer program to simulate the dynamics of residential segregation by social and ethnic status. *Race and Ethnic Studies Institute Technical Report and Program, Texas A&M University*, 1998.
- 26 Linton C. Freeman. Segregation in social networks. *Sociological Methods & Research*, 6(4):411–429, 1978.
- 27 Laetitia Gauvin, Jean-Pierre Nadal, and Jean Vannimenus. Schelling segregation in an open city: A kinetically constrained blume-emery-griffiths spin-1 system. *Physical Review E*, 81:066120, 2010.
- 28 Stefan Gerhold, Lev Glebsky, Carsten Schneider, Howard Weiss, and Burkhard Zimmermann. Computing the complexity for Schelling segregation models. *Communications in Nonlinear Science and Numerical Simulation*, 13(10):2236–2245, 2008.

- 29 Edgar N. Gilbert. Random graphs. *Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.
- 30 Mark S. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83:1420–1443, 1978.
- 31 Adam Douglas Henry, Paweł Prałat, and Cun-Quan Zhang. Emergence of segregation in evolving social networks. *Proceedings of the National Academy of Sciences*, 108(21):8605–8610, 2011.
- 32 Nicole Immorlica, Robert Kleinberg, Brendan Lucier, and Morteza Zadomighaddam. Exponential segregation in a two-dimensional Schelling model with tolerant individuals. In *SODA'17*, pages 984–993, 2017.
- 33 Panagiotis Kanellopoulos, Maria Kyropoulou, and Alexandros A. Voudouris. Modified Schelling games. In *SAGT'20*, pages 241–256, 2020. doi:10.1007/978-3-030-57980-7\_16.
- 34 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146, 2003.
- 35 Thomas M. Liggett. Coexistence in threshold voter models. *The Annals of Probability*, 22(2):764–802, 1994.
- 36 Thomas M. Liggett. *Voter Models*, pages 139–208. Springer, 1999.
- 37 Francisco W.S. Lima, Alexandre O. Sousa, and Muneer A. Sumuor. Majority-vote on directed Erdős–Rényi random graphs. *Physica A: Statistical Mechanics and its Applications*, 387(14):3503–3510, 2008.
- 38 Michael W. Macy. Chains of cooperation: Threshold effects in collective action. *American Sociological Review*, 56(6):730–747, 1991.
- 39 Hamed Omidvar and Massimo Franceschetti. Self-organized segregation on the grid. *Journal of Statistical Physics*, 170(4):1572–9613, 2018.
- 40 Hamed Omidvar and Massimo Franceschetti. Shape of diffusion and size of monochromatic region of a two-dimensional spin system. In *STOC'18*, pages 100–113, 2018.
- 41 Romans Pancs and Nicolaas J. Vriend. Schelling’s spatial proximity model of segregation revisited. *Journal of Public Economics*, 91(1):1–24, 2007.
- 42 Mathew Penrose. *Random Geometric Graphs*. Oxford University Press, 1st edition, 2003.
- 43 Luiz F. C. Pereira and F. G. Brady Moreira. Majority-vote model on random graphs. *Physical Review E*, 71:016123, 2005.
- 44 Alexis Poindron. A general model of binary opinions updating. *Mathematical Social Sciences*, 109:52–76, 2021.
- 45 Tim Rogers and Alan J McKane. A unified framework for Schelling’s model of segregation. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(07):P07006, 2011.
- 46 Thomas C. Schelling. Models of segregation. *American Economic Review*, 59(2):488–93, 1969.
- 47 Thomas C Schelling. *Micromotives and Macrobehavior*. WW Norton & Company, 2006.
- 48 Sorin Solomon and Dietrich Stauffer. Ising, Schelling and self-organising segregation. *The European Physical Journal B*, 57(4):473–479, 2007.
- 49 Michèle J. White. Segregation and diversity measures in population distribution. *Population index*, 52 2:198–221, 1986.
- 50 Zhi-Xi Wu and Petter Holme. Majority-vote model on hyperbolic lattices. *Physical Review E*, 81:011133, 2010.
- 51 H. Peyton Young. *Individual strategy and social structure: An evolutionary theory of institutions*. Princeton University Press Princeton, N.J, 1998.





# Piecewise-Linear Farthest-Site Voronoi Diagrams

Franz Aurenhammer ✉ 

Institute for Theoretical Computer Science, TU Graz, Austria

Evanthia Papadopoulou ✉ 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

Martin Suderland ✉ 

Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland

---

## Abstract

Voronoi diagrams induced by distance functions whose unit balls are convex polyhedra are piecewise-linear structures. Nevertheless, analyzing their combinatorial and algorithmic properties in dimensions three and higher is an intriguing problem. The situation turns easier when the farthest-site variants of such Voronoi diagrams are considered, where each site gets assigned the region of all points in space farthest from (rather than closest to) it.

We give asymptotically tight upper and lower worst-case bounds on the combinatorial size of farthest-site Voronoi diagrams for convex polyhedral distance functions in general dimensions, and propose an optimal construction algorithm. Our approach is uniform in the sense that (1) it can be extended from point sites to sites that are convex polyhedra, (2) it covers the case where the distance function is additively and/or multiplicatively weighted, and (3) it allows an anisotropic scenario where each site gets allotted its particular convex distance polytope.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** Voronoi diagram, farthest-site, polyhedral distance, polyhedral sites, general dimensions

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.30

**Funding** The first author was supported by Projects I 1836-N15 and I 5270-N, Austria Science Fund (FWF). The last two authors were supported in part by Projects 200021E\_154387 and 200021E\_201356, Swiss National Science Foundation (SNF).

## 1 Introduction

The *Voronoi diagram* of a set of  $n$  geometric objects, called *sites*, is a well-known space partitioning structure with numerous applications in diverse fields of science. In its *closest-site* variant, this diagram partitions the underlying space into maximal regions such that all points within one region have the same closest site. The Euclidean Voronoi diagram of point sites in  $\mathbb{R}^d$  is well studied; see e.g. [4, 7, 11, 15]. It is a piecewise-linear cell complex of worst-case complexity  $\Theta(n^{\lceil \frac{d}{2} \rceil})$  and can be constructed in  $O(n^{\lceil \frac{d}{2} \rceil} + n \log n)$  time. There are many ways to modify this standard diagram, for example, by using different distance measures, by considering sites of more general shape, or by assigning individual weights to them. Adapting to practical needs, such generalizations (among several others) have been studied extensively, and many satisfactory results are available nowadays [4, 18].

For most of these generalized Voronoi diagrams, the partition of space they define is not piecewise linear any more, but is rather composed of curved geometric objects of various dimensions and shapes. This complicates their structural analysis as well as their computational construction, especially in dimensions higher than two (where results are becoming comparatively sparse). For instance, already in three-dimensional space  $\mathbb{R}^3$ , the algebraic description of the edges and facets of the Euclidean Voronoi diagram of straight lines becomes exceedingly complicated [12]. What is more, the combinatorial complexity of



© Franz Aurenhammer, Evanthia Papadopoulou, and Martin Suderland;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 30; pp. 30:1–30:11

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

this diagram is a major open problem in computational geometry [17]. There is a gap of an order of magnitude between the  $\Omega(n^2)$  lower bound [1] and the only known upper bound of  $O(n^{3+\epsilon})$ , for any  $\epsilon > 0$  [21].

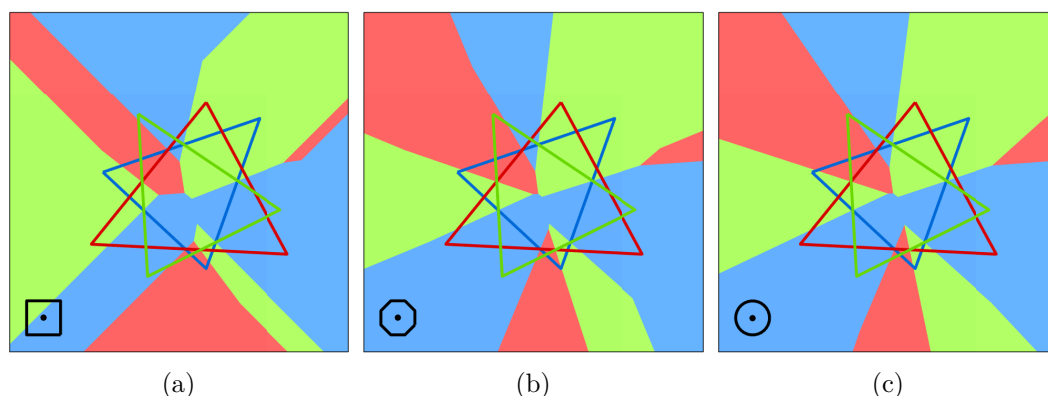
Certain types of Voronoi diagrams retain their piecewise linear structure, however. For example, the so-called power diagram [2] has this property. Another prominent class, and the one of interest in the present note, is induced by (convex) *polyhedral distance functions*. Intuitively speaking, the distance from a point  $x$  to a site  $s$  is now measured as the extent at which a given convex polyhedron  $\mathcal{P}$ , which being centered at  $x$ , has to expand till it starts touching  $s$ .

Several authors succeeded in deriving strong bounds on the combinatorial complexity of such Voronoi diagrams. If the  $n$  sites are points and the distance polytope  $\mathcal{P}$  is a simplex or a cube – the latter just giving the  $L_\infty$  distance – then this complexity in  $\mathbb{R}^d$  is  $\Theta(n^{\lceil \frac{d}{2} \rceil})$ , see [6]. (The dimension  $d$  is considered a constant throughout this paper.) In  $\mathbb{R}^3$ , the same bound still applies when any constant-sized convex polytope is chosen for  $\mathcal{P}$  [13]. For the sites being  $n$  straight lines in  $\mathbb{R}^3$ , with  $\mathcal{P}$  defined as before, near-quadratic bounds of  $\Omega(n^2\alpha(n))$  and  $O(n^2\alpha(n)\log(n))$  can be obtained [9]. Here  $\alpha(n)$  is the extremely slowly growing inverse Ackermann function. If we consider as sites disjoint convex polyhedra with  $n$  faces in total, then the complexity is  $O(n^{2+\epsilon})$ , as has been shown in [16]; this sharpens to  $O(n^2\alpha(n)\log(n))$  if all the sites are line segments.

Though Voronoi diagrams for convex polyhedral distance functions – in comparison to Euclidean Voronoi diagrams – thus proved easier to deal with concerning their combinatorial aspects, this does not seem to carry over to their algorithmic aspects. In fact, the papers cited above do not provide algorithms for computing such diagrams, and we are not aware of any construction algorithm particular to them.

As we shall show in this note, the situation changes if the so-called *farthest-site* variant of the diagram is considered (rather than the closest-site variant as above). The farthest-site Voronoi diagram is a partition of the underlying space into regions, such that the points within one region have the same farthest site (with respect to a given convex polyhedral distance function, in our case). We will show that the complexity of this diagram in  $\mathbb{R}^d$  is  $\Theta(n^{\lceil \frac{d}{2} \rceil})$  in the worst-case, and that it can be computed in optimal time  $O(n^{\lceil \frac{d}{2} \rceil} + n \log n)$ , mainly by using higher-dimensional convex hull algorithms. This result holds under rather general conditions: Sites can be arbitrary convex polyhedra (which may be unbounded or overlapping, having a total of  $n$  faces of various dimensions), the distance polytope  $\mathcal{P}$  may be unbounded (though constant-sized), and the resulting distance function can be additively and/or multiplicatively weighted for each site. Moreover, each site may get allotted a particular distance polytope, in order to generate an anisotropic scenario where sites can influence their surrounding in an individual way.

Farthest-site Voronoi diagrams are useful for performing farthest neighbor queries among the sites, for computing the smallest ball that contacts all sites, and for finding the largest gap to be bridged between any two sites – to name a few of their applications. Unfortunately, Euclidean farthest-site Voronoi diagrams have their peculiarities (unless all sites are points, in which case their combinatorial and computational behavior is much like their closest-point counterparts [20]). Their regions may disconnect into a large number of nonconvex parts, and the close relationship between nonempty regions and the convex hull of the sites is lost; see [3] for line segment sites in  $\mathbb{R}^2$ , and [19] for a generalization to arbitrary  $L_p$ -metrics. The only result for non-point sites in higher dimensions we are aware of is [5], who derive structural and



■ **Figure 1** Two approximations (a) and (b) of a Euclidean farthest-site Voronoi diagram (c). The sites are three overlapping triangles. Their boundaries are visualized in individual colors, and their farthest regions are painted accordingly. The distance polygons used – a square in (a) and a regular 8-gon in (b) – are shown in the bottom-left corner.

combinatorial properties for the farthest-site diagram of lines and line segments in  $\mathbb{R}^d$ . They characterize its unbounded cells, of which there are up to  $\Theta(n^{d-1})$  many in the worst-case, and describe an algorithm to compute these in near-optimal time.

With our results in the present note, a large class of Euclidean farthest-site Voronoi diagrams for convex sites in  $\mathbb{R}^d$ , even in their weighted and/or anisotropic variants, can be approximated in a piecewise-linear manner, and are computable by a simple and uniform approach: In  $\mathbb{R}^3$  for example, being probably the most interesting case, the Euclidean ball can be  $\beta$ -approximated by a convex polytope  $\mathcal{P}$  with  $O(1/\beta)$  vertices [16], such that the convex distance induced by  $\mathcal{P}$  is at most  $1 + \beta$  times the Euclidean distance. As a particularly useful result, a simple method for computing a piecewise-linear approximation of size  $O(n^2)$  of the Euclidean farthest-site Voronoi diagram for lines and/or line segments in  $\mathbb{R}^3$  becomes available. Even the planar instance is interesting: The fastest known algorithm for the Euclidean farthest-site Voronoi diagram for polygonal sites in  $\mathbb{R}^2$  runs in time  $O(n \log^3 n)$  [8], whereas our approximation can be computed in time  $O(n \log n)$  if the polygonal sites are convex. Figure 1 illustrates the similarity between these diagrams.

## 2 Convex polyhedral distance

We define a *polyhedron* in  $\mathbb{R}^d$  as the nonempty and finite (but possibly unbounded) intersection of closed halfspaces of  $\mathbb{R}^d$ . Note that a polyhedron does not need to be full-dimensional: For example lines, line segments, and single points are included as lower-dimensional instances.

Any  $d$ -dimensional polyhedron  $\mathcal{P}$  which contains the origin in its interior can be used to define a so-called *convex polyhedral distance*, from a point  $x \in \mathbb{R}^d$  to a point  $q \in \mathbb{R}^d$ :

$$\delta_{\mathcal{P}}(x, q) = \inf_{t \geq 0} \{ t \mid q \in x + t \cdot \mathcal{P} \}.$$

In other words,  $\delta_{\mathcal{P}}(x, q)$  describes the amount  $t \geq 0$  by which  $\mathcal{P}$ , when being placed at  $x$ , has to be scaled so as to cover  $q$ ; see Figure 2. Note that  $\delta_{\mathcal{P}}$  is a directed distance. We shall call  $\mathcal{P}$  the *distance polytope* that induces the distance function  $\delta_{\mathcal{P}}$ .

Let  $\mathcal{P}^R = \{ -p \mid p \in \mathcal{P} \}$  denote the reflection of the distance polytope about the origin.

► **Observation 1.** We have  $\delta_{\mathcal{P}}(x, q) = \delta_{\mathcal{P}^R}(q, x)$ .

**Proof.** Suppose that  $q \in x + t \cdot \mathcal{P}$ . Then there is a point  $p \in \mathcal{P}$  with  $q = x + t \cdot p$ , that is,  $x = q - t \cdot p$ . Thus we have  $x \in q - t \cdot \mathcal{P}$ , which by the identity  $-t \cdot \mathcal{P} = t \cdot \mathcal{P}^R$  means  $x \in q + t \cdot \mathcal{P}^R$ . ◀

Consider a set  $S$  of point sites in  $\mathbb{R}^d$ , and identify  $\mathbb{R}^d$  with the hyperplane  $x_{d+1} = 0$  in  $(d+1)$ -dimensional space  $\mathbb{R}^{d+1}$ . Observation 1 suggests to associate the distance polytope  $\mathcal{P}$  with a *distance cone*  $C_{\mathcal{P}}$  in  $\mathbb{R}^{d+1}$ , such that  $C_{\mathcal{P}}$  reflects with its height the polyhedral distance induced by  $\mathcal{P}$ .

$$C_{\mathcal{P}} = \bigcup_{t \geq 0} \begin{pmatrix} t \cdot \mathcal{P}^R \\ t \end{pmatrix} \quad (1)$$

$C_{\mathcal{P}}$  is a polyhedral cone obtained from scaling the reflected polytope  $\mathcal{P}^R$ . Its apex is at the origin. Let  $C_{\mathcal{P}}(q_i)$  be the translate of  $C_{\mathcal{P}}$  with its apex at some point site  $q_i \in S$ . Then for any point  $x \in \mathbb{R}^d$ , the  $(d+1)^{\text{st}}$  coordinate (called *height*) of the vertical projection of  $x$  to  $C_{\mathcal{P}}(q_i)$  equals the distance  $\delta_{\mathcal{P}}(x, q_i)$ .

Let now, more generally, the set  $S$  consist of polyhedral sites  $s_i$  in  $\mathbb{R}^d$ . We construct for each site  $s_i \in S$  a distance cone as follows. Take the Minkowski sum  $s_i \oplus C_{\mathcal{P}}$ . (The *Minkowski sum* of point sets  $A$  and  $B$  is defined as  $A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$ .) Because the Minkowski sum of two convex polyhedra is again a convex polyhedron, the object  $s_i \oplus C_{\mathcal{P}}$  is the intersection of halfspaces of  $\mathbb{R}^{d+1}$ . One of them is bounded from below by the hyperplane  $x_{d+1} = 0$  (if the site  $s_i$  is full-dimensional). We ignore this halfspace, and intersecting the remaining ones we obtain an unbounded polyhedron in  $\mathbb{R}^{d+1}$ , which we call the *distance cone* of  $s_i$ , and denote with  $C_{\mathcal{P}}(s_i)$ .

$C_{\mathcal{P}}(s_i)$  exhibits the following useful properties.

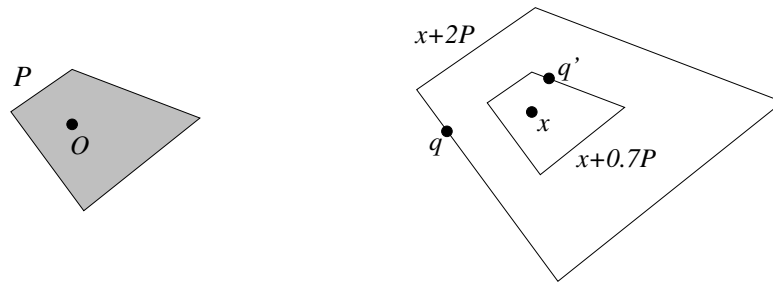
- For the special case of  $s_i$  being a point site  $q_i$ , the definition of  $C_{\mathcal{P}}(s_i)$  is consistent with that of  $C_{\mathcal{P}}(q_i)$  before.
- Let  $d_{\mathcal{P}}(x, s_i)$  be the height of the vertical projection of a point  $x \in \mathbb{R}^d$  to  $C_{\mathcal{P}}(s_i)$ . If  $x$  does not lie in the interior of  $s_i$ , then  $d_{\mathcal{P}}(x, s_i)$  is non-negative and equals the polyhedral distance of  $x$  to  $s_i$ , which is commonly defined as

$$\delta_{\mathcal{P}}(x, s_i) = \inf_{t \geq 0} \{ t \mid x + (t \cdot \mathcal{P}) \cap s_i \neq \emptyset \}.$$

- If  $x$  lies in the interior of  $s_i$  then  $d_{\mathcal{P}}(x, s_i)$  is negative, and measures how much  $x$  is inside the (full-dimensional) polyhedral site  $s_i$  by taking the minimum polyhedral distance to its facets; see Figure 1. This is because the part of  $C_{\mathcal{P}}(s_i)$  that lies below the hyperplane  $x_{d+1} = 0$  is determined solely by halfspaces which stem from  $s_i$  (and not from  $C_{\mathcal{P}}$  in Formula (1)). That is,  $d_{\mathcal{P}}$  is related to a generalized medial axis of  $s_i$  in this case.

### 3 Farthest-site Voronoi diagram

The so-called *farthest-site Voronoi diagram* of a set  $S$  of sites in  $\mathbb{R}^d$ , for short FVD( $S$ ), is a partition of  $\mathbb{R}^d$  into regions such that all points within a fixed region have the same farthest site. As before, let the sites  $s_i$  in  $S$  be polyhedra. These may be of any dimension  $k$ , for  $0 \leq k \leq d$ , and are not required to be disjoint or bounded.



■ **Figure 2** Polyhedral distance induced by  $\mathcal{P}$ :  $d_{\mathcal{P}}(x, q) = 2$  and  $d_{\mathcal{P}}(x, q') = 0.7$ .

We are interested in the diagram  $\text{FVD}(S)$  induced by the convex polyhedral distance function  $d_{\mathcal{P}}$  in Section 2, for a given distance polytope  $\mathcal{P}$ . Being a farthest-site diagram,  $\text{FVD}(S)$  corresponds to the pointwise maximum of the functions  $d_{\mathcal{P}}(x, s_i)$ , for  $s_i \in S$ , on  $\mathbb{R}^d$ .  $\text{FVD}(S)$  thus corresponds to the *upper envelope* of the boundaries of the distance cones  $C_{\mathcal{P}}(s_i)$  that define these functions, which, in turn, is given by the common intersection of these cones. Let us formulate this result in the following way.

► **Theorem 2.** *Let  $I$  be the (unbounded) convex polyhedron in  $\mathbb{R}^{d+1}$  that results from intersecting the distance cones  $C_{\mathcal{P}}(s_i)$ , for all sites  $s_i \in S$ . Then  $\text{FVD}(S)$  is the vertical projection of  $I$  onto the hyperplane  $x_{d+1} = 0$  of  $\mathbb{R}^{d+1}$ .*

One of the consequences of Theorem 2 is that  $\text{FVD}(S)$  is a piecewise linear diagram. Each region of  $\text{FVD}(S)$  is pre-partitioned into convex polyhedra (the projected facets of  $I$ ), and these regions define a partition of  $\mathbb{R}^d$ . Let us point out that, in earlier papers on Voronoi diagrams for polyhedral distance functions (e.g. in [16]), the distance of a point  $x$  to a site was set to zero in case  $x$  falls in the interior of that site. As a consequence, when the sites are not chosen to be pairwise disjoint, the part of  $\mathbb{R}^d$  covered by their union does not get partitioned by the diagram. Our more general definition of polyhedral distance, via distance cones, remedies this shortcoming.

The combinatorial complexity of  $\text{FVD}(S)$  is given by that of the projection polyhedron  $I$  in  $\mathbb{R}^{d+1}$ .  $I$  is the intersection of distance cones, and each distance cone  $C_{\mathcal{P}}(s_i)$ , in turn, is the intersection of halfspaces of  $\mathbb{R}^{d+1}$ . It is clear from Section 2 that the number of such halfspaces per cone is bounded by the number of facets of the Minkowski sum  $s_i \oplus \mathcal{P}^R$ , for the reflected distance polytope  $\mathcal{P}^R$ . A single face of  $s_i$ , combined with a single face of  $\mathcal{P}^R$ , can yield at most one facet of  $s_i \oplus \mathcal{P}^R$ ; see e.g. [10]. Therefore, if we assume that  $\mathcal{P}$  (and with it  $\mathcal{P}^R$ ) is of constant size, and that  $s_i$  has a total of  $n_i$  faces of different dimensions, then  $C_{\mathcal{P}}(s_i)$  is defined by  $O(n_i)$  halfspaces of  $\mathbb{R}^{d+1}$ . In conclusion, when putting  $n = \sum_{s_i \in S} n_i$ , the polyhedron  $I$  is the intersection of  $O(n)$  halfspaces of  $\mathbb{R}^{d+1}$ , and its complexity is bounded from above by  $O(n^{\lceil \frac{d}{2} \rceil})$  (provided  $d = O(1)$ ), by the well-known upper bound theorem. We will show in Section 4 that this complexity can be asymptotically attained in the worst case. Observe that  $I$  has  $O(n)$  facets, and that  $\text{FVD}(S)$  thus has this very number of full-dimensional cells.

Concerning computational aspects, the halfspaces defining a particular cone  $C_{\mathcal{P}}(s_i)$  can be singled out by (basically) computing the Minkowski sum  $s_i \oplus \mathcal{P}^R$ . This can be done [10], for instance, by pairwise adding up the  $O(n_i)$  vertices of  $s_i$  and the  $O(1)$  vertices of  $\mathcal{P}^R$ , and computing the convex hull of the resulting  $O(n_i)$  points in  $\mathbb{R}^d$ , spending a total of  $O(n^{\lfloor \frac{d}{2} \rfloor} + n \log n)$  time for all sites  $s_i \in S$ , when the optimal convex hull algorithm in [7]

## 30:6 Piecewise-Linear Farthest-Site Voronoi Diagrams

is applied. The construction of the projection polyhedron  $I$  is more time-consuming and takes  $O(n^{\lceil \frac{d}{2} \rceil} + n \log n)$  time; we use the convex hull algorithm in [7] again, but now for intersecting  $O(n)$  halfspaces in  $\mathbb{R}^{d+1}$ .

We may summarize as follows:

► **Theorem 3.** *Let  $S$  be a set of arbitrary polyhedral sites in  $\mathbb{R}^d$ , with a total combinatorial complexity of  $n$ . The farthest-site Voronoi diagram  $\text{FVD}(S)$  of  $S$  under the convex distance function induced by a polytope of constant size is of complexity  $O(n^{\lceil \frac{d}{2} \rceil})$ , and it can be computed in  $O(n^{\lceil \frac{d}{2} \rceil} + n \log n)$  time. The number of  $d$ -dimensional cells of  $\text{FVD}(S)$  is bounded by  $O(n)$ .*

The dependence on  $d$  of the bounds stated above is the same as for convex hulls of finite point sets.

### 4 More properties of FVD

The maximal size of farthest-site diagrams may be much smaller than that of their closest-site counterparts; several examples can be found in [4]. The question arises whether the upper bound given in Theorem 3 is asymptotically tight.

For special sets of polyhedral sites in  $\mathbb{R}^d$ , the diagram  $\text{FVD}(S)$  is indeed small, namely, when the sites in  $S$  have only a constant number of orientations. Then the halfspaces defining the projection polyhedron  $I$  in  $\mathbb{R}^{d+1}$  will have only a constant number of orientations as well, and all but  $O(1)$  of them will be redundant because their bounding hyperplanes are parallel. Consequently, the polyhedron  $I$  and its projection  $\text{FVD}(S)$  will be of constant size, and can be found in  $O(n)$  time.

Observe that the case of  $S$  being a set of  $n$  point sites in  $\mathbb{R}^d$  is covered above, because each point site can be described by the intersection of  $d+1$  halfspaces of  $\mathbb{R}^d$ , having the same fixed orientations. Not included, however, is the case of  $n$  line segment sites in  $\mathbb{R}^d$ , because the  $d+2$  halfspaces describing a line segment will be of different orientation for different sites, in general. In fact, sites of very simple shape can induce large diagrams, as is shown below.

► **Lemma 4.** *There exists a set  $S$  of  $n$  sites in  $\mathbb{R}^d$  of constant description such that the diagram  $\text{FVD}(S)$  has a complexity of  $\Omega(n^{\lceil \frac{d}{2} \rceil})$ .*

**Proof.** There exist two hyperplanes  $h_1, h_2$  in  $\mathbb{R}^{d+1}$ , and two point sets  $Y_1 \subset h_1$  and  $Y_2 \subset h_2$  each of size  $\frac{n}{2}$ , such that the lower convex hull of  $Y_1 \cup Y_2$  has  $\Theta(n^{\lceil \frac{d+1}{2} \rceil}) = \Theta(n^{\lceil \frac{d}{2} \rceil})$  complexity; this follows from Corollary 12 in [14]. W.l.o.g., we may assume (by applying an affine transformation) that

$$\begin{aligned}
 h_1 : \sum_{i=1}^d x_i = 1, \quad h_2 : \sum_{i=1}^{d-1} x_i - x_d = 1, \quad \text{and} \\
 Y_1 \subset \mathbb{R} \times \left(0, \frac{1}{d}\right]^d, \quad Y_2 \subset \mathbb{R} \times \left(0, \frac{1}{d}\right]^{d-1} \times \left[-\frac{1}{d}, 0\right),
 \end{aligned} \tag{2}$$

which fixes the hyperplanes and guarantees that most coordinates in  $Y_1 \cup Y_2$  are small. We now choose a distance polytope  $\mathcal{P}$  and a set  $S = S_1 \cup S_2$  of sites such that the projection polyhedron  $I$  for  $\text{FVD}(S)$  is dual to the lower convex hull of  $Y_1 \cup Y_2$ .

Let the hypercube  $[-1, 1]^d$  serve as  $\mathcal{P}$ . The set  $S_1 \cup S_2$  will consist of  $n$  halfspace sites in  $\mathbb{R}^d$ . For  $S_1$ , each of its halfspaces  $s$  is constructed from a point  $y = (y_1, \dots, y_{d+1}) \in Y_1$ . In particular, we describe  $s$  by the inequality  $\sum_{i=1}^d a_i x_i \leq b$ , where

$$a_1 = 1, \quad a_i = \frac{y_i}{1 - \sum_{j=2}^d y_j} \quad \text{for } i = 2, \dots, d, \quad \text{and } b = \frac{y_{d+1}}{1 - \sum_{j=2}^d y_j}. \quad (3)$$

Note that all  $a_i$  and  $b$  are positive because of our assumption (2). Moreover, we have  $\mathcal{P}^R = \mathcal{P}$ . Therefore, the Minkowski sum  $s \oplus \mathcal{P}^R$  is just a translate of  $s$  by the vector  $(1, 1, \dots, 1)^T$  in  $\mathbb{R}^d$ , which implies that the distance cone  $C_{\mathcal{P}}(s)$  is a single halfspace in  $\mathbb{R}^{d+1}$ , bounded from below by the hyperplane

$$x_{d+1} = \frac{1}{A} \cdot \left( \sum_{i=1}^d a_i x_i - b \right), \quad \text{where } A = \sum_{j=1}^d a_j. \quad (4)$$

By a well-known duality transform, the hyperplane in (4) is dual to the point

$$q = (q_1, \dots, q_{d+1}) = \frac{1}{A} \cdot (a_1, \dots, a_d, b)$$

in  $\mathbb{R}^{d+1}$ . Substituting the values in (3) and simple calculations give

$$q_i = y_i \quad \text{for } i = 2, \dots, d+1, \quad \text{and } q_1 = 1 - \sum_{j=2}^d y_j. \quad (5)$$

But this implies  $q = y$ , because both points lie in the hyperplane  $h_1$  in (2): We have  $y \in h_1$  by assumption, and  $q \in h_1$  by (5).

In a similar manner, we can construct suitable halfspace sites  $s$  for  $S_2$  from the points  $y$  in  $Y_2$ . (We omit these details here.) In conclusion, the projection polyhedron  $I$ , being the intersection of all the sites' distance cones, is the intersection of the upper halfspaces  $C_{\mathcal{P}}(s)$  for all  $s \in S_1 \cup S_2$ , and thus  $I$  is dual to the lower convex hull of  $Y_1 \cup Y_2$ . ◀

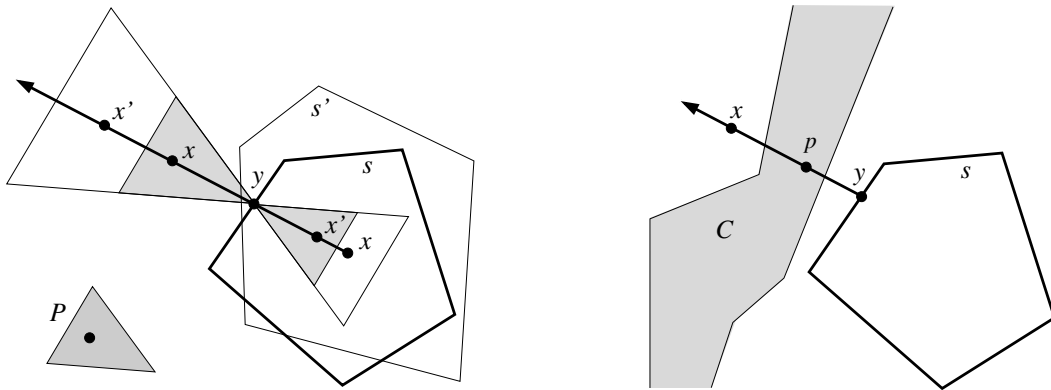
By Lemma 4, the runtime in Theorem 3 is asymptotically optimal in the worst case for  $d \geq 3$ . For  $d = 2$ , a reduction from sorting proves optimality.

Being the projection of  $I$ , the diagram  $\text{FVD}(S)$  is a polyhedral cell complex in  $\mathbb{R}^d$  which is face-to-face. Its *cells* (polyhedra of dimension  $d$ ) are nonconvex in general, as are its *facets* (polyhedra of dimension  $d-1$ ). Since the distance polytope  $\mathcal{P}$  is (more or less) an approximation of the Euclidean ball, quite a few properties of the Euclidean farthest-site diagram of  $S$  carry over to  $\text{FVD}(S)$ ; see e.g. [3, 5]. For example, the *region* of a site  $s_i \in S$  in  $\text{FVD}(S)$  (the set of all points in  $\mathbb{R}^d$  being farthest from  $s_i$ ) is disconnected in general, and it may consist of various cells of  $\text{FVD}(S)$ . Moreover, the following properties of the cells are preserved.

► **Lemma 5.** *All cells of  $\text{FVD}(S)$  are unbounded, and cells cannot contain voids of any dimension.*

**Proof.** Let  $C$  be some cell of  $\text{FVD}(S)$ , and assume that  $C$  is part of the region of the site  $s \in S$ . The assertion of the lemma can be easily derived from the following fact: Let  $x$  be an arbitrary point in  $C$ , and consider the point  $y$  on the boundary of  $s$  that realizes the polyhedral distance  $d_{\mathcal{P}}(x, s)$ . Then the infinite ray  $r$  that starts at  $x$  and is directed away from  $y$  is totally contained in  $C$ .





■ **Figure 3** Illustrations of the proofs of Lemma 5 (left) and Lemma 6 (right).

To prove this fact, refer to Figure 3 (left). Assume first that  $x \notin s$ . Then  $t = d_{\mathcal{P}}(x, s) \geq 0$ , and the homothet  $H = x + t \cdot \mathcal{P}$  touches  $s$  at  $y$ . Since  $s$  is the site in  $S$  farthest from  $x$ ,  $H$  intersects all the other sites. Let now  $x'$  be any point on  $r$  such that  $x$  lies between  $x'$  and  $y$ . Put  $t' = d_{\mathcal{P}}(x', s)$ . Then  $H' = x' + t' \cdot \mathcal{P}$  touches  $s$  at  $y$  too, and  $H$  is covered by  $H'$ , which implies that  $H'$  intersects all other sites as well. This implies that  $x'$  lies in the region of  $s$ .

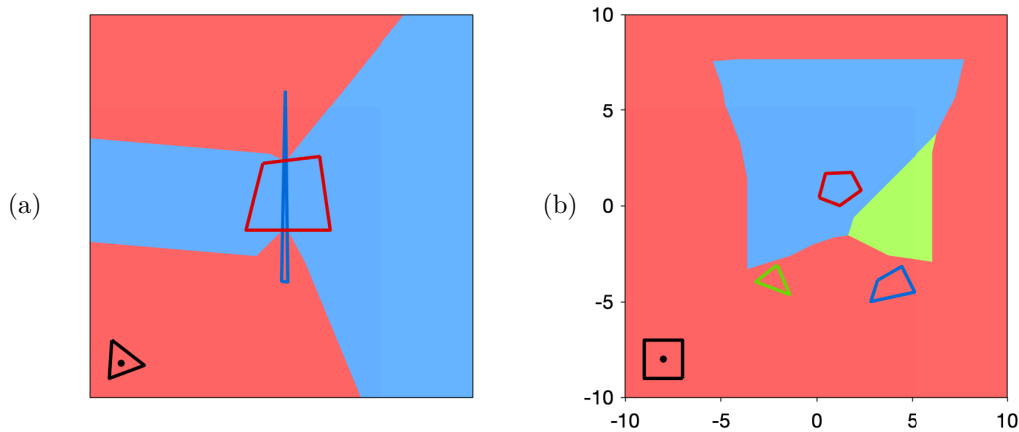
If  $x \in s$ , on the other hand, then  $t = d_{\mathcal{P}}(x, s) < 0$ , and we have  $t \cdot \mathcal{P} = u \cdot \mathcal{P}^R$  with  $u = -t > 0$ , for the reflected polytope  $\mathcal{P}^R$ . The homothet  $H = x + u \cdot \mathcal{P}^R$  touches  $s$  at  $y$ , and since  $s$  is farthest from  $x$ ,  $H$  is contained in all other sites now. For any point  $x'$  on  $r$  between  $x$  and  $y$ , and  $u' = -d_{\mathcal{P}}(x', s)$ ,  $H' = x' + u' \cdot \mathcal{P}^R$  touches  $s$  at  $y$  again, but is contained in  $H$  now and therefore also in all other sites. So  $x'$  has to lie in the region of  $s$ .

In summary, we conclude that the entire ray  $r$  lies in the cell  $C$  of the region of  $s$ . ◀

Let us define the  $(d-1)$ -skeleton of  $\text{FVD}(S)$  as the union of all the facets of  $\text{FVD}(S)$ . This skeleton can be disconnected, as a simple construction with only two sites shows; see Figure 4(a): Let site  $s_1$  be some polyhedron which approximates a line segment, and take as site  $s_2$  any polyhedron which contains the segment's midpoint but none of its endpoints. Then the region of  $s_1$  disconnects the  $(d-1)$ -skeleton of  $\text{FVD}(\{s_1, s_2\})$ . On the other hand, by the same argument as in [5], the following holds:

► **Lemma 6.** *The  $(d-1)$ -skeleton of  $\text{FVD}(S)$  is connected, provided that the sites in  $S$  are pairwise disjoint.*

**Proof.** Assume that this skeleton is not connected; see Figure 3 (right). Then there exists some cell  $C$  of  $\text{FVD}(S)$  that splits the skeleton into at least two parts. Let  $s$  be the farthest site corresponding to  $C$ . The site  $s$  does not touch the boundary of  $C$ , because of our assumption on the disjointness of the sites. Thus there exists some point  $x \notin C$  which is separated from  $s$  by  $C$ . Let  $y$  be the point on  $s$  that realizes the polyhedral distance from  $x$  to  $s$ . By construction, the line segment  $\overline{xy}$  intersects  $C$ , and we choose a point  $p$  in this intersection. Now, by the reasoning in the proof of Lemma 5, the infinite ray emanating from  $p$  in direction  $x$  is entirely contained in  $C$ . But this implies  $x \in C$ , which is a contradiction. ◀



■ **Figure 4** (a) The  $(d-1)$ -skeleton can be disconnected for non-disjoint sites. (b) A weighted farthest Voronoi diagram of three sites: The blue quadrangle has an additive weight of  $-1$ , and the red pentagon has a multiplicative weight of  $\frac{1}{2}$ .

## 5 Variants

In certain applications, a model of Voronoi diagram is required where the sites are capable of influencing their surrounding in an individual way; see [4, 18] for comprehensive treatments of this topic. One way to achieve this goal is to assign so-called *weights* to the sites, which affect the underlying distance function in an additive and/or multiplicative way.

Let each site  $s_i \in S$  have assigned two real numbers  $a(s_i)$  and  $m(s_i) > 0$ , and consider the weighted polyhedral distance:

$$\frac{d_{\mathcal{P}}(x, s_i)}{m(s_i)} - a(s_i)$$

In contrast to the nearest version, the sites' regions in the farthest Voronoi diagram shrink with increasing weights. Interestingly, and unlike the situation for the Euclidean farthest-site diagram, the FVD( $S$ ) induced by this distance is still a piecewise-linear cell complex. This becomes evident when the respective distance cones are considered: Additive weighting results in a vertical shift of these cones by an amount of  $a(s_i)$ , and multiplicative weighting enlarges by a factor of  $m(s_i)$  the value  $\tan \alpha_j$  of the dihedral angles  $\alpha_j$  of aperture of a cone's facets. In particular, each distance cone still is the intersection of  $O(n_i)$  halfspaces of  $\mathbb{R}^{d+1}$  when site  $s_i$  is of complexity  $n_i$ .

Multiplicative weighting leads to the occurrence of bounded regions in FVD( $S$ ), as simple examples show (Figure 4 (b)). However, purely additive weighting preserves the properties listed in Lemma 5. In particular, all cells are still unbounded: All facets of the projection polyhedron  $I$  for the unweighted FVD( $S$ ) are unbounded, and this fact cannot be altered by vertically shifting any of its defining halfspaces.

We may push things even further, and create an *anisotropic* scenario by allotting an individual distance polytope  $\mathcal{P}_i$  to each site  $s_i$ . In this way, each site is able to “interpret” its surrounding space in its own way – a concept useful in many situations [4]. In fact, the multiplicative weighting scheme is just the special case where  $\mathcal{P}_i = m(s_i) \cdot \mathcal{P}$ .

In all the extensions above, the properties of the distance cones needed for Theorem 3 to hold are preserved. We obtain the following general result:

► **Theorem 7.** *Theorem 3 remains valid for all the weighted and anisotropic variants of  $FVD(S)$  described above.*

Note finally that all these extensions can be combined, and lead to a very general class of easy-to-compute piecewise-linear farthest-site Voronoi diagrams in  $\mathbb{R}^d$ , where the impact of each site can be tuned by its shape, its weights, and its distance polytope including the choice of the polytope's center.

---

## References

- 1 Boris Aronov. A lower bound on Voronoi diagram complexity. *Information Processing Letters*, 83(4):183–185, 2002.
- 2 Franz Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.
- 3 Franz Aurenhammer, Scot Drysdale, and Hannes Krasser. Farthest line segment Voronoi diagrams. *Information Processing Letters*, 100(6):220–225, 2006.
- 4 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi diagrams and Delaunay Triangulations*. World Scientific Publishing Company, 2013.
- 5 Gill Barequet, Evanthia Papadopoulou, and Martin Suderland. Unbounded regions of high-order Voronoi diagrams of lines and segments in higher dimensions. In *30th International Symposium on Algorithms and Computation*, volume 149 of *LIPICs*, pages 62:1–62:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- 6 Jean-Daniel Boissonnat, Micha Sharir, Boaz Tagansky, and Mariette Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *Discrete & Computational Geometry*, 19(4):485–519, 1998.
- 7 Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.
- 8 Otfried Cheong, Hazel Everett, Marc Glisse, Joachim Gudmundsson, Samuel Hornus, Sylvain Lazard, Mira Lee, and Hyeon-Suk Na. Farthest-polygon Voronoi diagrams. *Computational Geometry: Theory and Applications*, 44(4):234–247, 2011.
- 9 L. Paul Chew, Klara Kedem, Micha Sharir, Boaz Tagansky, and Emo Welzl. Voronoi diagrams of lines in 3-space under polyhedral convex distance functions. *Journal of Algorithms*, 29(2):238–255, 1998.
- 10 Sandip Das and Swami Sarvottamananda. Computing the Minkowski sum of convex polytopes in  $\mathbb{R}^d$ . *CoRR*, abs/1811.05812, 2018. [arXiv:1811.05812](https://arxiv.org/abs/1811.05812).
- 11 Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986.
- 12 Hazel Everett, Daniel Lazard, Sylvain Lazard, and Mohab Safey El Din. The Voronoi diagram of three lines. *Discrete & Computational Geometry*, 42(1):94–130, 2009.
- 13 Christian Icking and Lihong Ma. A tight bound for the complexity of Voronoi diagrams under polyhedral convex distance functions in 3D. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing*, pages 316–321, 2001.
- 14 Menelaos I. Karavelas, Raimund Seidel, and Eleni Tzanaki. Convex hulls of spheres and convex hulls of disjoint convex polytopes. *Computational Geometry: Theory and Applications*, 46(6):615–630, 2013.
- 15 Victor Klee. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik*, 34(1):75–80, 1980.
- 16 Vladlen Koltun and Micha Sharir. Polyhedral Voronoi diagrams of polyhedra in three dimensions. *Discrete & Computational Geometry*, 31(1):83–124, 2004.

- 17 Joseph S. B. Mitchell and Joseph O'Rourke. Computational Geometry Column 42. *Intern. Journal of Computational Geometry & Applications*, 11(5):573–582, 2001.
- 18 Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams (2nd Edition)*, volume 501. John Wiley & Sons, 2009.
- 19 Evanthia Papadopoulou and Sandeep Kumar Dey. On the farthest line-segment Voronoi diagram. *Intern. Journal of Computational Geometry & Applications*, 23(6):443–460, 2013.
- 20 Raimund Seidel. On the number of faces in higher-dimensional Voronoi diagrams. In *Proceedings of the Third Annual Symposium on Computational Geometry*, pages 181–185, 1987.
- 21 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry*, 12:327–345, 1994.



# Effective Resistance and Capacitance in Simplicial Complexes and a Quantum Algorithm

Mitchell Black ✉

Oregon State University, Corvallis, OR, USA

William Maxwell ✉

Oregon State University, Corvallis, OR, USA

---

## Abstract

We investigate generalizations of the graph theoretic notions of effective resistance and capacitance to simplicial complexes and prove analogs of formulas known in the case of graphs. In graphs the effective resistance between two vertices is  $O(n)$ ; however, we show that in a simplicial complex the effective resistance of a null-homologous cycle may be exponential. This is caused by relative torsion in the simplicial complex. We provide upper bounds on both effective resistance and capacitance that are polynomial in the number of simplices as well as the maximum cardinality of the torsion subgroup of a relative homology group denoted  $\mathcal{T}_{\max}(\mathcal{K})$ . We generalize the quantum algorithm deciding  $st$ -connectivity in a graph and obtain an algorithm deciding whether or not a  $(d-1)$ -dimensional cycle  $\gamma$  is null-homologous in a  $d$ -dimensional simplicial complex  $\mathcal{K}$ . The quantum algorithm has query complexity parameterized by the effective resistance and capacitance of  $\gamma$ . Using our upper bounds we find that the query complexity is  $O(n^{5/2} \cdot d^{1/2} \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ . Under the assumptions that  $\gamma$  is the boundary of a  $d$ -simplex (which may or may not be included in the complex) and that  $\mathcal{K}$  is relative torsion-free, we match the  $O(n^{3/2})$  query complexity obtained for  $st$ -connectivity. These assumptions always hold in the case of  $st$ -connectivity. We provide an implementation of the algorithm whose running time is polynomial in the size of the complex and the relative torsion. Finally, we prove a duality theorem relating effective resistance and capacitance when  $\mathcal{K}$  is  $d$ -dimensional and admits an embedding into  $\mathbb{R}^{d+1}$ .

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Simplicial complexes, quantum computing

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.31

**Funding** This research was supported by NSF grants CCF-1816442 and CCF-1617951.

**Acknowledgements** We would like to thank Amir Nayyeri for a helpful discussion about Theorem 17.

## 1 Introduction

The *effective resistance* between two vertices  $s$  and  $t$  in a graph is a quantity that measures how “well-connected”  $s$  and  $t$  are; specifically, the more, shorter paths connecting  $s$  and  $t$ , the lower the effective resistance between  $s$  and  $t$ . While effective resistance was originally defined in the context of resistor networks, it has since been discovered that effective resistance has many other interpretations. It is a metric on the vertices of a graph [17]; it is proportional to the expected number of steps in a random walk from  $s$  to  $t$  and back to  $s$  [3]; if  $\{s, t\}$  is an edge, it is proportional to the probability  $\{s, t\}$  is in a random spanning tree of  $G$  [15]. Effective resistance also has applications. Sampling edges in a graph with probability proportional to the effective resistance between their endpoints produces a graph that approximates the spectrum of the Laplacian of the original graph [27]. Effective resistance is also a parameter in the running time of a quantum algorithm to decide if  $s$  and  $t$  are connected [1].

Recently, effective resistance has independently been generalized from graphs to simplicial complexes by several groups of authors, although each group defines effective resistance for a different object or objects in a simplicial complex. Kook and Lee [18] and Osting,



© Mitchell Black and William Maxwell;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 31; pp. 31:1–31:27

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Palande, and Wang [23] both define effective resistance as a quantity associated with the boundary of a simplex.<sup>1</sup> This definition is analogous to the graph definition, as a pair of vertices is the boundary of an edge. Hansen and Ghrist [9] give a more general definition and define effective resistance as a quantity between two homologous cycles<sup>2</sup>. Our definition is equivalent to Hansen and Ghrist's, as we define effective resistance as a quantity associated with a null-homologous cycle.

Recent research has shown that some properties of effective resistance in graphs generalize to simplicial complexes. For example, Kook and Lee prove that the effective resistance of the boundary a simplex is the probability that the simplex is in the high-dimensional equivalent of a spanning tree [18]. Osting, Palande, and Wang show that sampling  $d$ -simplices according to the effective resistance of their boundaries approximately preserves the spectrum of the  $(d-1)$ -dimensional up-Laplacian [23].

## 1.1 Our Contributions

In this paper, we continue this trend of investigating effective resistance in simplicial complexes. Our main contribution is to show that there is a quantum algorithm for testing if a cycle is null-homologous in a simplicial complex that is parameterized by the effective resistance and effective capacitance of a cycle. The effective capacitance of a cycle is inspired by a quantity on graphs, and to our knowledge, we are the first to explore effective capacitance in simplicial complexes. Null-homology testing is an important primitive in computational topology. For example, the iterative algorithm for computing Betti numbers works by adding simplices one by one to the complex and testing if the boundary of each is already null-homologous in the previous complex [4].

Motivated by our quantum algorithm, we investigate bounds on the effective resistance and effective capacitance. We prove a negative result. While the effective resistance between a pair of vertices in a graph is bounded above by the number of vertices in the graph, the effective resistance of a  $(d-1)$ -cycle in a simplicial complex can be exponential in the number of the  $d$ -simplices in the complex. The classical algorithm to determine if a cycle is null-homologous is to solve a system of linear equations in the  $d$ th boundary matrix, so the quantum algorithm is slower than the classical algorithm in the worst case. On the positive side, we prove this exponential behavior is the result of relative torsion in dimension  $(d-1)$ , and we prove bounds on effective resistance and effective capacitance in terms of the size of relative torsion groups of the complex.

We also prove a duality result between effective resistance and effective capacitance. For a  $d$ -dimensional simplicial complex  $\mathcal{K}$  embedded in  $\mathbb{R}^{d+1}$ , the effective capacitance of certain  $d$ -cycles in  $\mathcal{K}$  is the effective resistance between a pair of nodes in the dual graph. Our proof relies on a high dimensional generalization of planar graphs with two nodes  $s$  and  $t$  appearing on the same face. Due to space constraints, this result can be found in Appendix C.

## 2 Preliminaries

Given a set of vertices  $V$ , a **simplicial complex**  $\mathcal{K}$  on  $V$  is a subset of the power set  $\mathcal{K} \subseteq P(V)$  with the following property: for each  $\tau \in \mathcal{K}$ , if  $\sigma \subset \tau$ , then  $\sigma \in \mathcal{K}$ . We assume there is a fixed but arbitrary order on the vertices  $V = (v_1, \dots, v_n)$ . A simplex  $\sigma \in \mathcal{K}$  of size

<sup>1</sup> Kook and Lee do not require the simplex to be in the complex.

<sup>2</sup> Hansen and Ghrist define effective resistance on a cellular sheaf, which is a generalization of a simplicial complex.



$|\sigma| = d + 1$  is a **d-simplex**. The set of all  $d$ -simplices of  $\mathcal{K}$  is denoted  $\mathcal{K}_d$ . The **d-skeleton** of  $\mathcal{K}$ , denoted  $\mathcal{K}^d$ , is the simplicial complex of all simplices of  $\mathcal{K}$  of size at most  $d + 1$ . The **dimension** of  $\mathcal{K}$  is the largest  $d$  such that  $\mathcal{K}$  contains a  $d$ -simplex; a 1-dimensional simplicial complex is a **graph**.

The **d<sup>th</sup> chain group**  $C_d(\mathcal{K})$  is the vector space over  $\mathbb{R}$  with orthonormal basis  $\mathcal{K}_d$ . Unless otherwise stated, all vectors and matrices will be in the basis  $\mathcal{K}_d$ . An element of  $C_d(\mathcal{K})$  is a **d-chain**. Let  $\sigma = \{v_{i_0}, \dots, v_{i_d}\}$  be a  $d$ -simplex in  $\mathcal{K}$  with  $v_{i_j} \leq v_{i_k}$  whenever  $j \leq k$ . The **boundary** of  $\sigma$  is the  $(d - 1)$ -chain  $\partial\sigma = \sum_{j=0}^d (-1)^j (\sigma \setminus \{v_{i_j}\})$ . The **d<sup>th</sup> boundary map** is the linear map  $\partial_d : C_d(\mathcal{K}) \rightarrow C_{d-1}(\mathcal{K})$  defined  $\partial_d f = \sum_{\sigma \in \mathcal{K}_d} f(\sigma) \partial\sigma$  where  $f(\sigma)$  denotes the component of  $f$  indexed by the simplex  $\sigma$ . A key property of the boundary map is that  $\partial_{d-1} \circ \partial_d = 0$ . An element in  $\ker \partial_d$  is a **cycle**, and an element in  $\text{im } \partial_d$  is a **boundary** or a **null-homologous cycle**. The boundary maps have the property that  $\partial_d \circ \partial_{d+1} = 0$ , so  $\text{im } \partial_{d+1} \subset \ker \partial_d$ . The **d<sup>th</sup> homology group** is the quotient group  $H_d(\mathcal{K}) = \ker(\partial_d) / \text{im}(\partial_{d+1})$ . The **d<sup>th</sup> Betti number**  $\beta_d$  is the dimension of  $H_d(\mathcal{K})$ . The **d<sup>th</sup> coboundary map** is the map  $\delta_d := \partial_{d+1}^T : C_d(\mathcal{K}) \rightarrow C_{d+1}(\mathcal{K})$ . An element of  $\ker \delta_d$  is a **cocycle**, and an element in  $\text{im } \delta_{d-1}$  is a **coboundary**. We will use the notation  $\partial[\mathcal{K}]$  and  $\delta[\mathcal{K}]$  when we want to specify the complex associated with the (co)boundary operator. For some of our results we will need to consider integral homology. The **integral chain group**  $C_d(\mathcal{K}, \mathbb{Z})$  is the free abelian group generated by the set  $\mathcal{K}_d$  whose elements are formal sums  $\sum_{\sigma_i \in \mathcal{K}_d} \alpha_i \sigma_i$  for  $\alpha_i \in \mathbb{Z}$ . The integral homology groups  $H_d(\mathcal{K}, \mathbb{Z})$  are constructed in the same way as with coefficients over the reals.

The **d<sup>th</sup> up Laplacian**<sup>3</sup> is  $L_d = \partial_{d+1} \delta_d$ . There are two variants of the up Laplacian: the weighted up Laplacian and the normalized up Laplacian. Let  $w : \mathcal{K}_{d+1} \rightarrow \mathbb{R}^+$  be a weight function on the  $(d+1)$ -simplices. Let  $W : C_{d+1}(\mathcal{K}) \rightarrow C_{d+1}(\mathcal{K})$  be the diagonal matrix with  $W_{\tau, \tau} = w(\tau)$ . The **d<sup>th</sup> weighted up Laplacian** is  $L_d^W = \partial_{d+1} W \delta_d$ . The **degree** of a  $d$ -simplex  $\sigma$  is  $\text{deg}(\sigma) = \sum_{\tau \in \mathcal{K}_{d+1} : \sigma \subset \tau} w(\tau)$ . Let  $D : C_d(\mathcal{K}) \rightarrow C_d(\mathcal{K})$  be the diagonal matrix with  $D_{\sigma, \sigma} = \text{deg}(\sigma)$ . The **d<sup>th</sup> normalized up Laplacian** is  $\tilde{L}_d = D^{-1/2} \partial_{d+1} W \delta_d D^{-1/2}$ .

We will make use of the **bra-ket notation** for vectors when discussing the quantum algorithm as this is the convention in the quantum computing literature. A **bra** is a row vector represented by the notation  $\langle v|$ . A **ket** is a column vector represented by the notation  $|v\rangle$ . The inner product of  $u$  and  $v$  is represented as  $\langle u|v\rangle$ .

Finally, we will often want to refer to the set of simplices given a non-zero value by a chain  $f$ . We call this set the **support** of  $f$  and denote it  $\text{supp}(f) = \{\sigma_i \in \mathcal{K}_d : f(\sigma_i) \neq 0\}$ .

### 3 Effective Resistance and Effective Capacitance

Let  $\gamma \in C_{d-1}(\mathcal{K})$  be a cycle in a simplicial complex. We associate two quantities with  $\gamma$ , its *effective resistance* and *effective capacitance*. The effective resistance is finite if and only if  $\gamma$  is null-homologous, and the effective capacitance is finite if and only if  $\gamma$  is not null-homologous. We begin with the definition of effective resistance.

► **Definition 1.** Let  $\mathcal{K}$  be a simplicial complex with weight function  $w : \mathcal{K}_d \rightarrow \mathbb{R}^+$ . Let  $\gamma$  be a  $(d-1)$ -cycle in  $\mathcal{K}$ . If  $\gamma$  is null-homologous, the **effective resistance** of  $\gamma$  is  $\mathcal{R}_\gamma(\mathcal{K}, W) = \gamma^T (L_d^W)^+ \gamma$ , where  $(L_d^W)^+$  is the Moore-Penrose pseudoinverse of  $L_d^W$ ; if  $\gamma$  is not null-homologous, then  $\mathcal{R}_\gamma(\mathcal{K}, W) = \infty$ . When obvious, we drop the weights from the notation and write  $\mathcal{R}_\gamma(\mathcal{K})$ .

<sup>3</sup> There are related operators called the *down Laplacian* and the *Laplacian* (see [8]), but we won't use either in this paper.

## 31:4 Effective Resistance and Capacitance in Simplicial Complexes

This definition of effective resistance is consistent with effective resistance in graphs (see [26]) and other definitions of effective resistance in simplicial complexes [18, 23, 9]. However, this definition gives little intuition about effective resistance. We now prove there is an alternative definition of effective resistance in terms of chains with boundary  $\gamma$ . We begin with two definitions.

► **Definition 2.** *Given a  $d$ -dimensional simplicial complex  $\mathcal{K}$  and a  $(d-1)$ -dimensional null-homologous cycle  $\gamma$ , a **unit  $\gamma$ -flow** is a  $d$ -chain  $f \in C_d(\mathcal{K})$  such that  $\partial f = \gamma$ .*

In the case of graphs a unit  $\gamma$ -flow is a flow sending 1 unit of flow from its source to its sink. We now define the *flow energy* of a unit  $\gamma$ -flow, which quantifies the size of the flow.

► **Definition 3.** *Given a  $d$ -dimensional simplicial complex  $\mathcal{K}$  with weight function  $w : C_d(\mathcal{K}) \rightarrow \mathbb{R}^+$  and a unit  $\gamma$ -flow  $f$ , the **flow energy** of  $f$  on  $\mathcal{K}$  is  $J(f) = \sum_{\sigma \in \mathcal{K}(d)} \frac{f(\sigma)^2}{w(\sigma)} = f^T W^{-1} f$  where  $W$  is the diagonal matrix whose entries are the weights of the  $d$ -simplices.*

We will now relate unit  $\gamma$ -flows and their energy to the definition of effective resistance.

► **Lemma 4.** *Let  $\mathcal{K}$  be a simplicial complex and let  $\gamma$  be a null-homologous  $d$ -cycle. The effective resistance of  $\gamma$  is the minimum flow energy over all unit  $\gamma$ -flows, i.e.  $\mathcal{R}_\gamma(\mathcal{K}) = \min\{J(f) \mid \partial f = \gamma\}$*

**Proof.** We use two well-known properties of the pseudoinverse.

1. If a matrix  $B = AA^T$ , then  $B^+ = (A^T)^+ A^+$ .
2. Let  $B$  be a matrix and  $|v\rangle \in \text{im } B$ . The vector  $B^+v$  is the minimum-norm vector that  $B$  maps to  $v$ , i.e.  $B^+v = \arg \min\{\|u\| : Bu = v\}$ .

Our first observation is that we can factor the weighted Laplacian as

$$\begin{aligned} L_d^W &= \partial_{d+1} W \delta_d \\ &= (\partial_{d+1} W^{1/2})(\partial_{d+1} W^{1/2})^T. \end{aligned}$$

By property 1 above,  $(L_d^W)^+ = ((\partial_{d+1} W^{1/2})^T)^+(\partial_{d+1} W^{1/2})^+$ . Therefore,

$$\begin{aligned} \mathcal{R}_\gamma(\mathcal{K}) &= \gamma^T ((\partial_{d+1} W^{1/2})^T)^+(\partial_{d+1} W^{1/2})^+ \gamma \\ &= \|(\partial W^{1/2})^+ \gamma\|^2. \end{aligned}$$

By property 2 above,  $\mathcal{R}_\gamma(\mathcal{K})$  is the minimum squared-norm of a vector that  $\partial_{d+1} W^{1/2}$  maps to  $\gamma$ . Let  $f = (\partial W^{1/2})^+ \gamma$ ; the vector  $f$  is the unit  $\gamma$ -flow of minimum flow energy, which we now prove. A vector  $v$  is mapped to  $\gamma$  by  $\partial W^{1/2}$  if and only if  $W^{1/2}v$  is mapped to  $\gamma$  by  $\partial$  as  $W^{1/2}$  is a bijection, i.e.  $W^{1/2}v$  is a unit  $\gamma$ -flow. Moreover, the flow energy of  $W^{1/2}v$  is  $J(W^{1/2}v) = (W^{1/2}v)^T W^{-1} (W^{1/2}v) = v^T v = \|v\|^2$ . Therefore, the minimum flow energy of a unit  $\gamma$ -flow is the minimum squared-norm of a vector that  $\partial W^{1/2}$  maps to  $\gamma$ , which we previously saw was  $\mathcal{R}_\gamma(\mathcal{K})$ . ◀

We say that  $(\partial W^{1/2})^+ \gamma$  is the **minimum-energy** unit  $\gamma$ -flow.

The definition of effective capacitance is less intuitive than the definition for effective resistance, both in graphs and simplicial complexes. As well, there are fewer results about effective capacitance in graphs than effective resistance and no previous results about effective capacitance in simplicial complexes. The effective capacitance of  $\gamma$  in a graph is the minimum energy of any *unit  $\gamma$ -potential*, which is analogous to effective resistance that was the minimum energy of any unit  $\gamma$ -flow. Before providing the definition of unit  $\gamma$ -potential in a simplicial complex we will begin by reviewing the definition of a unit *st*-potential in a graph, which can be found in [12].

Let  $G$  be a graph such that  $s$  and  $t$  are connected in  $G$ , and let  $H \subseteq G$  be a subgraph such that  $s$  and  $t$  are not connected in  $H$ . A **unit  $st$ -potential** is a function  $p: V(G) \rightarrow \mathbb{R}$  such that  $p(t) = 1, p(s) = 0$ , and for any two vertices  $u, v$  in the same connected component of  $H$ ,  $p(u) = p(v)$ . Viewing  $p$  as a 0-chain we see by the last property that its coboundary is zero in  $H$ . Intuitively, our definition of a unit  $\gamma$ -potential measures “how far” a cycle  $\gamma$  is from null-homologous in a subcomplex  $\mathcal{L}$  of  $\mathcal{K}$ . The definition of a unit  $\gamma$ -potential is analogous to the definition of a unit  $st$ -potential.

► **Definition 5.** Let  $\mathcal{L} \subset \mathcal{K}$  be simplicial complexes, and let  $\gamma \in C_{d-1}(\mathcal{L})$  be a  $(d-1)$ -cycle such that  $\gamma$  is null-homologous in  $\mathcal{K}$  but not  $\mathcal{L}$ . A **unit  $\gamma$ -potential** in  $\mathcal{L}$  is a  $(d-1)$ -chain  $p$  such that  $\delta[\mathcal{L}]p = 0$  and  $p^T\gamma = 1$ .

► **Definition 6.** Given simplicial complexes  $\mathcal{L} \subset \mathcal{K}$  with weight function  $w: C_d(\mathcal{K}) \rightarrow \mathbb{R}$  and a  $\gamma$ -potential  $p$  in  $\mathcal{L}$ , the **potential energy** of  $p$  on  $\mathcal{K}$  is  $\mathcal{J}(p) = \sum_{\sigma \in \mathcal{K}_d} ((\delta[\mathcal{K}]p)^T \sigma)^2 w(\sigma) = (\delta[\mathcal{K}]p)^T W(\delta[\mathcal{K}]p)$ .

Figure 1 in the appendix shows a  $\gamma$ -potential. It is not obvious from the definition that a unit  $\gamma$ -potential will even exist for  $\gamma$ . We prove this in the following lemma.

► **Lemma 7.** Let  $\mathcal{L} \subset \mathcal{K}$  be simplicial complexes whose  $(d-1)$ -skeletons are equal, and let  $\gamma \in C_{d-1}(\mathcal{L})$  be a cycle. Then there exists a unit  $\gamma$ -potential in  $\mathcal{L}$  if and only if  $\gamma$  is not null-homologous in  $\mathcal{L}$ .

**Proof.** Observe that  $\ker \delta_{d-1}[\mathcal{L}] = (\text{im } \partial_d[\mathcal{L}])^\perp$  as  $\delta_{d-1}[\mathcal{L}] = \partial[\mathcal{L}]^T$ . Assume there is a  $\gamma$ -potential  $p$  in  $\mathcal{L}$ . As  $\delta[\mathcal{L}]p = 0$ , then  $p \in \ker \delta_{d-1}[\mathcal{L}] = (\text{im } \partial_d[\mathcal{L}])^\perp$ . As  $p^T\gamma = 1$  we see that  $\gamma$  has a non-zero component in  $(\text{im } \partial_d[\mathcal{L}])^\perp$ , so  $\gamma \notin \text{im } \partial[\mathcal{L}]$ . Alternatively, suppose that  $\gamma$  is not null-homologous in  $\mathcal{L}$ . Then  $\gamma$  has a non-zero component in  $(\text{im } \partial_d[\mathcal{L}])^\perp = \ker \delta[\mathcal{L}]$ . Let  $q = \Pi_{\ker \delta[\mathcal{L}]} \gamma$ , where  $\Pi_{\ker \delta[\mathcal{L}]}$  is the projection onto  $\ker \delta[\mathcal{L}]$ . Then  $q^T\gamma \neq 0$  and  $\delta[\mathcal{L}]q = 0$ . The vector  $q$  is not necessarily a unit  $\gamma$ -potential as it is not necessarily the case that  $q^T\gamma = 1$ , but the scaled vector  $p = \frac{1}{q^T\gamma} q$  is a unit  $\gamma$ -potential. ◀

Just as the effective resistance of  $\gamma$  was the minimum energy of any unit  $\gamma$ -flow, the effective capacitance of  $\gamma$  is the minimum energy of any unit  $\gamma$ -potential.

► **Definition 8.** Let  $\mathcal{L} \subset \mathcal{K}$  be simplicial complexes, and let  $\gamma \in C_{d-1}(\mathcal{L})$  be a  $(d-1)$ -cycle that is null-homologous in  $\mathcal{K}$ . If  $\gamma$  is not null-homologous in  $\mathcal{L}$ , the **effective capacitance** of  $\gamma$  in  $\mathcal{L}$  is  $\mathcal{C}_\gamma(\mathcal{L}) = \min_p \mathcal{J}(p)$  where  $p$  is a  $\gamma$ -potential. If  $\gamma$  is null-homologous in  $\mathcal{L}$ , then  $\mathcal{C}_\gamma(\mathcal{L}) = \infty$ .

## 4 Basic Properties: Parallel, Series, and Monotonicity Formulas

We now prove there are formulas for effective resistance in simplicial complexes analogous to the series and parallel formulas for effective resistance in graphs. These formulas not only are useful for calculating effective resistance, but they also provide intuition for effective resistance. In particular, they provide justification for the claim that effective resistance measures “how null-homologous” a cycle is in a complex.

► **Theorem 9 (Series Formula).** Let  $\mathcal{K}_1$  and  $\mathcal{K}_2$  be simplicial complexes with  $\gamma \in C_{d-1}(\mathcal{K}_1) \cap C_{d-1}(\mathcal{K}_2)$ ,  $C_d(\mathcal{K}_1) \cap C_d(\mathcal{K}_2) = \emptyset$ , and  $\gamma$  null-homologous in  $\mathcal{K}_1$  and  $\mathcal{K}_2$ . Let  $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2$ . Then  $\mathcal{R}_\gamma(\mathcal{K}) \leq \mathcal{R}_{\gamma_1}(\mathcal{K}_1) + \mathcal{R}_{\gamma_2}(\mathcal{K}_2)$ . Equality is achieved when  $\gamma_1$  and  $\gamma_2$  are the unique chains in  $\mathcal{K}_1$  and  $\mathcal{K}_2$  that sum to  $\gamma$ .

**Proof.** Let  $\gamma_1$  and  $\gamma_2$  be null-homologous cycles in  $\mathcal{K}_1$  and  $\mathcal{K}_2$  respectively that sum to  $\gamma$ , and let  $f_1$  and  $f_2$  be the minimum-energy unit  $\gamma_1$ - and  $\gamma_2$ -flows, respectively. Then  $f_1 + f_2$  is a unit  $\gamma$ -flow, and we can bound  $\mathcal{R}_\gamma(\mathcal{K}) \leq J(f) = J(f_1) + J(f_2) = \mathcal{R}_{\gamma_1}(\mathcal{K}_1) + \mathcal{R}_{\gamma_2}(\mathcal{K}_2)$ ; the equality  $J(f) = J(f_1) + J(f_2)$  follows from the fact that  $\mathcal{K}_1$  and  $\mathcal{K}_2$  have disjoint sets of  $d$ -simplices.

To prove the other direction, observe that  $\gamma$  can always be written as the sum of two null-homologous chains  $\gamma_1 \in C_{d-1}(\mathcal{K}_1)$  and  $\gamma_2 \in C_{d-1}(\mathcal{K}_2)$ . Any unit  $\gamma$ -flow  $g$  defines null-homologous  $(d-1)$ -cycles  $\gamma_1$  and  $\gamma_2$  that sum to  $\gamma$ ; namely, if  $g_1$  and  $g_2$  are the restriction of  $g$  to  $\mathcal{K}_1$  and  $\mathcal{K}_2$  respectively, then  $\gamma_1 = \partial g_1$  and  $\gamma_2 = \partial g_2$ .

If  $\gamma$  can be uniquely decomposed as  $\gamma = \gamma_1 + \gamma_2$ , then any unit  $\gamma$ -flow  $f$  can be decomposed as a unit  $\gamma_1$ -flow  $f_1$  and a unit  $\gamma_2$ -flow  $f_2$ . It follows that the energy of  $f$  is minimized when the energy of  $f_1$  and  $f_2$  are both minimized. Hence,  $\mathcal{R}_\gamma(\mathcal{K}) = \mathcal{R}_{\gamma_1}(\mathcal{K}_1) + \mathcal{R}_{\gamma_2}(\mathcal{K}_2)$ . ◀

► **Theorem 10 (Parallel Formula).** *Let  $\mathcal{K}_1$  and  $\mathcal{K}_2$  be simplicial complexes with  $\gamma \in C_{d-1}(\mathcal{K}_1) \cap C_{d-1}(\mathcal{K}_2)$ ,  $C_d(\mathcal{K}_1) \cap C_d(\mathcal{K}_2) = \emptyset$ , and  $\gamma$  null-homologous in  $\mathcal{K}_1$  and  $\mathcal{K}_2$ . Let  $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2$ . Then  $\mathcal{R}_\gamma(\mathcal{K}) \leq \left( \frac{1}{\mathcal{R}_\gamma(\mathcal{K}_1)} + \frac{1}{\mathcal{R}_\gamma(\mathcal{K}_2)} \right)^{-1}$ . Equality is achieved when  $\text{im } \partial[\mathcal{K}_1] \cap \text{im } \partial[\mathcal{K}_2] = \text{span}\{\gamma\}$ .*

**Proof.** Let  $f_1$  and  $f_2$  be the minimum energy unit  $\gamma$ -flows in  $\mathcal{K}_1$  and  $\mathcal{K}_2$  resp. For any real number  $t$ , the chain  $g_t = tf_1 + (1-t)f_2$  is a unit  $\gamma$ -flow in  $\mathcal{K}$ . We can therefore bound the effective resistance as  $\mathcal{R}_\gamma(\mathcal{K}) \leq \min_t J(g_t)$ .

To get the tightest bound of  $\mathcal{R}_\gamma(\mathcal{K})$ , we now derive  $t_{\text{opt}} := \arg \min_t J(g_t)$ . Observe that  $J(g_t) = t^2 J(f_1) + (1-t)^2 J(f_2) = t^2 \mathcal{R}_\gamma(\mathcal{K}_1) + (1-t)^2 \mathcal{R}_\gamma(\mathcal{K}_2)$ ; this follows from the fact that  $\mathcal{K}_1$  and  $\mathcal{K}_2$  have disjoint sets of  $d$ -simplices. The quantity  $J(g_t)$  is a positive quadratic in  $t$ , so  $t_{\text{opt}}$  is the value of  $t$  where the derivative of  $J(g_t)$  with respect to  $t$  is 0. Taking the derivative, we find that  $t_{\text{opt}} = \mathcal{R}_\gamma(\mathcal{K}_2) / (\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2))$ . Plugging  $t_{\text{opt}}$  into  $J(g_{t_{\text{opt}}})$ , we find that

$$\begin{aligned} J(g_{t_{\text{opt}}}) &= \left( \frac{\mathcal{R}_\gamma(\mathcal{K}_2)}{\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2)} \right)^2 \mathcal{R}_\gamma(\mathcal{K}_1) + \left( 1 - \frac{\mathcal{R}_\gamma(\mathcal{K}_2)}{\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2)} \right)^2 \mathcal{R}_\gamma(\mathcal{K}_2) \\ &= \left( \frac{\mathcal{R}_\gamma(\mathcal{K}_2)}{\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2)} \right)^2 \mathcal{R}_\gamma(\mathcal{K}_1) + \left( \frac{\mathcal{R}_\gamma(\mathcal{K}_1)}{\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2)} \right)^2 \mathcal{R}_\gamma(\mathcal{K}_2) \\ &= \left( \frac{1}{\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2)} \right)^2 \left( \mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2) \right) \mathcal{R}_\gamma(\mathcal{K}_1) \mathcal{R}_\gamma(\mathcal{K}_2) \\ &= \frac{\mathcal{R}_\gamma(\mathcal{K}_1) \mathcal{R}_\gamma(\mathcal{K}_2)}{\mathcal{R}_\gamma(\mathcal{K}_1) + \mathcal{R}_\gamma(\mathcal{K}_2)} \\ &= \left( \frac{1}{\mathcal{R}_\gamma(\mathcal{K}_1)} + \frac{1}{\mathcal{R}_\gamma(\mathcal{K}_2)} \right)^{-1}. \end{aligned}$$

This implies the upper bound on  $\mathcal{R}_\gamma(\mathcal{K})$  in the theorem statement. To get the lower bound, observe that any unit  $\gamma$ -flow  $g$  in  $\mathcal{K}$  can be orthogonally decomposed into chains  $g_1 \in C_d(\mathcal{K}_1)$  and  $g_2 \in C_d(\mathcal{K}_2)$ , so  $\partial[\mathcal{K}]g_1 + \partial[\mathcal{K}]g_2 = \gamma$ . We claim that  $\partial[\mathcal{K}_1]g_1 = t\gamma$  and  $\partial[\mathcal{K}_2]g_2 = (1-t)\gamma$  for some value of  $t$ ; if not, then  $\partial[\mathcal{K}_1]g_1 = t\gamma + \eta$  and  $\partial[\mathcal{K}_2]g_2 = (1-t)\gamma - \eta$  for some non-zero  $\eta \notin \text{span}\{\gamma\}$ , which cannot be the case as  $\text{im } \partial[\mathcal{K}_1] \cap \text{im } \partial[\mathcal{K}_2] = \text{span}\{\gamma\}$ . This proves the chain  $g$  is a linear combination of a unit  $\gamma$ -flow in  $\mathcal{K}_1$  and a unit  $\gamma$ -flow in  $\mathcal{K}_2$ . The chain  $g_t$  is the lowest energy such linear combination. ◀

Figure 2 in the appendix shows examples of unit  $\gamma$ -flows in series and parallel. These formulas justify the claim that the effective resistance of a null-homologous cycle  $\gamma$  is a measure of *how* null-homologous  $\gamma$  is. The more chains with boundary  $\gamma$ , the smaller the effective resistance of  $\gamma$  by the parallel formula. The smaller the chains bounding  $\gamma$ , the lower the effective resistance by the series formula.

A similar result to the series and parallel formula for effective resistance in graphs is *Rayleigh monotonicity*. Rayleigh monotonicity says that adding edges to a graph can only decrease the effective resistance between any pair of vertices; this reinforces the notion that effective resistance measures how well-connected a pair of vertices are, as adding an edge can only make a pair of vertices better connected. We prove a similar result for simplicial complexes.

► **Theorem 11** (Rayleigh Monotonicity). *Let  $\mathcal{K} \subset \mathcal{L}$  be simplicial complexes. Let  $|\gamma\rangle \in C_{d-1}(\mathcal{K}) \cap C_{d-1}(\mathcal{L})$  be a null-homologous cycle in both complexes. Then  $\mathcal{R}_\gamma(\mathcal{L}) \leq \mathcal{R}_\gamma(\mathcal{K})$ .*

**Proof.** As  $C_d(\mathcal{K}) \subset C_d(\mathcal{L})$ , then any unit  $\gamma$ -flow in  $\mathcal{K}$  is also a unit  $\gamma$ -flow in  $\mathcal{L}$ . As the effective resistance is the minimum energy of a unit  $\gamma$ -flow, then clearly  $\mathcal{R}_\gamma(\mathcal{L}) \leq \mathcal{R}_\gamma(\mathcal{K})$ . ◀

## 5 Bounds on resistance and capacitance

In this section, we provide upper bounds on the resistance and capacitance of a cycle  $\gamma$  in an unweighted simplicial complex  $\mathcal{K}$ . Our upper bounds are polynomial in the number of  $d$ -simplices and the cardinality of the torsion subgroup of the relative homology groups. In particular, our bounds on resistance and capacitance are dependent on the maximum cardinality of the torsion subgroup of the relative homology group  $H_{d-1}(\mathcal{L}, \mathcal{L}_0, \mathbb{Z})$ , where  $\mathcal{L} \subset \mathcal{K}$  is a  $d$ -dimensional subcomplex and  $\mathcal{L}_0 \subset \mathcal{L}$  is a  $(d-1)$ -dimensional subcomplex. In the worst case, our upper bounds are exponential in the number of  $d$ -simplices. There exist simplicial complexes such that the torsion subgroup of  $H_{d-1}(\mathcal{K}, \mathbb{Z})$  has cardinality  $n$  while  $\mathcal{K}$  only has  $O(\log^{1/d} n)$  vertices [22]. Note that such a complex contains at most  $O(n^d)$   $d$ -simplices.

In Theorem 17 we provide an example of a simplicial complex containing a cycle  $\gamma$  whose effective resistance is exponential in the number of simplices in the complex. It is important to reiterate that our bounds are in terms of the torsion of the relative homology groups. There exist simplicial complexes with no torsion in their homology groups but that do have torsion in their relative homology groups. An example of this is the Möbius strip. The Möbius strip has no torsion, but it has torsion relative to its boundary [5].

Our results rely on a change of basis on the boundary matrix called the *normal form* which reveals information about the torsion subgroup of  $H_{d-1}(\mathcal{K}, \mathbb{Z})$ . We state the normal form theorem below.

► **Theorem 12** (Munkres, Chapter 1 Section 11 [21]). *There are bases for  $C_d(\mathcal{K}, \mathbb{Z})$  and  $C_{d-1}(\mathcal{K}, \mathbb{Z})$  such that the matrix for the boundary operator  $\partial_d: C_d(\mathcal{K}, \mathbb{Z}) \rightarrow C_{d-1}(\mathcal{K}, \mathbb{Z})$  is in **normal form**, i.e.  $\tilde{\partial}_d = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}$  where  $D$  is a diagonal matrix with entries  $d_1, \dots, d_m$  such that each  $d_i$  divides  $d_{i+1}$  and each 0 is a zero matrix of appropriate dimensionality. The normal form of  $\partial_d$  satisfies the following properties:*

1. *The entries  $d_1, \dots, d_m$  correspond to the torsion coefficients of  $H_{d-1}(\mathcal{K}, \mathbb{Z}) \cong \mathbb{Z}^{\beta_d} \oplus \mathbb{Z}_{d_1} \oplus \dots \oplus \mathbb{Z}_{d_m}$  (where  $\mathbb{Z}_1 = 0$ ),*
  2. *The number of zero columns is equal to the dimension of  $\ker(\partial_d)$ .*
- Moreover, the boundary matrix  $\partial$  in the standard basis can be transformed to  $\tilde{\partial}$  by a set of elementary row and column operations. If  $\partial$  is square, these operations multiply  $\det \partial$  by  $\pm 1$ .*

Using Theorem 12, we obtain an upper bound on the determinants of the square submatrices of the boundary matrix  $\partial_d[\mathcal{K}]$  in terms of the *relative homology groups* of  $\mathcal{K}$ . Let  $\mathcal{L}$  be  $d$ -dimensional subcomplex of  $\mathcal{K}$ , and let  $\mathcal{L}_0$  be a  $(d-1)$ -dimensional subcomplex of  $\mathcal{K}$ . The **relative boundary matrix**  $\partial_d[\mathcal{L}, \mathcal{L}_0]$  is the submatrix of  $\partial_d$  obtained by including the columns of the  $d$ -simplices in  $\mathcal{L}$  and excluding the rows of the  $(d-1)$ -simplices in  $\mathcal{L}_0$ . With the relative boundary matrices, one can define the **relative homology groups** as  $H_k(\mathcal{L}, \mathcal{L}_0, \mathbb{Z}) = \ker \partial_k[\mathcal{L}, \mathcal{L}_0] / \text{im } \partial_{k+1}[\mathcal{L}, \mathcal{L}_0]$ . More information on the relative boundary matrix can be found in [5]. We denote the cardinality of the torsion subgroup of the relative homology group  $H_{d-1}(\mathcal{L}, \mathcal{L}_0, \mathbb{Z})$  by  $\mathcal{T}(\mathcal{L}, \mathcal{L}_0)$ . Similarly, we denote the maximum  $\mathcal{T}(\mathcal{L}, \mathcal{L}_0)$  over all relative homology groups as  $\mathcal{T}_{\max}(\mathcal{K})$ .

► **Lemma 13.** *Let  $\partial_d[\mathcal{L}, \mathcal{L}_0]$  be a  $k \times k$  square submatrix of  $\partial_d$  constructed by including columns for the  $d$ -simplices  $\mathcal{L}$  and excluding rows for the  $(d-1)$ -simplices  $\mathcal{L}_0$ . The magnitude of the determinant of  $\partial_d[\mathcal{L}, \mathcal{L}_0]$  is bounded above by the cardinality of the torsion subgroup of  $H_{d-1}(\mathcal{L}, \mathcal{L}_0, \mathbb{Z})$ , i.e.  $|\det(\partial_d[\mathcal{L}, \mathcal{L}_0])| \leq \mathcal{T}(\mathcal{L}, \mathcal{L}_0)$ .*

**Proof.** Without loss of generality, we assume that  $\det(\partial_d[\mathcal{L}, \mathcal{L}_0]) \neq 0$ ; if  $\det(\partial_d[\mathcal{L}, \mathcal{L}_0]) = 0$ , the bound is trivial. Since  $\partial_d[\mathcal{L}, \mathcal{L}_0]$  is a non-singular square matrix, its normal form  $\tilde{\partial}_d[\mathcal{L}, \mathcal{L}_0]$  is a diagonal matrix  $D = \text{diag}(d_1, \dots, d_k)$ . By Theorem 12, the determinant of  $\partial_d[\mathcal{L}, \mathcal{L}_0]$  is equal to  $\pm \det(\tilde{\partial}_d[\mathcal{L}, \mathcal{L}_0]) = \prod_{i=1}^k d_i$ . Also by Theorem 12, the torsion subgroup of  $H_{d-1}(\mathcal{L}, \mathcal{L}_0)$  is  $\mathbb{Z}_{d_1} \oplus \dots \oplus \mathbb{Z}_{d_k}$  which has cardinality  $\mathcal{T}(\mathcal{L}, \mathcal{L}_0) = \prod_{i=1}^k d_i$ . ◀

We are now ready to upper bound the effective resistance of a cycle  $\gamma$  in a simplicial complex  $\mathcal{K}$ .

► **Theorem 14.** *Let  $\mathcal{K}$  be an unweighted  $d$ -dimensional simplicial complex with  $n$   $d$ -simplices. Let  $\gamma$  be a unit-length null-homologous  $(d-1)$ -cycle in  $\mathcal{K}$ . The effective resistance of  $\gamma$  is bounded above as  $\mathcal{R}_\gamma(\mathcal{K}) = O(n^2 \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ .*

**Proof.** First, we remove  $d$ -simplices from  $\mathcal{K}$  to create a new complex  $\mathcal{L}$  such that  $\ker(\partial_d[\mathcal{L}]) = 0$  and  $\text{im } \partial_d[\mathcal{K}] = \text{im } \partial_d[\mathcal{L}]$ . Theorem 11 proves that removing  $d$ -simplices only increases the effective resistance, so  $\mathcal{R}_\gamma(\mathcal{K}) \leq \mathcal{R}_\gamma(\mathcal{L})$ . As  $\ker(\partial_d[\mathcal{L}]) = 0$ , there is a unique unit  $\gamma$ -flow  $f \in C_{d-1}(\mathcal{L})$ , which implies  $\mathcal{R}_\gamma(\mathcal{L}) = \|f\|^2$ .

The matrix  $\partial_d[\mathcal{L}]$  has full column rank, so we can find a non-singular  $n_d \times n_d$  square submatrix of  $\partial_d[\mathcal{L}]$ ; call this submatrix  $B$ . Let  $\mathcal{L}_0$  be the  $(d-1)$ -dimensional subcomplex that contains the  $(d-1)$ -simplices corresponding to rows excluded from  $B$ ;  $B$  is the relative boundary matrix  $\partial_d[\mathcal{L}, \mathcal{L}_0]$ . We have that  $Bf = c$ , where  $c$  is the restriction of  $\gamma$  to the rows of  $B$ . Observe that  $\|c\| \leq \|\gamma\| = 1$

We will apply Cramer's rule to upper bound the size of  $f$ . Let  $f(\sigma)$  denote the component of  $f$  indexed by the  $d$ -simplex  $\sigma$ . Cramer's rule gives the equality  $f(\sigma) = \frac{\det(B_{\sigma,c})}{\det(B)}$  where  $B_{\sigma,c}$  is the matrix obtained by replacing the column of  $B$  indexed by  $\sigma$  with the vector  $c$ . Since  $\det(B)$  is integral,  $|\det(B)| \geq 1$ , we can drop the denominator and use the inequality  $|f(\sigma)| \leq |\det(B_{\sigma,c})|$ . We bound  $|\det(B_{\sigma,c})|$  by its cofactor expansion on the column  $c$ , specifically  $|\det(B_{\sigma,c})| = |\sum_{i=1}^{n_d} (-1)^{i+j} \cdot c_i \cdot \det(B_{\sigma,c}^{c,i})| \leq \sum_{i=1}^{n_d} |c_i| \cdot t(\mathcal{K}) = O(\sqrt{n} \cdot \mathcal{T}_{\max}(\mathcal{K}))$  where  $B_{\sigma,c}^{c,i}$  denotes the submatrix obtained by removing the column  $c$  and removing the  $i$ th row and  $c_i$  denotes the  $i$ th component of  $c$ . The first inequality comes from Lemma 13, as  $B_{\sigma,c}^{c,i}$  is the relative boundary matrix  $\partial[\mathcal{L} \setminus \{\sigma\}, \mathcal{L}_0 \cup \sigma_i]$ , where  $\sigma_i$  is the  $(d-1)$ -simplex corresponding to the  $i$ th row of  $B$ . The factor of  $\sqrt{n}$  comes from our assumption that  $\|c\| \leq 1$  and the fact that  $\sum_{i=1}^{n_d} |c_i| = \|c\|_1 \leq \sqrt{|\text{supp}(c)|} \cdot \|c\|_2 \leq \sqrt{n} \cdot \|c\|_2$ , which can be shown using the Cauchy-Schwarz inequality. Finally, we compute the flow energy of  $f$  as  $J(f) = \sum_{\sigma \in \mathcal{K}^d} f(\sigma)^2 \leq \sum_{i=1}^{n_d} n \cdot \mathcal{T}_{\max}(\mathcal{K})^2 = O(n^2 \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ . The effective resistance of  $\gamma$  is the flow energy of  $f$ , so the result follows. ◀



The same argument also applies for any subcomplex  $\mathcal{L} \subset \mathcal{K}$  where  $\gamma$  is null-homologous in  $\mathcal{L}$  which gives us the following corollary.

► **Corollary 15.** *Let  $\mathcal{L} \subset \mathcal{K}$  be an unweighted  $d$ -dimensional simplicial complex and  $\gamma$  a null-homologous  $(d-1)$ -cycle in  $\mathcal{L}$ . The effective resistance of  $\gamma$  in  $\mathcal{L}$  is bounded above by  $\mathcal{R}_\gamma(\mathcal{L}) = O(n^2 \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ .*

In order to upper bound the effective capacitance of  $\gamma$ , we need to make two additional assumptions. We need to assume that as input  $\gamma$  has integral coefficients and that each coefficient is bound above by a constant. Even though  $\gamma$  is given as input as an integral cycle, we still normalize  $\gamma$  as a preprocessing step. Normalizing  $\gamma$  does not change whether or not  $\gamma$  is null-homologous. We state the theorem but leave the proof to the appendix. The proof idea is similar to that of the bound on effective resistance, but with a few extra technical details.

► **Theorem 16.** *Let  $\mathcal{L} \subset \mathcal{K}$  be an unweighted  $d$ -dimensional simplicial complexes, and let  $\hat{\gamma} \in C_{d-1}(\mathcal{L})$  be a  $(d-1)$ -cycle that is null-homologous in  $\mathcal{K}$  but not in  $\mathcal{L}$ . Assume also that  $\hat{\gamma}$  is integral and each of the coefficients  $|\hat{\gamma}_i| = O(1)$ . The effective capacitance of  $\gamma := \hat{\gamma}/\|\hat{\gamma}\|$  in  $\mathcal{K}(x)$  is bounded above by  $\mathcal{C}_\gamma(\mathcal{L}) = O(n^3 \cdot d \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ .*

Recall that  $t(\mathcal{K})$  could be exponential in the size of the complex. To end the section, we now provide an example of a simplicial complex containing a cycle  $\gamma$  such that the effective resistance of  $\gamma$  is exponential in the size of the complex.

► **Theorem 17.** *There exists a 2-dimensional simplicial complex with  $\Theta(n)$  triangles and a cycle  $\gamma$  such that the effective resistance of  $\gamma$  is  $\Theta(2^{2n})$ .*

**Proof.** Let  $\mathbb{RP}_\gamma$  denote a simplicial complex homeomorphic to the real projective plane with a disk removed; the cycle  $\gamma$  boundary of the removed disk. Hence, we have that the boundary of the sum of the triangles in the complex is  $\partial_2 \mathbb{RP}_\gamma = 2\alpha + \gamma$  for some 1-cycle  $\alpha$ . We require  $\mathbb{RP}_\gamma$  to be triangulated in such a way that  $|\text{supp}(\alpha)| = |\text{supp}(\gamma)|$ ; that is,  $\alpha$  and  $\gamma$  contain the same number of edges. Let the constant  $c$  denote the number of triangles in  $\mathbb{RP}_\gamma$ . See Figure 3 in the appendix for a triangulation of  $\mathbb{RP}_\gamma$ .

We consider a collection of disjoint complexes  $\mathbb{RP}_{\gamma_0}, \mathbb{RP}_{\gamma_1}, \dots, \mathbb{RP}_{\gamma_{n-1}}, \mathcal{D}_{\gamma_n}$ . Each  $\mathbb{RP}_{\gamma_i}$  is constructed in the same way was  $\mathbb{RP}_\gamma$ , and  $\mathcal{D}_{\gamma_n}$  triangulation of a disk using  $c$  triangles with boundary  $\gamma_n$  such that  $|\text{supp}(\gamma_n)| = |\text{supp}(\gamma)|$ . The sum of the triangles of each  $\mathbb{RP}_{\gamma_i}$  has boundary  $\partial_2 \mathbb{RP}_{\gamma_i} = 2\alpha_i + \gamma_i$ .

We consider the simplicial complex  $\mathcal{K}$  constructed by taking the quotient space under the identification  $\alpha_i \sim \gamma_{i+1}$ . That is, we glue the cycle  $\alpha_i$  in  $\mathbb{RP}_{\gamma_i}$  along the cycle  $\gamma_{i+1}$  in  $\mathbb{RP}_{\gamma_{i+1}}$ . The resulting complex contains a unique unit  $\gamma_0$ -flow  $f$ . The chain  $f$  assigns a value of 1 to each triangle in  $\mathbb{RP}_{\gamma_0}$ , a value of -2 to each triangle in  $\mathbb{RP}_{\gamma_1}$ , and in general, a value of  $(-1)^i 2^{i+1}$  to each triangle in  $\mathbb{RP}_{\gamma_i}$  and  $\mathcal{D}_{\gamma_n}$ . To see that  $f$  is indeed a unit  $\gamma_0$ -flow we compute the boundary

$$\begin{aligned} \partial f &= \left( \sum_{i=0}^{n-1} (-1)^i \cdot 2^i \partial \mathbb{RP}_{\gamma_i} \right) + (-1)^n \cdot 2^n \partial \mathcal{D}_{\gamma_n} \\ &= \left( \sum_{i=0}^{n-1} (-1)^i \cdot 2^i (\gamma_i + 2\gamma_{i+1}) \right) + (-1)^n \cdot 2^n \gamma_n \\ &= \gamma_0. \end{aligned}$$



## 31:10 Effective Resistance and Capacitance in Simplicial Complexes

The chain  $f$  is unique because the value of  $f$  must be equal to 1 on each triangle in  $\mathbb{RP}_{\gamma_0}$ , and the values on the triangles of  $\mathbb{RP}_{\gamma_i}$  determines the values on the triangles of  $\mathbb{RP}_{\gamma_{i+1}}$  and  $\mathcal{D}_{\gamma_n}$  as the  $\gamma_i$  terms must cancel out in  $\partial f$ .

As  $f$  is the unique unit  $\gamma_0$ -flow, the effective resistance of  $f$  is the flow energy of  $\gamma$ . The flow energy of  $f$  is  $J(f) = \sum_{i=0}^n c \cdot (2^i)^2 = \frac{c}{3} (2^{2n+2} - 1) = \Theta(2^{2n})$ . ◀

## 6 A Quantum Algorithm for Null-Homology Testing

In this section we provide an application of effective resistance and capacitance in simplicial complexes. We show that a quantum algorithm based on the *span program* model can be used to decide whether or not a cycle  $\gamma$  is null-homologous in a simplicial complex  $\mathcal{K}$ . An introduction to the span program model can be found in Appendix 6.1. The query complexity of this algorithm is parameterized by the maximum (finite) effective resistance and capacitance of  $\gamma$  over all subcomplexes of  $\mathcal{K}$ .

Our algorithm is a generalization of the quantum algorithm developed by Belovs and Reichardt to decide *st*-connectivity in a graph [1]. Their algorithm is parameterized by the effective resistance and capacitance of the 0-cycle  $|t\rangle - |s\rangle$  in the graph. Upper bounds on the effective resistance and capacitance imply a query complexity of  $O(n^{3/2})$ , where  $n$  is the number of vertices [12].

Our upper bounds on effective resistance and capacitance imply that the query complexity is polynomial in both the number of  $d$ -simplices as well as the cardinality of the largest torsion subgroup of a relative homology group of  $\mathcal{K}$ . In the case that  $\mathcal{K}$  is a graph, we match the  $O(n^{3/2})$  upper bound. Under the assumptions that  $\mathcal{K}$  is relative torsion free and that  $\gamma$  is the boundary of a  $d$ -simplex (which may or may not be included in the complex) we also match the  $O(n^{3/2})$  upper bound. Note that these assumptions are always true for *st*-connectivity in graphs.

### 6.1 A brief introduction to span programs

Span programs were first defined by Karchmer and Wigderson [14] and were first used for quantum algorithms by Reichardt and Špalek [24]. Intuitively, a span program is a model of computation which encodes a boolean function  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  into the geometry of two vector spaces and a linear operator between them. Encoding  $f$  into a span program implies the existence of a quantum query algorithm evaluating  $f$  (Theorem 20.)

► **Definition 18.** A *span program*  $\mathcal{P} = (\mathcal{H}, \mathcal{U}, |\tau\rangle, A)$  over the set of strings  $\{0, 1\}^n$  is a 4-tuple consisting of:

1. A finite dimensional Hilbert space  $\mathcal{H} = \mathcal{H}_1 \oplus \dots \oplus \mathcal{H}_n$  where  $\mathcal{H}_i = \mathcal{H}_{i,0} \oplus \mathcal{H}_{i,1}$ ,
2. a vector space  $\mathcal{U}$ ,
3. a non-zero vector  $|\tau\rangle \in \mathcal{U}$ , called the **target vector**
4. a linear operator  $A: \mathcal{H} \rightarrow \mathcal{U}$ .

For every string  $x = (x_1, \dots, x_n) \in \{0, 1\}^n$  we associate the Hilbert space  $\mathcal{H}(x) = \mathcal{H}_{1,x_1} \oplus \dots \oplus \mathcal{H}_{N,x_n}$  and the linear operator  $A(x) = A\Pi_{\mathcal{H}(x)}: \mathcal{H} \rightarrow \mathcal{U}$  where  $\Pi_{\mathcal{H}(x)}$  is the projection of  $\mathcal{H}$  onto  $\mathcal{H}(x)$ .

The quantum query complexity of evaluating  $\mathcal{P}$  depends on the sizes of the positive and negative witnesses, which we now define.

► **Definition 19.** Let  $\mathcal{P}$  be a span program and let  $x \in \{0, 1\}^N$ . A **positive witness** for  $x$  is a vector  $|w\rangle \in \mathcal{H}(x)$  such that  $A|w\rangle = |\tau\rangle$ . The **positive witness size** of  $x$  is

$$w_+(x, \mathcal{P}) = \min\{\| |w\rangle \|^2 : |w\rangle \in \mathcal{H}(x), A|w\rangle = |\tau\rangle\}.$$

If no positive witness exists for  $x$ , then  $w_+(x, \mathcal{P}) = \infty$ . If there is a positive witness for  $x$ , then  $x$  is a **positive instance**.

A **negative witness** for  $x$  is a linear map  $\langle w| : \mathcal{U} \rightarrow \mathbb{R}$  such that  $\langle w|A\Pi_{\mathcal{H}(x)} = 0$  and  $\langle w|\tau\rangle = 1$ . The **negative witness size** of  $x$  is

$$w_-(x, \mathcal{P}) = \min\{\|\langle w|A\|^2 : \langle w| : \mathcal{U} \rightarrow \mathbb{R}, \langle w|A\Pi_{\mathcal{H}(x)} = 0, \langle w|\tau\rangle = 1\}.$$

If no negative witness exists for  $x$ , then  $w_-(x, \mathcal{P}) = \infty$ . If there is a negative witness for  $x$ , then  $x$  is a **negative instance**.

A string  $x \in \{0, 1\}^N$  will either be a positive or negative instance of  $\mathcal{P}$ . A span program  $\mathcal{P}$  **decides** the function  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  if  $f(x) = 1$  when  $x$  is a positive instance and  $f(x) = 0$  when  $x$  is a negative instance. A span program can also evaluate a partial boolean function  $g: D \rightarrow \{0, 1\}$  where  $D \subset \{0, 1\}^n$  by the same criteria.

Span programs are a popular method in quantum computing because there are upper bounds on the complexity of evaluating span programs in the **query model**. The query model evaluates the complexity of a quantum algorithm by its **query complexity**, the number of times it queries an input oracle. In our case, the input oracle returns the bits of the binary string  $x$ . The **input oracle**  $\mathcal{O}_x$  takes  $\mathcal{O}_x: |i\rangle|b\rangle \rightarrow |i\rangle|b \oplus x_i\rangle$  where  $i \in [N]$ . Observe that the states  $|i\rangle$  can be stored on  $\lceil \log N \rceil$  qubits. Reichardt [25] showed that the query complexity of a span program is a function of the positive and negative witness sizes of the program.

► **Theorem 20** (Reichardt [25]). Let  $D \subset \{0, 1\}^N$  and  $f: D \rightarrow \{0, 1\}$ . Let  $\mathcal{P}$  be a span program that decides  $f$ . Let  $W_+(f, \mathcal{P}) = \max_{x \in f^{-1}(1)} w_+(x, \mathcal{P})$  and  $W_-(f, \mathcal{P}) = \max_{x \in f^{-1}(0)} w_-(x, \mathcal{P})$ . There is a bounded error quantum algorithm that decides  $f$  with query complexity  $O\left(\sqrt{W_+(f, \mathcal{P})W_-(f, \mathcal{P})}\right)$ .

A caveat to the query complexity model is that in general the time complexity of an algorithm can be much larger than the query complexity. We will provide details on bounding the time complexity of our problem in Section A.

## 6.2 A span program to decide if a cycle is null-homologous

In this section we present a span program for testing if a cycle is null-homologous in a simplicial complex. This span program is a generalization of the span program for  $st$ -connectivity defined in [14] and used to develop quantum algorithms in [1, 2, 12, 13]. Let  $\mathcal{K}$  be a  $d$ -dimensional simplicial complex. Let  $|\gamma\rangle \in C_{d-1}(\mathcal{K})$  be a  $(d-1)$ -cycle. Let  $n$  be the number of  $d$ -simplices in  $\mathcal{K}$ . Order the  $d$ -simplices  $\{\sigma_1, \dots, \sigma_n\}$ . Let  $w: \mathcal{K}_d \rightarrow \mathbb{R}$  be a weight function on the  $d$ -simplices. We define a span program over the strings  $\{0, 1\}^n$  in the following way.

1.  $\mathcal{H} = C_d(\mathcal{K})$ , with  $\mathcal{H}_{i,1} = \text{span}\{|\sigma_i\rangle\}$  and  $\mathcal{H}_{i,0} = \{0\}$ .
2.  $\mathcal{U} = C_{d-1}(\mathcal{K})$
3.  $A = \partial_d \sqrt{W}: C_d(\mathcal{K}) \rightarrow C_{d-1}(\mathcal{K})$
4.  $|\tau\rangle = \gamma$

## 31:12 Effective Resistance and Capacitance in Simplicial Complexes

We denote the above span program by  $\mathcal{P}_{\mathcal{K}}$ . Let  $x \in \{0, 1\}^N$  be a binary string. We define the subcomplex  $\mathcal{K}(x) := \mathcal{K}^{d-1} \cup \{\sigma_i : x_i = 1\}$ . That is,  $\mathcal{K}(x)$  contains the  $d$ -simplices  $\sigma_i$  such that  $x_i = 1$ . There exists a solution to the linear system  $\partial_d \sqrt{W} \Pi_{\mathcal{K}(x)} |v\rangle = |\gamma\rangle$  if and only if the cycle  $|\gamma\rangle$  is null-homologous in  $\mathcal{K}(x)$  if and only if  $x$  is a positive instance of  $\mathcal{P}_{\mathcal{K}}$ . The span program  $\mathcal{P}_{\mathcal{K}}$  decides the boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  where  $f(x) = 1$  if and only if  $\gamma$  is a null-homologous cycle in the subcomplex  $\mathcal{K}(x)$ .

Given a string  $x \in \{0, 1\}^n$  we show in the following two lemmas that  $w_+(x, \mathcal{P}_{\mathcal{K}}) = \mathcal{R}_{\gamma}(\mathcal{K}(x))$  and  $w_-(x, \mathcal{P}_{\mathcal{K}}) = \mathcal{C}_{\gamma}(\mathcal{K}(x))$ . The proofs are simple calculations following from the definitions of effective resistance and capacitance.

► **Lemma 21.** *Let  $x \in \{0, 1\}^N$  be a positive instance. There is a bijection between positive witnesses  $|w\rangle$  for  $x$  and unit  $\gamma$ -flows  $|f\rangle$  in  $\mathcal{K}(x)$ . Moreover, the positive witness size is equal to the effective resistance of  $\gamma$  in  $\mathcal{K}(x)$ ; that is,  $w_+(x, \mathcal{P}_{\mathcal{K}}) = \mathcal{R}_{\gamma}(\mathcal{K}(x))$ .*

**Proof.** Let  $|w\rangle \in C_d(\mathcal{K})$  be a positive witness for  $x$ , so  $\partial_d \sqrt{W} |w\rangle = |\gamma\rangle$ . We construct a unit  $\gamma$ -flow  $|f\rangle$  in  $\mathcal{K}(x)$  by  $|f\rangle = \sqrt{W} |w\rangle$ ;  $|f\rangle$  is indeed a unit  $\gamma$ -flow as  $\partial_d |f\rangle = \partial_d \sqrt{W} |w\rangle = |\gamma\rangle$ . Moreover,  $|w\rangle = W^{-1/2} |f\rangle$ . The flow energy of  $\gamma$  is  $J(f) = \langle f | W^{-1} |f\rangle = \langle W^{-1/2} f | W^{-1/2} f \rangle = \langle w | w \rangle = \| |w\rangle \|^2$ . Hence, the flow energy of  $|f\rangle$  equals the witness size of  $|w\rangle$ . Conversely, let  $|f\rangle$  be a unit  $\gamma$ -flow in  $\mathcal{K}(x)$  and define the positive witness for  $x$  as  $|w\rangle = W^{-1/2} |f\rangle$ . The same computation in the above paragraph shows that the flow energy of  $f$  equals the positive witness size of  $|w\rangle$ . ◀

► **Lemma 22.** *Let  $x \in \{0, 1\}^N$  be a negative instance. There is a bijection between negative witnesses  $\langle w |$  for  $x$  and unit  $\gamma$ -potentials  $\langle p |$  in  $\mathcal{K}(x)$ . Moreover, the negative witness size is equal to the effective capacitance of  $\gamma$  in  $\mathcal{K}(x)$ ; that is,  $w_-(x, \mathcal{P}_{\mathcal{K}}) = \mathcal{C}_{\gamma}(\mathcal{K}(x))$ .*

**Proof.** Let  $\langle w |$  be a negative witness for  $x$ . We will verify that  $\langle w |$  is a unit  $\gamma$ -potential. We have by the definition of negative witness that  $\langle w | \gamma \rangle = 1$ . We must show that the coboundary of  $\langle w |$  is zero in  $\mathcal{K}(x)$ . By the definition of a negative witness we have  $\langle w | \partial_d \sqrt{W} \Pi_{\mathcal{K}(x)} = 0$ . Since  $\sqrt{W}$  is a diagonal matrix and  $\Pi_{\mathcal{K}(x)}$  restricts the coboundary to the subcomplex  $\mathcal{K}(x)$ , we see that  $\langle w | \partial_d \sigma = 0$  for any  $\sigma \in \mathcal{K}(x)_d$ . To show that the witness size of  $\langle w |$  is equal to the potential energy, we have

$$\begin{aligned} \|\langle w | \partial_d \sqrt{W}\|^2 &= \langle w | \partial_d \sqrt{W} \sqrt{W} \partial_d^T |w\rangle \\ &= \langle \delta_{d-1} w | W | \delta_{d-1} w \rangle \\ &= \mathcal{J}(w). \end{aligned}$$

Conversely, let  $\langle p |$  be a unit  $\gamma$ -potential for  $\mathcal{K}(x)$ . We will prove that  $\langle p |$  is a negative witness for  $\gamma$ . Since the coboundary of  $\langle p |$  is zero in  $\mathcal{K}(x)$  we have  $\langle \delta p | \sigma \rangle = 0$  for each  $\sigma \in \mathcal{K}(x)_d$ , which implies  $\langle p | \partial_d \sqrt{W} \Pi_{\mathcal{K}(x)} = 0$  by the reasoning in the previous paragraph. Also by the previous paragraph, we have that the potential energy of  $\langle p |$  is equal to the negative witness size of  $\langle p |$  which concludes the proof. ◀

From these two lemmas we obtain the main theorem of the section, the quantum query complexity of the span program.

► **Theorem 23.** *Given a  $d$ -dimensional simplicial complex  $\mathcal{K}$ , a  $(d-1)$ -dimensional cycle  $\gamma$  that is null-homologous in  $\mathcal{K}$ , and a  $d$ -dimensional subcomplex  $\mathcal{K}(x) \subseteq \mathcal{K}$ , there exists a quantum algorithm deciding whether or not  $\gamma$  is null-homologous in  $\mathcal{K}(x)$  whose quantum query complexity is  $O\left(\sqrt{\mathcal{R}_{\max}(\gamma) \mathcal{C}_{\max}(\gamma)}\right)$ , where  $\mathcal{R}_{\max}$  is the maximum effective resistance of  $\gamma$  in any subcomplex  $\mathcal{K}(y)$  and  $\mathcal{C}_{\max}$  is the maximum effective capacitance of  $\gamma$  in any subcomplex  $\mathcal{K}(y)$ .*

**Proof.** By Theorem 20, the span program  $\mathcal{P}_{\mathcal{K}}$  can be converted into a quantum algorithm whose query complexity is  $O\left(\sqrt{W_+(f, \mathcal{P}_{\mathcal{K}})W_-(f, \mathcal{P}_{\mathcal{K}})}\right)$  where  $W_+(f, \mathcal{P}_{\mathcal{K}}) = \max_{x \in f^{-1}(1)} \mathcal{R}_{\gamma}(\mathcal{K}(x)) = \mathcal{R}_{\max}(\gamma)$  and  $W_-(f, \mathcal{P}_{\mathcal{K}}) = \max_{x \in f^{-1}(0)} \mathcal{C}_{\gamma}(\mathcal{K}(x)) = \mathcal{C}_{\max}(\gamma)$ . ◀

By Theorems 14 and 16 we obtain an upper bound on the query complexity parameterized by the number of simplices and the cardinality of the torsion subgroups of the relative homology groups.

► **Theorem 24.** *Let  $\mathcal{K}$  be an unweighted  $d$ -dimensional simplicial complex and  $\mathcal{K}(x)$  a  $d$ -dimensional subcomplex  $\mathcal{K}(x) \subseteq \mathcal{K}$ . Let  $\hat{\gamma} \in C_{d-1}(\mathcal{K})$  be a  $(d-1)$ -cycle such that  $\hat{\gamma}$  is integral and each of the coefficients  $|\hat{\gamma}_i| = O(1)$ . There exists a quantum algorithm deciding whether or not  $\gamma := \hat{\gamma}/\|\hat{\gamma}\|$  is null-homologous in  $\mathcal{K}(x)$  whose quantum query complexity is  $O(n^{5/2} \cdot d^{1/2} \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ .*

Finally, we state the query complexity under some assumptions that arise in the case of  $st$ -connectivity in graphs. In this case, the input cycle is  $|t\rangle - |s\rangle$ , and the support of  $|t\rangle - |s\rangle$  is equal to 2. A factor of  $n$  in both the upper bounds on resistance and capacitance is actually a factor of  $|\text{supp}(\gamma)|$  as seen in the proofs of these bounds. Under the assumption that the support of  $\gamma$  is bounded above by  $O(d)$ , we can replace a factor of  $n$  from both the flow energy and potential energy of any unit  $\gamma$ -flow and unit  $\gamma$ -potential with a factor of  $d$ . This assumption on the size of  $\text{supp}(\gamma)$  is true when  $\gamma$  is the boundary of a  $d$ -simplex.

Furthermore, graphs do not contain relative torsion<sup>4</sup>, so we make the additional assumption that  $\mathcal{K}$  is relative torsion-free. Under these assumptions our query complexity matches the query complexity arising from the span program deciding  $st$ -connectivity.

► **Corollary 25.** *Let  $\mathcal{K}$  be an unweighted  $d$ -dimensional simplicial complex and  $\mathcal{K}(x) \subseteq \mathcal{K}$  be a  $d$ -dimensional subcomplex. Let  $\hat{\gamma} \in C_{d-1}(\mathcal{K})$  be a  $(d-1)$ -cycle such that  $\hat{\gamma}$  is integral and each of the coefficients  $|\hat{\gamma}_i| = O(1)$ . Further assume that  $\mathcal{K}$  is relative torsion-free and  $|\text{supp}(\gamma)| = O(d)$ . There exists a quantum algorithm deciding whether or not  $\gamma := \hat{\gamma}/\|\hat{\gamma}\|$  is null-homologous in  $\mathcal{K}(x)$  whose quantum query complexity is  $O((dn)^{3/2})$ .*

### 6.3 Time efficient implementations

We have given bounds on the query complexity of null-homology testing; however, this does not imply a bound on the time complexity of evaluating this span program. There are two obstacles to a time-efficient implementation of the span program: the weights and the input cycle  $\gamma$ . The weights on the  $d$ -simplices make it difficult to implement the matrix  $\partial\sqrt{W}$ , as the weights on the simplices can be arbitrary real numbers. The input cycle  $\gamma$  is difficult to create on a quantum computer for the same reason, as the entries of  $\gamma$  can also be arbitrary real numbers. We explore the implementation details of this algorithm in the appendix.

We can give a quantum algorithm of bounded time complexity in one particular instance: when  $\mathcal{K}$  is unweighted and  $\gamma$  is the boundary of a  $d$ -simplex. We do not require said  $d$ -simplex to actually appear in the complex. The time complexity of this case is given in the following theorem.

<sup>4</sup> Graphs do not have relative torsion as the boundary matrix of a graph  $\partial_1[G]$  is *totally unimodular*. See Section 5.1 of the paper [5] for an explanation of the relationship between total unimodularity of the boundary matrix and relative torsion of a simplicial complex.

► **Theorem 26.** Let  $\mathcal{K}$  be a simplicial complex,  $\gamma \in C_{d-1}(\mathcal{K})$  a null-homologous cycle, and  $\mathcal{K}(x) \subset \mathcal{K}$  be a simplicial complex. Furthermore, assume that  $\gamma$  is the boundary of a  $d$ -simplex and the complex is unweighted. There is a quantum algorithm for deciding if  $\gamma$  is null-homologous in  $\mathcal{K}(x)$  that runs in time  $\tilde{O}\left(\frac{(dn)^{3/2} \cdot \mathcal{T}_{\max}(\mathcal{K})^2}{\sqrt{\lambda}} \left(\sqrt{d} + \sqrt{d_{\max}}\right) + \left(\sqrt{\frac{1}{\mathcal{R}_\gamma(\mathcal{K})}} + \sqrt{\mathcal{R}_\gamma(\mathcal{K})}\right) \left(\sqrt{d} + \sqrt{d_{\max}}\right)\right)$  where  $d_{\max}$  is the maximum degree of a  $(d-1)$ -simplex in  $\mathcal{K}$  and  $\lambda$  is the smallest eigenvalue of the normalized up-Laplacian.

The factor of  $\tilde{O}\left(\left(\sqrt{\frac{1}{\mathcal{R}_\gamma(\mathcal{K})}} + \sqrt{\mathcal{R}_\gamma(\mathcal{K})}\right) + \left(\sqrt{d} + \sqrt{d_{\max}}\right)\right)$  comes from the time necessary to create the initial state to the algorithm. Assuming such a state is already provided the running time reduces to  $\tilde{O}\left(\frac{(dn)^{3/2} \cdot \mathcal{T}_{\max}(\mathcal{K})^2}{\sqrt{\lambda}} \left(\sqrt{d} + \sqrt{d_{\max}}\right)\right)$ .

We can get a tighter analysis when  $\mathcal{K}$  is a pseudomanifold. When  $\mathcal{K}$  is a pseudomanifold, the maximum degree  $d_{\max} \leq 2$  and the running time reduces to  $\tilde{O}\left(\frac{d^2 \cdot n^{3/2} \cdot \mathcal{T}_{\max}(\mathcal{K})^2}{\sqrt{\lambda}}\right)$ . Steenbergen, Klivans, and Mukherjee provide a lower bound on  $\lambda$  for pseudomanifolds similar to the Cheeger inequality for graphs [28]. Their lower bound is in terms of the **boundary expansion** of the complex which is defined as  $h_d := \min_{\phi \in C_d(\mathbb{Z}_2)} \frac{\|\partial_d \phi\|}{\min_{\psi \in \text{im } \partial_{d+1}} \|\phi + \psi\|}$ . For pseudomanifolds they prove a lower bound of  $\lambda \geq \frac{h_d^2}{2(d+1)}$ .

Finally, in the case of graphs there is no relative torsion and we can state the running time as  $\tilde{O}\left(\frac{n^{3/2}}{\sqrt{\lambda}}\right)$  where  $\lambda$  is bounded below by the well-known Cheeger inequality.

---

## References

- 1 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for  $st$ -connectivity and claw detection. In *Algorithms – ESA 2012*, pages 193–204. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-33090-2\_18.
- 2 Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Info. Comput.*, 18(1–2):18–50, February 2018.
- 3 A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky. The electrical resistance of a graph captures its commute and cover times. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, pages 574–586, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/73007.73062.
- 4 Cecil Jose A. Delfinado and Herbert Edelsbrunner. An incremental algorithm for betti numbers of simplicial complexes. In *Proceedings of the Ninth Annual Symposium on Computational Geometry*, SCG '93, pages 232–239, New York, NY, USA, 1993. Association for Computing Machinery. doi:10.1145/160985.161140.
- 5 Tamal K. Dey, Anil N. Hirani, and Bala Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM J. Comput.*, 40(4):1026–1044, July 2011. doi:10.1137/100800245.
- 6 Tamal K. Dey, Tao Hou, and Sayan Mandal. Computing minimal persistent cycles: Polynomial and hard cases. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, pages 2587–2606, USA, 2020. Society for Industrial and Applied Mathematics.
- 7 Christoph Dürr, Mark Heiligman, Peter Hoyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, January 2006. doi:10.1137/050644719.
- 8 Timothy Goldberg. Combinatorial laplacians of simplicial complexes. B.S. Thesis, Bard College, Annandale-on-Hudson, NY, 2002.

- 9 Jakob Hansen and Robert Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 3(4):315–358, August 2019. doi:10.1007/s41468-019-00038-7.
- 10 Allen Hatcher. *Algebraic topology*. Cambridge Univ. Press, Cambridge, 2000. URL: <https://cds.cern.ch/record/478079>.
- 11 Tsuyoshi Ito and Stacey Jeffery. Approximate span programs. *Algorithmica*, 81(6):2158–2195, November 2018. doi:10.1007/s00453-018-0527-1.
- 12 Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithms for connectivity and related problems. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2018.49.
- 13 Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, August 2017. doi:10.22331/q-2017-08-17-26.
- 14 M. Karchmer and A. Wigderson. On span programs. In *[1993] Proceedings of the Eighth Annual Structure in Complexity Theory Conference*. IEEE Comput. Soc. Press, 1993. doi:10.1109/sct.1993.336536.
- 15 G. Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148:497–508, 1847.
- 16 A. Yu. Kitaev. Quantum measurements and the Abelian Stabilizer Problem. *arXiv e-prints*, pages quant-ph/9511026, November 1995. arXiv:quant-ph/9511026.
- 17 D. J. Klein and M. Randić. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, December 1993. doi:10.1007/BF01164627.
- 18 Woong Kook and Kang-Ju Lee. Simplicial networks and effective resistance. *Advances in Applied Mathematics*, 100:71–86, 2018. doi:10.1016/j.aam.2018.05.004.
- 19 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, January 2011. doi:10.1137/090745854.
- 20 William Maxwell and Amir Nayyeri. Generalized max-flows and min-cuts in simplicial complexes. *CoRR*, abs/2106.14116, 2021. arXiv:2106.14116.
- 21 James R. Munkres. *Elements of Algebraic Topology*. CRC Press, March 1984. doi:10.1201/9780429493911.
- 22 Andrew Newman. Small simplicial complexes with prescribed torsion in homology. *Discrete & Computational Geometry*, 62(2):433–460, March 2018. doi:10.1007/s00454-018-9987-y.
- 23 Braxton Osting, Sourabh Palande, and Bei Wang. Spectral sparsification of simplicial complexes for clustering and label propagation, 2019. arXiv:1708.08436.
- 24 Ben Reichardt and Robert Spalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8(1):291–319, 2012. doi:10.4086/toc.2012.v008a013.
- 25 Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, October 2009. doi:10.1109/focs.2009.55.
- 26 Daniel Spielman. Spectral and algebraic graph theory, 2019.
- 27 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 563–568, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374456.
- 28 John Steenbergen, Caroline Klivans, and Sayan Mukherjee. A cheeger-type inequality on simplicial complexes. *Advances in Applied Mathematics*, 56:56–77, 2014. doi:10.1016/j.aam.2014.01.002.
- 29 Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 32–41, November 2004. doi:10.1109/FOCS.2004.53.



## A

 Evaluating the span program for null-homology

In this section, we give a quantum algorithm for evaluating the null-homology span program. Our algorithm is inspired by the quantum algorithm for evaluating  $st$ -connectivity span program in graphs. The first quantum algorithm for evaluating the  $st$ -connectivity span program was given by Belovsz and Reichardt in [1]; however, we follow the slightly different algorithm introduced by Ito and Jefferies in [11]. We are also greatly indebted to the presentation of this algorithm given by Jeffery and Kimmel in [13].

The algorithm for evaluating a general span program  $\mathcal{P} = (\mathcal{H}, \mathcal{U}, |\tau\rangle, A)$  is to perform phase estimation of the vector  $|w_0\rangle := A^+|\tau\rangle$  on the unitary operator  $U = R_{\mathcal{H}(x)}R_{\ker A}$  where the notation  $R_S$  denotes the reflection about the subspace  $S$ . (The unitary  $R_S = 2\Pi_S - I$ , where  $\Pi_S$  is the projection onto  $S$ .) Intuitively, if  $x$  is a positive instance, then  $|w_0\rangle$  will be close to an eigenvector of  $U$  with phase 0. If  $x$  is a negative instance, then  $|w_0\rangle$  will be far from any eigenvector of  $U$  of phase 0. If we want to evaluate the function  $f : D \rightarrow \{0, 1\}$ , we need to perform phase estimation to precision  $O\left(1/\sqrt{W_-(f, \mathcal{P})W_+(f, \mathcal{P})}\right)$ . The algorithm for phase estimation of a unitary  $U$  to precision  $O(\delta)$  performs  $O(1/\delta)$  implementations of the unitary  $U$  [16], so the algorithm for evaluating the span program  $\mathcal{P} = (\mathcal{H}, \mathcal{U}, |\tau\rangle, A)$  requires  $O\left(\sqrt{W_-(f, \mathcal{P})W_+(f, \mathcal{P})}\right)$  implementations of  $U$ .

We now analyze the time complexity of implementing the unitary  $U$ . The reflection  $R_{\mathcal{H}(X)}$  can be implemented with one query to  $\mathcal{O}_x$ . This reflection is the same as the reflection across the good states in Grover's Algorithm. The rest of this section is devoted to an implementation of  $R_{\ker \partial}$ .

Recall that  $\ker \partial_d \subset C_d(\mathcal{K})$ . The idea behind the implementation of  $R_{\ker \partial}$  is that instead of reflecting across  $\ker \partial_d$  directly, we can embed  $C_d(\mathcal{K})$  into  $C_{d-1}(\mathcal{K}) \otimes C_d(\mathcal{K})$  by sending  $|\tau\rangle \rightarrow c|\partial\tau\rangle|\tau\rangle$  (where  $c$  is a normalization constant). We can then implement the reflection  $R_{\ker \partial}$  by implementing a series of "local reflections" on the basis  $|\partial\tau\rangle|\tau\rangle$ .

We consider two subspaces  $B$  and  $C$  of  $C_{d-1}(\mathcal{K}) \otimes C_d(\mathcal{K})$ . The spaces  $B$  and  $C$  are defined:

$$B = \text{span} \left\{ |b_\tau\rangle := \frac{1}{\sqrt{d+1}} |\partial\tau\rangle|\tau\rangle : \tau \in \mathcal{K}_d \right\}$$

and

$$C = \text{span} \left\{ |c_\sigma\rangle := \sum_{\sigma \subset \tau} \sqrt{\frac{w(\sigma)}{\deg(\sigma)}} |\sigma\rangle|\tau\rangle : \sigma \in \mathcal{K}_{d-1} \right\}.$$

The space  $C_{d-1}(\mathcal{K}) \otimes C_d(\mathcal{K})$  has basis  $\{|\sigma\rangle|\tau\rangle \mid \sigma \in \mathcal{K}_{d-1}, \tau \in \mathcal{K}_d\}$ . The vector  $|b_\tau\rangle$  is non-zero on a basis element  $|\sigma\rangle|\tau\rangle$  if and only if  $\sigma$  is on the boundary of  $\tau$ . Similarly, a component of  $|c_\sigma\rangle$  is non-zero on  $|\sigma\rangle|\tau\rangle$  if and only if  $\tau$  is on the coboundary of  $\sigma$ . The vector  $|b_\tau\rangle$  can be thought of as being *like* the boundary of  $\tau$ , with the additional property that the set  $\{|b_\tau\rangle \mid \tau \in C_d(\mathcal{K})\}$  is orthonormal. Similarly, the vector  $|c_\sigma\rangle$  is *like* the coboundary of  $\sigma$  but orthonormal.

We also define operators that embed  $C_d(\mathcal{K})$  and  $C_{d-1}(\mathcal{K})$  into  $B$  and  $C$  respectively. We define linear operators  $M_B : C_d(\mathcal{K}) \rightarrow B$  and  $M_C : C_{d-1}(\mathcal{K}) \rightarrow C$  as follows:

$$M_B := \sum_{\tau \in \mathcal{K}_d} |b_\tau\rangle\langle\tau|,$$

and

$$M_C := \sum_{\sigma \in \mathcal{K}_{d-1}} |c_\sigma\rangle\langle\sigma|.$$

As the columns of  $M_B$  and  $M_C$  are orthonormal, both operators are isometries.



We introduce the matrices  $M_C$  and  $M_B$  as they have the property that  $\ker M_C^\dagger M_B = \ker \partial$ , which we prove in the follow lemma. This fact will give us a way to implement  $R_{\ker \partial}$ .

► **Lemma 27.**  $\ker M_C^\dagger M_B = \ker \partial$ .

**Proof.** We first calculate the matrix  $M_C^\dagger M_B$ . We then argue that  $\ker M_C^\dagger M_B = \ker \partial$ . For a  $(d-1)$ -simplex  $\sigma$  and a  $d$ -simplex  $\tau$ , we have that

$$\langle c_\sigma | b_\tau \rangle = \sum_{\tau' \in \mathcal{K}_d: \sigma \subset \tau'} \frac{w(\tau)}{\sqrt{\deg(\sigma)}} \langle \sigma | \partial \tau \rangle \langle \tau' | \tau \rangle = \begin{cases} \sqrt{\frac{w(\tau)}{(d+1)\deg(\sigma)}} \langle \sigma | \partial \tau \rangle & \text{if } \sigma \subset \tau \\ 0 & \text{otherwise} \end{cases}.$$

So  $\langle c_\sigma | b_\tau \rangle$  is non-zero if and only if  $\sigma$  is in the boundary of  $\tau$ . We use this to calculate the product  $M_C^\dagger M_B$ :

$$\begin{aligned} M_C^\dagger M_B &= \sum_{\sigma \in \mathcal{K}_{d-1}} \sum_{\tau \in \mathcal{K}_d} |\sigma\rangle \langle c_\sigma | b_\tau \rangle \langle \tau | \\ &= \frac{1}{\sqrt{(d+1)}} \sum_{\sigma \subset \tau} \frac{\sqrt{w(\tau)} \langle \sigma | \partial \tau \rangle}{\sqrt{\deg(\sigma)}} |\sigma\rangle \langle \tau | \\ &= \frac{1}{\sqrt{(d+1)}} \left( \sum_{\sigma \in \mathcal{K}_{d-1}(\mathcal{K})} \frac{|\sigma\rangle \langle \sigma |}{\sqrt{\deg(\sigma)}} \right) \sum_{\tau \in \mathcal{K}_d} \sqrt{w(\tau)} |\partial \tau\rangle \langle \tau | \\ &= \frac{1}{\sqrt{(d+1)}} \left( \sum_{\sigma \in \mathcal{K}_{d-1}(\mathcal{K})} \frac{|\sigma\rangle \langle \sigma |}{\sqrt{\deg(\sigma)}} \right) \partial \sqrt{W} =: \hat{\partial}. \end{aligned}$$

The term  $\frac{|\sigma\rangle \langle \sigma |}{\sqrt{\deg(\sigma)}}$  is all-zeros matrix except for the  $(\sigma, \sigma)$ -entry, which is  $\frac{1}{\sqrt{\deg(\sigma)}}$ . The sum  $\sum_{\sigma \in \mathcal{K}_{d-1}} \frac{|\sigma\rangle \langle \sigma |}{\sqrt{\deg(\sigma)}}$  is a diagonal matrix. Accordingly, the matrix  $\hat{\partial}$  is  $\partial \sqrt{W}$  with each row scaled. Scaling the rows of a matrix does not change its row space or kernel, so  $\ker M_C^\dagger M_B = \ker \partial$ . ◀

The spaces  $B$  and  $C$  and the matrices  $M_B$  and  $M_C$  are inspired by the follow lemma of Szegedy which is necessary for implementing  $R_{\ker \partial}$ .

► **Lemma 28** (Szegedy [29], Theorem 1). *Let  $M_B$  and  $M_C$  be matrices with the same number of rows and orthonormal columns, and let  $B = \text{span } M_B$  and  $C = \text{span } M_C$ . The matrix  $M_C^\dagger M_B$  has singular values at most 1. Let  $\cos \theta_1, \dots, \cos \theta_k$  be the singular values of  $M_C^\dagger M_B$  in the range  $(0, 1)$ . Let  $U = R_C R_B$ . We can decompose the eigenspaces of  $U$  as*

- The  $(+1)$ -eigenspace of  $U$  is  $(B \cap C) \oplus (B^\perp \cap C^\perp)$ .
- The  $(-1)$ -eigenspace of  $U$  is  $(B \cap C^\perp) \oplus (B^\perp \cap C)$ .
- The remaining eigenvalues of  $U$  are  $e^{\pm 2i\theta_j}$  for  $1 \leq j \leq k$ .

The following lemma gives us a way to implment the  $R_{\ker \partial}$ . Let  $R_{U^-}$  be the rotation about  $(-1)$ -eigenspace of  $U$ , and let  $V = M_B^\dagger R_{U^-} M_B$ . The matrix  $V$  embeds  $C_d(\mathcal{K})$  into  $B$  with  $M_B$ , performs a reflection on  $B$  about the  $(-1)$ -eigenspace of  $U$ , and unembeds with  $M_B^\dagger$ . The following lemma proves that  $V = R_{\ker \partial}$ .

► **Lemma 29.** *The matrix  $V = M_B^\dagger R_{U^-} M_B$  satisfies the equality  $V = R_{\ker \partial}$ .*

**Proof.** We first verify that  $V$  is a reflection; that is, we show the eigenvalues of  $V$  are 1 and  $-1$ . The matrices  $M_B$  and  $M_C$  have orthonormal columns, so we can use Lemma 28 to characterize the eigenspaces of  $U$ . The  $(-1)$ -eigenspace of  $U$  is  $(B \cap C^\perp) \oplus (B^\perp \cap C)$  and the  $(+1)$ -eigenspace of  $U$  is  $(B \cap C) \cap (B^\perp \cap C^\perp)$ . As the spaces  $(B \cap C)$  and  $(B \cap C^\perp)$  span  $B$ ,

then  $R_{U^-}$  restricted to  $B$  has eigenvalues 1 and  $-1$ . As  $B = \text{im } M_B$  and  $V = M_B^\dagger R_{U^-} M_B$ , then we conclude that  $V$  has eigenvalues 1 and  $-1$  as well.

Now that we have determined that  $V$  is a reflection, we need to determine which subspace  $V$  reflects across. A corollary of the previous paragraph is that a vector  $|\psi\rangle \in C_d(\mathcal{K})$  is in the  $(+1)$ -eigenspace of  $V$  if and only if  $M_B|\psi\rangle$  is in the  $(-1)$ -eigenspace of  $U$ . Specifically, a vector  $|\psi\rangle$  is in the  $(+1)$ -eigenspace of  $V$  if and only if  $M_B|\psi\rangle \in C^\perp$ . As  $C^\perp = \ker M_C^\dagger$ , the vector  $|\psi\rangle$  is in the  $(+1)$ -eigenspace of  $V$  if and only if  $|\psi\rangle \in \ker M_C^\dagger M_B$ . We proved in Lemma 27 that  $\ker M_C^\dagger M_B = \ker \partial$ , so we conclude that  $V = R_{\ker \partial}$   $\blacktriangleleft$

We have a matrix  $V$  that implements  $R_{\ker \partial}$ ; next, we analyze the complexity of implementing  $V$ . We start by analyzing the complexity of implementing  $R_{U^-}$ , the reflection across the  $(-1)$ -eigenspace of  $U$ .

We implement the reflection around the  $(-1)$ -eigenspace of  $U$  using phase estimation, an algorithm introduced by Magniez et al. [19]. The algorithm is as follows. We first estimate the phase of  $U$  to some degree of accuracy to be specified shortly. Intuitively, we need to estimate the phase of  $U$  to high enough accuracy to distinguish between  $-1$  eigenvalues of  $U$  and eigenvalues of  $U$  close to  $-1$ . We then perform a reflection controlled on the estimated phase.

The **phase gap** of a unitary  $U$  with eigenvalues  $\{e^{i\theta_1}, \dots, e^{i\theta_k}\}$  is  $\min\{|\theta_i| : \theta_i \neq 0\}$ . The following lemma shows that the phase gap determines the complexity of reflecting across the 1-eigenspace of  $U$ .

► **Lemma 30** (Magniez et al. [19], Paraphrase of Theorem 6). *Let  $U$  be a unitary with phase gap  $\theta$ . A reflection around the 1-eigenspace of  $U$  can be performed to constant precision with  $O(\frac{1}{\theta})$  applications of  $U$ .*

The phase gap measures gap between the 1-eigenspace of a unitary and all other eigenvalues. We are interested in the gap in phase between the  $(-1)$ -eigenspace of  $U$  and the other eigenvalues of  $U$ . This is precisely the phase gap of  $-U$ . The following lemma analyzes the phase gap of  $-U$  and gives the complexity of reflecting about the  $(-1)$ -eigenspace of  $U$ .

► **Lemma 31.** *We can implement  $R_{U^-}$  with  $O\left(\sqrt{\frac{d+1}{\lambda}}\right)$  calls to  $U$ , where  $\lambda$  is the smallest non-zero eigenvalue of the normalized up-Laplacian.*

**Proof.** We need to calculate the phase gap of  $-U$  to determine the precision to which we need to estimate the phase of  $U$ . Observe that if  $\theta$  is the phase of an eigenvalue of  $U$ , then  $\theta + \pi$  is the phase of an eigenvalue of  $-U$ . We can bound the phase gap of  $-U$  using Lemma 28. The non-zero eigenvalues of  $U$  are  $\{e^{\pm i2\theta_j}\}_j$ , where  $\{\cos \theta_j\}_j$  were the singular values of  $M_C^\dagger M_B$ . Therefore, the phases of  $-U$  are  $\{\pm|\pi - 2\theta_j|\}_j$ . Using the inequality that  $\pi/2 - \theta_j \geq \cos \theta_j$  for  $\theta_j \in [0, \pi/2]$ , then the phase gap of  $-U$  is bounded below by

$$|\pi - 2\theta_j| \geq 2 \cos \theta_j \geq 2 \cdot \sigma_{\min}(M_C^\dagger M_B)$$

where  $\sigma_{\min}(M_C^\dagger M_B)$  is the smallest singular value of  $M_C^\dagger M_B$ .

We can actually relate the smallest singular value of  $M_C^\dagger M_B$  to something more meaningful. By the proof of Lemma 27, the matrix  $M_C^\dagger M_B = \frac{1}{\sqrt{d+1}} D^{-1/2} \partial \sqrt{W}$ , where  $D$  is the diagonal matrix with the degrees of the  $(d-1)$ -simplices on the diagonal. Thus,  $(M_C^\dagger M_B)(M_C^\dagger M_B)^\dagger = \frac{1}{d+1} D^{-1/2} \partial W \delta D^{-1/2} = \frac{1}{d+1} D^{-1/2} L D^{-1/2}$ . Recall from Section 2 that the matrix  $D^{-1/2} L D^{-1/2}$  is the normalized up-Laplacian. The singular values of a matrix  $A$  are the square roots of the eigenvalues of  $AA^T$ . Thus, the smallest singular value of  $M_C^\dagger M_B$ ,

and the phase gap of  $-U$ , is  $\Omega\left(\sqrt{\frac{\lambda}{d+1}}\right)$ , where  $\lambda$  is the smallest eigenvalue of  $\Delta$ . Therefore, by Lemma 30, we can implement  $R_{U^-}$  with  $O\left(\sqrt{\frac{d+1}{\lambda}}\right)$  calls to  $U$ . ◀

We are almost ready to give the running time for  $V = R_{\ker \partial}$ , but first, we need to make a delicate distinction. The matrices  $M_B$  and  $M_C$  have orthonormal columns, but they are *not* unitary. We can see this as  $\ker M_B^\dagger \neq 0$  and  $\ker M_C^\dagger \neq 0$ . As  $M_B$  and  $M_C$  are not unitary, they cannot be implemented on a quantum computer. Fortunately, it suffices to implement unitaries  $U_B$  and  $U_C$  such that  $U_B|_{C_d(\mathcal{K})} = M_B$  and  $U_C|_{C_{d-1}(\mathcal{K})} = M_C$ . Now we can give the running time for  $V = R_{\ker \partial}$ .

► **Lemma 32.** *There is an algorithm to perform  $R_{\ker \partial}$  in time  $\tilde{O}\left(\sqrt{\frac{d+1}{\lambda}}(T_B + T_C)\right)$ , where  $T_B$  and  $T_C$  are the times to perform  $U_B$  and  $U_C$  respectively.*

**Proof.** Lemma 29 shows that  $R_{\ker \partial} = V = M_B^\dagger R_{U^-} M_B$ . We can equivalently run  $U_B^\dagger R_{U^-} U_B$ . As  $U_B$  takes  $T_B$  by definition, we only need to show we can implement  $R_{U^-}$  in  $\tilde{O}\left(\sqrt{(d+1)/\lambda}(T_B + T_C)\right)$  time. Lemma 31 shows we can implement  $R_{U^-}$  with  $O(\sqrt{(d+1)/\lambda})$  calls to  $U$ , so we need to show we can implement  $U$  in  $\tilde{O}(T_B + T_C)$ . The unitary  $U = R_C R_B$ , and we claim we can implement  $R_C$  and  $R_B$  in  $\tilde{O}(T_B)$  and  $\tilde{O}(T_C)$  respectively. We can implement  $R_B$  as  $U_B R_{\mathcal{K}_d} U_B^\dagger$ , where  $R_{\mathcal{K}_d}$  reflects across the basis states  $\{|0\rangle|\sigma\rangle \mid \sigma \in \mathcal{K}_d\}$ . We can check if a quantum state is of the form  $|0\rangle|\sigma\rangle$  in  $O(\log n_d)$  gates (specifically, by checking if the basis state is within a certain range), so the unitary  $R_{\mathcal{K}_d}$  takes  $O(\log n_d)$  gates, and  $R_B$  takes  $\tilde{O}(T_B)$  time. The unitary  $R_C$  takes  $\tilde{O}(T_C)$  time by the same argument. ◀

The running time  $T_B$  is dependent on how the boundary maps are loaded into the quantum algorithm. We propose a method of storing the boundary maps in a quantum computer called the **incidence array**. The incidence array is adapted from the *adjacency array* introduced by Durr et al. in [7] to store the adjacency between pairs of vertices in a graph.

For a  $d$ -simplex  $\tau = \{v_0, \dots, v_d\}$ , the **down-incidence array** is the function  $g : |\tau\rangle|j\rangle|0\rangle \rightarrow |\tau\rangle|j\rangle|\tau \setminus \{v_j\}\rangle$  for  $0 \leq j \leq d$ . The simplices in the boundary of  $\tau$  have alternating sign. To address this, we also perform a negation conditioned on the parity of  $|j\rangle$  to compute  $(-1)^j |\tau\rangle|j\rangle|\tau \setminus \{v_j\}\rangle$ .

Durr et al. [7] claim that queries to the incidence array can be performed in logarithmic time. As the down-incidence array is identical to the adjacency array<sup>5</sup>, queries to the down-incidence also take logarithmic time. We can compute the state  $|\partial\tau\rangle|\tau\rangle$  with the down-incidence array and the following lemma.

► **Lemma 33** (Cade, Montanaro, Belovs [2], Implicit in the proof of Lemma 2). *Let  $f : [m] \rightarrow [k]$  be a function, and let  $\mathcal{O}_f$  be an oracle that computes  $\mathcal{O}_f : |i\rangle|0\rangle \rightarrow |i\rangle|f(i)\rangle$ . The state  $\frac{1}{\sqrt{m}} \sum_{i=1}^m |f(i)\rangle$  can be computed with  $O(\sqrt{m})$  queries to  $\mathcal{O}_f$  and  $O(\text{polylog}(m))$  additional gates.*

► **Corollary 34.** *The unitary  $U_B$  can be implemented in  $O(\sqrt{d})$  queries to the down-incidence array and  $\tilde{O}(\sqrt{d})$  time.*

<sup>5</sup> The down-incidence array is actually an adjacency array of a graph related to simplicial complexes, namely, the incidence graph between the  $(d-1)$ - and  $d$ -simplices.

## 31:20 Effective Resistance and Capacitance in Simplicial Complexes

It is harder to produce a generic implementation of  $U_C$  than  $U_B$ . The  $d$ -simplices can have arbitrary weights, so constructing the states  $|c_\sigma\rangle$  in general requires constructing arbitrary quantum states with real coefficients. However, the weights on the simplices do not affect whether or not a cycle is null-homologous. Therefore, we can always run our null-homology test on the unweighted complex; the trade-off is that the effective resistance or effective capacitance might be higher in the unweighted complex. We analyze the running time of this case in Section A.

We now analyze the complexity of constructing the initial state  $|w_0\rangle/||w_0\rangle||$ . To construct the dummy state, we start by adding an additional “ $d$ -cell”  $|\emptyset\rangle$  to the complex with boundary  $|\gamma\rangle$  (really, we just add  $|\gamma\rangle$  as a column to  $\partial$ .) The new cell will have non-trivial overlap with  $|w_0\rangle$ , so we can construct  $|w_0\rangle$  by amplifying this component of  $|\emptyset\rangle$ . We outline this method in the proof of Theorem 36, but first, we state Lemma 35 which is a generalization of the parallel formula for effective resistance; its proof is nearly identical to the proof of Theorem 10.

► **Lemma 35.** *Let  $V = V_1 \oplus V_2$  be a vector space. Let  $A : V \rightarrow U$  be a linear map, and let  $A_1 : U_1 \rightarrow V$  and  $A_2 : U_2 \rightarrow V$  be the restriction of  $A$  to  $U_1$  and  $U_2$ . Let  $|t\rangle \in \text{im } A_1 \cap \text{im } A_2 \subset U$ . If  $|s\rangle = A^+|t\rangle$ ,  $|s_1\rangle = A_1^+|t\rangle$ , and  $|s_2\rangle = A_2^+|t\rangle$ , then*

$$||s||^2 \leq \left( \frac{1}{||s_1||^2} + \frac{1}{||s_2||^2} \right)^{-1}$$

*Equality is achieved when  $\text{im } A_1 \cap \text{im } A_2 = \text{span}\{|t\rangle\}$ . In this case,  $|s\rangle = t|s_1\rangle + (1-t)|s_2\rangle$  where  $t = ||s_2||^2 / (||s_1||^2 + ||s_2||^2)$ .*

► **Theorem 36.** *Let  $\mathcal{O}_\gamma$  be the oracle that takes  $\mathcal{O}_\gamma : |0\rangle \rightarrow |\gamma\rangle$ . Let  $T_\gamma$  be the time it takes to implement  $\mathcal{O}_\gamma$ . The state  $|w_0\rangle = \partial^+|\gamma\rangle$  can be created in  $\tilde{O}((\sqrt{1/\mathcal{R}_\gamma(\mathcal{K})} + \sqrt{\mathcal{R}_\gamma(\mathcal{K})})(T_B + T_C + T_\gamma))$  time.*

**Proof.** We append  $|\gamma\rangle$  as a column to  $\partial$  to create a new matrix  $\hat{\partial}$ . Let  $|\emptyset\rangle$  be index of the new column, so  $\hat{\partial} = \partial + |\gamma\rangle\langle\emptyset|$ . Let  $|w'_0\rangle = \hat{\partial}^+|\gamma\rangle$ . We conclude that  $|\emptyset\rangle = |w'_0\rangle + |w'_0{}^\perp\rangle$  where  $|w'_0{}^\perp\rangle \in \ker \hat{\partial}$ , as the projection  $\Pi_{\ker \hat{\partial}}|\emptyset\rangle = \hat{\partial}^+\hat{\partial}|\emptyset\rangle = \hat{\partial}^+|\gamma\rangle = |w'_0\rangle$ .

We construct  $|w_0\rangle/||w_0\rangle||$  in two steps. First, we use amplitude amplification to amplify the  $|w'_0\rangle$  component of  $|\emptyset\rangle$ . We then use a second amplitude amplification to amplify the  $|w_0\rangle$  component of  $|w'_0\rangle$ . These amplitude amplifications are nested, as we need to perform the first to create the initial state for the second.

If we perform constant time phase estimation of  $|\emptyset\rangle$  on the unitary  $R_{\ker \hat{\partial}}$ , then we can map  $|\emptyset\rangle$  to  $|0\rangle|w'_0\rangle + |1\rangle|w'_0{}^\perp\rangle$ . We can then amplify the amplitude of  $|0\rangle|w_0\rangle$  part arbitrarily close to  $|w_0\rangle/||w_0\rangle||$  using  $O(||w_0\rangle||^{-1})$  calls to  $R_{\ker \hat{\partial}}$ .

We calculate  $||w_0\rangle||$  using the formula from the lemma. The vector  $|\emptyset\rangle$  has length 1, so Lemma 35 shows that

$$||w'_0\rangle||^2 = \left( \frac{1}{\mathcal{R}_\gamma(\mathcal{K})} + 1 \right)^{-1} = \frac{\mathcal{R}_\gamma(\mathcal{K})}{\mathcal{R}_\gamma(\mathcal{K}) + 1}.$$

Thus, we need to perform the reflection  $R_{\ker \hat{\partial}}$  a total of  $O(||w_0\rangle||^{-1}) = O(\sqrt{(\mathcal{R}_\gamma(\mathcal{K}) + 1)/\mathcal{R}_\gamma(\mathcal{K})})$  times to create  $|\hat{w}_0\rangle/||\hat{w}_0\rangle||$ .

The next step in our algorithm is to amplify the  $|w_0\rangle/\|w_0\rangle\|$  component of  $|w'_0\rangle/\|w'_0\rangle\|$ . By Lemma 35, the state  $\|w'_0\rangle\| = t\|w_0\rangle\| + (1-t)\|\emptyset\|$  for  $t = 1/(\mathcal{R}_\gamma(\mathcal{K}) + 1)$ . Therefore, the  $|w_0\rangle/\|w_0\rangle\|$  component of  $|w'_0\rangle/\|w'_0\rangle\|$  has norm

$$\begin{aligned} t\|w_0\rangle/\|w'_0\rangle\| &= \frac{1}{\mathcal{R}_\gamma(\mathcal{K}) + 1} \sqrt{\mathcal{R}_\gamma(\mathcal{K})} \sqrt{\frac{\mathcal{R}_\gamma(\mathcal{K}) + 1}{\mathcal{R}_\gamma(\mathcal{K})}} \\ &= \sqrt{\frac{1}{\mathcal{R}_\gamma(\mathcal{K}) + 1}} \end{aligned}$$

To return the state  $|w_0\rangle/\|w_0\rangle\|$ , we need to perform amplitude amplification again. We can create the state  $|w'_0\rangle$  using the amplitude amplification from the previous two paragraphs with  $O(\sqrt{(\mathcal{R}_\gamma(\mathcal{K}) + 1)/\mathcal{R}_\gamma(\mathcal{K})})$  applications of  $R_{\ker \partial'}$ , and we can reflect across  $|\emptyset\rangle$  in constant time as it is a basis state. To create  $|w_0\rangle/\|w_0\rangle\|$ , we need

$$O\left(\sqrt{(\mathcal{R}_\gamma(\mathcal{K}) + 1)/\mathcal{R}_\gamma(\mathcal{K})} \sqrt{(\mathcal{R}_\gamma(\mathcal{K}) + 1)}\right) = O\left(\sqrt{\mathcal{R}_\gamma(\mathcal{K})} + \sqrt{\frac{1}{\mathcal{R}_\gamma(\mathcal{K})}}\right)$$

applications of  $R_{\ker \hat{\partial}}$ .

We now argue that we can compute  $R_{\ker \hat{\partial}}$  in  $O(T_C + T_B + T_\gamma)$  time. As was the case with  $R_{\ker \partial}$ , we decompose  $R_{\ker \hat{\partial}} = M_B^\dagger R_{\hat{U}} M_B$  for space  $\hat{B}$  and  $\hat{C}$  defined

$$\hat{B} = B \cup \{b_\emptyset := |\gamma\rangle\}$$

$$\hat{C} = \text{span} \left\{ |c_\sigma\rangle := \frac{1}{\sqrt{\deg(\sigma) + 1}} |\sigma\rangle |\emptyset\rangle + \sum_{\sigma \subset \tau} \frac{w(\sigma)}{\sqrt{\deg(\sigma) + 1}} |\sigma\rangle |\tau\rangle : \sigma \in \mathcal{K}_{d-1} \right\}.$$

The unitaries  $M_{\hat{B}}$ ,  $M_{\hat{C}}$ , and  $\mathbb{R}_{\hat{U}}$  are defined analogously to  $M_B$  and  $M_C$ . We can implement the unitary version of these matrices  $U_{\hat{B}}$  in  $O(T_B + T_\gamma)$  and  $U_{\hat{C}}$  in  $O(T_C)$ . ◀

We summarize this section in the following theorem.

► **Theorem 37.** *Let  $\mathcal{K}$  be a simplicial complex,  $\gamma \in C_{d-1}(\mathcal{K})$  a null-homologous cycle, and  $\mathcal{K}(x) \subset \mathcal{K}$  be a simplicial complex. There is a quantum algorithm for deciding if  $\gamma$  is null-homologous in  $\mathcal{K}(x)$  that runs in time*

$$\tilde{O}\left(\sqrt{\frac{(d+1)\mathcal{R}_{\max}(\mathcal{K})\mathcal{C}_{\max}(\mathcal{K})}{\lambda}}(\sqrt{d} + T_C) + \left(\sqrt{\frac{1}{\mathcal{R}_\gamma(\mathcal{K})}} + \sqrt{\mathcal{R}_\gamma(\mathcal{K})}\right)(\sqrt{d} + T_C + T_\gamma)\right)$$

## Special cases

We now consider a few special cases of the null-homology span program. These special cases will allow us to replace the terms  $T_B$  and  $T_\gamma$  in Theorem 37 with concrete running times.

### Unweighted simplicial complexes

We now consider the case where there are no weights on the  $d$ -simplices, or equivalently, when  $w(\tau) = 1$  for each  $d$ -simplex  $\tau$ . While computing  $M_C$  is hard in general, in the unweighted case, we can implement the unitary  $M_C$  using a straightforward oracle. For a  $(d-1)$ -simplex  $\sigma$  that is incident to the  $d$ -simplices  $\{\tau_1, \dots, \tau_m\}$ , the **up-incidence array** is the oracle is the function that maps  $h : |\sigma\rangle |j\rangle |0\rangle \rightarrow |\sigma\rangle |j\rangle |\tau_j\rangle$ . By Lemma 33, the up-incidence array can be used to compute  $|c_\tau\rangle = \frac{1}{\sqrt{m}} |\tau\rangle |\delta\tau\rangle$  in  $O(\sqrt{m})$  time. The unitary  $M_C$  computes the state  $|c_\sigma\rangle$  in parallel, so computing  $M_C$  will take  $\sqrt{d_{\max}}$  queries, where  $d_{\max} = \max_{\tau \in C_{d-1}(\mathcal{K})} \deg(\tau)$  time. This is summarized in the following lemma.

## 31:22 Effective Resistance and Capacitance in Simplicial Complexes

► **Lemma 38.** *If  $\mathcal{K}$  is an unweighted simplicial complex, the unitary  $U_C$  can be implemented in  $O(\sqrt{d_{\max}})$  queries to the up-incidence array and  $\tilde{O}(\sqrt{d_{\max}})$  time.*

### Cycle is the boundary of a $d$ -simplex

We now consider the case that the input cycle  $\gamma$  is the boundary of a  $d$ -simplex. In this case, we can implement the oracle  $\mathcal{O}_\gamma$  with the down incidence array used to implement  $M_B$ . We get the same running time for  $T_\gamma$  as  $T_B$ .

► **Lemma 39.** *If  $\gamma$  is the boundary of a  $d$ -simplex, there is a quantum algorithm implementing  $\mathcal{O}_\gamma$  in  $O(\sqrt{d})$  queries to the down incidence array and  $\tilde{O}(\sqrt{d})$  time.*

## B Omitted proofs

### Proof of Theorem 16

Before obtaining our upper bound on the effective capacitance of a cycle we need to prove one lemma. In the following lemma, we provide an upper bound on the largest singular value of the coboundary matrix.

► **Lemma 40.** *The largest singular value of the coboundary matrix  $\delta_{d-1}$  is  $\sigma_{\max}(\delta_{d-1}) = O(\sqrt{dn})$ .*

**Proof.** Recall that the  $(d-1)$  up-Laplacian is  $L = \delta_{d-1}^T \delta_{d-1}$ . The squared singular values of  $\delta_{d-1}$  are the eigenvalues of  $L$ ; this follows from the generic theorem that the squared singular values of a matrix  $A$  are the eigenvalues of  $A^T A$ . Thus,  $\sigma_{\max}(\delta_{d-1})^2 \leq \sum_i \sigma_i(\delta_{d-1})^2 = \text{trace}(L)$ , where the  $\sigma_i(\delta_{d-1})$  are the singular values of  $\delta_{d-1}$ . We can obtain an upper bound on  $\sigma_{\max}(\delta_{d-1})$  by computing the trace of  $L$ . The diagonal elements of  $L$  are the degrees of the  $(d-1)$ -simplices [8, Proposition 3.3.2]. Each  $d$ -simplex is the coface of  $d+1$   $(d-1)$ -simplices, so summing up the diagonal of  $L$ , we find  $\text{trace}(L) = O(dn)$ . Thus,  $\sigma_{\max}(\delta_{d-1})$  is  $O(\sqrt{dn})$ . ◀

► **Theorem 16.** *Let  $\mathcal{L} \subset \mathcal{K}$  be an unweighted  $d$ -dimensional simplicial complexes, and let  $\hat{\gamma} \in C_{d-1}(\mathcal{L})$  be a  $(d-1)$ -cycle that is null-homologous in  $\mathcal{K}$  but not in  $\mathcal{L}$ . Assume also that  $\hat{\gamma}$  is integral and each of the coefficients  $|\hat{\gamma}_i| = O(1)$ . The effective capacitance of  $\gamma := \hat{\gamma}/\|\hat{\gamma}\|$  in  $\mathcal{K}(x)$  is bounded above by  $\mathcal{C}_\gamma(\mathcal{L}) = O(n^3 \cdot d \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ .*

**Proof.** Observe that because the coefficients  $\hat{\gamma}_i = O(1)$  then  $\|\hat{\gamma}\| = O(\sqrt{n})$ . We will use this fact later in the proof. Let  $p$  be a  $\gamma$ -potential. By definition,  $\delta[\mathcal{L}]p = 0$  and  $\gamma^T p = 1$ . We can express these constraints as the linear system

$$\begin{bmatrix} \delta[\mathcal{L}] \\ \gamma^T \end{bmatrix} p = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

We first remove linearly-dependent columns from this linear system until this system has full column rank. Columns of the matrix correspond to  $(d-1)$  simplices of  $\mathcal{L}$ , and rows correspond to  $d$ -simplices of  $\mathcal{L}$ . Removing columns from  $\delta[\mathcal{L}]$  changes  $\delta[\mathcal{L}]$  to the relative coboundary matrix  $\delta[\mathcal{L}, \mathcal{L}_0]$  where  $\mathcal{L}_0$  is the  $(d-1)$ -subcomplex corresponding to the columns that were removed.

Removing linearly-dependent columns does not change the image of the system of equation, so there is still a solution  $r$ , i.e.

$$\begin{bmatrix} \delta[\mathcal{L}, \mathcal{L}_0] \\ c \end{bmatrix} r = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

where  $c$  is the subvector of  $\gamma^T$  after removing the columns. The vector  $r$  is not a  $\gamma$ -potential as it is a vector in  $C_{d-1}(\mathcal{L} \setminus \mathcal{L}_0)$ , not  $C_{d-1}(\mathcal{L})$ . However, we can extend  $r$  to be a  $\gamma$ -potential by adding zeros in the entries indexed by  $\mathcal{L}_0$ . Adding zero-valued entries preserves the length of  $r$ .

We now want to remove rows from this matrix so that it has full row rank. Topologically, removing rows corresponds to removing  $d$ -simplices from the complex  $\mathcal{L}$  to create a new complex  $\mathcal{L}_1$ . Note that we must always include the row  $c$  to have full row rank; otherwise,  $r$  would be a non-zero vector in the kernel of this system, meaning the system does not have full rank. Removing these rows gives the linear system

$$\begin{bmatrix} \delta[\mathcal{L}_1, \mathcal{L}_0] \\ c \end{bmatrix} r = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Let  $C = [\delta[\mathcal{L}_1, \mathcal{L}_0]^T \quad c^T]^T$  and  $b = [0 \quad 0 \quad \dots \quad 1]^T$ . Note that  $C$  is an square matrix of size (say)  $m \times m$ .

We now use Cramer's rule to bound the size of  $\|r\|$ . By Cramer's rule,  $r_i$ , the  $i$ th entry of  $r$ , is  $r_i = \frac{\det(C_{i,b})}{\det(C)}$ . where  $C_{i,b}$  is the matrix obtained by replacing the  $i$ th column with  $b$ . We first lower bound  $|\det(C)|$ . We can express  $\det(C)$  by its cofactor expansion on the row of  $c$  as  $\det(C) = \sum_{i=1}^m (-1)^i \cdot c_i \cdot \det(\delta[\mathcal{L}_1, \mathcal{L}_0]_i)$  where  $\delta[\mathcal{L}_1, \mathcal{L}_0]_i$  is  $\delta[\mathcal{L}_1, \mathcal{L}_0]$  without the  $i$ th column. Each term  $\delta[\mathcal{L}_1, \mathcal{L}_0]_i$  is integral as  $\delta[\mathcal{L}_1, \mathcal{L}_0]$  is an integral matrix. Moreover, each term  $c_i = \hat{\gamma}_{j_i} / \|\hat{\gamma}\|$  where  $\hat{\gamma}_{j_i}$  is an integer, as  $c$  is a subvector of  $\gamma$ . We can then derive the lower bound

$$\begin{aligned} |\det(C)| &= \left| \sum_{i=1}^m (-1)^i \cdot c_i \cdot \det(\delta[\mathcal{L}_1, \mathcal{L}_0]_i) \right| \\ &= \frac{1}{\|\hat{\gamma}\|} \left| \sum_{i=1}^m (-1)^i \cdot \hat{\gamma}_{j_i} \cdot \det(\delta[\mathcal{L}_1, \mathcal{L}_0]_i) \right| \\ &\geq \frac{1}{\|\hat{\gamma}\|} \quad (\text{as } \sum_{i=1}^m (-1)^i \cdot \hat{\gamma}_{j_i} \cdot \det(\delta[\mathcal{L}_1, \mathcal{L}_0]_i) \text{ is integral}) \\ &= \Omega\left(\frac{1}{\sqrt{|\text{supp}(\hat{\gamma})|}}\right) \quad (\text{as each entry } \hat{\gamma}_i \text{ is } O(1)) \\ &= \Omega\left(\frac{1}{\sqrt{n}}\right). \end{aligned}$$

We now upper bound  $|\det(C_{i,b})|$ . We calculate  $\det(C_{i,b})$  with the cofactor expansion on the column replaced by  $b$ . As  $b$  has 1 in its last entry and 0s elsewhere, the cofactor expansion is  $\det(C_{i,b}) = \det(C_{i,b}^{i,c})$  where  $C_{i,b}^{i,c}$  is the matrix where we dropped the  $i$ th column and the row  $c$  from  $C_{i,b}$ . The matrix  $C_{i,b}^{i,c}$  is a square submatrix of  $\delta[\mathcal{K}]$ , so we can bound



$|\det(C_{i,b})| \leq \mathcal{T}(\mathcal{K})$ . Thus,  $r_i = \det(C_{i,b})/\det(C) \leq \sqrt{n} \cdot \mathcal{T}_{\max}(\mathcal{K})$  and  $\|r\| = \sqrt{\sum_{i=1}^m r_i^2} \leq \sqrt{n^2 \cdot \mathcal{T}_{\max}(\mathcal{K})^2} = n \cdot \mathcal{T}_{\max}(\mathcal{K})$ . The potential energy of  $r$  is  $\|\delta[\mathcal{K}]r\|^2$ . We can bound this using Lemma 40 to obtain  $\|\delta[\mathcal{K}]r\|^2 = O(n^3 \cdot d \cdot \mathcal{T}_{\max}(\mathcal{K})^2)$ . ◀

## C Embedded complexes

In this section, we consider the special case when  $\mathcal{K}$  is a  $d$ -dimensional simplicial complex with a given embedding into  $\mathbb{R}^{d+1}$ . In this case,  $\mathcal{K}$  is an **embedded complex**. Embedded complexes serve as a high dimensional generalization of planar graphs and naturally admit a dual graph. More specifically, we will generalize the special case of planar graphs for which the vertices  $s$  and  $t$  appear on the boundary of the same face. Throughout this section we assume we are given the embedding as input. Computing the dual graph from an embedding can be done in polynomial time [6]. We will show that the effective capacitance of a  $(d-1)$ -dimensional cycle  $\gamma$  in  $\mathcal{K}$  is equal to the effective resistance between a pair of vertices that are “dual” to  $\gamma$ . Hence, we can parameterize the quantum algorithm deciding if  $\gamma$  is null-homologous in terms of the effective resistance in  $\mathcal{K}$  and the effective resistance in the dual graph of  $\mathcal{K}$ . This section generalizes the analysis of planar graphs given by Jeffery and Kimmel [13]. The setup for our analysis has appeared in the author’s previous work [20].

The Alexander duality theorem [10, Corollary 3.45], states that for a  $d$ -dimensional simplicial complex  $\mathcal{K}$  with an embedding into  $\mathbb{R}^{d+1}$  the subspace  $\mathbb{R}^{d+1} \setminus \mathcal{K}$  consists of  $\beta_d + 1$  connected components where  $\beta_d$  is the dimension of  $H_d(\mathcal{K})$ . We call these connected components **voids** and exactly one of these voids is unbounded. We denote the bounded voids as  $V_i$  for  $1 \leq i \leq \beta_d$  and the unbounded void as  $V_\infty$ . Moreover, the boundaries of the bounded voids generate the homology group  $H_d(\mathcal{K})$ . The embedding implies that each  $d$ -simplex is contained on the boundary of at most two voids, and we make the assumption that the  $d$ -simplices are oriented consistently with respect to the voids. That is, if a  $d$ -simplex is on the boundary of two voids it is oriented positively on one void, and negatively on the other. We have a boundary matrix  $\partial_{d+1}$  whose columns are the voids and whose rows are the  $d$ -simplices. From the embedding and the consistent orientation we see that  $\partial_{d+1}$  is the edge-vertex incident matrix of the **directed dual graph**: the directed graph whose vertices are in bijection with the voids and whose edges are in bijection with the  $d$ -simplices of  $\mathcal{K}$ . The direction of the edges are inherited from the orientations of the  $d$ -simplices. For a  $d$ -simplex  $\sigma$  on the boundary of voids  $V_1$  and  $V_2$  we denote the dual edge by  $\sigma^* = (v_1^*, v_2^*)$  and we define the dual weight function by  $w^*(\sigma^*) = 1/w(\sigma)$ .

We construct an additional chain group  $C_{d+1}(\mathcal{K})$  whose basis elements are the bounded voids. This is a purely algebraic construction and gives rise to a new chain complex  $\dots \rightarrow C_{d+1}(\mathcal{K}) \xrightarrow{\partial_{d+1}} C_d(\mathcal{K}) \xrightarrow{\partial_d} \dots \xrightarrow{\partial_1} C_0(\mathcal{K})$ . Since the boundaries of the voids generate the  $d^{\text{th}}$  homology group of  $\mathcal{K}$  and  $C_{d+1}(\mathcal{K})$  is generated by these voids we obtain a valid chain complex. Moreover, we have that  $\dim H_d(\mathcal{K}) = 0$  in our new chain complex. More generally, we define the **dual complex** of  $\mathcal{K}$ , denoted  $\mathcal{K}^*$ , by the isomorphism  $C_{d-k+1}(\mathcal{K}) \cong C_k(\mathcal{K})$ . That is, the  $(d-k+1)$ -simplices of  $\mathcal{K}$  are in bijection with the  $k$ -simplices of  $\mathcal{K}^*$ . The dual graph is the 1-skeleton of  $\mathcal{K}^*$ . Moreover, we define the dual boundary operator  $\partial_k^*: C_k(\mathcal{K}^*) \rightarrow C_{k-1}(\mathcal{K}^*)$  to be the coboundary operator  $\delta_{d-k+1}: C_{d-k+1}(\mathcal{K}) \rightarrow C_{d-k}(\mathcal{K})$  of  $\mathcal{K}$ , and the dual coboundary operator  $\delta_k^*: C_{k-1}(\mathcal{K}^*) \rightarrow C_k(\mathcal{K}^*)$  to be the boundary operator  $\partial_{d-k+1}: C_{d-k+1}(\mathcal{K}) \rightarrow C_{d-k}(\mathcal{K})$  of  $\mathcal{K}$ . In other words the (co)boundary operators commute with the duality isomorphism. We summarize the construction in Figure 4 in the appendix.

We need to make one additional assumption on the location of the input  $(d-1)$ -dimensional cycle  $\gamma$  which makes our setup a generalization of a planar graph with two vertices  $s$  and  $t$  appearing on the same face. To achieve this we assume that there exists a void  $V_i$  with two unit

$\gamma$ -flows  $\Gamma_1$  and  $\Gamma_2$  such that  $\text{supp}(\Gamma_1) \cap \text{supp}(\Gamma_2) = \emptyset$  and  $\text{supp}(\Gamma_1) \cup \text{supp}(\Gamma_2) = \text{supp}(\partial_{d+1} V_i)$ . That is, there exist two unit  $\gamma$ -flows whose supports partition the boundary of the void  $V_i$ . This generalizes the fact in planar graphs that when  $s$  and  $t$  are on the same face we can find two  $st$ -paths which partition the boundary of the face. In planar graphs we are guaranteed to find two such paths, however for an arbitrary  $(d-1)$ -cycle  $\gamma$  we are not guaranteed to find two unit  $\gamma$ -flows partitioning the boundary of some void. More specifically, we take  $\Gamma_2$  to be a unit  $(-\gamma)$ -flow so that  $\partial_d \Gamma_2 = -\gamma$ . In the planar graph analogy this is equivalent as viewing  $\Gamma_1$  as a path from  $s$  to  $t$  and viewing  $\Gamma_2$  as a path from  $t$  to  $s$ . We add an additional basis element  $\Sigma$  to  $C_d(\mathcal{K})$  such that  $\partial_d \Sigma = -\gamma$ . In planar graphs this is equivalent to adding an edge directed from  $t$  to  $s$ . In a planar graphs the addition of this edge splits the face containing  $s$  and  $t$  into two. In higher dimensions the geometry is more complicated, but the addition of  $\Sigma$  allows us to perform a purely algebraic operation makes our chain complex behave as if  $V_i$  has been split into two. We remove  $V_i$  from  $C_{d+1}(\mathcal{K})$  and replace it with two new basis elements  $V_s$  and  $V_t$ . Next, we extend the boundary operator to  $V_s$  and  $V_t$  in the following way:  $\partial_{d+1} V_s = \Gamma_1 - \Sigma$  and  $\partial_{d+1} V_t = \Gamma_2 + \Sigma$ . In the dual complex the vertices dual to  $V_s$  and  $V_t$  are denoted  $s^*$  and  $t^*$ . Dual to  $\Sigma$  is an edge  $\Sigma^* = (t^*, s^*)$ . In the next section we will show that the effective capacitance of  $\gamma$  in a subcomplex  $\mathcal{K}(x)$  is equal to the effective resistance between  $s^*$  and  $t^*$  in the subgraph of the dual graph which is the 1-skeleton of  $\mathcal{K}^*(x)$ . Note that the 1-skeleton of  $\mathcal{K}^*(x)$  contains all of the vertices of  $\mathcal{K}^*$  but only includes the edges dual to the  $d$ -simplices in  $\mathcal{K}(x)$ .

### Effective capacitance is dual to effective resistance

The effective resistance between  $s^*$  and  $t^*$  in  $\mathcal{K}^*(x)$  is determined by the unit  $s^*t^*$ -flows in  $\mathcal{K}^*(x)$ . However, it will be convenient to work with **circulations** instead of flows. A unit  $s^*t^*$ -circulation  $f$  is a cycle; that is, an element of  $\ker \partial_1^*$ , such that  $f(\Sigma^*) = 1$ . Recall that  $\Sigma^*$  is the edge directed from  $t^*$  to  $s^*$ , so a unit  $s^*t^*$ -circulation is just a unit  $s^*t^*$ -flow with the additional edge  $\Sigma^*$  completing the cycle. Clearly, there is a bijection between unit  $s^*t^*$ -flows and unit  $s^*t^*$ -circulations and we define the flow energy of a circulation to be equal to the flow energy of its corresponding flow.

► **Theorem 41.** *Let  $\mathcal{K}$  be a  $d$ -dimensional simplicial complex embedded into  $\mathbb{R}^{d+1}$ , and let  $\gamma$  be a  $(d-1)$ -cycle such that there exist two unit  $\gamma$ -flows  $\Gamma_1$  and  $\Gamma_2$  whose supports partition the boundary of some void  $V_i$ . The effective capacitance  $\mathcal{C}_\gamma(\mathcal{K}(x))$  is equal to the effective resistance  $\mathcal{R}_{s^*t^*}(\mathcal{K}^*(x))$ .*

**Proof.** Let  $p$  be a unit  $\gamma$ -potential in  $\mathcal{K}(x)$  and we define  $f$  to be the image of  $\delta p$  under the duality isomorphism; that is,  $f = \partial_2^* p^*$ , which makes  $f$  a circulation in the 1-skeleton of  $\mathcal{K}^*(x)$ . Further, since  $\Sigma^* = (t^*, s^*)$  the circulation  $f$  corresponds to a unit  $s^*t^*$ -flow by the following calculation:  $f^T \Sigma^* = p^T (\partial_d \Sigma) = p^T \gamma = 1$ . Next, we calculate the flow energy of  $f$  and show it is equal to the potential energy of  $p$ .

$$\begin{aligned}
 J(f) &= \sum_{e^* \in \mathcal{K}^*(x)_1} \frac{f(e^*)^2}{w^*(e^*)} \\
 &= \sum_{e^* \in \mathcal{K}^*(x)_1} f(\sigma^*)^2 w(e) \\
 &= \sum_{\sigma \in \mathcal{K}(x)_d} ((\delta p)^T \sigma)^2 w(\sigma) \\
 &= \mathcal{J}(p)
 \end{aligned}$$

### 31:26 Effective Resistance and Capacitance in Simplicial Complexes

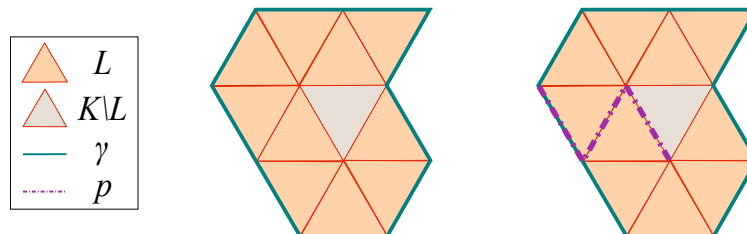
Conversely, let  $f^*$  be a unit  $s^*t^*$ -circulation in  $\mathcal{K}^*(x)$ . By the assumptions outlined in the beginning of the section we have  $\dim H_d(\mathcal{K}) = 0$  which in turn gives us  $\dim H_1(\mathcal{K}^*) = 0$ . Hence,  $f^*$  can be written as a linear combination of boundaries  $f^* = \sum \alpha_i B_i$  where  $B_i \in \text{im } \partial_2^*$ . Let  $p^*$  be the 2-chain in  $\mathcal{K}^*(x)$  with  $\partial_2^* p^* = f^*$ ; we will show that  $p$  is the unit  $\gamma$ -potential in  $\mathcal{K}(x)$  in bijection with  $f^*$ . To see that  $p$  is a unit  $\gamma$ -potential we compute its inner product with  $\gamma$ :

$$\begin{aligned} p^T \gamma &= p^T (\partial_d \Sigma) \\ &= (\delta_{d-1} p)^T \Sigma \\ &= (\partial_2^* p^*)^T \Sigma^* \\ &= (f^*)^T \Sigma^* \\ &= 1. \end{aligned}$$

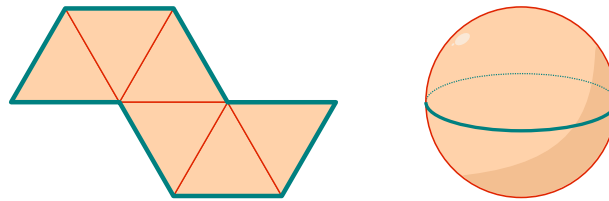
It remains to show that the potential energy of  $p$  is equal to the flow energy of  $|f^*|$ . We have the following calculation:

$$\begin{aligned} \mathcal{J}(p) &= \sum_{\sigma \in \mathcal{K}(x)_d} ((\delta_{d-1} p)^T \sigma)^2 w(\sigma) \\ &= \sum_{\sigma \in \mathcal{K}(x)_d} \frac{((\delta_{d-1} p)^T \sigma)^2}{w^*(\sigma^*)} \\ &= \sum_{\sigma \in \mathcal{K}(x)_d} \frac{((\partial_2^* p^*)^T \sigma^*)^2}{w^*(\sigma^*)} \\ &= \sum_{\sigma \in \mathcal{K}(x)_d} \frac{f^*(\sigma)^2}{w^*(\sigma^*)} \\ &= \mathcal{J}(f^*). \end{aligned}$$

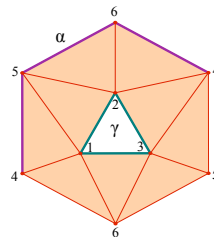
#### D Figures



■ **Figure 1** Left: A 1-cycle  $\gamma$ . Right: A unit  $\gamma$ -potential  $p$ . If this complex is unweighted, then the potential energy of  $p$  is 1.



■ **Figure 2** Left: The unique unit  $\gamma$ -flow is the 6 triangles in series. If the complex is unweighted, then the effective resistance of  $\gamma$  is 6. Right: The cycle  $\gamma$  is the equator of the sphere, and the two hemispheres are two unit  $\gamma$ -flows in parallel. If each hemisphere has potential energy 1, then the effective resistance of  $\gamma$  is  $\frac{1}{2}$ .



■ **Figure 3** The complex  $\mathbb{R}\mathbb{P}_\gamma$ . The inner cycle is  $\gamma$ , and half of the outer cycle is  $\alpha$ . The boundary of the sum of the triangles is  $2\alpha + \gamma$ .

$$\begin{array}{ccccccc}
 C_{d+1}(\mathcal{K}) & \xleftarrow{\partial_{d+1}} & C_d(\mathcal{K}) & \xleftarrow{\partial_d} & \dots & \xleftarrow{\partial_1} & C_0(\mathcal{K}) \\
 \uparrow \cong & & \uparrow \cong & & & & \uparrow \cong \\
 C_0(\mathcal{K}^*) & \xleftarrow{\delta_0^*} & C_1(\mathcal{K}^*) & \xleftarrow{\delta_1^*} & \dots & \xleftarrow{\delta_d^*} & C_{d+1}(\mathcal{K}^*)
 \end{array}$$

■ **Figure 4** The commutative diagram summarizing the dual complex construction.



# Anonymity-Preserving Space Partitions

Úrsula Hébert-Johnson ✉

Department of Computer Science, University of California, Santa Barbara, CA, USA

Chinmay Sonar ✉ 🏠

Department of Computer Science, University of California, Santa Barbara, CA, USA

Subhash Suri ✉

Department of Computer Science, University of California, Santa Barbara, CA, USA

Vaishali Surianarayanan ✉

Department of Computer Science, University of California, Santa Barbara, CA, USA

---

## Abstract

We consider a multidimensional space partitioning problem, which we call ANONYMITY-PRESERVING PARTITION. Given a set  $P$  of  $n$  points in  $\mathbb{R}^d$  and a collection  $H$  of  $m$  axis-parallel hyperplanes, the hyperplanes of  $H$  partition the space into an arrangement  $\mathcal{A}(H)$  of rectangular cells. Given an integer parameter  $t > 0$ , we call a cell  $C$  in this arrangement *deficient* if  $0 < |C \cap P| < t$ ; that is, the cell contains at least one but fewer than  $t$  data points of  $P$ . Our problem is to remove the minimum number of hyperplanes from  $H$  so that there are no deficient cells. We show that the problem is NP-complete for all dimensions  $d \geq 2$ . We present a polynomial-time  $d$ -approximation algorithm, for any fixed  $d$ , and we also show that the problem can be solved exactly in time  $(2d - 0.924)^k m^{O(1)} + O(n)$ , where  $k$  is the solution size. The one-dimensional case of the problem, where all hyperplanes are parallel, can be solved optimally in polynomial time, but we show that a related INTERVAL ANONYMITY problem is NP-complete even in one dimension.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Anonymity, Hitting Set, LP, Constant Approximation, Fixed-Parameter Tractable, Space Partitions, Parameterized Complexity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.32

**Funding** *Subhash Suri*: in part supported by NSF grant CCF-1814172.

**Acknowledgements** We thank Daniel Lokshtanov for discussions relating to our approximation algorithm for ANONYMITY-PRESERVING PARTITION, as well as an anonymous reviewer, whose comments helped to improve the aforementioned result.

## 1 Introduction

Consider the following geometric problem. We are given a set  $P$  of  $n$  points and a family  $H$  of  $m$  axis-parallel hyperplanes in  $\mathbb{R}^d$ . The hyperplanes of  $H$  partition the space into an arrangement  $\mathcal{A}(H)$  of rectangular cells. Given an integer parameter  $t > 0$ , we call a cell  $C$  *deficient* if  $0 < |C \cap P| < t$ ; that is, the cell contains at least one but fewer than  $t$  data points of  $P$ . We then ask: What is the minimum number of hyperplanes we must delete so that there are no deficient cells? See Figure 1 for an example. The problem turns out to be nontrivial even in two dimensions and, in fact, also in one dimension under a dual formulation.

While we are mainly interested in this as a natural geometric problem, it can also be relevant in the study of data anonymity. For instance, given a real-valued *scalar* data set, a common technique for *group anonymization* is to partition the domain into *buckets*, defined by a set of boundary values  $\{x_1, x_2, \dots, x_t\}$ . Given an integer target  $t > 0$ , the buckets are chosen to ensure that any bucket  $[x_i, x_{i+1}]$  is either empty or contains at least  $t$  different



© Úrsula Hébert-Johnson, Chinmay Sonar, Subhash Suri, and Vaishali Surianarayanan; licensed under Creative Commons License CC-BY 4.0

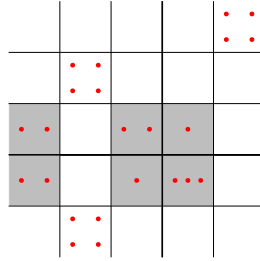
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 32; pp. 32:1–32:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** A 2-dimensional Anonymity-Preserving Partition instance with  $t = 4$ . The deficient cells are highlighted in gray and the two bold lines denote the optimal solution.

data records, thereby ensuring  $t$ -anonymity for each individual data value. Generalizing this to multidimensional data, the buckets are defined independently for each of the  $d$  axes, which geometrically creates a set of axis-parallel hyperplanes – the hyperplanes with normals parallel to the  $i$ -th coordinate axis correspond to the bucketing of the  $i$ -th dimension. Given a set of multidimensional data points and a set of candidate hyperplanes, the problem of discarding the fewest number of hyperplanes to achieve  $t$ -anonymity is precisely our space partitioning problem. For instance, one can imagine points being user locations in a two-dimensional coordinate system, and the problem is to specify those locations to within some “longitude” and “latitude” values so that every user’s location is  $t$ -anonymized. Inspired by these connections, we have chosen to call our problem ANONYMITY-PRESERVING PARTITION for convenience, but our research focus in this work is purely algorithmic, and not related to anonymity.

Space partitioning problems are fundamental to many domains, including computational geometry, databases, robotics, etc. [12, 4, 6, 9, 5, 2]; however, to the best of our knowledge, this particular partition problem has not been studied. In computational geometry, for instance, space partitioning is frequently used for *range query* data structures such as  $kD$ -trees, range trees, etc. [7, 22, 1, 18, 20]. The primary focus in those algorithms is a hierarchical partitioning of the space to represent a set of points so that all points inside a query range can be reported efficiently. In contrast, our goal is to sparsify the (flat) partition induced by a given set of hyperplanes. A different type of multidimensional partitioning is investigated in [15, 21], where the goal is to partition a  $d$ -dimensional array, with nonnegative entries, into a fixed number of subarrays with roughly equal weights. Those approaches are motivated by an interest in constructing a compact *histogram* of the multidimensional data. In contrast, in our anonymizing partition, the goal is not to balance the weight but rather to avoid small-weight regions. In addition, while in the histogram problem the array is partitioned into arbitrarily arranged rectangular boxes, in our setting the partition is induced by full hyperplanes. In [17], LeFevre et al. also consider an anonymity-related partitioning problem, but they compute an arbitrary rectangular subdivision, not an arrangement of hyperplanes. They also show that their problem is NP-complete, but their proof requires the dimension of the space to be unbounded – in particular,  $d \geq n$  in the constructed instances. In contrast, we show our problem is NP-complete even for dimension  $d = 2$ .

## 1.1 Our Contributions

We now discuss the main results of this paper. Given a set  $P$  of  $n$  points in  $\mathbb{R}^d$ , a set  $H$  of  $m$  axis-parallel hyperplanes, and an integer target  $0 < t \leq n$ , we define a *deletion set* to be a subset of hyperplanes so that no cell in the remaining arrangement is deficient. The goal of the ANONYMITY-PRESERVING PARTITION problem is to find a minimum deletion set.



For notational convenience, suppose  $H_i \subseteq H$  is the subset of planes whose normals are parallel to the  $i$ -th coordinate axis, for  $i = 1, 2, \dots, d$ . Then, if the number of nonempty families  $H_i$  is  $p$ , then our problem is essentially a  $p$ -dimensional problem, for  $p \leq d$ . If  $p = 1$ , then it is easy to solve the problem optimally using dynamic programming in time  $O(nm)$ . Surprisingly, we show that the problem is already NP-hard if  $p = 2$ , namely, the input is two-dimensional.

We then propose a polynomial-time  $p$ -approximation algorithm for the problem for any fixed  $p \leq d$ . For this, we reduce the problem to a variant of the well-known HITTING SET problem which we show to have an approximation algorithm using LP rounding. The approximate solution for the reduced HITTING SET instance will yield a  $p$ -approximate solution for our problem. We also give an FPT algorithm for the problem, with running time  $(2d - 0.924)^k m^{O(1)} + O(n)$ . From now on, for convenience of the reader, we assume that  $p = d$  and state the results in terms of  $d$ .

Finally, we also introduce an interval anonymity problem in one dimension which can be viewed as a geometric dual of ANONYMITY-PRESERVING PARTITION when  $d = 1$  – the roles of lines and points are interchanged. Specifically, we are given a set  $P$  of  $n$  points, which we call *markers*, a multiset  $S$  of  $m$  *segments* (intervals) on the real line  $\mathbb{R}$ , and an (integer) anonymity parameter  $0 < t \leq n$ . The set of markers  $P$  partitions  $S$  into equivalence classes, where two segments  $s, s'$  are in the same class if they contain the same set of marker points, namely,  $s \cap P = s' \cap P$ . We say a segment is *nonempty* if it contains at least one marker. We call an equivalence class consisting of nonempty segments *deficient* if it contains less than  $t$  segments. In the INTERVAL ANONYMITY problem, the aim is to remove a minimum number of points from  $P$  so that every nonempty segment of  $S$  belongs to a non-deficient equivalence class. For motivation, one can imagine segments as movement trajectories of  $m$  users, and markers as location sensors, and the goal is to report user locations in such a way that each user has  $t$ -anonymity. Somewhat surprisingly, this one-dimensional problem turns out to be NP-hard.

## 2 NP-Hardness of Anonymity-Preserving Partition

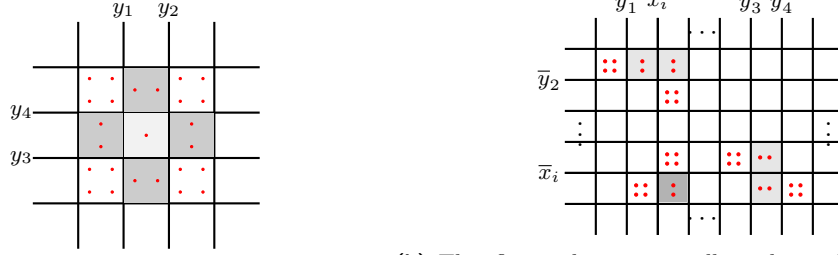
In this section, we prove that ANONYMITY-PRESERVING PARTITION is NP-hard even in two dimensions. This problem is easy to solve in one dimension, which we discuss in Section 3.

Let  $(P, H, t)$  be an instance of ANONYMITY-PRESERVING PARTITION in two dimensions. Without loss of generality, we assume that  $H_1, H_2 \subseteq H$  are the sets of hyperplanes having normals parallel to the  $x$ - and  $y$ -axes, respectively. Furthermore, we denote the hyperplanes  $h_1 \in H_1$  and  $h_2 \in H_2$  by equations of the form  $h_1 = x'$  and  $h_2 = y'$ , respectively, where  $x', y' \in \mathbb{R}$  are constants. To show NP-hardness, we reduce from a structured variant of SAT called LINEAR NEAR EXACT SATISFIABILITY (LNES), which is known to be NP-complete [11]. The main idea here is to associate literals with hyperplanes and clauses with deficient cells, and to make satisfying assignments correspond to deletion sets.

► **Theorem 1.** ANONYMITY-PRESERVING PARTITION is NP-complete for all dimensions  $d \geq 2$ .

**Proof.** Clearly, the decision version of our problem belongs to NP. We now show NP-hardness for just  $d = 2$  as these instances can be easily embedded into any higher dimension. An instance  $J$  of LNES consists of  $5s$  clauses, for  $s \in \mathbb{N}$ , and is denoted by

$$\mathcal{C} = \{U_1, V_1, U'_1, V'_1, \dots, U_s, V_s, U'_s, V'_s\} \cup \{C_1, \dots, C_s\}.$$



(a) This figure shows nine nonempty cells corresponding to an auxiliary clause  $C := (y_1 \vee y_2 \vee y_3 \vee y_4)$ . The middle cell with one point is an *auxiliary cell*, and the four gray cells on its boundary are *shadow auxiliary cells*. The nonempty white cells denote the *helpers*.

(b) This figure shows core cells and variable cells. We consider the following four core clauses:  $U_i := (\bar{y}_1 \vee x_i)$ ,  $V_i := (\bar{y}_2 \vee x_i)$ ,  $U'_i := (\bar{y}_3, \bar{x}_i)$ ,  $V'_i := (\bar{y}_4, \bar{x}_i)$ . Moreover, we assume the literals  $y_1, y_3, y_4$  are associated with the hyperplanes in  $H_2$  forming the auxiliary cells, and  $y_2$  is associated with the hyperplane in  $H_1$ . The *core cells* are colored light gray, and the *variable cell* is colored dark gray.

■ **Figure 2** Example construction of auxiliary, core, and variable cells.

We refer to the first  $4s$  clauses as the *core* clauses, and the remaining  $s$  clauses as the *auxiliary* clauses. The set of variables consists of  $s$  *main variables*  $x_1, \dots, x_s$  and  $4s$  *shadow variables*  $y_1, \dots, y_{4s}$ . Each core clause consists of two literals (one corresponding to a main variable, and the other to a shadow variable) and it has the following structure:  $\forall i \in [s], U_i \cap V_i = \{x_i\}$  and  $U'_i \cap V'_i = \{\bar{x}_i\}$ .

Each main variable  $x_i$  occurs exactly twice as a positive literal and twice as a negative literal. The main variables only occur in the core clauses. Each shadow variable makes two appearances: as a positive literal in an auxiliary clause and as a negative literal in a core clause. Each auxiliary clause consists of four literals, each corresponding to a positive occurrence of a shadow variable.

The LNES problem asks whether, given a set of clauses with the aforementioned structure, there exists an assignment  $\tau$  of truth values to the variables such that *exactly one* literal in every core clause and *exactly two* literals in every auxiliary clause evaluate to TRUE under  $\tau$ .

**Construction.** We construct the set of hyperplanes  $H = H_1 \cup H_2$  by adding hyperplanes placed at integer coordinates starting at one, i.e.,  $H = \{h_1 = x' \mid x' \in \{1, 2, \dots, 3qs\}\} \cup \{h_2 = y' \mid y' \in \{1, 2, \dots, 3qs\}\}$ . These hyperplanes are numbered from left to right and top to bottom. For  $i, j \in \mathbb{N}$ , let  $\square_{(i,j)}$  denote a  $1 \times 1$  cell  $[i, i+1] \times [j, j+1]$  on  $\mathcal{A}(H)$ . We set  $q = 5s + 4$  (recall  $s$  is a parameter from the LNES instance) which is sufficiently larger than the desired size of the deletion set ( $5s$ ). During the construction, we use  $q$  hyperplanes between a cluster of non-empty cells introduced so the sets remain independent, i.e., deleting lines from one cluster does not affect the other. We set the target  $t$  to 4. We associate a hyperplane from  $H$  with each of the  $10s$  literals ( $H$  may contain additional hyperplanes which are not associated with any literal). Of these  $10s$  hyperplanes,  $8s$  are associated with the shadow literals and  $2s$  with the main literals. By default, each cell in  $\mathcal{A}(L)$  is empty. We introduce the nonempty cells and organize them into the following three groups (also, we describe the locations of the  $4s$  hyperplanes associated with the positive shadow literals in the auxiliary cells group, and the locations of the remaining hyperplanes in the core cells group):

- **Auxiliary cells:** We introduce a set of nine nonempty cells for each auxiliary clause. For  $i \in [s]$ , we call  $\square_{(qi, qi)}$  the *auxiliary cell* for clause  $C_i$ . The first two literals in  $C_i$  are associated with the two adjacent hyperplanes  $x = qi$  and  $x = qi + 1$  from  $H_1$ , and the

remaining two literals are associated with the hyperplanes  $y = qi$  and  $y = qi + 1$  from  $H_2$ .<sup>1</sup> We add one point to  $\square_{(qi, qi)}$  (note that  $1 < t/2$ ). Moreover, we add  $t/2$  points to each of  $\square_{(qi-1, qi)}$ ,  $\square_{(qi+1, qi)}$ ,  $\square_{(qi, qi+1)}$ ,  $\square_{(qi, qi-1)}$ , and refer to them as *shadow cells*, while we add  $t$  points to each of  $\square_{(qi-1, qi-1)}$ ,  $\square_{(qi-1, qi+1)}$ ,  $\square_{(qi+1, qi-1)}$ ,  $\square_{(qi+1, qi+1)}$ , and refer to them as *helpers* (see Fig. 2a). Observe that for each  $C_i$ , one needs to remove at least two of the four hyperplanes associated with the shadow literals appearing in  $C_i$  forming the corresponding auxiliary cell  $\square_{(qi, qi)}$ . This is to ensure that we have at least  $t$  points in all the remaining cells among the nine initial cells without exceeding the 5s deletion limit.

- **Core cells:** For each *core clause*, we introduce two nonempty cells. For each *main variable*  $x_i$ , we construct eight cells for the four core clauses  $U_i, V_i, U'_i, V'_i$  together. Without loss of generality, let  $U_i := (\bar{y}_1 \vee x_i)$ , and  $V_i := (\bar{y}_2 \vee x_i)$ . Define  $z_i = q(s + 2i)$  for convenience.<sup>2</sup> We call  $\square_{(z_i, z_i)}$  and  $\square_{(z_i+1, z_i)}$  the *core cells* corresponding to the clauses  $U_i, V_i$ , respectively. We add two points to each of these cells and associate the common hyperplane  $x = z_i + 1$  from  $H_1$  to the literal  $x_i$ . Next, two cases arise according to the orientation of the hyperplanes associated with the literals  $y_1, y_2$ , say  $p(y_1), p(y_2)$  (recall that orientation of these hyperplanes is decided while constructing the *auxiliary cells*):

1.  $p(y_1) \in H_1$ : We associate the hyperplane  $y = z_i$  from  $H_2$  which forms the upper boundary of  $\square_{(z_i, z_i)}$  with  $\bar{y}_1$ , and add four points to  $\square_{(z_i, z_i-1)}$ . Similarly, if  $p(y_2) \in H_1$ , we associate the hyperplane  $y = z_i + 1$  from  $H_2$  which forms the lower boundary of  $\square_{(z_i+1, z_i)}$  with  $\bar{y}_2$ , and add four points to  $\square_{(z_i+1, z_i+1)}$ .
2.  $p(y_1) \in H_2$ : We associate the hyperplane  $x = z_i$  from  $H_1$  which is the left boundary of  $\square_{(z_i, z_i)}$  with  $\bar{y}_1$ , and add four points to  $\square_{(z_i-1, z_i)}$ . Similarly, if  $p(y_2) \in H_2$ , we associate the hyperplane  $x = z_i + 1$  from  $H_1$  which is the right boundary of  $\square_{(z_i+1, z_i)}$  with  $\bar{y}_2$ , and add four points to  $\square_{(z_i+2, z_i)}$ .

The construction above ensures that hyperplanes associated with  $y_i$  and  $\bar{y}_i$  have *orthogonal* normals. We call the two nonempty cells introduced in either of the cases above as *shadow core cells*.

We associate the literal  $\bar{x}_i$  to the hyperplane  $y = z_i + q + 1$  from  $H_2$ , and use a procedure symmetric to the one above to construct four nonempty cells. Here,  $\square_{(z_i+1, z_i+q)}$  and  $\square_{(z_i+1, z_i+q+1)}$  are *core cells* for the clauses  $U'_i, V'_i$ , respectively (note that, here, the two core cells are one below the other as opposed to side-by-side as we did for  $x_i$ ). We complete the rest of the construction as described above. For an example, refer to Fig. 2b. Observe that removal of the hyperplane associated with the positive literal  $x_i$  makes both core cells (corresponding to  $U_i, V_i$ ) non-deficient as these are merged together. Alternatively, removing the hyperplane corresponding to each  $\bar{y}_1, \bar{y}_2$  makes the core cells non-deficient. The case of the literal  $\bar{x}_i$  and the core clauses  $U'_i, V'_i$  is symmetric.

- **Variable cells:** Recall that our construction of core cells ensures that for each main and shadow variable, the two hyperplanes associated with its two literals have orthogonal normals. Next, we introduce three nonempty cells for each of these variables. For each main variable  $x_i$ , the two hyperplanes associated with  $x_i$  and  $\bar{x}_i$  form the top and left boundaries of the cell  $\square_{(z_i+1, z_i+q+1)}$ . We refer to  $\square_{(z_i+1, z_i+q+1)}$  as a *variable cell*, and add two points to it. Furthermore, we add four points each to  $\square_{(z_i, z_i+q+1)}$ ,  $\square_{(z_i+1, z_i+q)}$ , and call them *literal cells*. These cells are adjacent to the left and the upper boundaries of the variable cell. Refer to Fig. 2b.

<sup>1</sup> If for a *main variable*  $x_i$ , the two shadow variables appearing in the core clauses  $U_i, V_i$  are also the first two or the last two literals for some auxiliary clause, then we associate those literals with a pair of orthogonal hyperplanes  $y = qi$  and  $x = qi$  rather than with the default of a pair of parallel hyperplanes described earlier.

<sup>2</sup> Observe that we add an offset of  $qs$  so that the core and auxiliary cells are independent.

Next, we repeat the same procedure of introducing three nonempty cells for each shadow variable at the intersection of the hyperplanes associated with its literals. Notice that it is imperative to remove at least one of the two hyperplanes associated with the two literals for **every** variable so as to merge and make the variable cell non-deficient while staying within the deletion budget of  $5s$  hyperplanes.

For the constructed ANONYMITY-PRESERVING PARTITION instance  $I$ , we ask if there exists a deletion set with size at most  $5s$ . We now turn to the argument of equivalence.

**Forward direction.** Recall that we start with an instance  $J$  of LNES. Let  $\tau$  be a satisfying assignment for  $J$ ; then we claim that the set  $S$  consisting of  $5s$  hyperplanes associated with  $5s$  literals set to TRUE under  $\tau$  gives a valid deletion set for  $I$ . We now show that  $\mathcal{A}(H \setminus S)$  does not contain any deficient cell. First, we observe that  $\tau$  sets exactly one of the two literals associated with each of the  $5s$  variables to TRUE (since  $\tau$  is a valid assignment). Hence, the deficient variable cell introduced for each variable (see the dark gray cell from Fig. 2b) is merged with one of the literal cells and becomes non-deficient. Next, for each auxiliary clause  $C_i$  for  $1 \leq i \leq s$ , exactly two literals are set to TRUE. From the construction of the auxiliary cells group, one can verify that removing exactly two of the four hyperplanes associated with the four literals in  $C_i$  makes the auxiliary cell and the four shadow cells non-deficient (see Fig. 2a). Similarly,  $\tau$  sets exactly one literal from each core clause to TRUE. Hence, we remove exactly one hyperplane on the boundary of each deficient core cell. Due to this, the core cell merges with either a shadow core cell or another core cell, making it non-deficient (see Fig. 2b). This accounts for all the deficient cells in  $I$ ; hence, we conclude our argument for the forward direction.

**Reverse direction.** Let  $S$  be a valid deletion set of size at most  $5s$ ; we construct an assignment  $\tau$  for  $J$  by setting the literals associated with hyperplanes in  $S$  to TRUE. From the construction of the variable cells, we first observe that  $S$  contains exactly one of the two hyperplanes associated with the two literals for each of the  $5s$  variables in  $J$  (since  $|S| \leq 5s$ ). Hence,  $S$  is a valid SAT assignment, i.e., each variable is either set to TRUE or FALSE. Next, using a counting argument, we show that  $\tau$  is a satisfying assignment for  $J$ . Recall that each main variable  $x_i$  occurs twice as a positive literal and twice as a negative literal in the core clauses. Hence, the  $s$  literals associated with the  $s$  main variables set to TRUE under  $\tau$  satisfy exactly  $2s$  core clauses. Next, for the remaining  $2s$  core clauses,  $\tau$  sets exactly one negative shadow literal appearing in each of those clauses to TRUE. This is because from the construction of a core cell corresponding to each core clause, at least one of the two hyperplanes associated with the literals in the clause must be in  $S$  (and literals corresponding to main variables cannot be set to TRUE for this set of core clauses). Similarly,  $\tau$  sets at least two positive shadow literals appearing in each auxiliary clause to TRUE. At this stage, we use a counting argument: Among the  $4s$  shadow literals set to TRUE under  $\tau$ , exactly  $2s$  negative shadow literals and exactly  $2s$  positive shadow literals are TRUE (due to the argument above). Hence, with  $s$  main literals and  $2s$  negative shadow literals set to TRUE, each core clause is satisfied exactly once. With  $2s$  positive shadow literals set to TRUE, each auxiliary clause is satisfied exactly twice. This completes the proof for the reverse direction. ◀

### 3 Approximation and FPT Algorithms

In this section, we present a  $d$ -approximation algorithm for ANONYMITY-PRESERVING PARTITION. We first note that an  $O(d)$ -approximation can be easily achieved using a HITTING SET approximation, since we have a set system of VC dimension  $O(d)$  [13, 8].

Unfortunately, the constant factors in these HITTING SET approximations tend to be large, and in fact a much simpler greedy algorithm can directly give us a  $2d$ -approximation as follows: while there exists a deficient cell  $C$ , we remove all of its (at most)  $2d$  bounding hyperplanes, and iterate until no deficient cell remains. The approximation guarantee follows because for each deficient cell, the optimal solution must remove at least one hyperplane and the greedy algorithm removes  $2d$  hyperplanes. Thus, the main challenge is to improve on this naive bound, which is the main result of this section.

Our algorithm first reduces the ANONYMITY-PRESERVING PARTITION problem to a special case of HITTING SET in which all sets have a small size, and then we design an LP-rounding-based algorithm to obtain a  $d$ -approximation for this problem. We also present a fixed-parameter tractable algorithm running in time  $(2d - 0.924)^k m^{O(1)} + O(n)$  parameterized by the solution size  $k$ .<sup>3</sup>

The one-dimensional case of ANONYMITY-PRESERVING PARTITION can be easily solved in linear time; please see Appendix A for a proof of the following result:

► **Theorem 2.** *The ANONYMITY-PRESERVING PARTITION problem in one dimension can be solved in time  $O(mn)$ , where  $m$  is the number of hyperplanes and  $n$  is the number of points. Further, if every cell in the arrangement is nonempty, then it can be solved in time  $O(m + n)$ .<sup>4</sup>*

### 3.1 A $d$ -Approximation Algorithm

We start by defining a HITTING SET variant. Given a universe of elements  $U$  and a family  $\mathcal{F}$  of subsets of  $U$ , the HITTING SET problem asks us to find a minimum-sized set  $S \subseteq U$  such that  $S$  intersects with every set in  $\mathcal{F}$ . When every set in  $\mathcal{F}$  has size at most  $l$ , we call it the  $l$ -HITTING SET problem.

► **Lemma 3.** *Given an instance  $(P, H, t)$  of the  $d$ -dimensional ANONYMITY-PRESERVING PARTITION problem, we can construct an instance  $(U, \mathcal{F})$  of  $2d$ -HITTING SET such that  $U = H$ ,  $|\mathcal{F}| \leq |H|^{2d}$ , and  $(U, \mathcal{F})$  has a hitting set of size  $k$  if and only if  $(P, H, t)$  has a deletion set of size  $k$ , for any  $k \in \mathbb{N}$ .*

**Proof.** Given an instance  $(P, H, t)$  of ANONYMITY-PRESERVING PARTITION, we construct a  $2d$ -HITTING SET instance with universe  $U = H$  and the family  $\mathcal{F}$  being the set of all nonempty subsets  $X$  of  $H$  such that  $\mathcal{A}(X)$  has a deficient cell and such that  $X$  contains at most two hyperplanes from each  $H_i$  with  $1 \leq i \leq d$ .

▷ **Claim 4.** If  $(P, H, t)$  has a deletion set of size  $k$ , then  $(U, \mathcal{F})$  has a hitting set of size  $k$ .

**Proof.** Let  $H' \subseteq H$  be a deletion set of size  $k$  for  $(P, H, t)$ . Then, there is no deficient cell in  $\mathcal{A}(H \setminus H')$ . Since  $U = H$ , we now show that  $H'$  is also a hitting set of  $(U, \mathcal{F})$ . Suppose not; then there is a set  $X$  in  $\mathcal{F}$  that has no hyperplanes from  $H'$  in it. We know by the construction of  $\mathcal{F}$  that  $X$  has a cell that is deficient in  $\mathcal{A}(X)$ . Observe that even if we add any new hyperplanes to the arrangement  $\mathcal{A}(X)$ , there will still be a deficient cell. Thus,  $\mathcal{A}(H \setminus H')$  will have a deficient cell, which contradicts our assumption that  $H'$  was a deletion set. ◁

<sup>3</sup> Fixed-parameter tractability (FPT) is studied in the realm of parameterized complexity. FPT algorithms admit running time of the form  $f(k)n^{O(1)}$ , where  $k$  is the parameter under consideration and  $n$  is the size of the instance [10].

<sup>4</sup> We assume the points and hyperplanes in the input are sorted.

$$\begin{array}{ll}
\text{minimize} & \sum_{h \in H} x_h \\
\text{s.t.} & \sum_{h \in F} x_h \geq 1 \quad \forall F \in \mathcal{F} \\
& x_h \in [0, 1] \quad \forall h \in H
\end{array}$$

■ **Figure 3** LP for  $2d$ -Hitting Set.

▷ **Claim 5.** If  $(U, \mathcal{F})$  has a hitting set of size  $k$ , then  $(P, H, t)$  has a deletion set of size  $k$ .

*Proof.* Let  $H'$  be a hitting set of  $(U, \mathcal{F})$  of size  $k$ . Since  $U = H$ , we now show that  $H'$  is also a deletion set of  $(P, H)$ . Suppose not; then there is a cell  $C$  that is deficient in  $\mathcal{A}(H \setminus H')$ . Let  $X$  be the set of hyperplanes adjacent to  $C$  in this arrangement. Since all the hyperplanes in  $H$  are axis parallel and we are in the  $d$ -dimensional version of the problem, it follows that  $X$  contains at most two hyperplanes from each  $H_i$  with  $1 \leq i \leq p$ . Also, observe that  $\mathcal{A}(X)$  has the cell  $C$  in it. Since  $C$  is deficient, by construction of the family  $\mathcal{F}$ , we know  $X$  must be in  $\mathcal{F}$ . But since  $H' \cap X = \emptyset$ , this contradicts the fact that  $H'$  is a hitting set. ◁

This completes the proof of Lemma 3. Observe that the  $VC$ -dimension of the constructed set system is  $2d$ , hence, rounding algorithm from [13] would give an  $O(d)$ -approximation. ◀

We now observe the following simple fact:

► **Lemma 6.** For each set  $X \in \mathcal{F}$  of the  $2d$ -HITTING SET instance  $(U, \mathcal{F})$  obtained by applying the reduction in Lemma 3 to  $(P, H, t)$ , it holds that  $|H_i \cap X| \leq 2$ , for  $1 \leq i \leq d$ .

Our approximation algorithm uses LP rounding; see Figure 3. While the integrality gap of this LP is known to be at most  $d$ , the proof is *non-constructive* [3, Theorem 1]<sup>5</sup> and therefore it is not known how to efficiently compute a rounded solution with approximation factor less than  $2d$ . (The size of each set in the LP is  $2d$  and so in any fractional LP solution each set is only guaranteed to have some variable with value at least  $\frac{1}{2d}$ . Thus a straightforward rounding of the LP solution only leads to a  $2d$ -approximation.) Our main contribution, therefore, is to design a polynomial-time rounding algorithm that achieves a  $d$ -approximation for  $2d$ -HITTING SET, and thus also for  $d$ -dimensional ANONYMITY-PRESERVING PARTITION.

► **Theorem 7.** For every fixed dimension  $d \geq 2$ , there exists a polynomial-time algorithm that given a  $d$ -dimensional ANONYMITY-PRESERVING PARTITION instance, computes a deletion set with size at most  $d$  times the optimal size.

**Proof.** We describe our rounding algorithm for  $d = 2$  and defer the general case to Appendix B. We first use Lemma 3 to reduce the 2-dimensional ANONYMITY-PRESERVING PARTITION instance to a HITTING SET instance  $(U = H_1 \cup H_2, \mathcal{F})$ . Observe that by Lemma 6, for each set  $X \in \mathcal{F}$ , we have  $|H_1 \cap X| \leq 2$  and  $|H_2 \cap X| \leq 2$ . We now give a 2-approximation algorithm for  $(U, \mathcal{F})$  by extending the integrality gap result for the LP in [3] (see Figure 3).

<sup>5</sup> Note that in [3], Theorem 1 shows the integrality gap for a variant of hypergraph Vertex Cover. It is fairly straightforward to see that the Hitting Set instances obtained by applying the reduction in Lemma 3 can be equivalently expressed as instances of that same hypergraph Vertex Cover variant; hence, Lemma 3 also gives a reduction to hypergraph Vertex Cover.



For completeness, we first include the proof that the integrality gap is at most 2, and then describe our algorithm.

Let  $g : U \rightarrow [0, 1]$  be an optimal fractional hitting set of  $(U, \mathcal{F})$  with value  $\tau^*(U, \mathcal{F})$ . Also, let  $\tau(U, \mathcal{F})$  be the size of an optimal integral hitting set of  $(U, \mathcal{F})$ . Let  $B = \{(x_1, x_2) \in [0, \frac{1}{2}] \times [0, \frac{1}{2}] : x_1 + x_2 = \frac{1}{2}\}$ , and for each  $x = (x_1, x_2) \in B$ , let

$$T(x) = \{h \in H_1 : g(h) \geq x_1\} \cup \{h \in H_2 : g(h) \geq x_2\}.$$

In other words,  $B$  can be viewed as the set of all points on the line segment  $x_1 + x_2 = \frac{1}{2}$  for  $x_1, x_2 \in [0, \frac{1}{2}]$ , and  $T(x)$  can be viewed as the set obtained by rounding  $g$  using  $x_i$  as the threshold for each  $H_i$ .

We now prove that for any  $x \in B$ ,  $T(x)$  is a hitting set of  $(U, \mathcal{F})$ . Suppose not; then there must be a set  $X \in \mathcal{F}$  such that  $X \cap T(x) = \emptyset$ . By the definition of  $T(x)$ , for each hyperplane  $h \in X \cap H_i$ ,  $i \in \{1, 2\}$ , it holds that  $g(h) < x_i$ . Combining this with the fact that  $|X \cap H_1| \leq 2$  and  $|X \cap H_2| \leq 2$ , we get  $\sum_{h \in X} g(h) < 2(x_1 + x_2) = 1$ . This contradicts the fact that  $g$  is a feasible fractional hitting set of  $(U, \mathcal{F})$ , and thus  $T(x)$  is a hitting set.

Observe that for any given  $a, b \in [0, 1/2]$  with  $a \leq b$ , for a uniformly random  $x = (x_1, x_2) \in B$ , we have  $Pr(a \leq x_i \leq b) = \frac{b-a}{1/2}$  for  $i \in \{1, 2\}$ , i.e.,  $x_1$  and  $x_2$  have a uniform distribution over the interval  $[0, 1/2]$ . We will now use a probabilistic argument to prove that the integrality gap is bounded by 2. If we choose a uniformly random  $x = (x_1, x_2)$  from  $B$ , and let  $E(\cdot)$  denote the expected value, then we have

$$\begin{aligned} \tau(U, \mathcal{F}) \leq E(|T(x)|) &= \sum_{h \in H_i, i \in \{1, 2\}} Pr(g(h) \geq x_i) = \sum_{h \in U} \min\left(1, \frac{g(h)}{1/2}\right) \\ &\leq \sum_{h \in U} 2g(h) = 2\tau^*(U, \mathcal{F}). \end{aligned}$$

Let  $T := \{T(x) : x \in B\}$ . By the above argument, there exists  $x \in B$  such that  $T(x)$  is a hitting set of size at most  $2\tau^*(U, \mathcal{F})$ . Thus, to get a 2-approximation we will show that  $|T| \leq 2m + 2$  and that  $T$  can be constructed in polynomial time (see Appendix B, Algorithm 1 for pseudocode). We now build a set  $B' \subset B$  of size at most  $2m + 2$  such that  $T' := \{T(x) : x \in B'\} = T$ . We include one point for each hyperplane  $h \in H_i$  with  $g(h) \leq 1/2$ , and we include an arbitrarily chosen point between each consecutive pair of these points on the line  $x_1 + x_2 = 1/2$ .

Formally, define  $B_1$  and  $B_2$  as follows: For each  $h \in H_1$ , add  $(g(h), 1/2 - g(h))$  to  $B_1$  if  $g(h) \leq 1/2$ , and for each  $h \in H_2$ , add  $(1/2 - g(h), g(h))$  to  $B_1$  if  $g(h) \leq 1/2$ . Finally, add the point  $(1/2, 0)$  to  $B_1$ . Choose a value  $\varepsilon > 0$  such that for any distinct  $(x_1, x_2), (x'_1, x'_2) \in B_1$ , we have  $\varepsilon < |x'_1 - x_1|$ . For each  $x = (x_1, x_2) \in B_1$  such that  $x_1 \neq 1/2$ , add  $(x_1 + \varepsilon, 1/2 - x_1 - \varepsilon)$  to  $B_2$ . Finally, add  $(0, 1/2)$  to  $B_2$ . Now let  $B' = B_1 \cup B_2$ .

We now prove that  $T' = T$ . We only need to argue that for all  $x \in B \setminus B'$ ,  $T(x) \in T'$ . Given  $x = (x_1, x_2) \in B \setminus B'$ , let  $x' = (x'_1, x'_2)$  be the pair in  $B_1$  having the largest  $x'_1$  such that  $x'_1 < x_1$ . If such an  $x'$  does not exist, then it is easy to see that  $T(y = (0, 1/2)) = T(x)$ . If  $x'$  exists, then  $T(y = (x_1 + \varepsilon, 1/2 - x_1 - \varepsilon)) = T(x)$  since  $x \notin B'$ . In both cases  $y$  is in  $B'$  and thus  $T(y) = T(x)$  is in  $T'$ . This proves that  $T' = T$  and that  $|T| \leq 2m + 2$ . Our approximation algorithm constructs  $T$  and outputs the set in  $T$  having the smallest size. This completes the proof for  $d = 2$ . The complete algorithm as well as the details of the general case for dimensions  $d > 2$  are presented in Appendix B. ◀

The approximation ratio in Theorem 7 is the best possible that can be obtained using the particular LP formulation from Fig. 3 because it has an integrality gap of  $d$  for the constructed hitting set instances [3].



### 3.2 Fixed-Parameter Tractable Algorithm

Given the equivalence of  $2d$ -HITTING SET and ANONYMITY-PRESERVING PARTITION (refer to Lemma 3), an FPT algorithm follows easily (when  $d$  is a constant). This is because the  $l$ -Hitting Set problem is known to admit an exact algorithm running in time<sup>6</sup>  $(l - 0.924)^k |U|^{O(1)}$  [14], where  $k$  is the size of the hitting set.

► **Theorem 8.** *The ANONYMITY-PRESERVING PARTITION problem in  $d$  dimensions can be solved in time  $(2d - 0.92)^k (m)^{O(1)} + O(n)$ , where  $k$  is the size a minimum deletion set,  $m$  is the number of hyperplanes, and  $n$  is the number of points.*

## 4 An NP-hard Anonymity Problem on the Line

In this section, we show that the INTERVAL ANONYMITY problem is NP-complete and give an exact algorithm running in time  $3.08^k n^{O(1)} + O(m)$ , where  $k$  is the solution size. Recall that here we are given a set  $P$  of  $n$  points, which we call *markers*, a multiset  $S$  of  $m$  segments (intervals) on the real line  $\mathbb{R}$ , and an integral anonymity parameter  $t > 0$ . For convenience, when we consider any set of points, we consider them to be ordered from left to right according to their relative positions on the line. The set of markers  $P$  partitions  $S$  into equivalence classes, where two segments  $s$  and  $s'$  are in the same class if they contain the same set of marker points, namely,  $s \cap P = s' \cap P$ . We call an equivalence class consisting of nonempty segments *deficient* if it contains less than  $t$  segments. The INTERVAL ANONYMITY problem asks us to remove a minimum number of points from  $P$  so that every segment of  $S$  belongs to a non-deficient equivalence class. We now show that INTERVAL ANONYMITY is NP-complete.

► **Theorem 9.** *INTERVAL ANONYMITY is NP-complete, and is NP-hard to approximate within a factor of  $(2 - \varepsilon)$ , for any  $\varepsilon > 0$ , assuming the unique games conjecture (UGC).*

**Proof.** Clearly, the decision version of INTERVAL ANONYMITY belongs to NP. We give a polynomial-time approximation-preserving reduction from VERTEX COVER, which is NP-hard to approximate within a factor less than 2, assuming UGC [16].

**Construction.** Let  $G$  be a graph for which we seek a vertex cover of size at most  $k$ , and let  $n = |V(G)|$ . We can assume  $k \leq n$ . We construct an instance  $(P, S, t)$  of INTERVAL ANONYMITY having  $|P| = n + (n - 1)k$  and  $t = 2$ , where we seek the same solution size  $k$ . Let  $v_1, \dots, v_n$  be the vertices of  $G$ . For each vertex  $v_i$ , we create  $k + 2$  markers labeled as  $v_i, v_i^{(1)}, v_i^{(2)}, \dots, v_i^{(k+1)}$ , with one exception: the last vertex corresponds to just one marker,  $v_n$ . These markers occur in the following order:

$$v_1, v_1^{(1)}, \dots, v_1^{(k+1)}, \quad \dots, \quad v_{n-1}, v_{n-1}^{(1)}, \dots, v_{n-1}^{(k+1)}, v_n.$$

For each  $(v_i, v_j) \in E(G)$  with  $i < j$ , we add the following five (closed) intervals to  $S$ :  $[v_i, v_j]$ , two copies of  $[v_i, v_{j-1}^{(k+1)}]$ , and two copies of  $[v_i^{(1)}, v_j]$ . Since  $t = 2$ , we can see that the deficient intervals are exactly the ones of the form  $[v_i, v_j]$ .

<sup>6</sup> When  $2d \geq 15$ , there is an algorithm that runs in time  $O(c^k + m)$ ,  $c = d - 1 + \frac{1}{d-1}$  [19].

**Proof of equivalence.** For any vertex cover  $\mathcal{S}$  of  $G$ , if we remove the markers (without superscripts) corresponding to the vertices in  $\mathcal{S}$ , we obtain a solution for the Interval Anonymity instance. For the reverse direction, suppose we have a deletion set  $\overline{\mathcal{S}}$  for  $(P, S, t)$  of size at most  $k$ . Since our segments only have endpoints of the form  $v_i, v_i^{(1)},$  or  $v_i^{(k+1)}$ , we would have had to include in  $\overline{\mathcal{S}}$  all of the  $(k+1)$ -superscripted markers between two consecutive vertices if we wished for these to affect feasibility. Therefore, we can remove from  $\overline{\mathcal{S}}$  any superscripted markers and still maintain a feasible solution. Now,  $\overline{\mathcal{S}}$  naturally corresponds to a vertex cover for  $G$ . ◀

We now turn to a 4-approximation and an exact algorithm for the Interval Anonymity problem. Since this problem only cares about segments  $s$  such that  $s \cap P \neq \emptyset$ , we will from now on assume that for all segments  $s \in S$ ,  $s \cap P \neq \emptyset$ . Given an instance  $(P, S, t)$  of the Interval Anonymity problem, we now associate a set of at most four markers from  $P$  to every equivalence class  $X$ . We denote this set by  $M_X$ . Let  $s$  be a segment in  $X$ , and let  $l$  and  $r$  be the leftmost and the rightmost markers in the set  $s \cap P$ . Also, let  $l'$  and  $r'$  be the markers in  $P$  to the left of  $l$  and to the right of  $r$ , respectively, if they exist. Then,  $M_X = \{l', l, r, r'\}$  is the set containing these markers. Note that  $l$  might be equal to  $r$  and  $l'$  and  $r'$  might not exist, and thus  $M_X$  is a set of size at most four.

**4-Approximation.** The idea that each equivalence class can be associated with a set of at most four markers immediately gives us a polynomial-time 4-approximation algorithm and an exact algorithm running in time  $4^k(m+n)^{O(1)}$ , where  $k$  is the size of a minimum deletion set. The key here is to observe that (i) All segments in an equivalence class will remain in the same equivalence class in the final solution, and (ii) In order to make a deficient equivalence class  $X$  non-deficient, we need to remove at least one of the markers from  $M_X$ .

Then, the 4-approximation algorithm is as follows: (i) Initialize the deletion set  $D = \emptyset$ ; (ii) Repeatedly pick an arbitrary deficient equivalence class  $X$  and add all the markers in  $M_X$  to  $D$ , as long as there is a deficient equivalence class; (iii) Finally, output  $D$ . For the exact algorithm, instead of adding all of the markers from  $M_X$  to the deletion set, we guess which one of these markers to add to the deletion set (branching).

We obtain a better exact algorithm for this problem, similarly to the ANONYMITY-PRESERVING PARTITION problem, by reducing to 4-Hitting Set.

► **Theorem 10.** *The INTERVAL ANONYMITY problem can be solved in time  $3.08^k n^{O(1)} + O(m)$ , where  $k$  is the size a minimum deletion set.*

**Proof.** We first reduce our problem to 4-HITTING SET and then use the known  $(3.08)^k |U|^{O(1)}$  time algorithm [14] for 4-HITTING SET to solve our problem. Our focus now is to describe the reduction. Given an instance  $(P, S, t)$  of the INTERVAL ANONYMITY problem, we construct a 4-HITTING SET instance with universe  $U = P$  and family  $\mathcal{F}$  being the set of all nonempty subsets  $Q$  of  $P$  of size at most four such that the instance  $(Q, S, t)$  contains some deficient equivalence class.

Now we prove the forward direction: If  $(P, S, t)$  has a deletion set of size  $k$ , then  $(U, \mathcal{F})$  has a hitting set of size  $k$ . Let  $P' \subseteq P$  be a deletion set of size  $k$  of  $(P, S, t)$ . Then, there is no equivalence class in  $(P \setminus P', S, t)$  that is deficient. Since  $U = P$ , we now show that  $P'$  is also a hitting set of  $(U, \mathcal{F})$ . Suppose not; then there is a set  $Q \in \mathcal{F}$  that contains no markers from  $P'$ . We know by construction of  $\mathcal{F}$  that there is some deficient equivalence class  $X$  in  $(Q, S, t)$ . Let  $s$  be a segment in  $X$ , and let  $X'$  be the equivalence class that  $s$  belonged to in  $(P \setminus P', S, t)$ . Since segments in  $X'$  always remain together in their resulting equivalence class even after removing additional markers, it is easy to see that if  $X'$  is not deficient in  $(P \setminus P', S, t)$ , then  $X$  is not deficient in  $(Q, S, t)$ . This contradicts the fact that  $X$  is deficient and thus completes the forward direction.

Next, we show the reverse direction: If  $(U, \mathcal{F})$  has a hitting set of size  $k$ , then  $(P, S, t)$  has a deletion set of size  $k$ . Let  $P'$  be a hitting set of  $(U, \mathcal{F})$  of size  $k$ . Since  $U = P$ , we now show that  $P'$  is also a deletion set of  $(P, S, t)$ . Suppose not; then there is a deficient equivalence class  $X$  in  $(P \setminus P', S, t)$ . We show that  $M_X$  from  $(P \setminus P', S, t)$  belongs to  $\mathcal{F}$ , thus contradicting the fact that  $P'$  is a hitting set of  $(U, \mathcal{F})$  since  $M_X$  does not have any marker from  $P'$ . To satisfy an equivalence class  $E$ , at least one of the markers in  $M_E$  must be deleted. Therefore, deleting all markers from  $P \setminus P'$  except those from  $M_X$  will make  $X$  a deficient equivalence class in  $(M_X, S, t)$ . Thus, by construction,  $M_X$  belongs to  $\mathcal{F}$ . ◀

## 5 Conclusion

We considered a natural multidimensional space partitioning problem, showed that it is NP-complete in all dimensions  $d \geq 2$ , and designed a  $d$ -approximation algorithm and an FPT algorithm parameterized by solution size. Although we described our results for the case  $p = d$ , it is easy to see that the algorithm in fact guarantees a  $p$ -approximation for the more general case, where  $p \leq d$  is the number of nonempty families of hyperplanes. We also showed that a simple INTERVAL ANONYMITY problem is NP-complete even in one dimension, and gave approximation and FPT algorithms for that as well. Improving our approximation factors is an interesting open problem.

---

## References

- 1 P K Agarwal. Geometric Partitioning and Its Applications. *Discrete and Computational Geometry: Papers from the DIMACS Special Year (J. Goodman, R. Pollack, and W. Steiger, eds.)*, pages 1–37, 1991.
- 2 Pankaj K Agarwal and Micha Sharir. Applications of a new space-partitioning technique. *Discrete & Computational Geometry*, 9(1):11–38, 1993.
- 3 Ron Aharoni, Ron Holzman, and Michael Krivelevich. On a Theorem of Lovász on Covers in  $r$ -partite Hypergraphs. *Combinatorica*, 16(2):149–174, 1996.
- 4 Walid G Aref and Ihab F Ilyas. Sp-gist: An extensible database index for supporting space partitioning trees. *Journal of Intelligent Information Systems*, 17(2-3):215–240, 2001.
- 5 J. Baltés and J. Anderson. Flexible binary space partitioning for robotic rescue. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 4, pages 3144–3149 vol.3, 2003.
- 6 Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The  $R^*$ -tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, 1990.
- 7 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- 8 Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
- 9 J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.
- 10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 11 Pratyush Dayal and Neeldhara Misra. Deleting to Structured Trees. In *International Computing and Combinatorics Conference*, pages 128–139. Springer, 2019.
- 12 Adrian Dumitrescu, Joseph SB Mitchell, and Micha Sharir. Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. *Discrete & Computational Geometry*, 31(2):207–227, 2004.

- 13 Guy Even, Dror Rawitz, and Shimon Moni Shahar. Hitting sets when the vc-dimension is small. *Information Processing Letters*, 95(2):358–362, 2005.
- 14 Fedor V Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. *Theoretical Computer Science*, 411(7-9):1045–1053, 2010.
- 15 Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On Approximating Rectangle Tiling and Packing. In *Proceedings of the ninth annual ACM-SIAM Symposium On Discrete Algorithms*, volume 95, page 384. SIAM, 1998.
- 16 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within  $2 - \epsilon$ . *Journal of Computer and System Sciences*, 74(3):335–349, 2008.
- 17 Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional  $k$ -anonymity. In *22nd International conference on data engineering (ICDE'06)*, pages 25–25. IEEE, 2006.
- 18 Jiri Matousek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.
- 19 Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003.
- 20 Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.
- 21 Adam Smith and Subhash Suri. Rectangular tiling in multidimensional arrays. *Journal of Algorithms*, 37(2):451–467, 2000.
- 22 Marc Van Kreveld, Otfried Schwarzkopf, Mark de Berg, and Mark Overmars. *Computational geometry algorithms and applications*. Springer, 2000.

## A Proof of Theorem 2

We show that the Anonymity-Preserving Partition problem is easy to solve in the one-dimensional case in time  $O(mn)$ . Furthermore, this special case can be solved in time  $O(m + n)$  if every cell in the arrangement is nonempty. In both cases, we assume the points and hyperplanes in the input are pre-sorted.

**Proof (of Theorem 2).** We design a dynamic-programming algorithm to solve the problem in the one-dimensional case. Let  $i$  be the dimension in which we have a nonempty set of hyperplanes. We have  $m = |H_i| = |H|$ . We will denote the cells by  $f_1, \dots, f_{m+1}$  and the hyperplanes by  $h_1, \dots, h_m$ , so that they occur in the following order in space:

$$f_1, h_1, f_2, h_2, \dots, h_m, f_{m+1}.$$

Let  $n_i$  be the number of points in the cell  $f_i$ . We will think of hyperplanes and cells with smaller indices in this ordering as being “to the left.”

For each  $1 \leq i \leq m + 1$ , let  $L_i$  be the set of hyperplanes to the left of the cell  $f_i$ . We have  $L_1 = \emptyset$ . For a set of hyperplanes  $H'$ , let  $f_i(H')$  denote the cell containing  $f_i$  in the arrangement  $\mathcal{A}(H \setminus H')$ . For example, if  $H' = \{h_1\}$ , then  $f_2(H')$  is the cell formed by the union of  $f_1$  and  $f_2$ . For every  $1 \leq i \leq m + 1$  and every  $0 \leq s \leq t$ , we define the following value:

$f(i, s) =$  minimum possible size of a set  $H' \subseteq L_i$  such that in the arrangement  $\mathcal{A}(H \setminus H')$ , any nonempty cell to the left of  $f_i(H')$  contains at least  $t$  points, and the cell  $f_i(H')$  contains at least  $s$  points.

## 32:14 Anonymity-Preserving Space Partitions

The value we need to compute is  $f(m+1, t)$ . We compute  $f(m+1, t)$  using the following recursive formula:

$$f(i, s) = \begin{cases} 0 & \text{if } i = 1 \text{ and } s \leq n_1 \\ \infty & \text{if } i = 1 \text{ and } s > n_1 \\ \min(f(i-1, 0) + 1, f(i-1, t)) & \text{if } i > 1 \text{ and } s \leq n_i \\ f(i-1, s - n_i) + 1 & \text{if } i > 1 \text{ and } s > n_i. \end{cases}$$

The return value  $f(m+1, t)$  is always finite since we assume  $n \geq t$ . This concludes the algorithm – we leave the formal proof of correctness to the reader. It is easy to see that the running time is  $O(mt + n)$ , which is bounded by  $O(mn)$ .

We now proceed to the case when the instance is not only one-dimensional, but also has the property that every cell in the arrangement is nonempty. In this case, the problem can be solved by a greedy algorithm, which proceeds as follows:

- Initially, set  $q = 1$  and set  $S = \emptyset$ .
- Repeat the following steps while  $q \leq m + 1$ :
  - Set  $j$  to be the smallest  $j$  such that  $\sum_{i=q}^j n_i \geq t$ . Set  $S' = \{h_q, \dots, h_{j-1}\}$ . (If  $j = q$ , then  $S'$  is empty.) If there is no such  $j$ , this means we have reached the last of the cells. In that case, set  $j$  to be the largest  $j$  such that  $\sum_{i=j}^{m+1} n_i \geq t$ , set  $S' = \{h_j, \dots, h_m\}$ , and break once this iteration is complete.
  - Set  $S = S \cup S'$ .
  - Set  $q = j + 1$ .
- Return  $S$ .

Note that there always exists a  $j$  such that  $\sum_{i=j}^{m+1} n_i \geq t$  since we assume  $\sum_{i=1}^{m+1} n_i = n \geq t$ . The formal proof of correctness is straightforward, and we leave it to the reader. ◀

### B Proof of Theorem 7 for $d \geq 3$

In this section, we prove Theorem 3 for  $d \geq 3$  and provide the pseudocode for the  $d = 2$  case. Recall that Theorem 7 promises a  $d$ -approximation algorithm for the  $d$ -dimensional Anonymity-Preserving Partition problem.

**Proof (of Theorem 3 – for  $d \geq 3$ ).** Given an instance  $(P, H, t)$  of the  $d$ -dimensional Anonymity-Preserving Partition problem, we use the reduction in Lemma 3 to obtain a  $2d$ -Hitting Set instance  $(U, \mathcal{F})$ . Recall that  $U = H = \bigcup_{1 \leq i \leq d} H_i$ , i.e.,  $U$  is a union of  $d$  disjoint sets of hyperplanes  $H_i$ .

Next, we partition  $U$  into three sets  $S_1, S_2, S_3$  such that for all  $X \in \mathcal{F}$ ,  $|X \cap S_i| \leq d$  for  $1 \leq i \leq 3$ . When  $d$  is even, we let

$$S_1 = \bigcup_{1 \leq i \leq \frac{d}{2}} H_i, \quad S_2 = \bigcup_{\frac{d}{2} + 1 \leq i \leq d} H_i, \quad S_3 = \emptyset.$$

When  $d$  is odd, we let

$$S_1 = \bigcup_{1 \leq i \leq \lfloor \frac{d}{2} \rfloor} H_i, \quad S_2 = \bigcup_{\lfloor \frac{d}{2} \rfloor + 1 \leq i \leq d-1} H_i, \quad S_3 = H_d.$$

We define  $s_i = \max_{X \in \mathcal{F}} |X \cap S_i|$ , for  $i \in \{1, 2, 3\}$ . From Lemma 2, we know that for all  $X \in \mathcal{F}$ ,  $|X \cap H_i| \leq 2$ ; hence,  $s_1 + s_2 + s_3 \leq 2d$ . We now describe a  $d$ -approximation algorithm for  $(U, \mathcal{F})$ . To this end, we first use a result from [3] which bounds the integrality gap for the

LP from Fig. 3 on the instance  $(U, \mathcal{F})$  by  $d$ . For completeness, we include the proof from [3], and then build upon it to give an approximation algorithm.

Let  $g : U \rightarrow [0, 1]$  be an optimal fractional hitting set of  $(U, \mathcal{F})$  with value  $\tau^*(U, \mathcal{F})$ . Furthermore, let  $\tau(U, \mathcal{F})$  be the size of an optimal integral hitting set. We now construct a set  $B \subseteq [0, 1/d]^3$ . Fix four points:

$$q_1 = \left( \frac{s_1 + s_2 - s_3}{2ds_1}, 0, \frac{1}{d} \right), \quad q_2 = \left( \frac{1}{d}, \frac{s_2 + s_3 - s_1}{2ds_2}, 0 \right)$$

$$q_3 = \left( \frac{s_1 + s_3 - s_2}{2ds_1}, \frac{1}{d}, 0 \right), \quad q_4 = \left( 0, \frac{s_1 + s_2 - s_3}{2ds_2}, \frac{1}{d} \right)$$

and let

$$B^{(1)} = [q_1, q_2], \quad B^{(2)} = [q_3, q_4], \quad B^{(3)} = [q_1, q_3], \quad B^{(4)} = [q_2, q_4],$$

where  $[q_i, q_j]$  denotes the line segment between the points  $q_i$  and  $q_j$ . We define  $B = B^{(1)} \cup B^{(2)} \cup B^{(3)} \cup B^{(4)}$ .

Notice that the coordinates of  $q_1, q_2, q_3, q_4$  all satisfy the equation  $s_1x_1 + s_2x_2 + s_3x_3 = 1$ , and hence, this equation is satisfied by all tuples  $x = (x_1, x_2, x_3) \in B$ . Hence, using an argument similar to that used for  $d = 2$ , the sets  $T(x)$  constructed as follows are indeed hitting sets:

$$T(x) = \{h \in S_1 : g(h) \geq x_1\} \cup \{h \in S_2 : g(h) \geq x_2\} \cup \{h \in S_3 : g(h) \geq x_3\}.$$

Let  $T = \{T(x) : x \in B\}$ . Next, we define a probability measure  $\mu$  over  $B$  such that for any given  $a, b \in [0, 1/d]$  with  $a \leq b$ , for a randomly chosen tuple  $(x_1, x_2, x_3) \in B$ , we have  $\Pr(a \leq x_i \leq b) = \frac{b-a}{1/d}$  for  $1 \leq i \leq 3$ , i.e., the  $x_i$ 's have a uniform distribution over the interval  $[0, 1/d]$ . For  $1 \leq i \leq 4$ , let  $\mu_i$  be the uniform measures on the line segments  $B^{(i)}$  such that

$$\mu_1(B^{(1)}) = \mu_2(B^{(2)}) = \frac{(s_1 + s_3 - s_2)(s_2 + s_3 - s_1)}{2s_3(s_1 + s_2 - s_3)},$$

$$\mu_3(B^{(3)}) = \frac{(s_2 - s_3)(s_2 + s_3 - s_1)}{s_3(s_1 + s_2 - s_3)},$$

$$\mu_4(B^{(4)}) = \frac{(s_1 - s_3)(s_1 + s_3 - s_2)}{s_3(s_1 + s_2 - s_3)}.$$

We set  $\mu = \mu_1 + \mu_2 + \mu_3 + \mu_4$ . It can be verified that  $\sum_{i=1}^4 \mu_i(B^{(i)}) = 1$ , and hence,  $\mu(B) = 1$ . At this stage, to argue as in the case  $d = 2$  in order to show the bound on the integrality gap, it remains to show that  $x_i$  indeed has a uniform distribution on  $[0, 1/d]$  for all  $1 \leq i \leq 3$ .

It is easy to see that for a randomly chosen  $x = (x_1, x_2, x_3) \in B$ ,  $x_3$  has a uniform distribution over  $[0, 1/d]$ . This is because each  $\mu_i$  is a uniform measure over  $B^{(i)}$ , and  $x_3$  takes all values from  $[0, 1/d]$  on each  $B^{(i)}$  with  $1 \leq i \leq 4$ . It is easy to see that  $x_1$  is uniform over  $B^{(4)}$  using the same argument. Next, we observe that  $x_1$  is uniform on

## 32:16 Anonymity-Preserving Space Partitions

each of the line segments  $\left[0, \frac{s_1+s_3-s_2}{2ds_1}\right]$ ,  $\left[\frac{s_1+s_3-s_2}{2ds_1}, \frac{s_1+s_2-s_3}{2ds_1}\right]$ ,  $\left[\frac{s_1+s_2-s_3}{2ds_1}, \frac{1}{d}\right]$ . Recall that  $\mu_1(B^{(1)}) = \mu_2(B^{(2)})$ ; hence, the situation for the first and the third line segment is the same. Without loss of generality, assume that  $0 \leq s_3 \leq s_2 \leq s_1 \leq d$ . Hence, we only need to check

$$\frac{\mu_3(B^{(3)})}{\mu_2(B^{(2)})} = \frac{\frac{s_1+s_2-s_3}{2ds_1} - \frac{s_1+s_3-s_2}{2ds_1}}{\frac{s_1+s_3-s_2}{2ds_1}},$$

which indeed holds. Hence,  $x_1$  is uniformly distributed. With a similar argument, it can be shown that  $x_2$  is uniformly distributed. At this stage, similarly to the  $d = 2$  case, we can compute the expected size of  $T(x)$  to obtain the desired bound  $d$  on the integrality gap.

Next, we show that there are only  $O(m)$  distinct rounded hitting sets  $T(x)$  constructed using  $x \in B$ . Observe that while traversing on any line segment  $B^{(i)}$  for  $1 \leq i \leq 4$ , the hitting set  $T(x)$  may change at points  $x \in B^{(i)}$  for which there exists  $1 \leq j \leq 3$  such that  $g(h) = x_j$  for some  $h \in S_j$ , i.e., when the plane  $x_j = g(h)$  intersects  $B$ . Note that the hitting set  $T(x)$  does not change for the points on the open line segment between two consecutive intersection points on  $B^{(i)}$  obtained from the aforementioned planes (here, the open line segment  $(x_i, x_j)$  is the set of all points on the line segment  $[x_i, x_j]$  except for the endpoints). Since each such plane can have at most four intersection points with  $B$ , the number of distinct rounded solutions is  $O(m)$ , where  $m = |U|$ .

We iterate through all distinct rounded solutions and return a hitting set with minimum cardinality. This completes the proof of Theorem 7.  $\blacktriangleleft$

**Algorithm 1** 2-approximation for Anonymity-Preserving Partition in 2 dimensions.

- 
- 1: **Input:** Anonymity-Preserving Partition instance  $(P, H = H_1 \cup H_2, t)$
  - 2: **Output:** 2-approximate Deletion Set
  - 3:  $U \leftarrow H$
  - 4:  $\mathcal{F} \leftarrow \{X \subseteq U : \mathcal{A}(X) \text{ is deficient, } |X \cap H_i| \leq 2, \forall i \in \{1, 2\}\}$
  - 5:  $g \leftarrow$  optimum fractional hitting set of  $(U, \mathcal{F})$   $\triangleright g : U \rightarrow [0, 1]$
  - 6:  $B_1 \leftarrow \{(g(v), 1/2 - g(v)) : v \in H_1, g(v) \leq 1/2\} \cup \{(1/2 - g(v), g(v)) : v \in H_2, g(v) \leq 1/2\} \cup \{(1/2, 0)\}$
  - 7:  $\varepsilon \leftarrow$  arbitrary positive value less than  $\min_{(x_1, x_2), (x'_1, x'_2) \in B_1} |x_1 - x'_1|$
  - 8:  $B_2 \leftarrow \{(x_1 + \varepsilon, x_2 - \varepsilon) : (x_1, x_2) \in B_1, x_1 \neq 1/2\} \cup \{(0, 1/2)\}$
  - 9:  $B \leftarrow B_1 \cup B_2$
  - 10: **for**  $x = (x_1, x_2) \in B$  **do**
  - 11:      $T_x \leftarrow \{v_1 \in H_1 : g(v_1) \geq x_1\} \cup \{v_2 \in H_2 : g(v_2) \geq x_2\}$
  - 12:  $x_{min} \leftarrow \arg \min_{x \in B} |T_x|$
  - 13: **return**  $T_{x_{min}}$   $\triangleright T_{x_{min}}$  is a 2-approximate deletion set
-



# Impatient PPSZ – A Faster Algorithm for CSP

Shibo Li ✉ 

Shanghai Jiao Tong University, China

Dominik Scheder ✉ 

Shanghai Jiao Tong University, China

---

## Abstract

PPSZ is the fastest known algorithm for  $(d, k)$ -CSP problems, for most values of  $d$  and  $k$ . It goes through the variables in random order and sets each variable randomly to one of the  $d$  colors, excluding those colors that can be ruled out by looking at few constraints at a time.

We propose and analyze a modification of PPSZ: whenever all but 2 colors can be ruled out for some variable, immediately set that variable randomly to one of the remaining colors. We show that our new “impatient PPSZ” outperforms PPSZ exponentially for all  $k$  and all  $d \geq 3$  on formulas with a unique satisfying assignment.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Randomized algorithms, Constraint Satisfaction Problems, exponential-time algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.33

**Related Version** *Full Version:* <https://arxiv.org/abs/2109.02795>

**Funding** Both authors acknowledge support from the National Natural Science Foundation of China under grant 61502300 and 11671258.

**Acknowledgements** Dominik Scheder wants to thank Timon Hertli, Isabelle Hurbain, Sebastian Millius, Robin A. Moser, and May Szedlák, his co-authors of [4]. The idea of impatient assignment came up when we were working on [4].

## 1 Introduction

A Constraint Satisfaction Problem, or CSP for short, consists of a finite set of variables  $x_1, \dots, x_n$ , a domain  $[d] := \{1, \dots, d\}$  of potential values, called *colors*, and a set of constraints. A constraint is of the form  $(x_{i_1}, \dots, x_{i_k}) \in S$ , where  $S \subseteq [d]^k$ . In analogy to CNF-SAT, we assume in this paper that  $|S| = d^k - 1$ , i.e., all but one possible assignments satisfy the constraint. We speak of a  $(d, k)$ -CSP if all constraints are over  $k$  variables. In a slight abuse of notation, we also use  $(d, k)$ -CSP to denote the associated *decision problem*: is there a way to assign values in  $[d]$  to the variables that satisfies all constraints? This is NP-complete except when  $d = 1$  or  $k = 1$  or  $k = d = 2$ , so researchers focus on finding *moderately exponential-time algorithms*: algorithms of running time  $c^n$  for  $c < d$ . Examples include Beigel and Eppstein’s randomized algorithm for  $(d, 2)$ -CSP with running time  $O((0.4518d)^n)$  [1]; Schönning’s random walk algorithm of running time  $O^*((\frac{d(k-1)}{k})^n)$  [11]; Paturi, Pudlák, and Zane encoding-based randomized algorithm called PPZ [7] for  $k$ -SAT (i.e.,  $d = 2$ ), which runs in time  $O(2^{(1-1/k)n})$ . Paturi, Pudlák, Saks and Zane [6] improved PPZ by introducing a pre-processing step using small-width resolution. Both PPZ and PPSZ can be easily modified to work for  $(d, k)$ -CSP as well, as done by Scheder [8] for PPZ and Hertli et al. [4] for PPSZ. In both cases, several subtleties and technical difficulties arise, which are not present for  $k$ -SAT. Furthermore, [4] is the currently fastest algorithm for  $(d, k)$ -CSP when  $k \geq 4$ .



© Shibo Li and Dominik Scheder;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 33; pp. 33:1–33:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1.1 The PPSZ Algorithm

Let us give an informal description of PPSZ, first for SAT, and then for CSP. In either case, it chooses a random ordering  $\pi$  on the variables  $x_1, \dots, x_n$ . Then it goes through the variables one by one, in the order of  $\pi$ ; when processing  $x_i$ , it fixes  $x_i$  randomly to **true** or **false**, unless there is a set of up to  $D$  clauses that logically implies  $x_i = b$  for some  $b \in \{0, 1\}$ ; in the latter case, we fix  $x_i$  to  $b$  and say that  $x_i$  has been inferred by  $D$ -implication. For CSP, the only difference is that when processing  $x_i$ , it checks (with brute force) for which colors  $c \in [d]$  the statement  $[x_i \neq c]$  can be inferred by a set of up to  $D$  constraints; if so, we say  $[x_i \neq c]$  is  $D$ -implied, and color  $c$  is obviously ruled out. It then fixes  $x_i$  randomly to one of the colors not yet ruled out (or declares failure if all colors have been ruled out).

**Unique-SAT versus general-SAT.** A peculiar feature of PPSZ, as analyzed in the seminal paper [6], is that it performs better if the input instance  $F$  has a *unique* satisfying assignment. Certain properties, such as the existence of critical clause trees, break down once  $F$  has multiple solutions. In [6], the authors proposed a clever but technical workaround, which incurred an exponential overhead for  $k = 3, 4$ . In his 2011 breakthrough paper, Hertli [3] showed that this peculiarity is in fact an artifact of the analysis, and gave a very abstract and high-level proof that PPSZ on formulas with many solutions is indeed no worse. His proof was later simplified by Scheder and Steinberger [10]. The proofs in [3] and [10] work only provided that the internal machinery of the PPSZ algorithm (e.g., checking  $D$ -implication) is “not too good”. Curiously, in [4] it turned out that, for  $k = 2, 3$  and certain values of  $d$ , the PPSZ machinery is indeed “too good”, and consequently their time complexity for the general case (multiple solutions) is worse than for the unique case (exactly one solution). For formulas with a unique solution, their analysis gives a running time of  $O(d^{nS_{d,k} + o(n)})$ , for  $S_{d,k}$  defined below. This is the best known running time for all  $d, k$  except for  $(d, k) = (3, 2)$  and  $(d, k) = (4, 2)$ .

**Improvements to PPSZ for  $k$ -SAT.** Two recent results improve PPSZ. Hansen, Kaplan, Zamir, and Zwick [2] define a *biased* version of PPSZ and show that it achieves an improvement for all  $k \geq 3$ . Scheder [9] shows that PPSZ itself performs exponentially better than in the analysis of [6]. We would not be surprised if both improvements carry over to  $(d, k)$ -CSP, although to our knowledge, this has not been analyzed so far. The improvement presented in this work is of a different quality: it is not a generalization of some idea for  $k$ -SAT; in fact, the main idea only makes sense for  $d \geq 3$  and thus is particular to  $(d, k)$ -CSP problems.

**The time complexity of PPSZ for Unique  $(d, k)$ -CSP.** A main result of [4] is that PPSZ solves Unique  $(d, k)$ -CSP in time  $O(2^{S_{d,k} n + o(n)})$ , where  $S_{d,k}$  is defined by the following random experiment: let  $T^\infty$  be the infinite rooted tree in which each node on even depth (which includes the root at depth 0) has  $k - 1$  children and every node on odd depth has  $d - 1$  children. Let  $T_1, \dots, T_{d-1}$  be disjoint copies of  $T^\infty$ , sample  $p \in [0, 1]$  uniformly, and delete every odd-level node with probability  $p$ , independently. Let  $J_c$  be the indicator variable that is 1 if the root of  $T_c$  is contained in an infinite component after this deletion step. Then

$$S_{d,k} := \mathbb{E}[\log_2(J_1 + \dots + J_{d-1} + 1)] . \quad (1)$$

Approximate values of  $S_{d,k}$  for small values of  $d, k$  can be found in Hertli et al. [4]<sup>1</sup> To understand  $S_{d,k}$  for large values  $d$ , it is most advantageous to write  $c_{d,k} := \log_2(d) - S_{d,k}$ . Intuitively,  $\log_2(d)$  is the number of bits required to specify a color and  $S_{d,k}$  is the number of coin tosses used by PPSZ to determine it, so  $c_{d,k}$  is the number of coin tosses per variable saved by PPSZ. Theorem 1.4 of [4] states that  $\lim_{d \rightarrow \infty} c_{d,k} = -\int_0^1 \log_2(1 - r^{k-1})$ .

Thus, for large  $d$ , PPSZ saves a *constant number of bits* per variable; this is somewhat unsatisfactory; we would like to save a *constant fraction of bits*, i.e., an algorithm of running time  $d^{(1-c_k)n}$  for some  $c_k > 0$  for arbitrary  $d$ . The existence of such an algorithm is posed as an open problem by Koucký, Rödl, and Talebanfard [5]. In the same paper, they give such an algorithm for CSP formulas where every variable appears in a constant number of constraints. Formally, their algorithm runs in time  $d^{(1-c_{k,r})n}$  on  $(d, k)$ -CSP formulas  $F$  where each of the  $n$  variables occurs in at most  $r$  constraints.

## 1.2 Our Contribution

In this work, we focus on the case that  $F$  has a unique satisfying assignment  $\alpha^*$ , without loss of generality  $\alpha^* := (d, d, \dots, d)$ . The idea behind our improvement is as follows: suppose  $x, y, z$  are variables appearing in the order  $y, x, z$  in  $\pi$ . Focus on the point in time when PPSZ processes  $x$ , and assume every assignment prior to  $x$  has been correct. For example, the variable  $y$  has already been replaced by the constant  $d$ . In other words, when PPSZ tries to infer statements like  $[x \neq c]$  from small sets of constraints, it can use the information  $[y = d]$ . It cannot use  $[z = d]$ , however. Or can it? Maybe PPSZ can already infer  $[z \neq 1], \dots, [z \neq d - 1]$ ; in this case, it can also infer  $[z = d]$ , and it would be safe to fix  $z$  to  $d$ . Let us propose the following rule:

**Rule of One.** Whenever  $[z = c]$  can be inferred by  $D$ -implication, fix  $z$  to  $c$ .

This rule is “uncontroversial” in the sense that it will never make a mistake. Indeed, the reader who is familiar with the literature about PPSZ, in particular with its original version using small-width resolution, will notice that resolution implicitly implements the above rule. We propose the following more aggressive rule:

**Rule of Two.** Whenever  $[z = c_1 \vee z = c_2]$  can be inferred by  $D$ -implication, i.e., if all but 2 colors can be ruled out, pick  $c \in \{c_1, c_2\}$  uniformly at random and fix  $z$  to  $c$ .

Obviously, this rule can introduce mistakes. On the plus side, it might be very unlikely that the range of plausible (i.e., not ruled out) colors for  $z$  further decreases from 2 to 1. Better to bite the bullet now, decide on a value for  $z$ , hope that it is correct, and use that information for subsequent  $D$ -implications. For example, it might be that using the information  $[z = d]$  lets us rule out additional colors for  $x$ , the variable currently being processed by PPSZ. Unfortunately, this rule does more bad than good: consider the variables coming towards the very end of  $\pi$ . For each of them, it is very likely that all but one color can be ruled out; thus, PPSZ would set them correctly with high probability; using our Rule of Two, this probability would go down from (almost) 1 to (roughly) 1/2 since we decide on a value once only two values are left. We propose a less impatient rule:

<sup>1</sup> In [4] they define  $S_{d,k}$  with  $\log_d$  instead of  $\log_2$ . We prefer the binary logarithm for purely cosmetic reasons: otherwise some of our expressions would contain the expression “ $\log_d(2)$ ”, but “1” is undoubtedly nicer.

**Conservative Rule of Two.** Apply the Rule of Two only to variables  $z$  that are among the first  $\theta n$  in  $\pi$ ; don't apply it to the last  $(1 - \theta)n$  variables.

We will show that for those early variables, it is extremely unlikely that the set of plausible colors gets narrowed down to only one color; and that it is somewhat more likely that the Rule of Two helps us rule out one additional colors for a variable. In particular, we prove the following theorem:

► **Theorem 1.** *For every  $d \geq 3$  and  $k \geq 2$ , there is some  $\epsilon > 0$  and a randomized algorithm solving  $(d, k)$ -CSP in time  $2^{n(S_{d,k} - \epsilon)} \text{poly}(n)$ , where  $S_{d,k}$  is as defined in (1).*

Note that the Rule of Two does not make any sense in the Boolean context. For every variable  $z$  we can trivially 0-imply  $[z = 0 \vee z = 1]$ , and thus every variable immediately qualifies for impatient assignment. We thus lose all control over the ordering  $\pi$  in which we process the variables. The Rule of One does of course make sense in the Boolean setting and is implicitly implement if you are running “strong” PPSZ, i.e., with small-width resolution.

### 1.3 Notation

Let  $V = \{x_1, \dots, x_n\}$  be a set of variables and  $[d] = \{1, \dots, d\}$  be the set of possible colors. A *literal* is an expression  $(x \neq c)$ , where  $x \in V, c \in [d]$ . A *clause* is a disjunction of literals:  $(v_1 \neq c_1 \vee v_2 \neq c_2 \vee \dots \vee v_k \neq c_k)$ . A  $(d, k)$ -CSP is a conjunction of clauses of size  $k$  each. An assignment  $\alpha$  is a function  $V \rightarrow [d]$ . It satisfies a literal  $(x \neq c)$  if  $\alpha(x) \neq c$ ; it satisfies a clause if it satisfies at least one literal therein; it satisfies a  $(d, k)$ -CSP  $F$  if it satisfies all clauses in  $F$ . If  $V' \subseteq V$  and  $\alpha : V' \rightarrow [d]$ , we call  $\alpha$  a *partial assignment*;  $\text{vbl}(\alpha)$  denotes its domain, i.e.,  $V'$ .  $F^{[\alpha]}$  is the simplified formula after setting all variables in  $V'$  according to  $\alpha$ . We will write partial assignments like this:  $[x \mapsto 2, y \mapsto 3, \dots]$  and therefore  $F^{[x \mapsto 2]}$  will denote the formula after replacing  $x$  with 2. For a set  $V' \subseteq V$  of variables, we take the liberty to write  $F^{[V' \mapsto d]}$ , where  $[V' \mapsto d]$  is the partial assignment  $[x \mapsto d \text{ for each } x \in V']$ . For a clause  $C$  and a  $(d, k)$ -CSP  $F$ ,  $\text{vbl}(C)$  and  $\text{vbl}(F)$  denote the sets of variables in  $C$  and  $F$ , respectively. For a rooted tree  $T$  and a node  $v$  therein, the *subtree of  $T$  rooted at  $v$*  is the tree containing  $v$  (as root) and all its descendants. We use the notation  $[\text{statement}]$ , which evaluate to 1 if **statement** holds, and to 0 otherwise.

### 1.4 PPSZ and impatient PPSZ

► **Definition 2** (*D-implication* [4]). *Let  $F$  be a  $(d, k)$ -CSP formula and  $u$  be a literal of  $F$ . We say  $F$  implies  $u$  and write  $F \models u$  if all assignment satisfying  $F$  also satisfy  $u$ . We say  $F$   $D$ -implies  $u$  and write  $F \models_D u$  if there is some  $G \subseteq F$  with  $|G| \leq D$  and  $G \models u$ .*

For the rest of the paper,  $D = D(n)$  will be some slowly growing function in  $n$ , so  $F \models_D u$  can be checked in time  $O(|F|^D \text{poly}(n))$ , which is subexponential in  $n$ .

► **Definition 3** (*Plausible values*). *Let  $F$  be a  $(d, k)$ -CSP formula and  $x$  a variable. We say color  $c \in [d]$  is  $D$ -plausible for  $x$  in  $F$  if  $F$  does not  $D$ -imply  $(x \neq c)$ . Let  $\text{Plaus}(x, F, D)$  denote the set of all colors that are  $D$ -plausible for  $x$ . We will drop the parameter  $D$  if it is understood from the context.*

Note that our code specifies  $\pi$  as an explicit input parameter; it is the responsibility of the “user” to make sure  $\text{PPSZ}(F, \pi)$  is called with a random  $\pi$ ; furthermore, we implicitly assume that PPSZ declares failure if the set  $\text{Plaus}(x, F^{[\alpha]})$  in Line 4 is empty.

---

**Algorithm 1** PPSZ algorithm.
 

---

```

1: procedure PPSZ( $F, \pi$ )
2:    $\alpha \leftarrow$  the empty assignment
3:   for  $x \in \text{vbl}(F)$  in the order of  $\pi$  do
4:     choose  $\iota \in \text{Plaus}(x, F^{[\alpha]})$  uniformly at random
5:      $\alpha := \alpha \cup [x \mapsto \iota]$ 
6:   end for
7:   return  $\alpha$  if it satisfies  $F$ , else failure
8: end procedure

```

---

From now on, we view  $\pi$  not as a permutation of the variables but as a *placement*, i.e., a function  $V \rightarrow [0, 1]$ ; note that if  $\pi : V \rightarrow [0, 1]$  is sampled uniformly at random, it will be an injection with probability 1; sorting  $V$  in ascending order by their  $\pi$ -value will give a permutation of  $V$ . Additionally, we fix two parameters  $\theta$  (to be determined later) and  $\zeta := 2 - \log_2(3)$ , and mark every variable  $x$  as *eligible for impatient assignment* as follows:

► **Definition 4** (Eligible for impatient assignment). *For each variable  $x$ , define  $\mathbb{I}_x \in \{0, 1\}$  as follows. (1) If  $\pi(x) \geq \theta$ , set  $\mathbb{I}_x := 0$ ; (2) if  $\pi(x) < \theta$ , set  $\mathbb{I}_x := 1$  with probability  $\zeta$  and to 0 with probability  $1 - \zeta$ , independently of all other choices. If  $\mathbb{I}_x = 1$  we say  $x$  is eligible for impatient assignment.*

The reasoning behind condition (1) is that a “late” variable  $x$  (one with  $\pi(x) \geq \theta$ ) is likely to end up with a unique plausible value when PPSZ reaches it; setting it prematurely will do more bad than good; our decision to mark “early” variables with  $\pi(x) < \theta$  as eligible only with probability  $\zeta$  (and not with probability 1) has technical reasons: we want a certain function to become concave, and our choice of  $\zeta$  is the largest value for which this happens.

---

**Algorithm 2** Impatient PPSZ.
 

---

```

1: procedure IMPATIENTPPSZ( $F, \pi$ )
2:    $\alpha :=$  the empty assignment
3:   for  $x \in \text{vbl}(F)$  in ascending order of  $\pi$  do
4:     while  $\exists y \in \text{vbl}(F) \setminus \text{vbl}(\alpha)$  with  $\mathbb{I}_y = 1$  and  $|\text{Plaus}(y, F^{[\alpha]})| \leq 2$  do
5:       choose  $\iota \in \text{Plaus}(y, F^{[\alpha]})$  uniformly at random
6:        $\alpha := \alpha \cup [y \mapsto \iota]$ 
7:     end while
8:     if  $x \notin \text{vbl}(\alpha)$  then
9:       choose  $\iota \in \text{Plaus}(x, F^{[\alpha]})$  uniformly at random
10:       $\alpha := \alpha \cup [x \mapsto \iota]$ 
11:    end if
12:  end for
13:  return  $\alpha$  if it satisfies  $F$ , else failure
14: end procedure

```

---

**Proof sketch of Theorem 1.** As the analysis of PPSZ, our proof relies heavily on the concept of *critical clause trees*. Intuitively, those trees are a neat and tidy way to describe all ways how  $[x = c]$  can be ruled out. For PPSZ, this can be described as an extinction event in a Galton-Watson process; for ImpatientPPSZ however, we need a more complicated notion. To make matters worse, while [6] and [4] showed that the worst case for their PPSZ analysis

happens if all trees are “nice”, this fails for our algorithm. We have to come up with a (short) list of non-niceties, and for each of them show that  $[x = c]$  is a bit more likely to be ruled out; we can then finally analyze the “nice” case. ◀

## 2 Conceptual Framework

**Notation for sets of variables coming before variable  $x$ :  $V_x$  and  $V_x^{\text{imp}}$ .** To analyze PPSZ and our variant ImpatientPPSZ, we need to talk about the point in time where the algorithm processes a variable  $x$ , and in particular, we need to talk about the set of variables that have already been assigned a value at this point. For PPSZ, this is easy: we define  $V_x := \{y \in \text{vbl}(F) \mid \pi(y) < \pi(x)\}$ . For ImpatientPPSZ, it’s a bit more complicated: imagine we run ImpatientPPSZ but feed it the “correct” values in every assignment; that is, whenever a color  $c$  is chosen, make sure that  $c = d$  (we manipulate this random source to always choose the correct color); pause the algorithm in the iteration when variable  $x$  is being processed, just after line 7, and look at the partial assignment  $\alpha$  built so far. We set  $V_x^{\text{imp}} := \text{vbl}(\alpha) \setminus \{x\}$ . We remove  $x$  for purely technical reasons; if  $x$  happens to be already set at that time, then line 9 and 10 will be skipped by the algorithm anyway.

► **Observation 5.** *If line 9 is executed then  $c$  is chosen uniformly at random from the set  $\text{Plaus}(x, F^{[V_x^{\text{imp}} \mapsto d]})$ .*

We define the following indicator variables:

$$A_{x,c} := \begin{cases} 1 & \text{if } c \in \text{Plaus}(x, F^{[V_x \mapsto d]}, D) \\ 0 & \text{else.} \end{cases}$$

$$A_{x,c}^{\text{imp}} := \begin{cases} 1 & \text{if } c \in \text{Plaus}(x, F^{[V_x^{\text{imp}} \mapsto d]}, D) \\ 0 & \text{else.} \end{cases}$$

and  $A_x := \sum_c A_{x,c}$  and  $A_x^{\text{imp}} := \sum_c A_{x,c}^{\text{imp}}$ . These are random variables in our random placement  $\pi$ . Note that  $A_{x,d} = A_{x,d}^{\text{imp}} = 1$  because color  $d$  is always plausible; also,  $A_{x,c}^{\text{imp}} \leq A_{x,c}$  simply because  $V_x \subseteq V_x^{\text{imp}}$ , i.e., ImpatientPPSZ has at least as much information as PPSZ.

► **Lemma 6 ([4]).** *For a fixed permutation  $\pi$ ,  $\Pr[\text{PPSZ}(F, \pi) \text{ finds } \alpha^*] = \prod_x \frac{1}{A_x(\pi)}$ . For a random permutation,  $\Pr_\pi[\text{PPSZ}(F, \pi) \text{ finds } \alpha^*] \geq 2^{-\sum_x \mathbb{E}_\pi[\log_2 A_x(\pi)]}$ .*

The second statement follows from the first by Jensen’s inequality. To obtain a similar formula for ImpatientPPSZ, we need to take into account that a variable  $x$  might be assigned in line 6 or in line 10.

► **Lemma 7.** *For a fixed permutation  $\pi$ , we have  $\Pr[\text{ImpatientPPSZ}(F, \pi) \text{ finds } \alpha^*] \geq \prod_x \frac{1}{\max(1 + \mathbb{I}_x, A_x^{\text{imp}}(\pi))}$ . For a random permutation, the probability that ImpatientPPSZ succeeds is at least*

$$\Pr_\pi[\text{ImpatientPPSZ}(F, \pi) \text{ finds } \alpha^*] \geq 2^{-E_\pi[\sum_x \log_2(\max(1 + \mathbb{I}_x, A_x^{\text{imp}}(\pi)))]}.$$

**Proof.** If  $\mathbb{I}_x = 0$  then  $x$  will be assigned in line 10 and thus its value will be correct with probability  $1/A_x^{\text{imp}}(\pi)$ , conditioned on all prior assignments being correct. If  $\mathbb{I}_x = 1$  then either it is assigned in line 6, and is correct with probability  $1/2$ ; or it is still assigned regularly in line 10, and is correct with probability  $1/A_x^{\text{imp}}(\pi)$ . This proves the first inequality. The second inequality in the lemma follows from the first by Jensen’s inequality. ◀

## 2.1 Independence between colors

The crucial quantity in the analysis of ImpatientPPSZ is the random variable  $A_x^{\text{imp}} = \sum_c A_{x,c}^{\text{imp}}$ . The next lemma states that we can focus on analyzing the indicator variables  $A_{x,c}^{\text{imp}}$  individually; that is, if we condition on  $\pi(x) = p$ , then the  $d$  indicator variables are independent in the worst case. More formally:

► **Lemma 8** (Independence between colors). *Let  $\pi : V \rightarrow [0, 1]$  be uniformly random and set  $p := \pi(x)$ . We sample  $d$  random variables  $\tilde{A}_{x,c}^{\text{imp}} \in \{0, 1\}$ ,  $c = 1, \dots, d$  by setting each  $\tilde{A}_{x,c}^{\text{imp}}$  to 1 with probability  $\Pr[A_{x,c}^{\text{imp}} = 1 \mid \pi(x) = p]$ , independently. Set  $\tilde{A}_x^{\text{imp}} := \sum_c \tilde{A}_{x,c}^{\text{imp}}$ . Then*

$$\mathbb{E}_{\pi} [\log_2 (\max (1 + \mathbb{I}_x, A_x^{\text{imp}}(\pi)))] \leq \mathbb{E}_{\pi} [\log_2 (\max (1 + \mathbb{I}_x, \tilde{A}_x^{\text{imp}}(\pi)))] \quad (2)$$

**Proof idea.** We would like to prove this along the lines of Lemma 3.5 of [4]. The additional problem here is that although the function  $f : t \mapsto \log(t)$  is concave, the function  $g : t \mapsto \log(\max(2, t))$  isn't. This is why, if  $\pi(x) < \theta$ , we set  $\mathbb{I}_x$  to 1 with probability  $\zeta$  and to 0 with probability  $1 - \zeta$ . The convex combination  $(1 - \zeta) \cdot f + \zeta \cdot g$  is concave<sup>2</sup> and the proof goes through just as for Lemma 3.5 in [4]. See Lemma 24 in the appendix for a complete proof. ◀

The upshot is that it is sufficient to bound  $\Pr[A_{x,c}^{\text{imp}} = 1 \mid \pi(x) = p]$  from above, for each variable  $x$  and color  $c$ , individually.

## 2.2 Critical Clause Trees and Brief Analysis of PPSZ

In this section, we define critical clause trees and review some results from [4]. We assume that  $\mathbf{d} = (d, \dots, d)$  is our unique satisfying assignment. Note that every unsatisfied literal is of the form  $(y = d)$  and every satisfied literal is of the form  $(y \neq c)$  for some  $c \neq d$ . Let  $x \in \text{vbl}(F)$  and  $c \in \{1, \dots, d - 1\}$ . The *critical clause tree*  $T_{x,c}^h$  of height  $h$  has two types of nodes: a node  $u$  on an even level (which includes the root at level 0) is a *clause node*, has a *clause label*  $\text{clauselabel}(u)$  and an *assignment label*  $\beta_u$ ; it has at most  $k - 1$  children (in fact, one for each unsatisfied literal in  $\text{clauselabel}(u)$ ). A node  $v$  on an odd level is a *variable node* and has a *variable label*  $\text{varlabel}(v)$ ; it has exactly  $d - 1$  children. An edge  $(v, w)$  from a variable node  $v$  to a clause node  $w$  has an edge color  $EC(e) \in [d - 1]$ . Thus, if  $\text{varlabel}(v) = y$  and  $(v, w)$  edge color  $EC(v, w) = i$ , then this edge represents the alternative assignment  $[y \mapsto i]$ . The critical clause tree  $T_{x,c}^h$  is constructed as in algorithm 3.

Let us assume  $h$  is always odd, so the lowest layer of  $T_{x,c}^h$  consists of variable nodes.  $T_{x,c}^h$  has two types of leaves: those variable nodes at height  $h$ ; we call them *safe leaves*; and clause nodes whose clause label does not contain any literal of the form  $(y \neq d)$ ; we call them *unsafe leaves*.

► **Proposition 9** ([4]).

1. *Suppose  $v$  is a clause node in  $T_{x,c}^h$  with clause label  $C$  and  $(y \neq i)$ ,  $i \in [d]$  is a literal in  $C$ . Then if  $i = d$ ,  $v$  has a child whose variable label is  $y$ . If  $i < d$  and  $y \neq x$  then  $v$  has an ancestor node whose variable label is  $y$ .*
2. *No variable appears more than once as variable label on a path from root to a leaf.*

<sup>2</sup> The attentive reader might notice: it's not concave; however, if we change the definition of "log" in the definition of  $f$  and  $g$  from the usual log to "log on  $\mathbb{N}$  and linear between integers, then it is concave, provided that  $\zeta \leq 2 - \log_2 3$ .



---

**Algorithm 3** BuildCCT( $F, x, c, h$ ).

---

```

1: Create a root node and set  $\beta_{\text{root}} := \alpha[x = c]$ 
2: while  $\exists$  clause node  $u$  of height less than  $h - 1$  without a clause label do
3:   Find a clause  $C$  which is not satisfied by  $\beta_u$ 
4:   Set  $\text{clauselabel}(u) := C$ 
5:   for each unsatisfied literal  $(y \neq d)$  in  $C$  do
6:     Create a new child  $v$  of  $u$ 
7:      $\text{varlabel}(v) := y$ 
8:     for  $i \in [d - 1]$  do
9:       Create a new child  $w$  of  $v$ 
10:      Set  $\beta_w := \beta_v[y = i]$ 
11:      Set  $EC(v, w) = i$ 
12:     end for
13:   end for
14: end while
15: remove clause nodes at height  $h + 1$ 
16: return  $T_{x,c}^h$ 

```

---

► **Definition 10** (labeled tree). *A labeled tree is a possibly infinite tree such that: (1) every node is either a variable node or a clause node; (2) a variable node  $u$  has a label  $\text{varlabel}(u) \in \mathbb{L}$  in some label space  $\mathbb{L} \supseteq V$ ; (3) they alternate, i.e., if a variable node has children, they are all clause nodes, and vice versa; (4) its degree is bounded: there is some  $\Delta \in \mathbb{N}$  such that every node has at most  $\Delta$  children. A leaf in a labeled tree is a safe leaf if it is a variable node; Otherwise, it is an unsafe leaf.*

Note that each subtree of a critical clause tree is a labeled tree. A *safe path* in a labeled tree is a path that starts at the root and is either infinite or ends at a safe leaf.

► **Definition 11** ( $\text{Cut}_p$  and  $\text{Cut}$ ). *Let  $T$  be a labeled tree. The event  $\text{Cut}_p(T)$  is an event in the probability space of all placements  $\pi : \mathbb{L} \rightarrow [0, 1]$  that happens if every safe path in  $T$  contains a node  $v$  with  $\pi(\text{varlabel}(v)) < p$ . Let  $x$  be the label of the root of  $T$ . We define  $\text{Cut}(T) := \text{Cut}_{\pi(x)}(T)$ .*

Suppose  $T$  is a labeled tree, and let  $T_1, \dots, T_l$  be the subtrees rooted at the  $l$  children of the root of  $T$ . Note that the  $T_i$  are themselves labeled trees. If the root of  $T$  is a clause node then  $\text{Cut}_p(T) = \bigwedge_{i=1}^l \text{Cut}_p(T_i)$ . If it is a variable node, let  $y := \text{varlabel}(\text{root}(T))$ , and observe that  $\text{Cut}_p(T) = [\pi(y) < p]$  if  $\text{root}(T)$  itself is a safe leaf (i.e., if  $l = 0$ ) and  $\text{Cut}_p(T) = [\pi(y) < p] \vee \bigwedge_{i=1}^l \text{Cut}_p(T_i)$  else .

Next, we connect the notion of cuts to our notion of being a plausible color. For this, set  $L := (d - 1)(k - 1)$  and observe that  $T_{x,c}^h$  has at most  $L^i$  clause nodes at depth  $2i$ . Choose  $\tilde{h}$  to be the largest integer for which  $1 + L + L^2 + \dots + L^{\tilde{h}} \leq D$  (recall  $D$ , our strength parameter in the definition of  $D$ -implication), and set  $h := 2\tilde{h} + 1$ . Then  $T_{x,c}^h$  has at most  $D$  clause nodes and  $h$  is also a slowly growing function in  $n$ .

► **Lemma 12** ([4]). *Let  $p = \pi(x)$ . If  $\text{Cut}_p(T_{x,c}^h)$  happens then  $A_{x,c} = 0$ .*

Recall the infinite trees  $T^\infty$  and  $T_1, \dots, T_{d-1}$  and the indicator variables  $J_1, \dots, J_{d-1}$  defined above, just before (1), and observe that  $J_c = 1$  iff  $\text{Cut}_p(T_c)$  does *not* happen. Let  $T_\infty$  be the subtree of  $T^\infty$  rooted at the first child of the root. Define  $Q(p) := \Pr[\text{Cut}_p(T^\infty)]$  and  $R(p) := \Pr[\text{Cut}_p(T_\infty)]$ . The next proposition is from [4], adapted for our purposes.

► **Proposition 13** ([4]). Set  $L = (k - 1)(d - 1)$ . If  $p \geq 1 - \frac{1}{L}$  then  $Q(p) = R(p) = 1$ ; otherwise,  $Q(p)$  and  $R(p)$  are the unique roots in  $[0, 1]$  of the equations  $Q = (p + (1 - p)Q^{d-1})^{k-1}$  and  $R = p + (1 - p)R^L$ , respectively. Furthermore,  $Q(p) = R(p)^{k-1}$ .

As our height parameter  $h$  grows (roughly logarithmic with our strength parameter  $D$ ), the critical clause trees  $T_{x,c}^h$  will look more and more like  $T^\infty$ , and thus the cut probability will converge to  $Q(p)$ . Formally, let  $\text{error}(d, k, h, p)$  and  $\text{error}(d, k, h)$  stand for any functions that converge to 0 as  $h \rightarrow \infty$ .

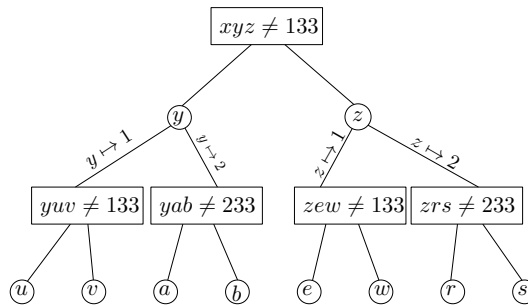
► **Proposition 14** (Lemma 3.6 in [4]).  $\Pr[\text{Cut}_p(T_{x,c}^h)] \geq \Pr[\text{Cut}_p(T_c)] - \text{error}(d, k, p, h)$ .

To summarize: conditioned on  $\pi(x) = p$ , the sum  $A_x = A_{x,1} + \dots + A_{x,d}$  has the worst behavior if all  $A_{x,c}$  are independent (Lemma 8); furthermore,  $A_{x,c} \leq J_c$  except with probability  $\text{error}(d, k, p, h)$ , for all  $c \leq d - 1$ , and therefore:

► **Lemma 15** ([4]).  $\mathbb{E}_\pi[\log_2(A_x)] \leq \mathbb{E}[\log_2(J_1 + \dots + J_{d-1} + 1)] + \text{error}(d, k, h) = S_{d,k} + \text{error}(d, k, h)$ .

### 3 The Probability of Inferring $x \neq c$

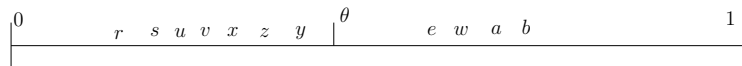
Just as [4] analyzes PPSZ by studying the random variables  $A_{x,c}$ , we have to study  $A_{x,c}^{\text{imp}}$ . We can always resort to the “old” analysis via  $A_{x,c}^{\text{imp}} \leq A_{x,c}$ . However, the whole point of this work is to show that this inequality is often strict. To understand how and when this might happen, we discuss an example for  $d = 3$ .



This is  $T_{x,1}^3$ , the critical clause tree for  $x$  and 1 built up to height 3. The formula  $F$  in question contains the constraints shown as clause labels, but of course contains many more constraints. Suppose that  $u, v, a, b, z$  come before  $x$  in  $\pi$ , and  $e, w, r, s, y$  come later. In the normal PPSZ, we have already set  $u, v, a, c, z \mapsto 3$  when considering  $x$ , and thus the clauses of  $F$  will have shrunk:

- $(yuv \neq 133)$  shrinks to  $(y \neq 1)$ ;
- $(yab \neq 233)$  shrinks to  $(y \neq 2)$ ;
- $(zew \neq 133)$  and  $(zrs \neq 233)$  don't shrink but disappear: they are satisfied by  $z \mapsto 3$ ;
- $(xyz \neq 133)$  shrinks to  $(xy \neq 13)$ .

Together, the three shrunk clauses  $(y \neq 1)$ ,  $(y \neq 2)$ , and  $(xy \neq 13)$  imply  $(x \neq 1)$ ; since  $D \geq 3$  this means that  $x = 1$  can be ruled out, i.e.,  $A_{x,1} = 0$ . Next, suppose  $\pi$ , viewed as a placement  $\pi : V \rightarrow [0, 1]$ , looks like this:



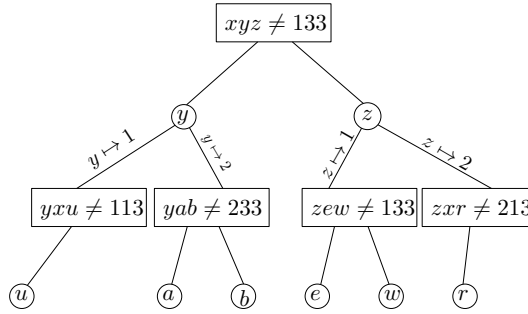
### 33:10 Impatient PPSZ

and assume for simplicity that all variables  $l$  with  $\pi(l) < \theta$  in  $\text{vbl}(F)$  are eligible for impatient assignment (i.e., have  $\mathbb{I}_l = 1$ ). Note that  $\text{Cut}(T_{x,1}^3)$  does not happen. Namely, the path from root to  $c$  contains two variable labels,  $y$  and  $b$ , and  $\pi(y), \pi(b) \geq \pi(x)$ . Analogously, the alternative assignment  $\alpha^*[x \mapsto 1, y \mapsto 2, b \mapsto 2]$  satisfies all clauses in the figure above, and thus the algorithm cannot infer  $x \neq 1$  from those clauses alone, and  $A_{x,1} = 1$ . Observe now what happens in ImpatientPPSZ:

- $r, s, u, v \mapsto 3$  before  $x$  is even considered;
- $(yuv \neq 133)$  shrinks to  $(y \neq 1)$ , and thus  $\text{Plaus}(y, F^{[\alpha]})$  shrinks to  $\{2, 3\}$ ;
- $y$  is assigned a value in line 6;
- the analogous thing happens to  $z$ ;
- $r, s, u, v \in V_x$ , and  $r, s, u, v, y, z \in V_x^{\text{imp}}$ ;
- $(xyz \neq 133)$  shrinks to  $(x \neq 1)$  and thus  $A_{x,1}^{\text{imp}} = 0$ .

We can now try to work out a formula for the probability that  $x = c$  is ruled out in this manner; however, our above example and analysis contains two silent assumptions that cannot be taken for granted in general:

1. All variable labels in  $T_{x,c}^3$  are distinct.
  2. All clause labels of  $T_{x,c}^3$  are critical clauses, i.e.,  $k - 1$  of its literals are of the form  $y \neq d$ .
- The original PPSZ paper [6] addresses Point 1 by using the FKG inequality to show that having multiple labels can never hurt us. But now we are talking about a more complicated event; it is not clear whether an FKG-like result applies. Point 2 is more troublesome. Consider the alternative scenario that  $T_{x,c}^3$  looks like this:



and consider the same  $\pi$  as above:  $r, s, u, v, x, z, y, \theta, e, w, a, b$ . After setting  $r, s, u, v \mapsto 3$ , the shrunk clauses are  $(yx \neq 11)$ ,  $(yab \neq 233)$ ,  $(zew \neq 133)$ , and  $(zx \neq 21)$ . Neither for  $y$  nor for  $z$  can we rule out any color, and therefore our impatient mechanism will not kick in. We will have  $V_x = V_x^{\text{imp}} = \{r, s, u, v\}$ . In other words, non-critical clauses seem useless for ImpatientPPSZ. But looking at the above example tree, we see what comes to the rescue: the right-most clause node is missing a child; it has at most  $k - 2$  children instead of  $k - 1$ . This alone will be enough to improve our success probability by a bit. It is time for some formal definitions.

► **Definition 16** (Privileged variables). *A variable  $x$  is privileged if there is some color  $c \in \{1, \dots, d - 1\}$  such that*

1.  $T_{x,c}^h$  has fewer than  $(k - 1)^2(d - 1)$  variable nodes at level 3 or
2.  $T_{x,c}^3$  has two variable nodes  $u$  and  $w$  with  $\text{varlabel}(u) = \text{varlabel}(w)$ .

► **Proposition 17.** *There is an  $\epsilon_{\text{privileged}} > 0$  for variable  $x$  which is privileged, depending only on  $d$  and  $k$ , such that*

$$\mathbb{E}[\log_2(A_x)] \leq S_{d,k} - \epsilon_{\text{privileged}} + \text{error}(d, k, h),$$

for every privileged variable  $x$  in  $F$ .

See Proposition 25 in the appendix for a proof.

► **Corollary 18.** *For every privileged variable  $x \in \text{vbl}(F)$  it holds that  $\mathbb{E}[\log_2(\max(1 + \mathbb{I}_x, A_x^{\text{imp}}))] \leq S_{d,k} - \epsilon_{\text{privileged}} + c\theta + \text{error}(d, k, h)$*

**Proof.** Since  $\max(a, b) \leq a \cdot b$  when  $a, b \geq 1$ , we get

$$\mathbb{E}[\log_2(\max(1 + \mathbb{I}_x, A_x^{\text{imp}}))] \leq \frac{\mathbb{E}[\log_2(1 + \mathbb{I}_x)]}{\pi} + \frac{\mathbb{E}[\log_2(A_x^{\text{imp}})]}{\pi}.$$

The first term equals  $\Pr[\mathbb{I}_x = 1] = c\theta$ ; the second is at most  $\mathbb{E}[\log_2(A_x)]$ , which by Proposition 17 is at most  $S_{d,k} - \epsilon_{\text{privileged}} + \text{error}(d, k, h)$ . This concludes the proof. ◀

► **Lemma 19.** *There is a constant  $\epsilon > 0$ , depending only on  $d$  and  $k$ , such that*

$$\mathbb{E}[\log_2(\max(1 + \mathbb{I}_x, A_x^{\text{imp}}))] \leq S_{d,k} - 0.1699 \left( \frac{c}{L+1} \theta^{L+1} + O(\theta^{L+2}) \right) + \text{error}(d, k, h).$$

for all non-privileged variables  $x$ . The constant factor hidden in the  $O(\cdot)$  depends only on  $d$  and  $k$ .

By choosing  $\theta$  sufficiently small, we can make sure that the bounds in Lemma 19 and Corollary 18 are both at most  $S_{d,k} - \epsilon_{d,k} + \text{error}(d, k, h)$ , for some  $\epsilon_{d,k}$  depending only on  $d$  and  $k$ . Together with Lemma 7, this proves Theorem 1.

**Proof of Lemma 19.** For a color  $1 \leq c \leq d - 1$ , fix the critical clause tree  $T_{x,c}^h$  and let us introduce a bit of notation. The root of  $T_{x,c}^h$  has a label

$$C_{\text{root}} = (x \neq c \vee y_1 \neq d \cdots \vee y_{k-1} \neq d).$$

It has  $k - 1$  children  $v_1, \dots, v_{k-1}$ , whose respective variable labels are  $y_1, \dots, y_{k-1}$ . Let  $T_i$  denote the subtree of  $T_{x,c}^h$  rooted at  $v_i$ . Each  $y_i$  in turn has  $d - 1$  children; each such level-2 node  $v$  has a clause label  $C_v$ ; note that  $C_v$  is a *critical clause*, i.e.,  $k - 1$  of its literals are of the form  $(z \neq d)$ , since otherwise it would have fewer than  $k - 1$  children, and  $T_{x,c}^h$  would have fewer than  $(k - 1)^2(d - 1)$  nodes at level 3; in other words,  $x$  would be privileged.

We need to define an event  $\text{ImpCut}_p(T_{x,c}^h)$  which, analogous to  $\text{Cut}_p(T_{x,c}^h)$ , describes the event  $A_{x,c}^{\text{imp}} = 0$  in terms of  $T_{x,c}^h$  only. Going for a full such characterization is possible but messy, and it is not clear what the worst-case structure of such  $T_{x,c}^h$  will be; this is the reason why we, when considering our impatient assignment mechanism, will look only up to depth 3 in  $T_{x,c}^h$ . For each node  $w$  of  $T_{x,c}^h$  at level 1, 2, or 3, we define event  $\text{LocalImpCut}_p(v)$  as follows:

1. If  $v$  is at level 3 of  $T_{x,c}^h$  then  $\text{LocalImpCut}_p(v)$  happens if  $\pi(\text{varlabel}(v)) < p$ .
2. If  $v$  is at level 2 of  $T_{x,c}^h$  then  $\text{LocalImpCut}_p(v)$  happens if  $\text{LocalImpCut}_p(w)$  happens for the  $k - 1$  children  $w$  of  $v$  (recall that  $\text{clauselabel}(v)$  is a critical clause and therefore  $v$  has exactly  $k - 1$  children);
3. If  $v$  is at level 1, set  $y := \text{varlabel}(v)$ ;  $\text{LocalImpCut}_p(v)$  happens if
  - a.  $\pi(y) < p$  or
  - b.  $\mathbb{I}_y = 1$  and  $\text{LocalImpCut}_p(v)$  happens for at least  $d - 2$  of the  $d - 1$  children of  $v$ .

Finally, we define

$$\text{ImpCut}_p(T_{x,c}^h) := \bigwedge_{i=1}^{k-1} (\text{Cut}_p(T_i) \vee \text{LocalImpCut}_p(v_i)) \quad (3)$$

The next lemma is the “impatient analog” of Lemma 12.

► **Lemma 20.** *Let  $p = \pi(x)$ . If  $\text{ImpCut}_p(T_{x,c}^h)$  happens then  $A_{x,c}^{\text{imp}} = 0$ .*

The proof is very similar to that of Lemma 12, just taking into account the impatient assignment mechanism. We restate and prove it as Lemma 26 in the appendix. Next, we prove a lower bound on  $\Pr[\text{ImpCut}_p(T_{x,c}^h)]$ . For  $q \in [0, 1]$  and  $l \in \mathbb{N}$ , define

$$\text{abamo}(q, l) := q^l + l(1 - q)q^{l-1} . \quad (4)$$

The name abamo is the acronym of “all but at most one” and is indeed the probability that, among  $l$  independent events of probability  $q$  each, all or all but one happen. Recall the definition of  $Q(p) := \Pr[\text{Cut}_p(T^\infty)]$  just before Proposition 13.

► **Lemma 21.** *If  $p < \theta$  then  $\Pr[\text{ImpCut}_p(T_{x,c}^h) \mid \pi(x) = p]$  is at least*

$$(p + c(\theta - p)\text{abamo}(p^{k-1}, d - 1) + (1 - p - c(\theta - p))Q(p)^{d-1})^{k-1} - \text{error}(d, k, h) .$$

*If  $p \geq \theta$  then it is at least  $Q(p) - \text{error}(d, k, h)$ .*

**Proof sketch.** For each subtree  $T_i$  of  $T_{x,c}^h$ , either  $\text{Cut}_p(T_i)$  or  $\text{LocalImpCut}_p(v_i)$  must happen. Now this happens if either (1)  $\pi(y) < p$ , which explains the first term of the sum in the parentheses; (2)  $\pi(y) \geq p$  and  $\mathbb{I}_{y_i} = 1$  and  $\text{LocalImpCut}_p(v_i)$ , which is the second term; or (3)  $\pi(y) \geq p$  and  $\mathbb{I}_{y_i} = 0$  and  $\text{Cut}_p(T_i)$ , which is the third term. See Lemma 27 for a complete proof. ◀

Let us summarize our reasoning so far. Define an ensemble  $J_1^{\text{imp}}, \dots, J_{d-1}^{\text{imp}}$  of random variables in  $\{0, 1\}$  as follows: set  $p := \pi(x)$ ; then independently set each  $J_c^{\text{imp}}$  to 0 with probability  $W^{k-1}$  and 1 with probability  $1 - W^{k-1}$ , where

$$W = W(p) := \begin{cases} p + c(\theta - p)\text{abamo}(p^{k-1}, d - 1) + (1 - p - c(\theta - p))Q(p)^{d-1} & \text{if } p < \theta \\ R(p) & \text{else.} \end{cases}$$

One checks that  $W(p)$  is continuous at  $p = \theta$  since  $R(p) = p + (1 - p)R(p)^{(k-1)(d-1)} = p + (1 - p)Q(p)^{d-1}$ . Set  $J^{\text{imp}} := J_1^{\text{imp}} + \dots + J_{d-1}^{\text{imp}} + 1$ . We have shown so far that

$$\begin{aligned} \mathbb{E} [\log_2 \max(1 + \mathbb{I}_x, A_{x,c}^{\text{imp}})] &\leq \mathbb{E} [\log_2 \max(1 + \mathbb{I}_x, J^{\text{imp}})] + \text{error}(d, k, h) \\ &= \Pr[J^{\text{imp}} = 1 \wedge \mathbb{I}_x] + \mathbb{E} [\log_2(J^{\text{imp}})] + \text{error}(d, h, k) . \end{aligned} \quad (5)$$

► **Proposition 22.**  $\Pr[J^{\text{imp}} = 1 \wedge \mathbb{I}_x] \leq \frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2})$ .

► **Proposition 23.**  $\mathbb{E} [\log_2(J^{\text{imp}})] - S_{d,k} \leq (d - 1) \log_2(1 - 1/d) \cdot \left( \frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2}) \right)$ .

We prove the two propositions in Section E in the appendix. Together with (5), they imply that  $\mathbb{E} [\log_2 \max(1 + \mathbb{I}_x, A_{x,c}^{\text{imp}})] - S_{d,k}$  is at most

$$\left( \frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2}) \right) (1 + (d - 1) \log_2(1 - 1/d)) + \text{error}(d, k, h) .$$

The expression in the first parenthesis is positive for sufficiently small  $\theta$ ; in fact, we have to choose  $\theta$  small enough to beat the hidden constant in the  $O(\cdot)$ , which in turn depends only on  $d$  and  $k$ . The expression in the second parenthesis,  $1 + (d - 1) \log_2(1 - 1/d)$ , is negative for all  $d \geq 3$ . It is maximized for  $d = 3$ , where it becomes  $2 - 2 \log_2(3) < -0.1699$ . Thus, we can choose  $\theta$  such that the whole expression is at most  $S_{d,k} - \epsilon_{d,k} + \text{error}(d, k, h)$  for some  $\epsilon_{d,k} > 0$  depending only on  $d$  and  $k$ . This concludes the proof of Lemma 19. ◀

## 4 Future Work

In the analysis of PPSZ, the worst case happens if all everything looks “nice”: all variable nodes in  $T_{x,1}, \dots, T_{x,d-1}$  have different labels; all clause labels are critical clauses.

In this scenario, our analysis for impatient assignment could go deeper than level 3; we could define a more powerful event ImpCut and obtain much better bounds on the running time. Indeed, future work hopefully will identify the worst-case shape of the  $T_{x,c}^h$  and allow us to analyze the full power impatient assignment.

The condition  $|\text{Plaus}(y, F^{[\alpha]})| \leq 2$  in Line 4 in Algorithm 2 is arbitrary. Why “ $\leq 2$ ”? Why not “ $\leq 3$ ”? For large  $d$ , what would the optimal cut-off value be?

---

### References

- 1 Richard Beigel and David Eppstein. 3-coloring in time  $O(1.3289^n)$ . *J. Algorithms*, 54(2):168–204, 2005. doi:10.1016/j.jalgor.2004.06.008.
- 2 Thomas Dueholm Hansen, Haim Kaplan, Or Zamir, and Uri Zwick. Faster  $k$ -SAT algorithms using biased-PPSZ. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 578–589. ACM, 2019. doi:10.1145/3313276.3316359.
- 3 Timon Hertli. 3-SAT faster and simpler – unique-SAT bounds for PPSZ hold in general. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science – FOCS 2011*, pages 277–284. IEEE Computer Soc., Los Alamitos, CA, 2011. doi:10.1109/FOCS.2011.22.
- 4 Timon Hertli, Isabelle Hurbain, Sebastian Millius, Robin A Moser, Dominik Scheder, and May Szedlák. The PPSZ algorithm for constraint satisfaction problems on more than two colors. In *International Conference on Principles and Practice of Constraint Programming*, pages 421–437. Springer, 2016.
- 5 Michal Koucký, Vojtech Rödl, and Navid Talebanfard. A separator theorem for hypergraphs and a CSP-SAT algorithm. *CoRR*, abs/2105.06744, 2021. arXiv:2105.06744.
- 6 Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An improved exponential-time algorithm for  $k$ -SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.
- 7 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 566–574. IEEE, 1997.
- 8 Dominik Scheder. PPZ for more than two truth values—an algorithm for constraint satisfaction problems. *arXiv preprint*, 2010. arXiv:1010.5717.
- 9 Dominik Scheder. PPSZ is better than you think. *Electron. Colloquium Comput. Complex.*, 28:69, 2021. URL: <https://eccc.weizmann.ac.il/report/2021/069>.
- 10 Dominik Scheder and John P. Steinberger. PPSZ for General  $k$ -SAT - making Hertli’s analysis simpler and 3-SAT faster. In Ryan O’Donnell, editor, *32nd Computational Complexity Conference, CCC 2017, July 6-9, 2017, Riga, Latvia*, volume 79 of *LIPICs*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.CCC.2017.9.
- 11 Uwe Schöningh. A probabilistic algorithm for  $k$ -SAT and constraint satisfaction problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 410–414. IEEE Computer Society, Los Alamitos, CA, 1999. doi:10.1109/SFPCS.1999.814612.

## A Independence between colors

► **Lemma 24** (Lemma 8, restated). *Let  $\pi : V \rightarrow [0, 1]$  be uniformly random and set  $p := \pi(x)$ . We sample  $d$  random variables  $\tilde{A}_{x,c}^{\text{imp}} \in \{0, 1\}$ ,  $c = 1, \dots, d$  by setting each  $\tilde{A}_{x,c}^{\text{imp}}$  to 1 with probability  $\Pr[A_{x,c}^{\text{imp}} = 1 \mid \pi(x) = p]$ , independently. Set  $\tilde{A}_x^{\text{imp}} := \sum_c \tilde{A}_{x,c}^{\text{imp}}$ . Then*

$$\mathbb{E}_{\pi} \left[ \log_2 \left( \max \left( 1 + \mathbb{I}_x, A_x^{\text{imp}}(\pi) \right) \right) \right] \leq \mathbb{E}_{\pi} \left[ \log_2 \left( \max \left( 1 + \mathbb{I}_x, \tilde{A}_x^{\text{imp}}(\pi) \right) \right) \right] \quad (6)$$

**Proof.** We prove (6) conditioned on  $\pi(x) = p$ . Let  $\mathbf{Z} \in \{0, 1\}^{V \setminus \{x\}}$  be defined by  $Z_y := [\pi(y) \geq p]$ . Note that each  $Z_y$  is 1 with probability  $1 - p$ , independently. Next, observe that each  $A_{x,c}^{\text{imp}}$  is a monotone increasing Boolean function  $f_c(Z)$ : moving some  $\pi(y)$  above  $p$  can only increase  $A_{x,c}^{\text{imp}}$ . Let  $\mathbf{Z}^{(1)}, \dots, \mathbf{Z}^{(d)}$  be  $d$  independent copies of  $\mathbf{Z}$ ; that is, each has the same distribution as  $\mathbf{Z}$  but they are independent. Conditioned on  $\pi(x) = p$ , we have  $(f_1(\mathbf{Z}), \dots, f_d(\mathbf{Z})) \sim (A_{x,1}^{\text{imp}}, \dots, A_{x,d}^{\text{imp}})$  and  $(f_1(\mathbf{Z}^{(1)}), \dots, f_d(\mathbf{Z}^{(d)})) \sim (\tilde{A}_{x,1}^{\text{imp}}, \dots, \tilde{A}_{x,d}^{\text{imp}})$ , where  $A \sim B$  means that the random variables  $A$  and  $B$  have the same distribution.

Now if  $p > \theta$  and therefore  $\mathbb{I}_x = 0$ , then the function  $\log_2(\max(1 + \mathbb{I}_x, \cdot))$  in (6) becomes  $\log_2(\cdot)$  and we can directly apply the Concave Correlation Lemma (Lemma A.1 of the full version of [4]).

If  $\mathbb{I}_x = 1$ , the trouble is that the function  $t \mapsto \log_2(\max(2, t))$  is not concave anymore. However, note that if  $\pi(x) < \theta$ , we set  $\mathbb{I}_x$  to 1 with probability  $\zeta$  and 0 with probability  $1 - \zeta$ . Conditioned on  $\pi(x) = p$ , the randomness in (6) comes from two sources: (1) the choice of  $\mathbb{I}_x$ ; (2) the randomness in  $\mathbf{Z}$  (or  $\mathbf{Z}^{(1)}, \dots, \mathbf{Z}^{(d)}$  for the right-hand side). We can break down both sides of (6) as follows:

$$\mathbb{E}_{\mathbf{Z}, \mathbb{I}_x} [\log_2(\max(1 + \mathbb{I}_x, A_x^{\text{imp}}))] = \mathbb{E}_{\mathbf{Z}} [\zeta \log_2(\max(2, A_x^{\text{imp}})) + (1 - \zeta) \log_2(A_x^{\text{imp}})] , \quad (7)$$

where  $\zeta = 2 - \log_2(3)$ . Now the function  $t \mapsto \zeta \log_2(\max(2, t)) + (1 - \zeta) \log_2(t)$  is still not concave. However, note that the arguments of  $\log_2(t)$  in (6) and (7) are integers; define  $g(t)$  to be the function that equals  $\log_2(t)$  if  $t$  is an integer, and is linear between integers. Now  $g$  is concave and  $t \mapsto \zeta g(\max(2, t)) + (1 - \zeta)g(t)$  is concave, too. In fact, this function is linear on  $[1, 3]$  and agrees with  $g$  for  $t \geq 3$ . Now the lemma again follows by the Concave Correlation Lemma (Lemma A.1 of [4]).  $\blacktriangleleft$

## B PPSZ for privileged variables

► **Proposition 25** (Proposition 17, restated). *Suppose  $x \in \text{vbl}(F)$  is a privileged variable. Then there is an  $\epsilon_{\text{privileged}} > 0$ , depending only on  $d$  and  $k$ , such that*

$$\mathbb{E} [\log_2(A_x)] \leq S_{d,k} - \epsilon_{\text{privileged}} + \text{error}(d, k, h) ,$$

for every privileged variable  $x$  in  $F$ .

**Proof.** This proof is similar in spirit and also technical details to the proof of Lemma 19 in [9], except that the latter is concerned with SAT (i.e., the case  $d = 2$ ).

Note that a variable  $x$  can be privileged for two reasons: first, there is some color  $c$  such that the critical clause tree  $T_{x,c}^h$  has fewer than  $(k - 1)L$  leaves at level 3; in other words, some clause node  $v$  at level 2 has fewer than  $k - 1$  children (note that the nodes at level 0 and 1 have the “right” number of children; the clause label of 0 is a critical clause, and therefore the root has always  $k - 1$  children; an odd-level node always has  $d - 1$  children). The second reason would be that, for some color  $c$ , level 1 and 3 of the critical clause tree  $T_{x,c}^h$  contain nodes  $u$  and  $v$  with  $\text{varlabel}(u) = \text{varlabel}(v)$ .

It is easy to see that the first kind of privilege is stronger: let  $v$  be the level-2 node with fewer than  $k - 1$  children. We can add “fictitious” subtrees until  $v$  has  $k - 1$  children, and make sure that one of the added children shares its variable label with an already-existing level-3 node. The result of this operation,  $T'_{x,c}$ , exhibits a privilege of the second kind, and  $\text{Cut}_p(T_{x,c}^h) \supseteq \text{Cut}_p(T'_{x,c})$ .

Thus, let us assume that  $x$  is privileged because  $T_{x,c}^h$  contains two nodes  $v$  and  $w$  with  $\text{varlabel}(v) = \text{varlabel}(w) = z$  and the depths of  $v$  and  $w$  are in  $\{1, 3\}$ . Analogous to the proof of Proposition 14 (Lemma 3.5 in [4]), we start with iteratively assign fresh labels to variable

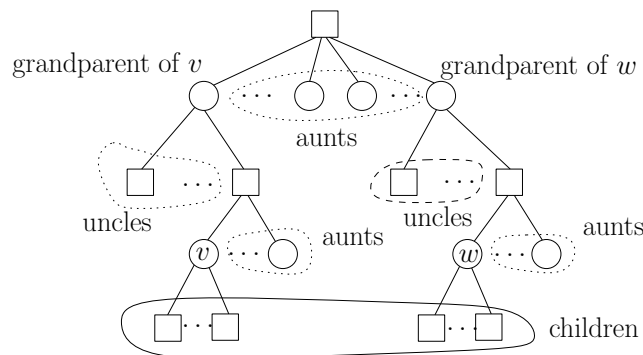


nodes; as shown in [4], this never increases  $\Pr[\text{Cut}_p(T_{x,c}^h)]$ . We apply this to all variable nodes except  $v$  and  $w$ , and obtain a new tree  $T$ . We make sure that there are no “missing children” in  $T$ , i.e., that every clause has  $k - 1$  children; this can be achieved by attaching fictitious subtrees, which does not increase  $\Pr[\text{Cut}_p(T)]$ . Also, we will for convenience assume that  $T$  is infinite, i.e., has no safe leaves (and no unsafe leaves, either). This does increase  $\Pr[\text{Cut}_p]$ , but by at most  $\text{error}(d, k, h)$ . In  $T$  we still have  $\text{varlabel}(v) = \text{varlabel}(w) = z$ , but all other labels are distinct. Let  $T'$  be the tree where  $v$  and  $w$  receive fresh labels  $z_v, z_w$ . We already know that  $\Pr[\text{Cut}_p(T')] = Q(p)$ . It remains to show that  $\Pr[\text{Cut}_p(T)]$  is substantially larger than  $\Pr[\text{Cut}_p(T')]$ . For this, let  $\mathbb{L}$  be the set of variable labels appearing in  $T$  and  $T'$ , and let  $\tau : \mathbb{L} \setminus \{z, z_v, z_w\} \rightarrow [0, 1]$ . We will analyze the difference

$$\Pr[\text{Cut}_p(T) \mid \tau] - \Pr[\text{Cut}_p(T') \mid \tau] \tag{8}$$

for fixed  $\tau$ . Introduce the three Boolean variables  $a := [\pi(z) < p]$ ,  $a_v := [\pi(z_v) < p]$ , and  $a_w := [\pi(z_w) < p]$ . Note that under  $\tau$ , the event  $\text{Cut}_p(T')$  reduces to  $f_\tau(a_v, a_w)$  for some monotone Boolean function and  $\text{Cut}_p(T)$  reduces to  $f_\tau(a, a)$ , for the same function  $f_\tau$ . There are only six possible such functions:  $f_\tau(a_v, a_w)$  is either 0, 1,  $a_v$ ,  $a_w$ ,  $a_v \wedge a_w$ , or  $a_v \vee a_w$ . If it is one of the first four, then  $\Pr[f_\tau(a_v, a_w)] = \Pr[f_\tau(a, a)]$  and (8) is 0. It cannot be  $a_v \vee a_w$ : the nodes  $v$  and  $w$  are not ancestors of each other. Finally, if  $f_\tau(a_v, a_w) = a_v \wedge a_w$  then we call  $\tau$  *pivotal* and observe that (8) becomes  $p - p^2$ .

From here on, our plan is to lower bound the probability that  $\tau$  is pivotal. We give a necessary and sufficient criterion for  $\tau$  to be pivotal.<sup>3</sup> It is best illustrated with a figure.



Squares are the clause nodes and circles are the variable nodes. Note that we assume that  $v$  and  $w$  are both on level 3, and their lowest common ancestor is the root. In the other cases, the picture and the subsequent calculation will be slightly different. To ease notation, we adopt the notation  $\text{Cut}_p(u) := \text{Cut}_p(T_u)$ , where  $T_u$  is the subtree of  $T'$  rooted at  $u$  (note that  $T'$  and  $T$  have the same node set, only some labels differ). In the case depicted in the figure,  $\tau$  is pivotal if and only if

1.  $\text{Cut}_p(u)$  happens for all aunts and uncles  $u$ ;
2.  $\text{Cut}_p(u)$  does not happen for all children  $u$  of  $v$ ; neither for all children  $u$  of  $w$ .
3.  $\pi(\text{grandparent of } v), \pi(\text{grandparent of } w) \geq p$ .

<sup>3</sup> Actually, it is sufficient for our purposes that the criterion be sufficient, and not necessary that it be necessary.

### 33:16 Impatient PPSZ

Furthermore, note that  $\Pr[\text{Cut}_p(u)]$  equals  $Q(p)$  if  $u$  is an uncle and  $R(p)$  if  $u$  is an aunt. Therefore,

$$\begin{aligned} \Pr[\text{Cut}_p(T)] - \Pr[\text{Cut}_p(T')] &\geq (p - p^2) \cdot \Pr[\tau \text{ is pivotal}] = \\ &(p - p^2)Q(p)^{\text{uncles}} \cdot R(p)^{\text{aunts}} \cdot (1 - Q(p)^{d-1})^2 (1 - p)^2 \\ &=: \delta(p) . \end{aligned}$$

It is clear that  $\delta(p) > 0$  for  $0 < p < 1 - 1/N$  and  $\delta(p) = 0$  for  $p \geq 1 - 1/N$ . Recalling the definition of  $S_{d,k} = \mathbb{E}[\log(J_1 + \dots + J_{d-1} + 1)]$  comparing it to  $\mathbb{E}[\log_2(A_x)] = \mathbb{E}[\log_2(A_{x,1} + \dots + A_{x,d-1} + 1)]$ , we can couple the ensembles  $\mathbf{A} := (A_{x,c})_{c=1}^{d-1}$  and  $\mathbf{J} := (J_c)_{c=1}^{d-1}$  such that  $\mathbf{A} \leq \mathbf{J}$  except with probability error( $d, k, p, h$ ), and  $A_{x,c} = 0, J_c = 1$ , conditioned on  $\pi(x) = p$ , happens with probability at least  $\delta(p) - \text{error}(d, k, p, h)$ . In fact, let us ignore the term error( $d, k, h$ ) for now and simply assume that  $\mathbf{A} \leq \mathbf{J}$  (more rigorously, we would have to replace every  $T_{x,c}^h$  by the appropriate infinite version; we decide to simply ignore error( $d, k, h$ ) in the following, lest we overload the reader with our notation). Set  $\Delta := J - A_x$ , and observe that  $\Delta \geq 0$  and  $\Pr[\Delta \geq 1 \mid \pi(x) = p] \geq \delta(p)$ .

$$\begin{aligned} \mathbb{E}[\log_2(J)] - \mathbb{E}[\log_2(A_x)] &= -\mathbb{E}\left[\log_2\left(\frac{J - \Delta}{J}\right)\right] \\ &= -\mathbb{E}\left[\log_2\left(1 - \frac{\Delta}{J}\right)\right] \\ &\geq -\mathbb{E}\left[\log_2\left(1 - \frac{\Delta}{d}\right)\right] \\ &\geq \frac{\log_2(e)}{d} \mathbb{E}[\Delta] \\ &\geq \frac{\log_2(e)}{d} \int_0^1 \delta(p) dp =: \epsilon_{\text{privileged}} . \end{aligned}$$

This is some positive number, and it depends only on  $d$  and  $k$ . ◀

## C Local reasoning for ImpatientPPSZ

► **Lemma 26** (Lemma 20, restated). *Suppose  $x \in \text{vbl}(F)$  is non-privileged. Let  $p = \pi(x)$ . If  $\text{ImpCut}_p(T_{x,c}^h)$  happens then  $A_{x,c}^{\text{imp}} = 0$ .*

**Proof.** We will prove the contrapositive: assume that  $A_{x,c}^{\text{imp}} = 1$  and show that  $\text{ImpCut}_p(T_{x,c}^h)$  does not happen. Let  $F(T_{x,c}^h)$  denote the set of clause labels appearing in  $T_{x,c}^h$ . Since  $A_{x,c}^{\text{imp}} = 1$  by assumption, the formula  $F^{[V_x^{\text{imp}} \mapsto d]}$  does not  $D$ -imply  $(x \neq c)$ . In particular,  $|F(T_{x,c}^h)| \leq D$  and therefore  $F(T_{x,c}^h)^{[V_x^{\text{imp}} \mapsto d]}$  does not imply  $(x \neq c)$ . This means that there is an assignment  $\gamma$  that (1) satisfies  $F(T_{x,c}^h)$ , (2)  $\gamma(x) = c$ , (3)  $\gamma(y) = d$  for all  $y \in V_x^{\text{imp}}$ .

As a first step, we will show that  $\text{Cut}_p(T_{x,c}^h)$  does not happen. For this, we will construct a sequence of clause nodes  $u_0, u_1, \dots$ , with  $u_0$  being the root and  $u_{i+1}$  being a grandchild of  $u_i$ , keeping the following invariant:

**Invariant.** For every clause node  $u$  in the sequence,  $\beta_u(y) \neq d \Rightarrow \gamma(y) = \beta_u(y)$ .

Note that the invariant is satisfied for the root:  $x$  is the only variable with  $\beta_{\text{root}}(x) \neq d$ , and  $\gamma(x) = c = \beta_{\text{root}}(x)$ . To find  $u_{i+1}$  from  $u_i$ , let  $C_i$  be the clause label of  $u_i$ , and write  $C_i$  as

$$C_i = (y_1 \neq c_1 \vee \dots \vee y_l \neq c_l \vee z_{l+1} \neq d \vee \dots \vee z_{k-1} \neq d) ,$$

where  $c_1, \dots, c_l \neq d$ . By construction,  $\beta_{u_i}$  violates  $C_i$ , and therefore  $\beta_{u_i}(y_j) = c_j$  for  $1 \leq j \leq l$ ; by the invariant,  $\gamma(y_j) = c_j$ , too. But  $\gamma$  satisfies  $C_i$  (it satisfies every clause label in  $T_{x,c}^h$ ), and therefore  $\gamma(z_j) = c \neq d$  for some  $l+1 \leq j \leq k-1$ . In particular,  $u_i$  has children. Let  $v$  be the child of  $u_i$  with variable label  $z_j$ . If  $v$  is a leaf (a safe leaf), terminate the process and call the path from root to  $v$  the *witness path*. Otherwise, and let  $u_{i+1}$  be the child of  $v$  with  $EC(v, u_{i+1}) = c$ . Note that  $u_{i+1}$  satisfies the invariant.

Since  $T_{x,c}^h$  is finite, this process terminates with a witness path. Note that  $\gamma(y) \neq d$  for all variable labels  $y$  appearing on that path. In particular, this means that  $y \notin V_x^{\text{imp}}$ , thus  $y \notin V_x$ , thus  $\pi(y) \geq \pi(x)$ . In other words,  $\text{Cut}_p(T_{x,c}^h)$  does not happen.

Without loss of generality, let  $v_1$  be the level-1-node on the witness path, and  $T_1$  be the tree rooted at  $v_1$ , and  $y_1 := \text{varlabel}(v_1)$ . Observe that  $\text{Cut}_p(T_1)$  does not happen. We will now show that  $\text{LocalImpCut}_p(v_1)$  does not happen, either. Assume, for the sake of contradiction, that  $\text{LocalImpCut}_p(v_1)$  happens. Does it happen because of Point 3a in the definition? Certainly not:  $\gamma(y_1) \neq d$  since  $v_1$  is on the witness path, and thus  $\pi(y_1) \geq p$ . So it happens because of Point 3b, and  $\mathbb{I}_{y_1} = 1$ ; without loss of generality, this means that  $\text{LocalImpCut}_p(v_1)$  happens for the first  $d-2$  children  $w_1, \dots, w_{d-2}$  of  $v_1$ ; let  $C_1, \dots, C_{d-2}$  be the respective clause labels. All those  $C_i$  are critical clauses ( $x$  is non-privilegeded, remember), and have  $k-1$  children each. So  $\text{LocalImpCut}_p$  happens for the first  $(k-1)(d-2)$  of the  $(k-1)(d-1)$  grandchildren of  $v_1$ . In other words, all their variable labels  $z$  have  $\pi(z) < p$  and thus  $z \in V_x$ . Under the assignment  $[V_x \mapsto d]$ , each of  $C_i$  reduces to a unit clause; this unit clause is still violated by  $\beta_{w_i}$  and is therefore either  $(y_1 \neq i)$  or  $(x \neq c)$ . If it was  $(x \neq c)$  then  $F(T_{x,c}^h)^{[V_x \mapsto d]}$  would imply  $(x \neq c)$  and therefore  $A_{x,c} = A_{x,c}^{\text{imp}} = 0$ , contradicting our assumption. So it is  $(y_1 \neq i)$ . In other words,  $F(T_{x,c}^h)^{[V_x \mapsto d]}$  contains the unit clauses  $(y_1 \neq 1), \dots, (y_1 \neq d-2)$ ; thus, when  $x$  is being processed by ImpatientPPSZ, the set of plausible values for  $y$  has been reduced to at most two values:  $d-1$  and  $d$ ; since  $\mathbb{I}_{y_1} = 1$ , the algorithm will assign  $y_1$  a value in Line 6, and  $y_1 \in V_x^{\text{imp}}$ . This is again a contradiction:  $\gamma(y_1) \neq d$  since  $v_1$  is on the witness path;  $\gamma(y_1) = d$  since  $y_1 \in V_x^{\text{imp}}$ . This concludes the proof.  $\blacktriangleleft$

## D ImpCut probability

Suppose  $x \in \text{vbl}(F)$  is non-privilegeded and  $T_{x,c}^h$  is a critical clause tree for  $x$  and  $c \in [d]$ .

► **Lemma 27** (Lemma 21, restated). *If  $p < \theta$  then  $\Pr[\text{ImpCut}_p(T_{x,c}^h) \mid \pi(x) = p]$  is at least*

$$(p + c(\theta - p)\text{abamo}(p^{k-1}, d-1) + (1 - p - c(\theta - p))Q(p)^{d-1})^{k-1} - \text{error}(d, k, h) .$$

*If  $p \geq \theta$  then it is at least  $Q(p) - \text{error}(d, k, h)$ .*

**Proof.** If  $p \geq \theta$  then this is obvious since already  $\text{Cut}_p(T_{x,c}^h)$  has probability at least  $Q(p) - \text{error}(d, k, h)$ , by Proposition 14. Thus we assume  $p < \theta$ . The root of  $T_{x,c}^h$  has  $k-1$  children  $v_1, \dots, v_{k-1}$ , whose respective variable labels are  $y_1, \dots, y_{k-1}$ . Let  $T_i$  denote the subtree of  $T_{x,c}^h$  rooted at  $v_i$ .

$$\begin{aligned} \Pr [\text{ImpCut}_p(T_{x,c}^h)] &= \Pr \left[ \bigwedge_{i=1}^{k-1} (\text{Cut}_p(T_i) \vee \text{LocalImpCut}_p(v_i)) \right] \\ &\geq \prod_{i=1}^{k-1} (\Pr[\text{Cut}_p(T_i) \vee \text{LocalImpCut}_p(v_i)]) . \quad (\text{FKG inequality}) \end{aligned}$$

We can apply the FKG inequality because each event  $\text{Cut}_p(T_i) \vee \text{LocalImpCut}_p(v_i)$  is a monotone increasing Boolean function in the variables  $[\pi(z) < p]$  and  $\mathbb{I}_{y_i}$ . It remains to show that, for each  $1 \leq i \leq k-1$ , the event  $\text{Cut}_p(T_i) \vee \text{LocalImpCut}_p(v_i)$  happens with probability at least

$$p + c(\theta - p)\text{abamo}(p^{k-1}, d-1) + (1 - p - c(\theta - p))Q(p)^{d-1} - \text{error}(d, k, h) \quad (9)$$

For this, let us abbreviate  $T := T_i$ ,  $v := v_i$  its root, and  $y := \text{varlabel}(v) = y_i$ ; also, we define the events  $A := \text{LocalImpCut}_p(v)$  and  $B := \text{Cut}_p(T)$ . We distinguish three cases:

- (i) if (1)  $\pi(y) < p$  then the desired event  $A \vee B$  happens;
- (ii) if  $\pi(y) \geq p$  and  $\mathbb{I}_y = 1$  (which implies  $\pi(y) < \theta$ ) then we ignore  $B$  and focus on  $A$ ;
- (iii) if  $\pi(y) \geq p$  and  $\mathbb{I}_y = 0$ , then  $A$  does not happen, so focus on  $B$ .

Formally,

$$\Pr[A \vee B] \geq \Pr[(i)] + \Pr[(ii)] \cdot \Pr[A \mid (ii)] + \Pr[(iii)] \cdot \Pr[B \mid (iii)]$$

Next, let us look at each case.

1.  $\Pr[(i)] = p$ ; this explains the first term in (9).
2.  $\Pr[(ii)] = c(\theta - p)$ . Furthermore, if (ii) happens, then  $A$  happens if and only if for at least  $d-2$  of the children  $w_1, \dots, w_{d-1}$ , the event  $A_j := \text{LocalImpCut}_p(w_j)$  happens. Each  $A_j$  happens with probability  $\rho := p^{k-1}$ ; they are independent since all  $(d-1)(k-1)$  grandchildren of  $v$  have distinct labels. Therefore,

$$\begin{aligned} \Pr[A \mid (ii)] &= \Pr[A_1 \wedge \dots \wedge A_{d-1}] + \sum_{j^*=1}^{d-1} \Pr[\neg A_{j^*} \wedge \bigwedge_{j \neq j^*} A_j] \\ &= \rho^{d-1} + (d-1)(1-\rho)\rho^{d-2} = \text{abamo}(p^{k-1}, d-1). \end{aligned}$$

This explains the second term in (9).

3.  $\Pr[(iii)] = 1 - p - c(\theta - p)$ . If (iii) happens, then  $B$  happens if and only if  $\text{Cut}_p(T)$  happens for each of the  $d-1$  subtrees of  $T$ . By Proposition 14, this happens with probability  $(Q(p) - \text{error}(d, k, h))^{d-1}$ . This explains the third and fourth term in (9).

This concludes the proof.  $\blacktriangleleft$

## E Bounding losses and gains. Proofs of Propositions 22 and 23

First, we need some good-enough estimates for our probabilities  $R(p)$ ,  $Q(p)$ , and  $W(p)$ . Note that  $R(p)$  and  $Q(p)$  are the roots of certain polynomials, and we do not have an explicit formula for them. The bounds in Proposition 28 are somewhat crude but sufficient for our purposes.

► **Proposition 28.**  $R(p) \leq p + 4p^L$ ;  $Q(p) \leq (p + 4p^L)^{k-1}$ ; and  $W(p) \leq p + O(\theta p^{(d-2)(k-1)})$ . The hidden constant in the  $O$  depends on  $d$  and  $k$  only.

**Proof.** One checks that  $R(p)$  is convex on the interval  $[0, 1 - 1/L]$ . To see this, note that for  $p \leq 1 - 1/L$ ,  $R(p)$  is the unique solution in  $[0, 1]$  of the equation

$$R = p + (1 - p)R^L,$$

by Proposition 13. We can solve explicitly for  $p$  and check that  $p(R)$  is concave, by elementary calculus. Since  $R$  is convex,  $R(0) = 0$ , and  $R(1 - 1/L) = 1$ , the graph of  $R(p)$  is below the line from  $(0, 0)$  to  $(1 - 1/L, 1)$ , and therefore  $R(p) \leq \frac{L}{L-1}p$ . This is not enough yet, but applying the equation of  $R$  to this estimate gives

$$R = p + (1-p)R^L \leq p + (1-p) \left( \frac{L}{L-1} p \right)^L \leq p + 4p^L.$$

The upper bound for  $Q$  follows directly from  $Q(p) = R(p)^{k-1}$ . It remains to prove the upper bound on  $W(p)$ :

$$\begin{aligned} W(p) &= p + c(\theta - p)\text{abamo}(p^{k-1}, d-1) + (1-p - c(\theta - p))Q(p)^{d-1} \\ &\leq p + \theta\text{abamo}(p^{k-1}, d-1) + Q(p)^{d-1} \\ &= p + \theta p^{(k-1)(d-1)} + \theta(d-1)(1-p^{k-1})p^{(k-1)(d-2)} + Q(p)^{d-1} \\ &\leq p + (d-1)\theta p^{(d-2)(k-1)} + R^L \\ &\leq p + (d-1)\theta p^{(d-2)(k-1)} + (p + 4p^L)^L \\ &\leq p + O\left(\theta p^{(d-2)(k-1)}\right). \end{aligned}$$

► **Proposition 29** (Proposition 22, restated).  $\Pr[J^{\text{imp}} = 1 \wedge \mathbb{I}_x] \leq \frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2})$ .

**Proof.** Recall that if  $\pi(x) < \theta$  then  $\mathbb{I}_x$  is 1 with probability  $c$  and 0 with probability  $1 - c$ . If  $\pi(x) \geq \theta$  then  $\mathbb{I}_x = 0$ . Also,  $J^{\text{imp}} = 1$  if and only if  $J_1^{\text{imp}} = \dots = J_{d-1}^{\text{imp}} = 0$ . Therefore,

$$\begin{aligned} \Pr[J^{\text{imp}} = 1 \wedge \mathbb{I}_x] &= c \cdot \int_0^\theta \Pr[J^{\text{imp}} = 1 \mid \pi(x) = p] dp = c \cdot \int_0^\theta W^{(d-1)(k-1)} dp \\ &= c \cdot \int_0^\theta (p + O(\theta p^{(d-2)(k-1)}))^L dp \leq c \cdot \int_0^\theta p^L (1 + O(\theta)) dp \\ &\hspace{15em} (\text{since } (d-2)(k-1) \geq 1) \\ &= \frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2}) \end{aligned}$$

This proves the proposition. ◀

► **Proposition 30** (Proposition 23, restated).  $\mathbb{E}[\log_2(J^{\text{imp}})] - S_{d,k} \leq (d-1)\log_2(1-1/d) \cdot \left( \frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2}) \right)$ .

**Proof.** Recall the definition of  $S_{d,k}$ : sample random variables  $J_1, \dots, J_{d-1}$  by setting  $p := \pi(x)$  and setting each  $J_c$  to 0 with probability  $Q(p)$  and to 1 with probability  $1 - Q(p)$ , and  $J = J_1 + \dots + J_{d-1} + 1$ . So the  $J_c$  are independent conditioned on  $\pi(x) = p$ . Then  $S_{d,k} = \mathbb{E}[\log_2(J)]$ . Set  $\Delta_c := J_c - J_c^{\text{imp}}$  and  $\Delta = \sum_c \Delta_c$ . Note that all  $\Delta_c$  have the same distribution.

► **Proposition 31.**  $\mathbb{E}[\Delta_1 \mid \pi(x) = p] \geq c(\theta - p)L(p^{L-1} - O(p^L))$  for all  $1 \leq c \leq d-1$ .

In particular, if  $p < \theta$  and  $\theta$  is sufficiently small then  $\mathbb{E}[\Delta_1] \geq 0$ . Therefore,  $\mathbb{E}[J_c] \leq \mathbb{E}[J_c^{\text{imp}}]$  and we can couple the ensemble  $(J_1, \dots, J_{d-1})$  and  $(J_1^{\text{imp}}, \dots, J_{d-1}^{\text{imp}})$  on a common probability space on which  $J_c \leq J_c^{\text{imp}}$ , always, and thus  $\Delta \geq 0$ . We therefore see that

### 33:20 Impatient PPSZ

$\mathbb{E}[\log_2(J^{\text{imp}})] - S_{d,k}$  is

$$\begin{aligned} \mathbb{E}[\log_2(J^{\text{imp}}) - \log_2(J)] &= \mathbb{E}\left[\log_2\left(1 - \frac{\Delta}{J}\right)\right] \\ &\leq \mathbb{E}\left[\log_2\left(1 - \frac{\Delta}{d}\right)\right] \leq \mathbb{E}\left[\log_2\left(\left(1 - \frac{1}{d}\right)^\Delta\right)\right] \\ &= \mathbb{E}[\Delta] \log_2\left(1 - \frac{1}{d}\right). \end{aligned}$$

Conditioned on  $\pi(x) = p$  and using Proposition 31, this is at most

$$c(\theta - p)L(p^{L-1} - O(p^L))(d-1)\log_2\left(1 - \frac{1}{d}\right).$$

We integrate this over  $p$  to get rid of the condition  $\pi(x) = p$  and see that

$$\mathbb{E}[\log_2(J^{\text{imp}})] - S_{d,k} \leq (d-1)\log_2\left(1 - \frac{1}{d}\right) \cdot \left(\frac{c}{L+1}\theta^{L+1} + O(\theta^{L+2})\right).$$

This concludes the proof of Proposition 30. ◀

It remains to prove Proposition 31.

#### Proof of Proposition 31.

$$\begin{aligned} \mathbb{E}[\Delta_1 \mid \pi(x) = p] &= \mathbb{E}[J_c - J_c^{\text{imp}} \mid \pi(x) = p] = (1 - Q) - (1 - W^{k-1}) = W^{k-1} - R^{k-1} \\ &\geq (k-1)(W - R)R^{k-2}, \end{aligned}$$

where the last inequality follows because  $W^{k-1} = (R + W - R)^{k-1} = R^{k-1} \left(1 + \frac{W-R}{R}\right)^{k-1} \geq R^{k-1} \left(1 + \frac{(k-1)(W-R)}{R}\right) = R^{k-1} + (k-1)(W-R)R^{k-2}$ . Now let us bound  $W - R$  from below. If  $p \geq \theta$  then  $W(p) = R(p)$  and  $W - R = 0$ . If  $p < \theta$ , we expand  $R(p)$  as follows:

$$R = p + (1-p)Q^{(d-1)} = p + c(\theta - p)Q^{d-1} + (1-p - c(\theta - p))Q^{d-1}$$

and therefore

$$\begin{aligned} W - R &= c(\theta - p) \left(\text{abamo}(p^{k-1}, d-1) - Q^{d-1}\right) \\ &= c(\theta - p) \left(p^{(k-1)(d-1)} + (d-1)(1-p^{k-1})p^{(k-1)(d-2)} - Q^{d-1}\right) \\ &\geq c(\theta - p) \left(p^L + (d-1)p^{(k-1)(d-2)} - (d-1)p^L - (p + O(p^2))^L\right) \\ &\geq c(\theta - p)(d-1) \left(p^{(k-1)(d-2)} - O(p^L)\right). \end{aligned}$$

Next, combining the previous two calculations, we see that

$$\begin{aligned} \mathbb{E}[\Delta_1 \mid \pi(x) = p] &\geq (k-1)(W - R)R^{k-2} \geq (k-1)(W - R)p^{k-2} \\ &\geq (k-1)c(\theta - p)(d-1) \left(p^{(k-1)(d-2)} - O(p^L)\right) p^{k-2} \\ &\geq c(\theta - p)L(p^{L-1} - O(p^L)). \end{aligned} \quad \blacktriangleleft$$

# Fine-Grained Meta-Theorems for Vertex Integrity

Michael Lampis  

Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

Valia Mitsou 

Université de Paris, IRIF, CNRS, 75205, Paris, France

---

## Abstract

---

Vertex Integrity is a graph measure which sits squarely between two more well-studied notions, namely vertex cover and tree-depth, and that has recently gained attention as a structural graph parameter. In this paper we investigate the algorithmic trade-offs involved with this parameter from the point of view of algorithmic meta-theorems for First-Order (FO) and Monadic Second Order (MSO) logic. Our positive results are the following: (i) given a graph  $G$  of vertex integrity  $k$  and an FO formula  $\phi$  with  $q$  quantifiers, deciding if  $G$  satisfies  $\phi$  can be done in time  $2^{O(k^2q + q \log q)} + n^{O(1)}$ ; (ii) for MSO formulas with  $q$  quantifiers, the same can be done in time  $2^{2^{O(k^2+kq)}} + n^{O(1)}$ . Both results are obtained using kernelization arguments, which pre-process the input to sizes  $2^{O(k^2)}q$  and  $2^{O(k^2+kq)}$  respectively.

The complexities of our meta-theorems are significantly better than the corresponding meta-theorems for tree-depth, which involve towers of exponentials. However, they are worse than the roughly  $2^{O(kq)}$  and  $2^{2^{O(k+q)}}$  complexities known for corresponding meta-theorems for vertex cover. To explain this deterioration we present two formula constructions which lead to fine-grained complexity lower bounds and establish that the dependence of our meta-theorems on  $k$  is best possible. More precisely, we show that it is not possible to decide FO formulas with  $q$  quantifiers in time  $2^{o(k^2q)}$ , and that there exists a constant-size MSO formula which cannot be decided in time  $2^{2^{o(k^2)}}$ , both under the ETH. Hence, the quadratic blow-up in the dependence on  $k$  is unavoidable and vertex integrity has a complexity for FO and MSO logic which is truly intermediate between vertex cover and tree-depth.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms

**Keywords and phrases** Model-Checking, Fine-grained complexity, Vertex Integrity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.34

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.10333>

**Funding** *Michael Lampis*: Partially supported by ANR JCJC project “ASSK” (ANR-18-CE40-0025-01).

## 1 Introduction

An algorithmic meta-theorem is a general statement proving that a large class of problems is tractable. Such results are of great importance because they allow one to quickly classify the complexity of a new problem, before endeavoring to design a fine-tuned algorithm. In the domain of parameterized complexity theory for graph problems, possibly the most well-studied type of meta-theorems are those where the class of problems in question is defined using a language of formal logic, typically a variant of First-Order (FO) or Monadic Second-Order (MSO) logic, which are the logics that allow quantification over vertices or sets of vertices respectively<sup>1</sup>. In this area, the most celebrated result is Courcelle’s theorem [6], which states

---

<sup>1</sup> Note that the version of MSO logic we use in this paper is sometimes also referred to as  $\text{MSO}_1$  to distinguish from the version that also allows quantification over sets of edges.



© Michael Lampis and Valia Mitsou;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiro Sadakane; Article No. 34; pp. 34:1–34:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



that all properties expressible in MSO logic are solvable in linear time, parameterized by treewidth and the size of the MSO formula. In the thirty years since the appearance of this fundamental result, numerous other meta-theorems in this spirit have followed (we give an overview of some such results below).

Despite its great success, Courcelle’s theorem suffers from one significant weakness: the algorithm it guarantees for deciding an MSO formula  $\phi$  on a graph  $G$  with  $n$  vertices and treewidth  $k$  has running time  $f(k, \phi) \cdot n$ , where  $f$  is, in the worst case, a tower of exponentials whose height can only be bounded as a function of  $\phi$ . Unfortunately, it has been known since the work of Frick and Grohe [20] that this terrible parameter dependence cannot be avoided, even if one only considers FO logic on trees (or MSO logic on paths [40]). This has motivated the study of the complexity of FO and MSO logic with parameters which are more restrictive than treewidth. In the context of such parameters, fixed-parameter tractability for all MSO-expressible problems is already given by Courcelle’s theorem, so the goal is to obtain more “fine-grained” meta-theorems which achieve a better dependence on  $\phi$  and  $k$ .

The two results from this line of research which are most relevant to our paper are the meta-theorems for vertex cover given in [39], and the meta-theorem for tree-depth given by Gajarský and Hliněný [21]. Regarding vertex cover, it was shown in [39] that FO and MSO formulas with  $q$  quantifiers can be decided on graphs with vertex cover  $k$  in time roughly  $2^{O(kq+q \log q)}$  and  $2^{2^{O(k+q)}}$  respectively. Both of these results were shown to be tight, in the sense that improving their dependence on  $k$  would violate the Exponential Time Hypothesis (ETH). For tree-depth, it was shown in [21] that FO and MSO formulas with  $q$  quantifiers can be decided on graphs with tree-depth  $k$  with a complexity that is roughly  $k$ -fold exponential. Hence, for fixed  $k$ , the complexity we obtain is elementary, but the height of the tower of exponentials increases with  $k$ , and this cannot be avoided under the ETH [40].

Vertex cover and tree-depth are among the most well-studied measures in parameterized complexity. In all graphs  $G$  we have  $\text{vc}(G)+1 \geq \text{td}(G) \geq \text{pw}(G) \geq \text{tw}(G)$ , so these parameters form a natural hierarchy with pathwidth and treewidth, with vertex cover being the most restrictive. As explained above, the distance between the performance of meta-theorems for vertex cover (which are double-exponential for MSO) and for tree-depth (which give a tower of exponentials of height  $\text{td}$ ) is huge, but conceptually this is perhaps not surprising. Indeed, one could argue that the structural distance between graphs of vertex cover  $k$  from the class of graphs of tree-depth  $k$  is also huge. As a reminder, a graph has vertex cover  $k$  if we can delete  $k$  vertices to obtain an independent set; while a graph has tree-depth  $k$  if there exists  $k' \leq k$  such that we can delete  $k'$  vertices to obtain a disjoint union of graphs of tree-depth  $k - k'$ . Clearly, the latter (inductive) definition is more powerful and covers vastly more graphs, so it is natural that model-checking should be significantly harder for tree-depth.

The landscape of parameters described above indicates that there should be space to investigate interesting structural parameters *between* vertex cover and tree-depth, exactly because the distance between these two is large in terms of generality and complexity. One notion that has recently attracted attention in this area is *Vertex Integrity* [11], denoted as  $\iota(G)$ . A graph has vertex integrity  $k$  if there exists  $k' \leq k$  such that we can delete  $k'$  vertices and obtain a disjoint union of graphs of *size* at most  $k - k'$ . Hence, the definition of vertex integrity is the same as for tree-depth, except that we replace the inductive step by simply bounding the size of the components that result after deleting a separator of the graph. This produces a notion that is more restrictive than tree-depth, but still significantly more general than vertex cover (where the resulting components must be singletons). In all graphs  $G$ , we have  $\text{vc}(G) + 1 \geq \iota(G) \geq \text{td}(G)$ , so it becomes an interesting question to investigate the complexity trade-off associated with these parameters, that is, how the complexity of various

problems deteriorates as we move from vertex cover, to vertex integrity, to tree-depth. This type of study was recently undertaken systematically for many problems by Gima et al. [29]. In this paper we make an investigation in the same direction from the lens of algorithmic meta-theorems.

**Our results.** We consider the problem of verifying whether a graph  $G$  satisfies a property given by an FO or MSO formula with  $q$  quantifiers, assuming  $\iota(G) \leq k$ . Our goal is to give a fine-grained determination of the complexity of this problem as a function of  $k$ . We obtain the following two positive results:

1. FO formulas with  $q$  quantifiers can be decided in time  $2^{O(k^2q + q \log q)} + n^{O(1)}$ .
2. MSO formulas with  $q$  vertex and set quantifiers can be decided in time  $2^{2^{O(k^2 + kq)}} + n^{O(1)}$ .

Hence, we obtain meta-theorems stating that any problem that can be expressed in FO or MSO logic can be solved in the aforementioned times. Both of these results are obtained through a kernelization argument, similar in spirit to the arguments used in the meta-theorems of [21, 39]. To describe the main idea, recall that if  $\iota(G) \leq k$ , then there exists a separator  $S$  of size at most  $k$ , such that removing it will disconnect the graph into components of size at most  $k$ . The key now is that these components can be partitioned into  $2^{k^2}$  equivalence *types*, where components of the same type are isomorphic. We then argue that if we have a large number of isomorphic components, it is always safe to delete any one of them from the graph, as this does not change whether the given formula holds (Lemmas 12 and 14). We then complete the argument by applying the standard brute-force algorithms for FO and MSO logic on the kernels.

We complement the results above by showing that the approach of kernelizing and then executing the brute-force algorithm is best possible. More precisely, we show that, under the ETH, it is not possible to obtain a model-checking algorithm for FO logic running in time  $2^{o(k^2q)}n^{O(1)}$ ; while for MSO we construct a constant-sized formula which cannot be model-checked in time  $2^{2^{o(k^2)}}$ . Hence, the quadratic dependence on  $k$ , which distinguishes our meta-theorems from the corresponding meta-theorems for vertex cover, cannot be avoided.

**Related work.** The study of structural parameters which trade off the generality of treewidth for improved algorithmic properties is by now a standard topic in parameterized complexity. The most common type of work here is to consider a problem that is intractable parameterized by treewidth and see whether it becomes tractable parameterized by vertex cover or tree-depth [2, 10, 13, 16, 17, 31, 32, 35, 34, 36, 42, 41]. See [1] for a survey of results of this type. In this context, vertex integrity has only recently started being studied as an intermediate parameter between vertex cover and tree-depth, and it has been discovered that fixed-parameter tractability for several problems which are W-hard by tree-depth can be extended from vertex cover to vertex integrity [4, 12, 25, 27, 29]. Note that some works use a measure called *core fracture* number, which is an equivalent notion to vertex integrity.

Algorithmic meta-theorems are a well-studied topic in parameterized complexity (see [30] for a survey). Courcelle's theorem has been extended to the more general notion of clique-width [7], and more efficient versions of these meta-theorems have been given for the more restricted parameters twin-cover [22], shrub-depth [24, 23], neighborhood diversity and max-leaf number [39]. Meta-theorems have also been given for even more general graph parameters, such as [5, 14, 19, 18], and for logics other than FO and MSO, with the goal of either targeting a wider class of problems [26, 37, 38, 44], or achieving better complexity [43]. Meta-theorems have also been given in the context of kernelization [3, 15, 28] and

approximation [9]. To the best of our knowledge, the complexity of FO and MSO model checking parameterized by vertex integrity has not been explicitly studied before, but since vertex integrity is a restriction of tree-depth and a generalization of vertex cover, the algorithms of [21] and the lower bounds of [39] apply in this case.

## 2 Definitions and Preliminaries

First, let us formally define the notion of vertex integrity of a graph.

► **Definition 1.** *A graph  $G$  is said to have vertex integrity  $\iota(G)$  when there exists a set  $S \subset V(G)$  such that, if  $S' \subset V(G)$  is the set of vertices of the largest connected component of  $G \setminus S$  then  $|S| + |S'| \leq \iota(G)$ .*

We recall that Drange et al. [11] have shown that deciding if a graph has  $\iota(G) \leq k$  admits a kernel of order  $O(k^3)$ . Hence, given a graph  $G$  that is promised to have vertex integrity  $k$ , we can execute this kernelization algorithm and then look for the optimal separator  $S$  in the kernel. As a result, finding a separator  $S$  proving that  $\iota(G) \leq k$  can be done in  $k^{O(k)} + n^{O(1)}$ . Since this running time is dominated by the running times of our meta-theorems, we will always silently assume that the separator  $S$  is given in the input when the input graph has vertex integrity  $k$ .

A main question that will interest us is whether a graph satisfies a property expressible in First-Order (FO) or Monadic Second-Order (MSO) logic. Let us briefly recall the definitions of these logics. We use  $x_i, i \in \mathbb{N}$  to denote vertex (FO) variables and  $X_i, i \in \mathbb{N}$  to denote set (MSO) variables. Vertex variables take values from a set of vertex constants  $U = \{u_i, i \in \mathbb{N}\}$ , whereas vertex set variables take values from a set of vertex set constants  $D = \{D_i, i \in \mathbb{N}\}$ .

Now, given a graph  $G$ , in order to say that the assignment of a vertex variable  $x_i$  or a vertex set variable  $X_i$  to a constant corresponds to a particular vertex or vertex set of  $G$ , we make use of a *labeling* function  $\ell$  that maps vertex constants to vertices of  $V(G)$  and of a *coloring* function  $\mathcal{C}$  that maps vertex set constants to vertex sets of  $V(G)$ . More formally,  $\ell, \mathcal{C}$  are partial functions  $\ell : U \rightarrow V(G)$  and  $\mathcal{C} : D \rightarrow 2^{V(G)}$ . The functions may be undefined for some constants, for example, if  $\ell$  is not defined for the constant  $u_i$  we write  $\ell(u_i) \uparrow$ .

► **Definition 2.** *Given a triplet  $G, \ell, \mathcal{C}$ , a vertex  $v \in V(G)$  is said to be unlabeled if  $\nexists u_i \in U$  such that  $\ell(u_i) = v$ . A set of vertices  $C_1 \subseteq V(G)$  is unlabeled if all the vertices of  $C_1$  are unlabeled.*

► **Definition 3.** *We say that two labeling functions  $\ell, \ell'$  agree on a constant  $u_i$  if either they are both undefined on  $u_i$  or  $\ell(u_i) = \ell'(u_i)$ . Similarly, two coloring functions  $\mathcal{C}, \mathcal{C}'$  agree on  $D_i$  if they are both undefined or  $\mathcal{C}(D_i) = \mathcal{C}'(D_i)$ .*

► **Definition 4.** *Given two triplets  $G_1, \ell_1, \mathcal{C}_1$  and  $G_2, \ell_2, \mathcal{C}_2$  and a bijective function  $f : V(G_1) \rightarrow V(G_2)$ . For  $C_1 \subseteq V(G_1)$ , we define  $f(C_1) = \bigcup_{v \in C_1} \{f(v)\}$ . We say that  $V(G_1)$  and  $V(G_2)$  have the same labelings for  $f$  if  $\forall u_i \in U$ , either both  $\ell_1(u_i), \ell_2(u_i)$  are undefined or  $f(\ell_1(u_i)) = \ell_2(u_i)$ ; we say that  $V(G_1)$  and  $V(G_2)$  have the same colorings for  $f$  if  $\forall D_i \in D$ , either both  $\mathcal{C}_1(D_i), \mathcal{C}_2(D_i)$  are undefined or  $f(\mathcal{C}_1(D_i)) = \mathcal{C}_2(D_i)$ .*

► **Definition 5.** *An isomorphism between two triplets  $G_1, \ell_1, \mathcal{C}_1$  and  $G_2, \ell_2, \mathcal{C}_2$  is a bijective function  $f : V(G_1) \rightarrow V(G_2)$  such that (i) for all  $v, w \in V(G_1)$  we have  $(v, w) \in E(G_1)$  if and only if  $(f(v), f(w)) \in E(G_2)$ , (ii)  $V(G_1)$  and  $V(G_2)$  have the same labelings and colorings for  $f$ . Two triplets  $G_1, \ell_1, \mathcal{C}_1$  and  $G_2, \ell_2, \mathcal{C}_2$  are isomorphic if there exists an isomorphism between them.*

► **Definition 6.** Given a triplet  $G, \ell, \mathcal{C}$ . We say that two sets  $C_1 \subseteq V(G)$  and  $C_2 \subseteq V(G)$  have the same type if there exist  $\ell', \mathcal{C}'$  and an isomorphism  $f : V(G) \rightarrow V(G)$  between the triplets  $G, \ell, \mathcal{C}$  and itself such that  $f$  maps elements of  $C_1$  to  $C_2$  and vice versa and elements from  $V(G) \setminus (C_1 \cup C_2)$  to themselves.

Notice that only for vertices that don't belong in the sets  $C_1$  and  $C_2$  (which  $f$  maps to themselves) we can have that  $f(\ell(u_i)) = \ell(u_i)$ . This leads to the following observation:

► **Observation 7.** In order for two disjoint sets  $C_1$  and  $C_2$  to have the same type, they should necessarily be unlabeled (that is,  $\forall u_i, \ell(u_i) \notin C_1 \cup C_2$ ).

► **Definition 8.** Given a triplet  $G, \ell, \mathcal{C}$  and a set  $C_1 \subset V(G)$ . The restriction of  $\mathcal{C}$  to  $G \setminus C_1$  is a function  $\mathcal{C}' : D \rightarrow V(G) \setminus C_1$  such that  $\mathcal{C}'(D_i) = \mathcal{C}(D_i) \setminus C_1$  for all  $D_i \in D$  for which  $\mathcal{C}(D_i) \cap C_1 \neq \emptyset$  and  $\mathcal{C}, \mathcal{C}'$  agree on the rest of  $D_i$ .

An MSO formula is a formula produced by the following grammar, where  $X$  represents a set variable,  $x$  a vertex variable,  $y$  a vertex variable or vertex constant, and  $Y$  a set variable or constant:

$$\phi \rightarrow \exists X.\phi \mid \exists x.\phi \mid \phi \vee \phi \mid \neg\phi \mid y \sim y \mid y = y \mid y \in Y$$

The operations above are vertex set quantification, vertex quantification, disjunction, negation, edge relation, vertex equality, and set inclusion respectively. Their semantics are defined inductively in the usual way: given a triplet  $G, \ell, \mathcal{C}$  and an MSO formula  $\phi$ , we say that the graph satisfies the property described by  $\phi$ , or simply that  $G, \ell, \mathcal{C}$  models  $\phi$ , and write  $G, \ell, \mathcal{C} \models \phi$  according to the following rules:

- $G, \ell, \mathcal{C} \models u_i \in D_j$  if  $\ell(u_i)$  is defined and  $\ell(u_i) \in \mathcal{C}(D_j)$ .
- $G, \ell, \mathcal{C} \models u_i = u_j$  if  $\ell(u_i), \ell(u_j)$  are defined and  $\ell(u_i) = \ell(u_j)$ .
- $G, \ell, \mathcal{C} \models u_i \sim u_j$  if  $\ell(u_i), \ell(u_j)$  are defined and  $(\ell(u_i), \ell(u_j)) \in E(G)$ .
- $G, \ell, \mathcal{C} \models \phi \vee \psi$  if  $G, \ell, \mathcal{C} \models \phi$  or  $G, \ell, \mathcal{C} \models \psi$ .
- $G, \ell, \mathcal{C} \models \neg\phi$  if it is not the case that  $G, \ell, \mathcal{C} \models \phi$ .
- $G, \ell, \mathcal{C} \models \exists x_i.\phi$  if there exists  $v \in V(G)$  such that  $G, \ell', \mathcal{C} \models \phi[x_i \setminus u_i]$ , where  $\ell(u_i) \uparrow$ ,  $\phi[x_i \setminus u_i]$  is the formula obtained from  $\phi$  if we replace every occurrence of  $x_i$  with the (new) constant  $u_i$  and  $\ell' : U \rightarrow V(G)$  is a partial function for which  $\ell'(u_i) = v$ , and  $\ell', \ell$  agree on all other values  $u_j \neq u_i$ .
- $G, \ell, \mathcal{C} \models \exists X_i.\phi$  if there exists  $S \subseteq V(G)$  such that  $G, \ell, \mathcal{C}' \models \phi[X_i \setminus D_i]$ , where  $\mathcal{C}(D_i) \uparrow$ ,  $\phi[X_i \setminus D_i]$  is the formula obtained from  $\phi$  if we replace every occurrence of  $X_i$  with the (new) constant  $D_i$  and  $\mathcal{C}' : D \rightarrow 2^{V(G)}$  is a partial function for which  $\mathcal{C}'(D_i) = S$  and  $\mathcal{C}', \mathcal{C}$  agree on all other values  $D_j \neq D_i$ .

If none of the above applies then  $G, \ell, \mathcal{C}$  does not model  $\phi$  and we write  $G, \ell, \mathcal{C} \not\models \phi$ . Observe that, from the syntactic rules presented above, a formula can have free (non-quantified) variables. However, we will only define model-checking for formulas without free variables (also called sentences). Slightly abusing notation, we will write  $G \models \phi$  to mean  $G, \ell, \mathcal{C} \models \phi$  for the nowhere defined functions  $\ell, \mathcal{C}$ . Note that our definition does not contain conjunctions or universal quantifiers, but these can be obtained from disjunctions and existential quantifiers using negations in the usual way, so we will use them freely when constructing formulas.

An FO formula is defined as an MSO formula that uses no set variables  $X_i$ . In the remainder, we will assume that all formulas are given to us in prenex form, that is, all quantifiers appear in the beginning of the formula. We call the problem of deciding whether  $G, \ell, \mathcal{C} \models \phi$  the model-checking problem.

We recall the following basic fact:

► **Lemma 9.** *Let  $G_1, \ell_1, \mathcal{C}_1$  and  $G_2, \ell_2, \mathcal{C}_2$  be two isomorphic triplets. Then, for all MSO formulas  $\phi$  we have  $G_1, \ell_1, \mathcal{C}_1 \models \phi$  if and only if  $G_2, \ell_2, \mathcal{C}_2 \models \phi$ .*

**Proof.**  $G_1, \ell_1, \mathcal{C}_1$  and  $G_2, \ell_2, \mathcal{C}_2$  are isomorphic. Thus there exists a bijective function  $f : V(G_1) \rightarrow V(G_2)$  such i)  $f$  preserves in  $G_2$  the (non-)edges between the pairs of images of vertices in  $G_1$  and ii)  $V(G_1)$  and  $V(G_2)$  have the same labelings and colorings for  $f$ .

We proceed by induction on the structure of  $\phi$ .

- For  $\phi := u_i \in D_j$ .  $G_1, \ell_1, \mathcal{C}_1 \models \phi$  iff  $\ell_1(u_i) \in \mathcal{C}_1(D_j)$  iff  $f(\ell_1(u_i)) \in f(\mathcal{C}_1(D_j))$  iff  $\ell_2(u_i) \in \mathcal{C}_2(D_j)$  iff  $G_2, \ell_2, \mathcal{C}_2 \models \phi$
- For  $\phi := u_i = u_j$ .  $G_1, \ell_1, \mathcal{C}_1 \models \phi$  iff  $\ell_1(u_i) = \ell_1(u_j)$  iff  $f(\ell_1(u_i)) = f(\ell_1(u_j))$  iff  $\ell_2(u_i) = \ell_2(u_j)$  iff  $G_2, \ell_2, \mathcal{C}_2 \models \phi$
- For  $\phi := u_i \sim u_j$ .  $G_1, \ell_1, \mathcal{C}_1 \models \phi$  iff  $(\ell_1(u_i), \ell_1(u_j)) \in E(G_1)$  iff  $(f(\ell_1(u_i)), f(\ell_1(u_j))) \in E(G_2)$  iff  $(\ell_2(u_i), \ell_2(u_j)) \in E(G_2)$  iff  $G_2, \ell_2, \mathcal{C}_2 \models \phi$
- For  $\phi := \phi' \vee \phi''$ , or  $\phi := \neg \phi'$  By the inductive hypothesis,  $G_1, \ell_1, \mathcal{C}_1 \models \phi'$  iff  $G_2, \ell_2, \mathcal{C}_2 \models \phi'$  and  $G_1, \ell_1, \mathcal{C}_1 \models \phi''$  iff  $G_2, \ell_2, \mathcal{C}_2 \models \phi''$ . Thus the statement also holds for  $\phi$ .
- For  $\phi := \exists x_i. \phi'$ . We prove the one direction, the other is identical if we use  $f^{-1}$  instead of  $f$  in our arguments.

$G_1, \ell_1, \mathcal{C}_1 \models \exists x_i. \phi'$  if there exists  $v \in V(G_1)$  such that  $G_1, \ell'_1, \mathcal{C}_1 \models \phi[x_i \setminus u_i]$ , where  $\ell_1(u_i) \uparrow$ ,  $\ell'_1(u_i) = v$ , and  $\ell'_1, \ell_1$  agree on all other values  $u_j \neq u_i$ . We define a partial labeling function  $\ell'_2 : U \rightarrow V(G_2)$ , such that  $\ell'_2(u_i) = f(\ell'_1(u_i)) = f(v)$  and  $\ell'_2, \ell_2$  agree on all other values. It is easy to see that  $G_1, \ell'_1, \mathcal{C}_1$  and  $G_2, \ell'_2, \mathcal{C}_2$  are isomorphic, thus by the inductive hypothesis  $G_2, \ell'_2, \mathcal{C}_2 \models \phi[x_i \setminus u_i]$ . Since  $\exists f(v) \in V(G_2)$  such that  $G_2, \ell'_2, \mathcal{C}_2 \models \phi[x_i \setminus u_i]$  and  $\ell_2(u_i) \uparrow$  (since  $\ell_1(u_i) \uparrow$  and  $V(G_1)$  and  $V(G_2)$  have the same labelings for  $f$ ), therefore  $G_2, \ell_2, \mathcal{C}_2 \models \exists x_i. \phi'$ .

- For  $\phi := \exists X_i. \phi'$ . The proof is similar with the above case. Once again we will only show the one direction.

$G_1, \ell_1, \mathcal{C}_1 \models \exists X_i. \phi'$  if there exists  $S \subseteq V(G_1)$  such that  $G_1, \ell_1, \mathcal{C}'_1 \models \phi[X_i \setminus D_i]$ , where  $\mathcal{C}_1(D_i) \uparrow$ ,  $\mathcal{C}'_1(D_i) = S$  and  $\mathcal{C}'_1, \mathcal{C}_1$  agree on all other values  $D_j \neq D_i$ .

We define a partial coloring function  $\mathcal{C}'_2 : D \rightarrow 2^{V(G_2)}$  such that  $\mathcal{C}'_2(D_i) = f(\mathcal{C}'_1(D_i)) = f(S)$  and  $\mathcal{C}'_2, \mathcal{C}_2$  agree on all other values. Once again,  $G_1, \ell_1, \mathcal{C}'_1$  and  $G_2, \ell_2, \mathcal{C}'_2$  are isomorphic, thus by the inductive hypothesis  $G_2, \ell_2, \mathcal{C}'_2 \models \phi[X_i \setminus D_i]$ . Since  $\exists f(S) \subseteq V(G_2)$  such that  $G_2, \ell_2, \mathcal{C}'_2 \models \phi[X_i \setminus D_i]$  and we have that  $\mathcal{C}_2(D_i) \uparrow$ , therefore  $G_2, \ell_2, \mathcal{C}_2 \models \exists X_i. \phi'$ . ◀

### 3 FPT algorithms for FO and MSO Model-Checking parameterized by vertex integrity

In this section we prove Theorems 10 and 11. The statements appear right below.

► **Theorem 10.** *Given a graph  $G$  with  $\iota(G) \leq k$  and an FO formula  $\phi$  in prenex form having at most  $q$  quantifiers. Then deciding if  $G \models \phi$  can be solved in time  $(2^{O(k^2)} \cdot q)^q + \text{poly}(|G|)$ .*

► **Theorem 11.** *Given a graph  $G$  with  $\iota(G) \leq k$  and an MSO formula  $\phi$  in prenex form having at most  $q_1$  vertex variable quantifiers and at most  $q_2$  vertex set variable quantifiers. Then deciding if  $G \models \phi$  can be solved in time  $(2^{2^{O(k^2+kq_2)}} \cdot q_1)^{q_1} + \text{poly}(|G|)$ .*

The proofs are heavily based on Lemmata 12 and 14. The first, which is about FO Model-Checking, says that if we have at least  $q+1$  components of the same type then we can erase one such component from the graph. The reason essentially is that, if  $G, \ell, \mathcal{C}$  models  $\phi$

by labeling a vertex  $v$  that belongs to the component to be removed, we can replace that vertex by a corresponding vertex in another component having the same type. Notice that the formula has  $q$  quantifiers and thus the graph will have  $q$  labels after the assignment. Since we have  $q + 1$  components of the same type, for one of these components the vertex that corresponds to  $v$  will be unlabeled.

The second, which is about MSO Model-Checking, says that since we can quantify over sets of vertices, unlike the case for FO, each set quantification can potentially affect a large number of components that originally had the same type (by coloring its intersection with each of them). However, since each component has size at most  $k$ , we have  $2^k$  ways that the quantified set can overlap with the components. Thus, if we originally had a sufficiently large number of same type components, even after the coloring, we will still have a sufficient number of components that are of the same type, such that even if we remove one such component the answer of the problem won't change.

Lemmata 12 and 14, together with the fact that there exist a bounded number of types of components, give the kernels (Lemma 13 for FO and Lemma 15 for MSO).

► **Lemma 12.** *Given a triplet  $G, \ell, \mathcal{C}$  having  $q + 1$  vertex sets  $C_1, C_2, \dots, C_{q+1}$  of the same type and  $\phi$  an FO formula in prenex form having  $q$  quantifiers. Then  $G, \ell, \mathcal{C} \models \phi$  if and only if  $G \setminus C_1, \ell, \mathcal{C}' \models \phi$ , where  $\mathcal{C}'$  is the restriction of  $\mathcal{C}$  to  $V(G) \setminus C_1$ .*

**Proof.** We proceed by induction on the structure of the formula  $\phi$ .

1. For  $\phi := u_i \in D_j$ ,  $\phi := u_1 = u_2$ , or  $\phi := u_1 \sim u_2$ . From Observation 7 the sets are unlabeled. Thus  $\nexists v \in C_1$  for which  $\ell(u_1) = v$  or  $\ell(u_2) = v$ . Thus the statement of the lemma holds for the base case.
2. For  $\phi := \phi_1 \vee \phi_2$  or  $\phi := \neg \phi_1$ . From the inductive hypothesis, we have that  $G, \ell, \mathcal{C} \models \phi_1$  if and only if  $G \setminus C_1, \ell, \mathcal{C}' \models \phi_1$  and that  $G, \ell, \mathcal{C} \models \phi_2$  if and only if  $G \setminus C_1, \ell, \mathcal{C}' \models \phi_2$ . It is easy to see that the statement of the lemma holds also for  $\phi$ .
3. The most interesting case is for  $\phi := \exists x_i. \phi'$ . If  $G, \ell, \mathcal{C} \models \phi$  then from the definition of the semantics of  $\phi$  there exists  $v \in V(G)$  such that  $G, \ell', \mathcal{C} \models \phi[x_i \setminus u_i]$  with  $\ell(u_i) \uparrow$  and  $\ell' : U \rightarrow V(G)$  being a partial function for which  $\ell'(u_i) = v$ , and  $\ell'$  agrees with  $\ell$  on all other values  $u_j \neq u_i$ .

First we prove that without loss of generality  $v \notin C_1$ . Suppose that  $v \in C_1$ . Since  $C_1$  and  $C_2$  have the same type on  $G, \ell, \mathcal{C}$ , by Definition 6 there exists an isomorphism  $f : C_1 \rightarrow C_2$ . Consider now a labeling function  $\ell'' : U \rightarrow V(G)$  where  $\ell''(u_i) = f(\ell'(u_i)) = f(v)$ , otherwise  $\ell', \ell''$  agree on  $u_j \neq u_i$ . Observe that  $G, \ell', \mathcal{C}$  and  $G, \ell'', \mathcal{C}$  are isomorphic, thus from Lemma 9 we have that  $G, \ell', \mathcal{C} \models \phi$  iff  $G, \ell'', \mathcal{C} \models \phi$ . In that case, instead of  $v \in C_1$  we shall consider  $f(v) \in C_2$ . Thus, from now on we can assume that  $v \notin C_1$ .

For the triplet  $G, \ell', \mathcal{C}$   $q$  of the sets  $C_1, C_2, \dots, C_{q+1}$  are still unlabeled and have the same type ( $C_1$  is among them). Also  $\phi'$  has  $q - 1$  quantifiers. Thus, by the inductive step,  $G, \ell', \mathcal{C} \models \phi'$  if and only if  $G \setminus C_1, \ell', \mathcal{C}' \models \phi'$ . Since  $v \in V(G) \setminus C_1$ , we have that  $G \setminus C_1, \ell, \mathcal{C}' \models \phi$ .

For the other direction, observe that  $v \in V(G) \setminus C_1$  implies that  $v \in V(G)$ . Thus the statement holds with similar reasoning as above. ◀

► **Lemma 13.** *For a triplet  $G, \ell, \mathcal{C}$  with vertex integrity  $\iota(G) \leq k$  and with  $\ell, \mathcal{C}$  everywhere undefined and for a formula  $\phi$  with  $q$  quantifiers, FO MODEL CHECKING has a kernel of size  $O(2^{k^2} \cdot q \cdot k)$ , assuming we are given in the input  $S \subseteq V(G)$  such that the largest component of  $G \setminus S$  has size at most  $k - |S|$ .*



**Proof.** We give a polynomial-time algorithm to calculate an upper bound on the number of components of  $G \setminus S$  having the same type. Observe that types are only specified by the neighborhoods of the vertices of the components ( $\ell$  and  $\mathcal{C}$  are everywhere undefined thus there are no labels or colors on  $G$ ).

First, we arbitrarily number the vertices of  $S$  and of each component. In order to classify the components into types, we map each component  $C_i$  to a vector  $[N_1, N_2, \dots, N_{|C_i|}]$ , where  $N_j$  is an ordered set containing the (numbered) neighbors of the  $j^{\text{th}}$  vertex of  $C_i$  (starting from the neighbors in  $S$ ). Clearly, two components having the same vectors also have the same type, using the isomorphism that maps the  $i$ -th vertex of one to the  $i$ -th vertex of the other.

Since each component has at most  $k$  vertices and each vertex has at most  $2^k$  different types of neighborhoods  $N_j$ , we can have at most  $2^{k^2}$  vectors, thus at most  $2^{k^2}$  types of components. Furthermore, since we are given  $S$ , we can test in polynomial time if two components have the same type under the arbitrary numbering we used. From Lemma 12, if more than  $q$  components have the same type we can remove one such component without changing the answer of the problem, thus we can in polynomial time either reduce the graph or conclude that each component type appears at most  $q$  times. In the end we will have at most  $2^{k^2} \cdot q$  components, each having at most  $k$  vertices, thus the result.  $\blacktriangleleft$

By applying the straightforward algorithm which runs in time  $|V(G)|^q \cdot \text{poly}(|G|)$  for FO MODEL CHECKING, together with Lemma 13 we get the complexity promised by Theorem 10.

In order to prove Theorem 11 we need a stronger version of Lemma 12.

► **Lemma 14.** *Given a triplet  $G, \ell, \mathcal{C}$  with at least  $q' = 2^{k \cdot q_2} \cdot q_1 + 1$  vertex sets  $C_1, C_2, \dots, C_{q'}$  having the same type and sizes at most  $k$  and an MSO formula  $\phi$  in prenex form with  $q_1$  FO quantifiers and  $q_2$  MSO quantifiers. Then  $G, \ell, \mathcal{C} \models \phi$  if and only if  $G \setminus C_1, \ell, \mathcal{C}_1 \models \phi$ , where  $\mathcal{C}_1$  is the restriction of  $\mathcal{C}$  to  $V(G) \setminus C_1$ .*

**Proof.** We proceed by induction on the structure of  $\phi$ . We can reuse the arguments of Lemma 12, except for the case where  $\phi := \exists X_i. \phi'$ , so we focus on this case.

For the one direction, if  $G, \ell, \mathcal{C} \models \phi$ , from the definition of the semantics of  $\phi$ , then there exists  $S \subseteq V(G)$  such that  $G, \ell, \mathcal{C}' \models \phi[X_i \setminus D_i]$  with  $\mathcal{C}(D_i) \uparrow$  and  $\mathcal{C}' : D \rightarrow 2^{V(G)}$  being a partial function for which  $\mathcal{C}'(D_i) = S$ , and  $\mathcal{C}'$  agrees with  $\mathcal{C}$  on all other values  $D_j \neq D_i$ .

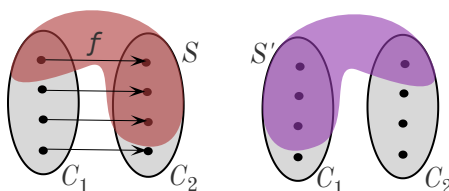
Since each of the vertex sets  $C_1, C_2, \dots, C_{q'}$  has size at most  $k$ , there are at most  $2^k$  possible ways for  $S$  to intersect with each of them. Therefore, by pigeonhole principle, one such intersection appears in at least  $\lceil \frac{q'}{2^k} \rceil = 2^{k(q_2-1)} \cdot q_1 + 1$  sets, call that group  $M$ . In order to be able to apply the inductive hypothesis, we need to prove that, without loss of generality,  $C_1 \in M$ .

Suppose that  $C_1 \notin M$ . We will do a “swapping” of  $C_1$  with a vertex set (say  $C_2$  without loss of generality) that does belong in the group  $M$ . Since  $C_1$  and  $C_2$  have the same type, that means that there exists an isomorphism  $f : C_1 \rightarrow C_2$ .

We consider a new coloring function  $\mathcal{C}''$  that agrees with  $\mathcal{C}'$  everywhere but on the constant  $D_i$ . This new coloring function will map  $D_i$  to the set of vertices  $S'$  (instead of  $S$ ), where we have replaced every  $v \in S \cap C_1$  with  $f(v)$  and every  $v \in S \cap C_2$  with  $f^{-1}(v)$  (see Figure 1). More formally,  $\mathcal{C}''(D_i) = S'$  where  $S' = (S \setminus (C_1 \cup C_2)) \cup f(C_1 \cap S) \cup f^{-1}(C_2 \cap S)$ . Then the triplets  $G, \ell, \mathcal{C}'$  and  $G, \ell, \mathcal{C}''$  are isomorphic and from Lemma 9 we have that  $G, \ell, \mathcal{C}' \models \phi$  iff  $G, \ell, \mathcal{C}'' \models \phi$ . From now on we assume that  $C_1$  belongs in  $M$ .

For the triplet  $G, \ell, \mathcal{C}'$ , the sets in  $M$  have all the same type and  $|M| \geq 2^{k(q_2-1)} \cdot q_1 + 1$ . Furthermore, the function  $\phi'$  has  $q_1$  FO and  $q_2 - 1$  MSO quantifiers. Therefore, by the inductive hypothesis we can remove a set from  $M$  and the answer of the problem won't change, in other words we have that  $G, \ell, \mathcal{C}' \models \phi'$  iff  $G \setminus C_1, \ell, \mathcal{C}'_1 \models \phi'$ , where  $\mathcal{C}'_1$  is the restriction of  $\mathcal{C}'$  on  $V(G) \setminus C_1$ . From the semantics of  $\phi$  we have that  $G \setminus C_1, \ell, \mathcal{C}_1 \models \phi$ .





■ **Figure 1** The way the vertex set  $S'$  intersects the vertex sets  $C_1$  and  $C_2$ .

For the other direction, if  $G \setminus C_1, \ell, \mathcal{C}_1 \models \phi$  then there exists  $S_1 \subseteq V(G) \setminus C_1$  such that  $G \setminus C_1, \ell, \mathcal{C}'_1 \models \phi[X_i \setminus D_i]$  with  $\mathcal{C}_1(D_i) \uparrow$  and  $\mathcal{C}_1$  being a partial coloring function for which  $\mathcal{C}'_1(D_i) = S_1$ , and  $\mathcal{C}'_1$  agrees with  $\mathcal{C}_1$  on all other values  $D_j \neq D_i$ .

As previously,  $S_1$  partitions  $C_2, \dots, C_{q'}$  into  $2^k$  equivalence classes, depending on the intersection of each set with  $S_1$ , such that sets placed in the same class (i.e. having isomorphic intersection with  $S_1$ ) have the same type in  $G \setminus C_1, \ell, \mathcal{C}'_1$ . Hence, one of these classes has size at least  $\frac{q'-1}{2^k} = 2^{k(q_2-1)} \cdot q_1$ , call this class  $M'$ . We construct a triplet  $G, \ell, \mathcal{C}^*$  as follows: let  $C_j \in M'$  and  $f'$  be the isomorphism from  $C_j$  to  $C_1$ ; We set that  $\mathcal{C}^*$  agrees with  $\mathcal{C}$  on all sets except  $D_i$ ; and for  $D_i$  we have  $\mathcal{C}^*(D_i) = \mathcal{C}'_1(D_i) \cup f'(S_1 \cap C_j)$ . In other words, we define  $\mathcal{C}^*$  in such a way that the set  $C_1$  has the same type as all sets of the class  $M'$ . But then we have  $|M' \cup \{C_1\}| \geq 2^{k(q_2-1)} \cdot q_1 + 1$  sets of the same type and by inductive hypothesis we have  $G, \ell, \mathcal{C}^* \models \phi[X_i \setminus D_i]$ . Therefore, by the semantics of MSO we have  $G, \ell, \mathcal{C} \models \phi$ . ◀

▶ **Lemma 15.** *For a triplet  $G, \ell, \mathcal{C}$  with vertex integrity  $\iota(G) \leq k$  and with  $\ell, \mathcal{C}$  everywhere undefined and for a formula  $\phi$  with  $q_1$  FO quantifiers and  $q_2$  MSO quantifiers, MSO MODEL CHECKING has a kernel of size  $O(2^{k^2+kq_2} \cdot q_1 \cdot k)$ , assuming we are given in the input  $S \subseteq V(G)$  such that the largest component of  $G \setminus S$  has size at most  $k - |S|$ .*

**Proof.** The proof is the same as for Lemma 13. The only thing that changes is the number of same-type components required to have before removing one such component ( $q'$  required by Lemma 14 versus  $q + 1$  required by Lemma 12). ◀

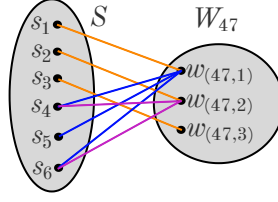
Applying the straightforward algorithm for MSO Model-Checking that runs in  $2^{q_2 \cdot V(G)} \cdot V(G)^{q_1} \cdot \text{poly}|G|$  and Lemma 15 gives the complexity promised by Theorem 11.

## 4 Lower Bounds

In this section we show that the dependence of our meta-theorems on vertex integrity cannot be significantly improved, unless the ETH is false. Our strategy will be to present a unified construction which, starting from an arbitrary graph  $G$  with  $n$  vertices, produces a new graph  $H(G)$ , with small vertex integrity, such that we can deduce if two vertices of  $G$  are connected using appropriate constant-sized FO formulas of  $H$ . This will, in principle, allow us to express an FO or MSO-expressible property of  $G$  as a corresponding property of  $H(G)$ , and hence, if the original property is hard, to obtain a lower bound on model-checking on  $H$ . Let us describe this construction in more details.

**Construction.** We are given a graph  $G$  on  $n$  vertices, say  $V(G) = \{v_1, \dots, v_n\}$ , and  $m$  edges. Let  $k = \lceil \sqrt{\log n} \rceil$ . We construct a graph  $H$  as follows:

1. We begin constructing  $V(H)$  by forming  $n + m + 1$  sets of vertices, called  $S, W_1, \dots, W_n$ , and  $Y_1, \dots, Y_m$ . We have  $|S| = 2k, |W_i| = k$  for all  $i \in [n]$ , and  $|Y_j| = 2k + 1$  for all  $j \in [m]$ . The vertices of  $S$  are numbered arbitrarily as  $s_1, s_2, \dots, s_{2k}$ .



■ **Figure 2** The connection between  $S$  and the set  $W_{47}$ . For this example  $k = 3$ , we can represent up to  $2^9$  numbers in binary. In order to represent  $47_{10} = 000101111_2$ , we shall connect  $w_{(47,1)}$  with  $s_4, s_5$  and  $s_6$  in order to represent the three least significant bits (which are all 1), and  $w_{(47,2)}$  with  $s_4$  and  $s_6$  to represent the next triad of bits. The three most significant bits are all 0.

2. Internally,  $S$  induces an independent set, each  $W_i$ , for  $i \in [n]$  induces a clique, and each  $Y_j$ , for  $j \in [m]$  induces a graph made up of two disjoint cliques of size  $k$ , denoted  $Y_j^1, Y_j^2$ , and a vertex connected to all  $2k$  vertices of the cliques  $Y_j^1, Y_j^2$ .
3. For each  $i \in [n]$ , we attach a leaf to each vertex of  $W_i$ . For each  $j \in [m]$ , we attach two leaves to each vertex of  $Y_j^1$ , three leaves to each vertex of  $Y_j^2$ , and four leaves to the remaining vertex of  $Y_j$ .
4. For each  $i \in [n]$ , number the vertices of  $W_i$  arbitrarily as  $w_{(i,1)}, w_{(i,2)}, \dots, w_{(i,k)}$ . For each  $\beta \in [k]$  we connect  $w_{(i,\beta)}$  to  $s_\beta$ . Furthermore, let  $b_1 b_2 \dots b_{k^2}$  be the binary representation of  $i - 1$  with the least significant digit first, that is, a sequence of bits such that  $\sum_\beta b_\beta 2^{\beta-1} = i - 1$ . Note that  $k^2 \geq \log n$ , therefore  $k^2$  bits are sufficient to represent all numbers from 0 to  $n - 1$ . We partition this binary representation into  $k$  blocks of  $k$  bits. For  $\beta \in [k]$  we consider the bits  $b_{(\beta-1)k+1} \dots b_{\beta k}$  and we use these bits to determine the connections between  $w_{(i,\beta)}$  and the vertices  $s_{k+1}, \dots, s_{2k}$ . More precisely, for  $\beta, \gamma \in [k]$ , we set that  $w_{(i,\beta)}$  is connected to  $s_{k+\gamma}$  if and only if  $b_{(\beta-1)k+\gamma}$  is equal to 1.
5. For each  $j \in [m]$  we do the following. Suppose the  $j$ -th edge of  $G$  has endpoints  $v_{i_1}, v_{i_2}$ . We number the vertices of  $Y_j^1$  as  $y_{(j,1)}^1, \dots, y_{(j,k)}^1$ , and the vertices of  $Y_j^2$  as  $y_{(j,1)}^2, \dots, y_{(j,k)}^2$  in some arbitrary way. Now for all  $\beta \in [k]$  we set that  $y_{(j,\beta)}^1$  has the same neighbors in  $S$  as  $w_{(i_1,\beta)}$  and  $y_{(j,\beta)}^2$  has the same neighbors in  $S$  as  $w_{(i_2,\beta)}$ .

The construction of our graph is now complete. The intuition behind this construction is that each clique  $W_i$  represents a vertex  $v_i \in V(G)$ . In order to distinguish the vertices, we use the  $k^2 \geq \log n$  possible edges between vertices in  $W_i$  and the second part of  $S$ , that is  $\{s_{k+1}, \dots, s_{2k}\}$ . These edges should represent the binary representation of  $i$ . See Figure 2 for an example.

Vertices of  $H$  may be (arbitrarily) labeled for the purpose of the construction but for the purpose of Model-Checking the graph  $H$  is unlabeled. In order to give a numbering to the vertices of  $W_i$ , we use the matching between  $W_i$  and the first  $k$  vertices of the set  $S$  (the first vertex of  $W_i$  connects to the first vertex of  $S$ , etc).

The sets  $Y_j$  represent edges in  $G$ . If the  $j^{\text{th}}$  edge in  $E(G)$  is the edge  $(v_{i_1}, v_{i_2})$ , then  $Y_j^1$  should have the same connections with  $S$  as the set  $W_{i_1}$  (similarly  $Y_j^2, W_{i_2}$ ). In order to check in  $H$  whether  $(v_{i_1}, v_{i_2})$  is an edge, we shall check if there exists a set  $Y_j$  such that each vertex of  $Y_j^1$  has the same neighborhood in  $S$  as a vertex of  $W_{i_1}$  and each vertex of  $Y_j^2$  has the same neighborhood in  $S$  as a vertex of  $W_{i_2}$ .

It is crucial here that the construction is such that  $W_i, W_{i'}$  are distinguishable for  $i \neq i'$  in terms of their neighborhoods in  $S$ , that is, there always exists  $w \in W_i$  for which no  $w' \in W_{i'}$  has  $N(w) \cap S = N(w') \cap S$ . We will show that it is not hard to express this property in FO logic. Furthermore, the leaves we have attached to various vertices will allow us to distinguish in FO logic whether a vertex belongs in a set  $W_i, Y_j^1$ , or  $Y_j^2$ .

We now establish some basic properties about  $H$  and what can be expressed about its vertices in FO logic:

- **Lemma 16.** *The graph  $H$  satisfies the following properties, for any coloring function  $\mathcal{C}$ .*
1. *We have  $\iota(H) = O(\sqrt{\log n})$  and  $|V(H)| = O(n^2\sqrt{\log n})$ .*
  2. *For each  $i, i' \in [n]$  with  $i \neq i'$ , there exists a vertex  $w \in W_i$  such that for all  $w' \in W_{i'}$  we have  $N(w) \cap S \neq N(w') \cap S$ .*
  3. *There exist constant-sized FO formulas  $\phi_W(x_1), \phi_{Y_1}(x_1), \phi_{Y_2}(x_1), \phi_S(x_1)$  using one free variable  $x_1$ , such that  $H, \ell, \mathcal{C} \models \phi_W[x_1 \setminus u_1]$  (respectively  $H, \ell, \mathcal{C} \models \phi_{Y_1}[x_1 \setminus u_1]$ ,  $H, \ell, \mathcal{C} \models \phi_{Y_2}[x_1 \setminus u_1]$ ,  $H, \ell, \mathcal{C} \models \phi_S[x_1 \setminus u_1]$ ) if and only if  $\ell(u_1) \in W_i$  for some  $i \in [n]$  (respectively  $\ell(u_1) \in Y_j^1$ ,  $\ell(u_1) \in Y_j^2$ , for some  $j \in [m]$ ,  $\ell(u_1) \in S$ ).*
  4. *There exists a constant-sized FO formula  $\phi_{WY}$  using only two free variables  $x_1, x_2$  such that  $H, \ell, \mathcal{C} \models \phi_{WY}[x_1 \setminus u_1][x_2 \setminus u_2]$  if and only if  $\ell(u_1) \in W_i$  for some  $i \in [n]$ ,  $\ell(u_2) \in Y_j^\alpha$  for some  $j \in [m], \alpha \in \{1, 2\}$ , and for all  $\beta \in [k]$  we have  $N(w_{(i,\beta)}) \cap S = N(y_{(j,\beta)}^\alpha) \cap S$ .*
  5. *There exists a constant-sized FO formula  $\phi_{adj}$  using only two free variables  $x_1, x_2$  such that  $H, \ell, \mathcal{C} \models \phi_{adj}[x_1 \setminus u_1][x_2 \setminus u_2]$  if and only if  $\ell(u_1) \in W_i$  and  $\ell(u_2) \in W_{i'}$  for some  $i, i' \in [n]$  such that  $(v_i, v_{i'}) \in E(G)$ .*

**Proof.** For the first property, we observe that the largest component of  $H \setminus S$  has size at most  $10\sqrt{\log n} + 2$ , while  $|S| \leq 2\sqrt{\log n} + 2$ . Furthermore, we have at most  $m + n = O(n^2)$  components after removing  $S$ .

For the second property, since  $i \neq i'$ , their binary representations differ in some bit. Let  $\beta, \gamma \in [k]$  be such that if  $b_1 \dots b_{k-2}$  is the binary representation of  $i - 1$  and  $b'_1 \dots b'_{k-2}$  is the binary representation of  $i' - 1$ , we have  $b_{(\beta-1)k+\gamma} \neq b'_{(\beta-1)k+\gamma}$ . But then, exactly one of  $w_{(i,\beta)}, w_{(i',\beta)}$  is connected to  $s_{k+\gamma}$ . Furthermore,  $w_{(i,\beta)}$  is connected to  $s_\beta$ , but the only neighbor of  $s_\beta$  in  $W_{i'}$  is  $w_{(i',\beta)}$ . Hence,  $w_{(i,\beta)}$  is the claimed vertex.

For the third property, observe that, in  $H$ , vertices of  $S$  have no leaves attached, vertices of each  $X_i$  have one leaf attached, vertices of  $Y_j^1$  have two leaves attached, vertices of  $Y_j^2$  have three leaves attached, and the remaining vertices have four leaves attached. Hence, it suffices to be able to express in FO, with a constant-sized formula, the property “ $x_1$  has exactly  $c$  leaves attached”, where  $c \in \{0, 1, 2, 3\}$ . This is not hard to do. For example, the formula  $\phi_2(x_1) := \exists x_2 \exists x_3 \forall x_4 ((x_2 \sim x_1) \wedge (x_3 \sim x_1) \wedge (x_2 \neq x_3) \wedge ((x_4 = x_1) \vee (\neg(x_4 \sim x_2) \wedge \neg(x_4 \sim x_3))))$  expresses the property that  $x_1$  has at least two leaves attached to it. Using the same ideas we can construct  $\phi_c(x_1)$ , for  $c \in \{1, 2, 3, 4\}$  and then  $\phi_S(x_1) := \neg\phi_1(x_1)$ ,  $\phi_W(x_1) := \phi_1(x_1) \wedge \neg\phi_2(x_1)$ ,  $\phi_{Y_1} := \phi_2(x_1) \wedge \neg\phi_3(x_1)$ ,  $\phi_{Y_2}(x_1) := \phi_3(x_1) \wedge \neg\phi_4(x_1)$ .

For the fourth property, we set  $\phi_{WY}(x_1, x_2) := \phi_{WY_1}(x_1, x_2) \vee \phi_{WY_2}(x_1, x_2)$ , where we define two formulas  $\phi_{WY_\alpha}$  depending on whether  $\alpha = 1$  or  $\alpha = 2$ . We have

$$\begin{aligned} \phi_{WY_\alpha}(x_1, x_2) &:= \phi_W(x_1) \wedge \phi_{Y_\alpha}(x_2) \wedge \forall x_3 ((\neg\phi_W(x_3)) \vee (\neg(x_3 \sim x_1) \wedge \neg(x_3 = x_1))) \vee \\ &\quad \exists x_4 (\phi_{Y_1}(x_4) \wedge (x_4 \sim x_2 \vee x_4 = x_2) \wedge \forall x_5 (\phi_S(x_5) \rightarrow (x_5 \sim x_3 \leftrightarrow x_5 \sim x_4))) \end{aligned}$$

What we are saying here is that  $\phi_{WY_1}[x_1 \setminus u_1][x_2 \setminus u_2]$  is satisfied if  $\ell(u_1) \in W_i, \ell(u_2) \in Y_j^1$ , for some  $i \in [n], j \in [m]$ , and for every  $x_3 \in W_i$  there exists  $x_4 \in Y_j^1$  such that  $N(x_3) \cap S = N(x_4) \cap S$ . Therefore, if this property holds, then  $W_i$  and  $Y_j^1$  represent the same vertex of  $V$  (similarly for  $\phi_{WY_2}$ ).

For the last property, we set

$$\begin{aligned} \phi_{adj}(x_1, x_2) &:= \phi_W(x_1) \wedge \phi_W(x_2) \wedge \exists x_3 \exists x_4 ((\phi_{Y_1}(x_3) \wedge \phi_{Y_2}(x_4)) \vee (\phi_{Y_1}(x_4) \wedge \phi_{Y_2}(x_3))) \wedge \\ &\quad \phi_{WY}(x_1, x_3) \wedge \phi_{WY}(x_2, x_4) \wedge \exists x_5 (\neg\phi_S(x_5) \wedge x_3 \sim x_5 \wedge x_4 \sim x_5) \end{aligned}$$

## 34:12 Fine-Grained Meta-Theorems for Vertex Integrity

In other words,  $H, \ell, \mathcal{C} \models \phi_{adj}[x_1 \setminus u_1][x_2 \setminus u_2]$  if (i)  $\ell(u_1) \in W_i$  and  $\ell(u_2) \in W_{i'}$ , for some  $i, i' \in [n]$  (ii) there exist  $x_3, x_4$  such that  $x_3 \in Y_j^1$  and  $x_4 \in Y_j^2$  for the same  $j$ ; this is verified because  $x_3, x_4$  have a common neighbor  $x_5$  that does not belong in  $S$  (iii)  $W_i, W_{i'}$  correspond to the same pair of vertices as the set  $Y_j = Y_j^1 \cup Y_j^2$ , which means that  $(v_i, v_{i'}) \in E(G)$ .  $\blacktriangleleft$

We are now ready to prove our lower bounds.

► **Theorem 17.** *If there exists an algorithm which, given a graph  $G$  with  $n$  vertices and  $\iota(G) = k$  and an FO formula  $\phi$  with  $q$  quantifiers, decides whether  $G \models \phi$  in time  $2^{o(k^2q)}n^{O(1)}$ , then the ETH is false.*

**Proof.** We perform a reduction from  $q$ -CLIQUE. It is well-known that, given a graph  $G$  on  $n$  vertices it is not possible to decide if  $G$  contains a clique of size  $q$  in time  $n^{o(q)}$ , unless the ETH is false [8]. We claim that we will construct the graph  $H(G)$ , as previously described, and an FO formula  $\phi_C$  such that  $\phi_C$  will contain  $O(q)$  quantifiers and  $H, \ell, \mathcal{C} \models \phi_C$  for the nowhere defined functions  $\ell, \mathcal{C}$  if and only if  $G$  has a  $q$ -clique. If we achieve this, then, since by Lemma 16 we have  $k = O(\sqrt{\log n})$ , and the size of  $H$  is polynomially related to the size of  $G$ , the stated running time would become  $2^{o(q(\sqrt{\log n})^2)}n^{O(1)} = n^{o(q)}$  and we refute the ETH. Our goal is then to define such an FO formula  $\phi_C$ . We define

$$\begin{aligned} \phi_C := & \exists x_1 \exists x_2 \dots \exists x_q \bigwedge_{i \in [q]} \phi_W(x_i) \wedge \bigwedge_{i, i' \in [q], i \neq i'} (x_i \neq x_{i'}) \\ & \forall x_{q+1} \forall x_{q+2} \bigwedge_{i \in [q]} (\neg(x_{q+1} = x_i)) \vee \bigwedge_{i \in [q]} (\neg(x_{q+2} = x_i)) \vee (x_{q+1} = x_{q+2}) \vee \\ & \phi_{adj}(x_{q+1}, x_{q+2}) \end{aligned}$$

We now claim that by the construction of  $H$ , we have that  $H, \ell, \mathcal{C} \models \phi_C$  if and only if  $G$  has a clique. If  $G$  has a clique  $\{v_{i_1}, v_{i_2}, \dots, v_{i_q}\}$ , we map  $x_1, x_2, \dots, x_q$  to arbitrary vertices of  $W_{i_1}, \dots, W_{i_q}$ . For the next part of the formula, either  $x_{q+1}, x_{q+2}$  correspond to some (different)  $x_i, x_{i'}$  or the formula is true. Last, we claim that  $H, \ell', \mathcal{C} \models \phi_{adj}[x_{q+1} \setminus u_i][x_{q+2} \setminus u_{i'}]$ , where  $x_i, x_{i'}$  are substituted by  $u_i, u_{i'}$  and  $\ell'(u_i) \in W_i, \ell'(u_{i'}) \in W_{i'}$ . Indeed, because we have a clique in  $G$ , by construction there exists a  $Y_j$  such that each vertex of  $Y_j^1$  has the same neighborhood in  $S$  as  $W_i$  and each vertex of  $Y_j^2$  has the same neighborhood in  $S$  as  $W_{i'}$  (or the same with the roles of  $Y_j^1, Y_j^2$  reversed). Hence,  $\phi_{adj}$  is satisfied.

For the converse direction, suppose that  $H, \ell, \mathcal{C} \models \phi_C$  for the nowhere defined labeling function  $\ell$ . Then there exists a labeling function  $\ell'$  that assigns  $\ell'(u_1), \ell'(u_2), \dots, \ell'(u_q)$  to some vertices of  $\bigcup_{i \in [n]} W_i$  and is undefined everywhere else such that  $\ell'(u_i) \neq \ell'(u_{i'})$  for  $i \neq i'$  and  $H, \ell', \mathcal{C} \models \phi_C$  where

$$\phi_{C'} := \forall x_{q+1} \forall x_{q+2} \bigwedge_{i \in [q]} (\neg(x_{q+1} = u_i)) \vee \bigwedge_{i \in [q]} (\neg(x_{q+2} = u_i)) \vee (x_{q+1} = x_{q+2}) \vee \phi_{adj}(x_{q+1}, x_{q+2})$$

We extract a multi-set  $S$  of  $q$  vertices of  $G$  as follows: for  $\beta \in [q]$ , if  $\ell'(u_\beta) \in W_i$ , then we add  $v_i$  to  $S$ . We claim that for any two elements  $v_i, v_{i'}$  of  $S$  we have  $(v_i, v_{i'}) \in E$ . If we prove this, then the vertices of  $S$  are distinct and form a  $q$ -clique in  $G$ .

Since we have universal quantifications for  $x_{q+1}, x_{q+2}$ , we can define a new labeling function  $\ell''$ , with  $\ell''(u_{q+1}) = \ell'(u_i)$  and  $\ell''(u_{q+2}) = \ell'(u_{i'})$ , for any  $i, i' \in [q], i \neq i'$ , with  $\ell'', \ell'$  agreeing everywhere else. Observe that this selection imposes that  $H, \ell'', \mathcal{C} \models \phi_{adj}[x_{q+1} \setminus u_i][x_{q+2} \setminus u_{i'}]$  and from property 5 of Lemma 16 we get that  $\ell'(u_i), \ell'(u_{i'})$  belong to two different  $W_j, W_{j'}$  that correspond to the endpoints of an edge of  $G$ .  $\blacktriangleleft$

► **Theorem 18.** *If there exists an algorithm which, given a graph  $G$  with  $n$  vertices and  $\iota(G) = k$  and an MSO formula  $\phi$  with constant size, decides whether  $G \models \phi$  in time  $2^{2^{o(k^2)}} n^{O(1)}$ , then the ETH is false.*

**Proof.** Our strategy is similar to that of Theorem 17, except that we will now reduce from 3-COLORING, which is known not to be solvable in  $2^{o(n)}$  on graphs on  $n$  vertices, under the ETH [33]. We will produce a constant-sized formula  $\phi_{Col}$  with the property that  $H, \ell, \mathcal{C} \models \phi_{Col}$  for the nowhere defined functions  $\ell, \mathcal{C}$  if and only if  $G$  is 3-colorable. Since  $k = O(\sqrt{\log n})$  an algorithm running in  $2^{2^{o(k^2)}}$  would imply a  $2^{o(n)}$  algorithm for 3-coloring  $G$ , contradicting the ETH. We define

$$\phi_{Col} := \exists X_1 \exists X_2 \exists X_3 \forall x_1 \forall x_2 (x_1 \in X_1 \vee x_1 \in X_2 \vee x_1 \in X_3) \wedge \bigwedge_{i=1,2,3} \phi_{adj}(x_1, x_2) \rightarrow (x_1 \in X_i \rightarrow \neg(x_2 \in X_i))$$

Assume that  $G$  has a proper 3-coloring  $c : V \rightarrow [3]$ . Then we define, for  $\alpha \in [2]$   $S_\alpha = \bigcup_{i:c(v_i)=\alpha} W_i$  and  $S_3 = V(H) \setminus (S_1 \cup S_2)$ . Let  $\mathcal{C}'$  be a coloring function such that  $\mathcal{C}'(D_\alpha) = S_\alpha$  for  $\alpha = 1, 2, 3$  and  $\mathcal{C}'(D_{\alpha'}) \uparrow$  for  $\alpha' \notin [3]$ . We claim that  $H, \ell, \mathcal{C}' \models \phi_{Col}[X_1 \setminus D_1][X_2 \setminus D_2][X_3 \setminus D_3]$ . Indeed, for any labeling function  $\ell'$  that defines only  $\ell'(u_1)$  and  $\ell'(u_2)$  we have (i)  $H, \ell', \mathcal{C}' \models u_1 \in D_1 \vee u_1 \in D_2 \vee u_1 \in D_3$  (since  $\mathcal{C}'(D_1), \mathcal{C}'(D_2), \mathcal{C}'(D_3)$  is a partition of  $V(H)$ ); (ii) If  $H, \ell', \mathcal{C}' \models \phi_{adj}[x_1 \setminus u_1][x_2 \setminus u_2]$  then  $\ell'(u_1) \in W_i, \ell'(u_2) \in W_{i'}$  for some  $i, i' \in [n], i \neq i'$  with  $(v_i, v_{i'}) \in E(G)$  (from property 5 of Lemma 16). Therefore  $c(v_i) \neq c(v_{i'})$  so for  $\alpha \in [3]$   $H, \ell', \mathcal{C}' \models u_1 \in D_\alpha \rightarrow \neg u_2 \in D_\alpha$ .

For the converse direction, suppose that  $H, \ell, \mathcal{C} \models \phi_{Col}$  for the nowhere defined  $\ell, \mathcal{C}$ . Then there exists a coloring function  $\mathcal{C}'$  such that  $\mathcal{C}'(D_\alpha) = S_\alpha$ , for  $\alpha \in [3]$  and  $H, \ell, \mathcal{C}' \models \phi_{Col}[X_1 \setminus D_1][X_2 \setminus D_2][X_3 \setminus D_3]$ . We extract a coloring of  $V(G)$  as follows: for  $i \in [n]$  we set  $c(v_i)$  to be the minimum  $\alpha$  such that  $W_i \cap S_\alpha \neq \emptyset$ . We show that the coloring  $c : V(G) \rightarrow [3]$  defined in this way is proper. Consider  $i, i' \in [n]$  such that  $(v_i, v_{i'}) \in E(G)$ . Let  $\ell'$  be a labeling function such that  $\ell'(u_1) \in W_i \cap S_{c(v_i)}$  and  $\ell'(u_2) \in W_{i'} \cap S_{c(v_{i'})}$ . Observe that  $W_i \cap S_{c(v_{i'})} \neq \emptyset$  by the definition of  $c(v_i)$ . Then  $H, \ell', \mathcal{C}' \models \phi_{adj}[x_1 \setminus u_1][x_2 \setminus u_2]$ . Therefore we have that for  $\alpha \in [3]$ ,  $H, \ell', \mathcal{C}' \models u_1 \in D_\alpha \rightarrow \neg(u_2 \in D_\alpha)$ . Therefore  $S_{c(v_i)} \neq S_{c(v_{i'})}$ , which means that  $c(v_i) \neq c(v_{i'})$ . ◀

## References

- 1 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In *ESA*, volume 173 of *LIPICs*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 2 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Parameterized (approximate) defective coloring. *SIAM J. Discret. Math.*, 34(2):1084–1106, 2020.
- 3 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (meta) kernelization. *J. ACM*, 63(5):44:1–44:69, 2016.
- 4 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph isomorphism on graph classes that exclude a substructure. *Algorithmica*, 82(12):3566–3587, 2020.
- 5 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. In *FOCS*, pages 601–612. IEEE, 2020.
- 6 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
- 7 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

- 8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 9 Anuj Dawar, Martin Grohe, Stephan Kreutzer, and Nicole Schweikardt. Approximation schemes for first-order definable optimisation problems. In *LICS*, pages 411–420. IEEE Computer Society, 2006.
- 10 Holger Dell, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Tobias Mömke. Complexity and approximability of parameterized max-csps. *Algorithmica*, 79(1):230–250, 2017.
- 11 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016.
- 12 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 607–613. ijcai.org, 2017. doi:10.24963/ijcai.2017/85.
- 13 Pavel Dvořák and Dusan Knop. Parameterized complexity of length-bounded cuts and multicuts. *Algorithmica*, 80(12):3597–3617, 2018.
- 14 Zdenek Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013.
- 15 Eduard Eiben, Robert Ganian, and Stefan Szeider. Meta-kernelization using well-structured modulators. *Discret. Appl. Math.*, 248:153–167, 2018.
- 16 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.
- 17 Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.
- 18 Markus Frick. Generalized model-checking over locally tree-decomposable classes. *Theory Comput. Syst.*, 37(1):157–191, 2004.
- 19 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.
- 20 Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Log.*, 130(1-3):3–31, 2004.
- 21 Jakub Gajarský and Petr Hliněný. Kernelizing MSO properties of trees of fixed height, and some consequences. *Log. Methods Comput. Sci.*, 11(1), 2015.
- 22 Robert Ganian. Improving vertex cover as a graph parameter. *Discret. Math. Theor. Comput. Sci.*, 17(2):77–100, 2015.
- 23 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1), 2019.
- 24 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In *MFCS*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012.
- 25 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021.
- 26 Robert Ganian and Jan Obdržálek. Expanding the expressive power of monadic second-order logic on restricted graph classes. In *IWOCA*, volume 8288 of *Lecture Notes in Computer Science*, pages 164–177. Springer, 2013.
- 27 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. On structural parameterizations of the edge disjoint paths problem. *Algorithmica*, 83(6):1605–1637, 2021. doi:10.1007/s00453-020-00795-3.
- 28 Robert Ganian, Friedrich Slivovsky, and Stefan Szeider. Meta-kernelization with structural parameters. *J. Comput. Syst. Sci.*, 82(2):333–346, 2016.



- 29 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrty. In *CIAC*, volume 12701 of *Lecture Notes in Computer Science*, pages 271–285. Springer, 2021.
- 30 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. *Model Theoretic Methods in Finite Combinatorics*, 558:181–206, 2011.
- 31 Gregory Z. Gutin, Mark Jones, and Magnus Wahlström. The mixed chinese postman problem parameterized by pathwidth and treedepth. *SIAM J. Discrete Math.*, 30(4):2177–2205, 2016. doi:10.1137/15M1034337.
- 32 Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos. Digraph coloring and distance to acyclicity. In *STACS*, volume 187 of *LIPICs*, pages 41:1–41:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 33 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 34 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for  $(k, r)$ -center. *Discret. Appl. Math.*, 264:90–117, 2019.
- 35 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized  $d$ -scattered set. *Discrete Applied Mathematics*, 2020. doi:10.1016/j.dam.2020.03.052.
- 36 Leon Kellerhals and Tomohiro Koana. Parameterized complexity of geodetic set. In *IPEC*, volume 180 of *LIPICs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 37 Dusan Knop, Martin Koutecký, Tomáš Masarík, and Tomáš Toufar. Simplified algorithmic metatheorems beyond MSO: treewidth and neighborhood diversity. *Log. Methods Comput. Sci.*, 15(4), 2019.
- 38 Dusan Knop, Tomáš Masarík, and Tomáš Toufar. Parameterized complexity of fair vertex evaluation problems. In *MFCS*, volume 138 of *LIPICs*, pages 33:1–33:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 39 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.
- 40 Michael Lampis. Model checking lower bounds for simple graphs. *Log. Methods Comput. Sci.*, 10(1), 2014.
- 41 Michael Lampis. Minimum stable cut and treewidth. In *ICALP*, volume 198 of *LIPICs*, pages 92:1–92:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 42 Michael Lampis and Valia Mitsou. Treewidth with a quantifier alternation revisited. In *IPEC*, volume 89 of *LIPICs*, pages 26:1–26:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.
- 43 Michal Pilipczuk. Problems parameterized by treewidth tractable in single exponential time: A logical approach. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 520–531. Springer, 2011.
- 44 Stefan Szeider. Monadic second order logic on graphs with local cardinality constraints. *ACM Trans. Comput. Log.*, 12(2):12:1–12:21, 2011.









# Essentially Tight Kernels For (Weakly) Closed Graphs

Tomohiro Koana  

Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Christian Komusiewicz  

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Frank Sommer  

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

---

## Abstract

We study kernelization of classic hard graph problems when the input graphs fulfill triadic closure properties. More precisely, we consider the recently introduced parameters closure number  $c$  and weak closure number  $\gamma$  [Fox et al., SICOMP 2020] in addition to the standard parameter solution size  $k$ . The weak closure number  $\gamma$  of a graph is upper-bounded by the minimum of its closure number  $c$  and its degeneracy  $d$ . For CAPACITATED VERTEX COVER, CONNECTED VERTEX COVER, and INDUCED MATCHING we obtain the first kernels of size  $k^{\mathcal{O}(\gamma)}$ ,  $k^{\mathcal{O}(\gamma)}$ , and  $(\gamma k)^{\mathcal{O}(\gamma)}$ , respectively. This extends previous results on the kernelization of these problems on degenerate graphs. These kernels are essentially tight as these problems are unlikely to admit kernels of size  $k^{\mathcal{O}(\gamma)}$  by previous results on their kernelization complexity in degenerate graphs [Cygan et al., ACM TALG 2017]. For CAPACITATED VERTEX COVER, we show that even a kernel of size  $k^{\mathcal{O}(c)}$  is unlikely. In contrast, for CONNECTED VERTEX COVER, we obtain a problem kernel with  $\mathcal{O}(ck^2)$  vertices. Moreover, we prove that searching for an induced subgraph of order at least  $k$  belonging to a hereditary graph class  $\mathcal{G}$  admits a kernel of size  $k^{\mathcal{O}(\gamma)}$  when  $\mathcal{G}$  contains all complete and all edgeless graphs. Finally, we provide lower bounds for the kernelization of INDEPENDENT SET on graphs with constant closure number  $c$  and kernels for DOMINATING SET on weakly closed split graphs and weakly closed bipartite graphs.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Parameterized complexity and exact algorithms; Theory of computation  $\rightarrow$  Graph algorithms analysis

**Keywords and phrases** Fixed-parameter tractability, kernelization,  $c$ -closure, weak  $\gamma$ -closure, Independent Set, Induced Matching, Connected Vertex Cover, Ramsey numbers, Dominating Set

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.35

**Related Version** A continuously updated version of the paper is available at: <https://arxiv.org/abs/2103.03914>

**Funding** Tomohiro Koana: Supported by the Deutsche Forschungsgemeinschaft (DFG), project DiPa, NI 369/21.

Frank Sommer: Supported by the Deutsche Forschungsgemeinschaft (DFG), project EAGR, KO 3669/6.

**Acknowledgements** We would like to thank the anonymous reviewers of ISAAC'21 for their many helpful remarks that have substantially improved the presentation of the results in this paper.

## 1 Introduction

A main tool for coping with hard computational problems is to shrink large input data to a computationally hard core by removing easy parts of the instance in polynomial time. Parameterized algorithmics provides the framework of *kernelization* for analyzing the power and limits of polynomial-time data reduction algorithms. In addition to the input instance  $I$ ,



© Tomohiro Koana, Christian Komusiewicz, and Frank Sommer;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 35; pp. 35:1–35:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a parameterized problem comes equipped with a parameter  $k$  which describes the structure of the input or is a bound on the solution size. A *kernelization* for a parameterized problem  $L$  is an algorithm that replaces every input instance  $(I, k)$  of  $L$  in polynomial time by an equivalent instance  $(I', k')$  of  $L$  (the kernel) whose size depends only on the parameter  $k$ , that is,  $|I'| + k' \leq g(k)$  for some computable function  $g$ . The kernel is guaranteed to be small if  $k$  is small and  $g$  grows only modestly. A particularly important special case is thus a kernelization where  $g$  is a polynomial function. Such kernels are referred to as *polynomial kernelizations*.

Many problems do not admit a kernel simply because they are believed to be not *fixed-parameter tractable*. That is, it is assumed that they are not solvable in  $f(k) \cdot |I|^{\mathcal{O}(1)}$  time. A classic example is DOMINATING SET parameterized by the solution size  $k$ . Moreover, even problems that do admit kernels are known to not admit polynomial kernels [2, 7, 21, 22]<sup>1</sup>; a classic example is CONNECTED VERTEX COVER parameterized by the solution size  $k$  [7].

To devise kernelization algorithms for such problems, one considers either additional parameters or restricted classes of input graphs. One example for this approach is kernelization in degenerate graphs [4, 5, 24]. A graph  $G$  is  $d$ -degenerate if every subgraph of  $G$  contains at least one vertex with degree at most  $d$ . DOMINATING SET, for example, admits a kernel of size  $k^{\mathcal{O}(d^2)}$  where  $d$  is the degeneracy of the input graph [24]. Thus, the exponent of the kernel size depends on  $d$ ; we will say that DOMINATING SET admits a polynomial kernel on  $d$ -degenerate graphs. This kernelization was shown to be tight in the sense that there is no kernel of size  $k^{\mathcal{O}(d^2)}$  [4]. The situation is different for INDEPENDENT SET which admits a trivial problem kernel with  $\mathcal{O}(dk)$  vertices: here the kernel size is polynomial in  $d + k$ .

Real-world networks have small degeneracy  $d$ , making  $d$  an interesting parameter from an application point of view. Moreover, bounded degeneracy imposes combinatorial structure that can be exploited algorithmically as evidenced by the discussion above. Recently, Fox et al. [13] discovered two new parameters that share these two features; they are well-motivated from a practical standpoint and describe interesting and useful combinatorial features of graphs. The first parameter is the *closure* of a graph, defined as follows.

► **Definition 1.1** ([13, Definition 1.1]). Let  $\text{cl}_G(v) := \max_{w \in V(G) \setminus N[v]} \{|N(v) \cap N(w)|, 0\}$  denote the closure number of a vertex  $v$  in a graph  $G$ . A graph  $G$  is  $c$ -closed if  $\text{cl}_G(v) < c$  for all  $v \in V(G)$ .

In other words, a graph is  $c$ -closed if every pair of nonadjacent vertices has at most  $c - 1$  common neighbors. The parameter models triadic closure in social networks, the observation that people with many common acquaintances are likely to know each other. Fox et al. [13] devised another parameter, the *weak closure* which relates to  $c$ -closure as degeneracy relates to maximum degree: instead of demanding a bounded closure number for every vertex, one demands that every induced subgraph has some vertex with bounded closure number.

► **Definition 1.2** ([13, Definition 1.3]). A graph  $G$  is weakly  $\gamma$ -closed if

- there exists a weak closure ordering  $\sigma := v_1, \dots, v_n$  of the vertices of  $G$  such that  $\text{cl}_{G_i}(v_i) < \gamma$  for all  $i \in [n]$  where  $G_i := G[\{v_i, \dots, v_n\}]$ , or, equivalently, if
- every induced subgraph  $G'$  of  $G$  has a vertex  $v \in V(G')$  such that  $\text{cl}_{G'}(v) < \gamma$ .

The weak closure number of a graph  $G$  is the minimum integer  $\gamma$  such that the graph  $G$  is weakly  $\gamma$ -closed.

<sup>1</sup> All kernelization lower bounds mentioned in this work are based on the assumption  $\text{coNP} \not\subseteq \text{NP/poly}$ .

Let  $G$  be a graph and let  $d, c$ , and  $\gamma$  be the degeneracy, the closure number and the weak closure number of  $G$ . The three parameters  $d, c$ , and  $\gamma$  are related as follows:

1. The weak closure number  $\gamma$  is at most  $\max(d + 1, c)$ .
2. The weak closure number  $\gamma$  can be arbitrarily smaller than  $d$  as witnessed by large complete graphs.
3. The degeneracy  $d$  and the closure number  $c$  are incomparable as witnessed by large complete graphs (these have large degeneracy and are 1-closed) and large complete bipartite graphs where one part has size two (these are 2-degenerate and have large closure number).
4. The latter example also shows that  $\gamma$  can be much smaller than the closure number  $c$  which is very often the case in real-world data [13, 19].

Akin to degeneracy,  $c$ -closure and weak  $\gamma$ -closure have proven to be very useful parameters. In particular, all maximal cliques of a graph can be enumerated in  $3^{\gamma/3} \cdot n^{\mathcal{O}(1)}$  time [13]. By the above discussion on the relation of  $\gamma$  and  $d$ , this result thus extends the range of tractable clique enumeration instances from the class of bounded-degeneracy graphs [9] to the larger class of graphs with bounded weak closure. The clique enumeration algorithm for weak closed graphs [13] has also been extended to the enumeration of other clique-like subgraphs [16, 19]. Concerning kernelization, in previous work, we showed that INDEPENDENT SET and INDUCED MATCHING admit polynomial kernels with respect to the parameter  $k + c$  and that DOMINATING SET admits a polynomial kernel on  $c$ -closed graphs [20]. Later, we extended the kernelization result for INDEPENDENT SET to parameterization by weak closure. More generally, we showed that  $\mathcal{G}$ -SUBGRAPH, where one wants to find a subgraph on at least  $k$  vertices belonging to  $\mathcal{G}$  admits a kernel with  $\mathcal{O}(\gamma k^2)$  vertices if  $\mathcal{G}$  is closed under taking subgraphs [19]. To the best of our knowledge, this is the only known kernelization result for the weak closure parameterization.

In this work we study the kernelization of several further hard graph problems on weakly closed graphs. In a nutshell, we provide kernels for a range of problems that have not been considered on weakly closed graphs so far. Our kernels are based on several combinatorial observations on the structure of weakly closed graphs that might be of more general interest.

**Our Results.** Building on a combinatorial lemma of Frankl and Wilson [14], we obtain a general lemma (Lemma 2.2) which can be used to bound the size of graphs in terms of their vertex cover number and weak closure number. More precisely, we show that in a graph  $G$  with vertex cover  $S$  of size  $k$  and weak closure  $\gamma$ , the number of different neighborhoods in the independent set  $I := V(G) \setminus S$  is  $k^{\mathcal{O}(\gamma)}$ . Lemma 2.2 gives a general strategy for obtaining kernels in weakly closed graphs: Devise reduction rules that 1) bound the size of the vertex cover and 2) decrease the size of neighborhood classes. We also show that Lemma 2.2 can be extended to a more general notion of neighborhood types (Lemma 2.4).

We then show that this strategy helps in obtaining kernels on weakly closed graphs for CAPACITATED VERTEX COVER, CONNECTED VERTEX COVER, CONNECTED  $\ell$ -COMPONENT ORDER CONNECTIVITY (CONNECTED  $\ell$ -COC), and INDUCED MATCHING all parameterized by the natural parameter solution size  $k$ . For these problems, polynomial kernels in degenerate graphs are known [4, 5, 11, 17]. Our results thus extend the class of graphs for which polynomial kernels are known for these problems. The kernels have size  $k^{\mathcal{O}(\gamma)}$  and  $(\gamma k)^{\mathcal{O}(\gamma)}$ , respectively, and by previous results on degenerate graphs the dependence on  $\gamma$  in the exponent cannot be avoided [4, 5]. We complement these findings with a study of CAPACITATED VERTEX COVER and CONNECTED VERTEX COVER on  $c$ -closed graphs. Interestingly, the kernelization complexity of the problems differs: CAPACITATED VERTEX COVER does not admit a kernel of size  $\mathcal{O}(k^{\frac{c-1}{2}-\epsilon})$  for all  $\epsilon > 0$  whereas CONNECTED VERTEX COVER admits a kernel with  $\mathcal{O}(ck^2)$  vertices.

Next, we study the kernelization complexity of INDEPENDENT SET on  $c$ -closed graphs. We show that INDEPENDENT SET does not admit a kernel of size  $\mathcal{O}(k^{2-\epsilon})$  on  $c$ -closed graphs for constant  $c$ . This complements previous kernels of size  $\mathcal{O}(c^2 k^3)$  [20] and  $\mathcal{O}(\gamma^2 k^3)$  [19], narrowing the gap between upper and lower bound for the achievable kernel size on (weakly) closed graphs. We also obtain a lower bound of  $\Omega(k^{4/3-\epsilon})$  on the number of *vertices* in the graph in case of constant  $c$  and show that at least a linear dependence on  $c$  is necessary in any kernelization of INDEPENDENT SET in  $c$ -closed graphs: under standard assumptions, there is no kernel of size  $c^{(1-\epsilon)} \cdot k^{\mathcal{O}(1)}$ . Some of our results also hold for Ramsey-type problems where one wants to find a large subgraph belonging to a class  $\mathcal{G}$  containing all complete and all edgeless graphs. In this context, we observe that weakly  $\gamma$ -closed graphs fulfill the Erdős–Hajnal property [10] with a linear dependence on  $\gamma$ : There is a constant  $q$  such that every weakly  $\gamma$ -closed graph on  $k^{q\gamma}$  vertices has either a clique of size  $k$  or an independent set of size  $k$ . We believe that this observation is of independent interest and that it will be useful in the further study of weakly  $\gamma$ -closed graphs.

Finally, we consider DOMINATING SET which admits a kernel of size  $k^{\mathcal{O}(c)}$  [20]. It is open whether DOMINATING SET admits a kernel of size  $k^{f(\gamma)}$  for some function  $f$ , which would extend the class of kernelizable input graphs from degenerate to weakly closed. We make partial progress towards answering this question by showing that there is a kernel of size  $k^{\mathcal{O}(\gamma^2)}$  on graphs with constant clique number (such as bipartite graphs) and a kernel of size  $(\gamma k)^{\mathcal{O}(\gamma)}$  in split graphs. In both cases these bounds are tight in the sense that kernels of size  $k^{\mathcal{O}(d^2)}$  and of size  $k^{\mathcal{O}(c)}$  are unlikely to exist [4, 19].

Due to lack of space, several proofs (marked with (\*)) and all results for DOMINATING SET on bipartite and split graphs are deferred to the full version of this article.

**Preliminaries.** By  $[n]$  we denote the set  $\{1, \dots, n\}$  for some  $n \in \mathbb{N}$ . For a graph  $G$ , let  $V(G)$  denote its *vertex set*,  $E(G)$  its *edge set*, and  $n := |V(G)|$  the number of vertices. Let  $X \subseteq V(G)$  be a vertex set. By  $G[X]$  we denote the *subgraph induced* by  $X$  and by  $G - X := G[V(G) \setminus X]$  we denote the graph obtained by removing the vertices of  $X$ . If the vertices of  $X$  are pairwise adjacent (nonadjacent), then  $X$  is a *clique* (an *independent set*, respectively). We denote by  $N_G(X) := \{y \in V(G) \setminus X \mid xy \in E(G), x \in X\}$  the *open neighborhood* of  $X$  and by  $N_G[X] := N_G(X) \cup X$  the *closed neighborhood* of  $X$ . The maximum degree of  $G$  is  $\Delta_G := \max_{v \in V(G)} \deg_G(v)$ . In the remainder of this paper we fix a weak closure ordering  $\sigma$ . Note that such an ordering can be computed in polynomial time [13]. We define  $P_G^\sigma(v) := \{u \in N_G(v) \mid u \text{ appears before } v \text{ in } \sigma\}$  and  $Q_G^\sigma(v) := \{u \in N_G(v) \mid u \text{ appears after } v \text{ in } \sigma\}$ . We say that  $P_G^\sigma(v)$  are *prior neighbors* of  $v$  and  $Q_G^\sigma(v)$  are *posterior neighbors* of  $v$ . A *matching*  $M$  is a set of vertex-disjoint edges. By  $V(M)$  we denote the union of all endpoints of edges in  $M$ . We omit the superscripts and subscripts when they are clear from the context. The following observation follows from the definition of weak closure.

► **Observation 1.3.** *For nonadjacent vertices  $u, v \in V(G)$ , it holds that  $|Q(u) \cap Q(v)| \leq |Q(u) \cap N(v)| \leq \gamma - 1$ .*

**Proof.** Let  $G_u$  and  $G_v$  be the graph induced by the vertices that appear after  $u$  and  $v$ , respectively. We have two cases based on whether  $u$  or  $v$  appears first in the weak closure ordering  $\sigma$ .

- $u$  precedes  $v$ . By the definition of  $Q$ -neighbors, we have  $Q_G(u) \cap Q_G(v) \subseteq Q_G(u) \cap N_G(v)$  and  $Q_G(u) \cap N_G(v) = N_{G_u}(u) \cap N_{G_u}(v)$ . Since  $|N_{G_u}(u) \cap N_{G_u}(v)| \leq \text{cl}_{G_u}(u)$ , it follows from the definition of weak closure that  $|N_{G_u}(u) \cap N_{G_u}(v)| \leq \gamma - 1$ .

- $v$  precedes  $u$ . Clearly,  $Q_G(u) \cap P_G(v) = \emptyset$ . We then have  $Q_G(u) \cap N_G(v) = Q_G(u) \cap (P_G(v) \cup Q_G(v)) = Q_G(u) \cap Q_G(v)$ . It follows from the definition of  $Q$ -neighbors that  $|Q_G(u) \cap Q_G(v)| = |N_{G_u}(u) \cap N_{G_v}(v)| \leq |N_{G_v}(u) \cap N_{G_v}(v)| \leq \text{cl}_{G_v}(v) \leq \gamma - 1$ . We have shown that  $|Q(u) \cap Q(v)| \leq |Q(u) \cap N(v)| \leq \gamma - 1$  for both cases. ◀

A parameterized problem is *fixed-parameter tractable* if every instance  $(I, k)$  can be solved in  $f(k) \cdot |I|^{\mathcal{O}(1)}$  time for some computable function  $f$ . An algorithm with such a running time is an *FPT algorithm*. A *kernelization* is a polynomial-time algorithm which transforms every instance  $(I, k)$  of a parameterized language  $Q$  into an equivalent instance  $(I', k')$  of  $Q$  such that  $|I'| + k' \leq g(k)$  for some computable function  $g$ . A *compression* of a parameterized language  $Q$  into a language  $R$  is an algorithm that takes as input an instance  $(x, k) \in \Sigma^* \times \mathbb{N}$  and returns a string  $y$  in time polynomial in  $|x| + k$  such that  $|y|$  is bounded by some polynomial in  $k$ , and  $y \in R$  if and only if  $(x, k) \in Q$ . It is widely believed that  $W[t]$ -hard problems ( $t \in \mathbb{N}$ ) do not admit an FPT algorithm. For more details on parameterized complexity, we refer to the standard monographs [3, 8].

## 2 Bounding the Size of Weakly Closed Graphs with Small Twin Sets

Frankl and Wilson [14] proved the following bound on the size of set systems where the number of different intersection sizes is bounded.

► **Proposition 2.1** ([14, Theorem 11]). *Let  $\mathcal{F}$  be a collection of pairwise distinct subsets of  $[n]$  and let  $L \subseteq \{0\} \cup [n]$  be some subset. If  $|S \cap S'| \in L$  for all distinct  $S, S' \in \mathcal{F}$ , then  $|\mathcal{F}| \in \mathcal{O}(n^{|L|})$ .*

We now use this proposition to achieve a bound on the size of weakly closed graphs when every vertex has few false twins and the size of the vertex cover is small. Herein, two vertices  $u$  and  $v$  are *false twins* if  $N(u) = N(v)$ .

► **Lemma 2.2.** *Let  $G$  be a weakly  $\gamma$ -closed graph and let  $I$  be an independent set of  $G$ . Suppose that each vertex  $v \in I$  has at most  $t - 1$  false twins. Then,  $|I| \in t \cdot \mathcal{O}(3^{\gamma/3} \cdot k^{2\gamma+3})$ , where  $k := n - |I|$ .*

**Proof.** We say that two vertices  $v, v' \in I$  are *P-equivalent*, *Q-equivalent*, and *N-equivalent* if  $P(v) = P(v')$ ,  $Q(v) = Q(v')$ , and  $N(v) = N(v')$ , respectively. Let  $\mathcal{P}$ ,  $\mathcal{Q}$ , and  $\mathcal{N}$  denote the collection of *P*-equivalence, *Q*-equivalence, and *N*-equivalence classes, respectively. We extend the notation of *P*, *Q*, and *N* to an equivalence classes  $A$  by defining  $P(A) := P(v)$ ,  $Q(A) := Q(v)$ , and  $N(A) := N(v)$  for some  $v \in A$ . Since there is at most one *N*-equivalence class for every pair of *P*-equivalent and *Q*-equivalent classes, we have  $|\mathcal{N}| \leq |\mathcal{P}| \cdot |\mathcal{Q}|$ . By the assumption that there are at most  $t$  vertices in each *N*-equivalence class, we also have  $|I| \leq t \cdot |\mathcal{N}|$ . Thus, it suffices to show suitable bounds on  $|\mathcal{P}|$  and  $|\mathcal{Q}|$ .

First, we prove that  $|\mathcal{Q}| \in \mathcal{O}(k^\gamma)$ , using the result of Frankl and Wilson (Proposition 2.1 [14]). Since  $I \supseteq A$  is an independent set,  $Q(A) \subseteq S := V(G) \setminus I$ . Moreover, for two distinct *Q*-equivalence classes  $A$  and  $A'$ , we have  $|Q(A) \cap Q(A')| < \gamma$  by Observation 1.3, and equivalently,  $|Q(A) \cap Q(A')| \in L$  for  $L := \{0\} \cup [\gamma - 1]$ . By Proposition 2.1 we obtain  $|\mathcal{Q}| \in \mathcal{O}(|S|^{|L|}) = \mathcal{O}(k^\gamma)$ .

Next, we bound the size of  $\mathcal{P}$ . Let  $I_0 := \{v \in I \mid \exists u, w \in P(v): uw \notin E(G)\}$  be the set of vertices in  $I$  with nonadjacent prior neighbors. By the definition of weak  $\gamma$ -closure, there are at most  $\gamma - 1$  vertices of  $I_0$  for every pair of nonadjacent vertices in  $S$ . Thus, we have  $|I_0| < \gamma \binom{|S|}{2} \in \mathcal{O}(\gamma k^2)$ .

Let  $I_1 := I \setminus I_0$  and let  $\mathcal{P}_1$  be the collection of  $P$ -equivalence classes in  $I_1$ . Note that for every  $A \in \mathcal{P}_1$ , its neighborhood  $P(A)$  is a clique. Since a weakly  $\gamma$ -closed graph on  $n$  vertices has  $\mathcal{O}(3^{\gamma/3}n^2)$  maximal cliques [13], there are  $\mathcal{O}(3^{\gamma/3}k^2)$  equivalence classes  $A$  such that  $P(A)$  constitutes a maximal clique in  $G[S]$ . Consider an equivalence class  $A$  such that  $P(A) \subset C$  for some maximal clique  $C$  in  $G[S]$ . We will show that there are  $k^{\mathcal{O}(\gamma)}$  such equivalence classes. Let  $u$  be the first vertex of  $C \setminus P(A)$  in the weak closure ordering  $\sigma$ . Since  $P(A) \subset C \subseteq N(u) = P(u) \cup Q(u)$ , we have  $P(A) = (P(A) \cap P(u)) \cup (P(A) \cap Q(u))$ . As  $P(A) \cap P(u) = C \cap P(u)$  by the choice of  $u$ , we can rewrite  $P(A) = (C \cap P(u)) \cup B$ , where  $B := P(A) \cap Q(u)$ . Thus, there is at most one equivalence class of  $\mathcal{P}_1$  for every maximal clique  $C$  in  $G[S]$ , vertex  $u \in S$ , and vertex subset  $B \subseteq S$ , and thereby, we have  $|\mathcal{P}_1| \in \mathcal{O}(3^{\gamma/3}k^2 \cdot k \cdot b)$ , where  $b$  denotes the number of choices for  $B$ . Observe that  $P(A) = P(v)$  for some vertex  $v \in I_1$  and thus that  $B = Q(u) \cap P(v) \subseteq Q(u) \cap N(v)$ . Recall that  $u$  and  $v$  are not adjacent by the choice of  $u$ . It follows that  $|B| \leq |Q(u) \cap N(v)| \leq \gamma - 1$  by Observation 1.3, and hence  $b \in \mathcal{O}(k^\gamma)$  and  $|\mathcal{P}_1| \in \mathcal{O}(3^{\gamma/3} \cdot k^3 \cdot k^\gamma) = \mathcal{O}(3^{\gamma/3} \cdot k^{\gamma+3})$ . Overall, we have  $|\mathcal{P}| \leq (|I_0| + |\mathcal{P}_1|) \in \mathcal{O}(3^{\gamma/3} \cdot k^{\gamma+3})$ . The total number of  $N$ -equivalence classes is thus at most  $|\mathcal{Q}| \cdot |\mathcal{P}| \in \mathcal{O}(3^{\gamma/3} \cdot k^{2\gamma+3})$ . ◀

We now show that Proposition 2.1 can be also applied to bound the graph size in terms of the  $\ell$ -COC number, which is the smallest size of a vertex set  $S$  such that every connected component in  $G - S$  has size at most  $\ell$ , where  $\ell$  is a fixed constant. The 1-COC number is the vertex cover number. To obtain this generalization, we extend the notion of twins.

► **Definition 2.3.** *Let  $G = (V, E)$  be a graph and let  $A, B \subseteq V(G)$  such that  $|A| = |B| = \ell$ . The sets  $A$  and  $B$  are  $\ell$ -twins if there exists an ordering  $a_1, \dots, a_\ell$  of  $A$  and an ordering  $b_1, \dots, b_\ell$  of  $B$  such that  $N(a_i) \setminus A = N(b_i) \setminus B$  for each  $i \in [\ell]$ .*

Note that  $u$  and  $v$  are false twins if and only if  $\{u\}$  and  $\{v\}$  are 1-twins.

► **Lemma 2.4 (\*)**. *Let  $G$  be a graph and let  $D \subseteq V(G)$  be such that each connected component in  $G[D]$  has size at most  $\ell$ . Suppose that for every connected component  $Z$  in  $G[D]$ , there are at most  $t - 1$  other connected components  $Z'$  in  $G[D] - Z$  such that  $Z$  and  $Z'$  are  $|Z|$ -twins. Then,  $|D| \in \mathcal{O}(t \cdot k^{\mathcal{O}(\gamma)})$ , where  $k = n - |D|$ .*

### 3 Applications of our Framework

We now apply Lemma 2.2 and Lemma 2.4 to obtain kernels for several well-known problems.

#### 3.1 Capacitated Vertex Cover

The first problem to which we apply Lemma 2.2 is CAPACITATED VERTEX COVER.

CAPACITATED VERTEX COVER

**Input:** A graph  $G$ , a capacity function  $\text{cap}: V(G) \rightarrow \mathbb{N}$ , and  $k \in \mathbb{N}$ .

**Question:** Is there a set  $S$  of at most  $k$  vertices and a function  $f$  mapping each edge of  $E(G)$  to one of its endpoints in  $S$  such that  $|\{e \in E(G) \mid f(e) = v\}| \leq \text{cap}(v)$  for all  $v \in S$ ?

CAPACITATED VERTEX COVER admits a kernel with  $\mathcal{O}(k^{d+1})$  vertices. Furthermore, this kernel is essentially tight: a kernel with  $\mathcal{O}(k^{d-\epsilon})$  vertices would imply  $\text{coNP} \subseteq \text{NP/poly}$  [4]. We will show that the reduction rule used to obtain a kernel in degenerate graphs also leads to a kernel in graphs with bounded weak closure. One may view this result as a way of showing that the rules are more powerful than what was previously known. The kernel uses the following rule.



► **Reduction Rule 3.1** ([4, Rule 3]). *If  $S \subseteq V(G)$  is a subset of false twin vertices with a common neighborhood  $N(S)$  such that  $|S| = k + 2 \geq |N(S)|$ , then remove a vertex with minimum capacity in  $S$  from  $G$ , and decrease all the capacities of vertices in  $N(S)$  by one.*

We omit the proof for the correctness of Reduction Rule 3.1, referring to Cygan et al. [4, Lemma 20]. One can easily verify that it does not increase the weak  $\gamma$ -closure. In the following theorem, we show that Reduction Rule 3.1 indeed gives us a kernel with  $k^{\mathcal{O}(\gamma)}$  vertices.

► **Theorem 3.2.** *CAPACITATED VERTEX COVER has a kernel of size  $k^{\mathcal{O}(\gamma)}$ .*

**Proof.** We show that a Yes-instance which is reduced with respect to Reduction Rule 3.1 has size  $k^{\mathcal{O}(\gamma)}$ . Let  $S$  be a capacitated vertex cover of size at most  $k$  of  $(G, \text{cap})$ . Let  $I := V(G) \setminus S$ . By definition,  $I$  is an independent set and  $N(v) \subseteq S$  for all  $v \in I$ . Moreover, since  $(G, \text{cap})$  is reduced with respect to Reduction Rule 3.1 there is no set of  $k + 2$  vertices in  $I$  that have the same neighborhood. Hence,  $I$  fulfills the condition of Lemma 2.2 with  $t = k + 2$ . Thus,  $|I| \in k \cdot k^{\mathcal{O}(\gamma)}$  which implies  $|V(G)| = |S| + |I| \in k^{\mathcal{O}(\gamma)}$ . ◀

We also show that this kernel is essentially tight even if  $\gamma$  is replaced by  $c$ .

► **Theorem 3.3 (\*)**. *For  $c \geq 4$ , CAPACITATED VERTEX COVER has no kernel of size  $\mathcal{O}(k^{\frac{c-1}{2}-\epsilon})$  unless  $\text{coNP} \subseteq \text{NP/poly}$ .*

## 3.2 Connected Vertex Cover

We now provide kernels for CONNECTED VERTEX COVER, a well-studied variant of VERTEX COVER which is notoriously hard and does not admit a polynomial kernel when parameterized  $k$  [7].

CONNECTED VERTEX COVER

**Input:** A graph  $G$  and  $k \in \mathbb{N}$ .

**Question:** Is there a vertex cover  $S$  of size at most  $k$  in  $G$  such that  $G[S]$  is connected?

We will show that by applying Lemma 2.2 we obtain a kernel of size  $k^{\mathcal{O}(\gamma)}$ . We may use the following known rule.

► **Reduction Rule 3.4** ([4, Rule 2]). *If  $S \subseteq V(G)$  is a set of at least two twin vertices with a common neighborhood  $N(S)$  such that  $|S| > |N(S)|$ , then remove one vertex  $v$  of  $S$  from  $G$ .*

After exhaustive application of Reduction Rule 3.4 we have, again by Lemma 2.2, that every Yes-instance has size  $k^{\mathcal{O}(\gamma)}$ . The proof is completely analogous to that of Theorem 3.2.

► **Theorem 3.5.** *CONNECTED VERTEX COVER admits a kernel of size  $k^{\mathcal{O}(\gamma)}$ .*

This kernel is essentially tight, because there is no kernel of size  $k^{o(d)}$  [4]. We now show a polynomial kernel for  $k + c$ .

► **Theorem 3.6 (\*)**. *CONNECTED VERTEX COVER has a kernel with  $\mathcal{O}(ck^2)$  vertices.*

This result stands in contrast to CAPACITATED VERTEX COVER, which has no kernel of size  $k^{o(c)}$  unless  $\text{coNP} \subseteq \text{NP/poly}$  (Theorem 3.3).

**An Extension to Connected  $\ell$ -COC.** In CONNECTED  $\ell$ -COC the task is to find a set  $S$  of at most  $k$  vertices such that  $G[S]$  is connected and every connected component of  $G - S$  has size at most  $\ell$ , where  $\ell$  is a fixed constant. We show that this problem also admits a kernel of size  $k^{\mathcal{O}(\gamma)}$ . The main idea lies in the extension of Reduction Rule 3.4:

► **Reduction Rule 3.7.** *Let  $T_1, \dots, T_x \subseteq V(G)$  be a set of  $x$  many  $r$ -twins, for some  $r \in [\ell]$ . If  $x \geq k + \ell + 2$ , then remove all vertices in  $T_x$  from  $G$ .*

Note that Reduction Rule 3.7 can be exhaustively performed in polynomial time since the  $r$ -twin relation can be computed in  $n^{2r+\mathcal{O}(1)}$  time. We then obtain the following theorem from Lemma 2.4.

► **Theorem 3.8 (\*).** *CONNECTED  $\ell$ -COC has a kernel of size  $k^{\mathcal{O}(\gamma)}$  for constant  $\ell$ .*

### 3.3 Induced Matching

In this section, we provide a kernel of size  $(\gamma k)^{\mathcal{O}(\gamma)}$  for INDUCED MATCHING:

INDUCED MATCHING

**Input:** A graph  $G$  and  $k \in \mathbb{N}$ .

**Question:** Is there a set  $M$  of at least  $k$  edges such that the endpoints of distinct edges are pairwise nonadjacent?

INDUCED MATCHING is W[1]-hard for the parameter  $k$  on general graphs. For  $c$ -closed graphs, we developed a kernel with  $\mathcal{O}(c^7 k^8)$  vertices [20]. For  $d$ -degenerate graphs, Kanj et al. [17] and Erman et al. [11] independently presented kernels of size  $k^{\mathcal{O}(d)}$ . Later, Cygan et al. [4] provided a matching lower bound  $k^{\mathcal{O}(d)}$  on the kernel size. Note that this also implies the nonexistence of  $k^{\mathcal{O}(\gamma)}$ -size kernels unless  $\text{coNP} \subseteq \text{NP/poly}$ .

It turns out that Lemma 2.2 is again helpful in designing a  $k^{\mathcal{O}(\gamma)}$ -size kernel for INDUCED MATCHING. In a nutshell, we show that the application of a series of reduction rules results in a graph with a  $(\gamma k)^{\mathcal{O}(1)}$ -size vertex cover. We do so by combining the kernelization of Erman et al. [11] for degenerate graphs with our previous one for  $c$ -closed graphs [20]. Lemma 2.2 and the reduction rule which removes twin vertices then give us a kernel of size  $(\gamma k)^{\mathcal{O}(\gamma)}$ .

Erman et al. [11] use the following observation for degenerate graphs.

► **Lemma 3.9** ([11, Proof of Theorem 2.10]). *Any graph  $G$  with a matching  $M$  has an induced matching of size  $|M|/(4d_G + 1)$ .*

Ideally, we would like to prove a lemma analogous to Lemma 3.9 on weakly  $\gamma$ -closed graphs. Note, however, that a complete graph on  $n$  vertices (which is weakly 1-closed) has no induced matching of size 2, although it contains a matching of size  $\lfloor n/2 \rfloor$ . So we follow a different route, and prove an analogous lemma on weakly  $\gamma$ -closed bipartite graphs (there exist bipartite 2-closed graphs whose degeneracy is unbounded; see e.g. Eschen et al. [12]). As we shall see, this serves our purposes.

► **Lemma 3.10.** *Suppose that  $G$  is a bipartite graph with a bipartition  $(A, B)$ . If  $G$  has a matching  $M$  of size  $f_\gamma(k) := 4\gamma k^2 + 3k$ , then  $G$  has an induced matching of size  $k$ .*

**Proof.** Recall that  $Q^\sigma(v) := \{u \in N(v) \mid u \text{ appears after } v \text{ in } \sigma\}$ . Let  $S \subseteq V(G)$  be the set of vertices  $v$  such that  $|Q(v)| \geq \gamma k$ . Suppose that  $|S| \geq 2k$ . Then, we may assume that  $|A \cap S| \geq k$ . Let  $A' \subseteq A \cap S$  be an arbitrary vertex set of size exactly  $k$  and consider

some vertex  $v \in A'$ . Since  $|Q(v) \cap N(v')| < \gamma$  for every  $v' \in A' \setminus \{v\}$  by Observation 1.3, we have  $|Q(v) \setminus \bigcup_{v' \in A' \setminus \{v\}} N(v')| > 0$  for each  $v \in A'$ . Consequently, there is at least one vertex  $q_v \in Q(v) \setminus \bigcup_{v' \in A' \setminus \{v\}} N(v')$ . Then, the edge set  $\{vq_v \mid v \in A'\}$  forms an induced matching of size  $k$  in  $G$ .

Now, consider the case  $|S| < 2k$ . By the definition of  $S$ , it holds that  $|Q_{G-S}(v)| \leq |Q_G(v)| \leq \gamma k$  for each vertex  $v \in V(G) \setminus S$ . Hence, the degeneracy of  $G - S$  is at most  $\gamma k$ . Since  $G - S$  has a matching  $M_{G-S}$  of size at least  $|M| - |S| \geq f_\gamma(k) - 2k = 4\gamma k^2 + k$ , Lemma 3.9 yields an induced matching of size  $|M_{G-S}|/(4d_{G-S} + 1) \geq k$ . ◀

We use the following reduction rule to sparsify the graph  $G$  so that every sufficiently large vertex set contains a large independent set (see Lemma 3.13).

► **Reduction Rule 3.11.** *If for some vertex  $v \in V(G)$ , there is a maximum matching  $M_v$  of size at least  $2\gamma k$  in  $G[Q(v)]$ , then delete  $v$ .*

► **Lemma 3.12.** *Reduction Rule 3.11 is correct.*

**Proof.** Let  $G' := G - v$ . Suppose that  $G$  has an induced matching  $M$  of size  $k$ . If  $v \notin V(M)$ , then  $M$  is also an induced matching in  $G'$ . So assume that  $vv' \in V(M)$  for some vertex  $v' \in V(G)$ . Then, we have  $|N(u) \cap Q(v)| < \gamma$  for any vertex  $u \in V(M \setminus \{vv'\})$  by Observation 1.3 and thus  $|V(M_v) \setminus \bigcup_{u \in V(M \setminus \{vv'\})} N(u)| \geq 2|M_v| - (\gamma - 1)(2k - 2) > |M_v|$ . By the pigeon-hole principle, this implies that there is an edge  $e \in M_v$  not incident with any vertex in  $V(M)$  and no endpoint of  $e$  is adjacent to any vertex in  $V(M \setminus \{vv'\})$ . Then,  $(M \setminus \{vv'\}) \cup \{e\}$  is an induced matching of size  $k$  in  $G'$ . ◀

► **Lemma 3.13.** *Suppose that  $G$  is a graph in which Reduction Rule 3.11 is applied on every vertex. Then, every vertex set  $S \subseteq V(G)$  of size at least  $g_\gamma(k) := 4\gamma k^2 + k^2$  contains an independent set  $I \subseteq S$  of size  $k$ .*

**Proof.** Suppose that there is no independent set of size  $k$  in  $G' := G[S]$  for some vertex set  $S$  of size  $g_\gamma(k)$ . For every vertex  $v \in S$ , let  $M_v$  be a maximum matching in  $Q_{G'}(v)$  and let  $I_v := Q_{G'}(v) \setminus V(M_v)$ . By Reduction Rule 3.11, we have  $|V(M_v)| = 2|M_v| \leq 4\gamma k$ . Since  $I_v$  is an independent set, we then have  $|Q_{G'}(v)| = |M_v| + |I_v| < 4\gamma k + k$  for every vertex  $v \in S$ , and thus  $d_{G'} < 4\gamma k + k$ . Note, however, that  $G'$  has an independent set of size  $|S|/(d_{G'} + 1) \geq k$ , which is a contradiction. ◀

To identify a part of the graph with a sufficiently large induced matching, we rely on the LP relaxation of VERTEX COVER, following our approach [20] to obtain a polynomial kernel on  $c$ -closed graphs. Recall that VERTEX COVER can be formulated as an integer linear program as follows, using a variable  $x_v$  for each  $v \in V(G)$ :

$$\min \sum_{v \in V(G)} x_v \quad \text{subject to} \quad \begin{array}{l} x_u + x_v \geq 1 \quad \forall uv \in E(G), \\ x_v \in \{0, 1\} \quad \forall v \in V(G). \end{array}$$

We will refer to the LP relaxation of VERTEX COVER as VCLP. We use the well-known facts that VCLP always admits an optimal solution in which  $x_v \in \{0, 1/2, 1\}$  for each  $v \in V(G)$  and that such a solution can be found in polynomial time. Suppose that we have such an optimal solution  $(x_v)_{v \in V(G)}$ . Let  $V_0 := \{v \in V(G) \mid x_v = 0\}$ ,  $V_1 := \{v \in V(G) \mid x_v = 1\}$ , and  $V_{1/2} := \{v \in V(G) \mid x_v = 1/2\}$ . Also, let  $\text{opt}(G)$  be the optimum of VCLP. We show that we can immediately return Yes, whenever  $\text{opt}(G)$  is sufficiently large:

► **Reduction Rule 3.14.** *If  $\text{opt}(G) \geq 2g_\gamma(g_\gamma(f_\gamma(k)))$ , then return Yes.*

## 35:10 Essentially Tight Kernels for (Weakly) Closed Graphs

Here, the functions  $f_\gamma$  and  $g_\gamma$  are as specified in Lemmas 3.10 and 3.13, respectively.

► **Lemma 3.15.** *Reduction Rule 3.14 is correct.*

**Proof.** We show that  $G$  has an induced matching of size  $k$  whenever  $\text{opt}(G) \geq 2g_\gamma(g_\gamma(f_\gamma(k)))$ . Let  $M$  be an arbitrary maximal matching in  $G$ . Since  $V(M)$  is a vertex cover, we have  $\text{opt}(G) \leq |V(M)| = 2|M|$ , and hence  $|M| \geq \text{opt}(G)/2 \geq g_\gamma(g_\gamma(f_\gamma(k)))$ . Let  $M := \{a_1b_1, \dots, a_{|M|}b_{|M|}\}$  and let  $A := \{a_1, \dots, a_{|M|}\}$  and  $B := \{b_1, \dots, b_{|M|}\}$ . By Lemma 3.13, there exists an independent set  $A' \subseteq A$  of size  $s' := g_\gamma(f_\gamma(k))$ . Without loss of generality, suppose that  $A' = \{a'_1, \dots, a'_{s'}\}$  and let  $B' := \{b'_1, \dots, b'_{s'}\}$  be the set of vertices matched to  $A'$  in  $M$ . Again by Lemma 3.13, we obtain an independent set  $B'' \subseteq B'$  of size  $s'' := f_\gamma(k)$ . We assume without loss of generality that  $B'' = \{b''_1, \dots, b''_{s''}\}$ . Let  $A'' := \{a''_1, \dots, a''_{s''}\}$  be the set of vertices matched to  $B''$  in  $M$ . Then,  $G[A'' \cup B'']$  is a bipartite graph with a matching of size at least  $s'' = f_\gamma(k)$ . By Lemma 3.10,  $G[A'' \cup B'']$  has an induced matching of size  $k$ . ◀

Since  $\text{opt}(G) = |V_{1/2}|/2 + |V_1|$ , it holds that  $|V_{1/2}|/2 + |V_1| \leq 2g_\gamma(g_\gamma(f_\gamma(k))) \in \mathcal{O}(\gamma^7 k^8)$  after the application of Reduction Rule 3.14. Hence, it remains to bound the size of  $V_0$ . To do so, it suffices to remove twins:

► **Reduction Rule 3.16.** *If  $N(u) = N(v)$  for some vertices  $u, v \in V(G)$ , then delete  $v$ .*

Since an induced matching contains at most one of  $u$  and  $v$ , the rule is obviously correct. We are finally ready to utilize Lemma 2.2 to derive an upper bound on  $V_0$ : Since  $V_0$  is an independent set, Lemma 2.2 gives us  $|V_0| \in |V_{1/2} \cup V_1|^{\mathcal{O}(\gamma)} \in (\gamma k)^{\mathcal{O}(\gamma)}$ . Thus, we have the following result.

► **Theorem 3.17.** *INDUCED MATCHING has a kernel of size  $(\gamma k)^{\mathcal{O}(\gamma)}$ .*

## 4 Independent Set and Ramsey-Type Problems

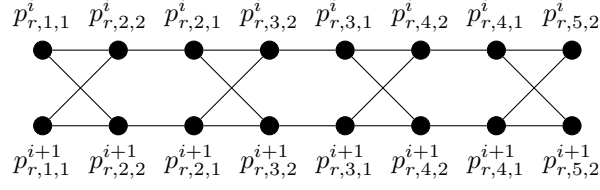
We now investigate the kernelization complexity of INDEPENDENT SET, where we are given a graph  $G$  and an integer  $k$ , and ask whether  $G$  has an independent set of size  $k$ . INDEPENDENT SET admits a kernel with  $\mathcal{O}(ck^2)$  vertices and  $\mathcal{O}(c^2k^3)$  edges [20] and a kernel with  $\mathcal{O}(\gamma k^2)$  and  $\mathcal{O}(\gamma^2 k^3)$  edges [19]. We show a lower bound for these parameterizations: unless  $\text{coNP} \subseteq \text{NP/poly}$ , INDEPENDENT SET admits no kernel of size  $k^{2-\epsilon}$  and no kernel with  $k^{4/3-\epsilon}$  vertices even if the  $c$ -closure is constant. We also show a kernel lower bound of size  $c^{1-\epsilon} k^{\mathcal{O}(1)}$  for any  $\epsilon$ . We also consider the following related problem where  $\mathcal{G}$  is a hereditary graph class containing all complete graphs and edgeless graphs.

$\mathcal{G}$ -SUBGRAPH

**Input:** A graph  $G$  and  $k \in \mathbb{N}$ .

**Question:** Is there a set  $S$  of at least  $k$  vertices such that  $G[S] \in \mathcal{G}$ ?

Khot and Raman [18] showed that  $\mathcal{G}$ -SUBGRAPH is FPT when parameterized by  $k$ , using Ramsey's theorem: for any  $k \in \mathbb{N}$ , any graph  $G$  on at least  $R(k) \in 2^{\mathcal{O}(k)}$  vertices contains a clique of size  $k$  or an independent set of size  $k$ . RAMSEY, the special case where  $\mathcal{G}$  is the family of all complete and edgeless graphs, admits no polynomial kernel unless  $\text{coNP} \subseteq \text{NP/poly}$  [21]. Similarly,  $\mathcal{G}$ -SUBGRAPH admits no polynomial kernel for several graph classes  $\mathcal{G}$ , such as cluster graphs [23]. Our contribution for  $\mathcal{G}$ -SUBGRAPH is two-fold: First, we observe that the lower bounds for INDEPENDENT SET on graphs with constant  $c$ -closure also



■ **Figure 1** An illustration of  $P_r^i$  and  $P_r^{i+1}$  for  $t = 5$ .

hold for RAMSEY. This complements a kernel for  $\mathcal{G}$ -SUBGRAPH with  $\mathcal{O}(ck^2)$  vertices [20].<sup>2</sup> Second, we provide a kernel of size  $k^{\mathcal{O}(\gamma)}$ . To show our kernel lower bounds, we will use *weak  $q$ -compositions*. Weak  $q$ -compositions exclude kernels of size  $\mathcal{O}(k^{q-\varepsilon})$  for  $\varepsilon > 0$ .

► **Definition 4.1** ([6, 15]). *Let  $q \geq 1$  be an integer, let  $L_1 \subseteq \{0, 1\}^*$  be a decision problem, and let  $L_2 \subseteq \{0, 1\}^* \times \mathbb{N}$  be a parameterized problem. A weak  $q$ -composition from  $L_1$  to  $L_2$  is a polynomial-time algorithm that on input  $x_1, \dots, x_{t^q} \in \{0, 1\}^n$  outputs an instance  $(y, k') \in \{0, 1\}^* \times \mathbb{N}$  such that:*

- $(y, k') \in L_2 \Leftrightarrow x_i \in L_1$  for some  $i \in [t^q]$ , and
- $k' \leq t \cdot n^{\mathcal{O}(1)}$ .

► **Lemma 4.2** ([3, 6, 15]). *Let  $q \geq 1$  be an integer, let  $L_1 \subseteq \{0, 1\}^*$  be an NP-hard problem, and let  $L_2 \subseteq \{0, 1\}^* \times \mathbb{N}$  be a parameterized problem. If there is a weak  $q$ -composition from  $L_1$  to  $L_2$ , then  $L_2$  has no compression of size  $\mathcal{O}(k^{q-\varepsilon})$  for any  $\varepsilon > 0$ , unless  $\text{coNP} \subseteq \text{NP/poly}$ .*

**Weak Composition.** We give a weak composition from the following problem:

MULTICOLORED INDEPENDENT SET

**Input:** A graph  $G$  and a partition  $(V_1, \dots, V_k)$  of  $V(G)$  into  $k$  cliques.

**Question:** Is there an independent set of size exactly  $k$ ?

A standard reduction from a restricted variant of 3-SAT (for instance, each literal appears exactly twice [1]) shows that MULTICOLORED INDEPENDENT SET is NP-hard even when  $\Delta_G \in \mathcal{O}(1)$  and  $|V_i| \in \mathcal{O}(1)$  for all  $i \in [k]$ . Let  $[t]^q$  be the set of  $q$ -dimensional vectors whose entries are in  $[t]$ . Suppose that  $q \geq 2$  is a constant and that we are given  $t^q$  instances  $\mathcal{I}_x = (G_x, (V_x^1, \dots, V_x^k))$  for  $x \in [t]^q$ , where  $\Delta_{G_x} \in \mathcal{O}(1)$  and  $|V_x^i| \in \mathcal{O}(1)$  for all  $x \in [t]^q$  and  $i \in [k]$ . We construct an INDEPENDENT SET instance  $(H, k')$ . The kernel lower bound of size  $k^{2-\varepsilon}$  will be based on the special case  $q = 2$ . To obtain the lower bound of  $c^{1-\varepsilon} k^{\mathcal{O}(1)}$ , however, we need the composition to work for all  $q \in \mathbb{N}$ . Hence, we give a generic description in the following. First, we construct a graph  $H_i$  as follows for every  $i \in [k]$ :

- For every  $x \in [t]^q$ , include  $V_x^i$  into  $V(H_i)$ .
- For every  $r \in [q]$ , introduce a path  $P_r^i$  on  $2t - 2$  vertices. We label the  $(2j - 1)$ -th vertex as  $p_{r,j,1}^i$  and the  $2j$ -th vertex as  $p_{r,j+1,2}^i$  (see Figure 1 for an illustration). Note that  $V(P_r^i) = \{p_{r,j,1}^i, p_{r,j+1,2}^i \mid j \in [t - 1]\}$ . For every  $j \in [t]$ , we now define the set  $P_{r,j}^i$ : let  $P_{r,1}^i = \{p_{r,1,1}^i\}$ ,  $P_{r,t}^i = \{p_{r,t,2}^i\}$ , and  $P_{r,j}^i = \{p_{r,j,1}^i, p_{r,j,2}^i\}$  for  $j \in [2, t - 2]$ .
- For every  $r \in [q]$  and  $j \in [t]$ , add edges such that  $P_{r,j}^i \cup \bigcup_{x \in [t]^q, x_r = j} V_x^i$  forms a clique (see Figure 1 for an illustration).

<sup>2</sup> Any  $c$ -closed  $n$ -vertex graph contains a clique or an independent set of size  $\Omega(\sqrt{n/c})$  [20].

## 35:12 Essentially Tight Kernels for (Weakly) Closed Graphs

Now, construct  $H$  by taking the disjoint union of the  $H_i$ ,  $i \in [k]$ , and adding the following:

- For every  $x \in [t]^q$ , add edges such that  $H[V(G_x)] = G_x$ .
- For every  $i \in [k-1]$ ,  $r \in [q]$ , and  $j \in [t-1]$ , add edges  $p_{r,j,1}^i p_{r,j+1,2}^{i+1}$  and  $p_{r,j,1}^{i+1} p_{r,j+1,2}^i$ .

This concludes the construction of  $H$ . Let  $k' := qkt - qk + k$ .

We call the vertices of  $\bigcup_{x \in [t]^q, i \in [k]} V_x^i$  the *instance vertices*. The other vertices, which are on  $P_r^i$  for some  $i \in [k]$  and  $r \in [q]$ , serve as *instance selectors*: As we shall see later, any independent set  $J$  of size  $k'$  in  $H$  contains exactly  $t-1$  vertices of  $P_r^i$  for every  $i \in [k]$  and  $r \in [q]$ . In fact, there is exactly one  $j \in [t]$  such that  $J \cap P_{r,j}^i = \emptyset$  and  $|J \cap P_{r,j'}^i| = 1$  for all  $j' \in [t] \setminus \{j\}$ . Consequently,  $J$  contains no instance vertex in  $V_x^i$  for  $x_r \neq j$ , and thereby,  $j$  is *selected* for the  $r$ -th dimension. We bound the  $c$ -closure of  $H$  and prove the correctness.

► **Lemma 4.3 (\*)**. *It holds that  $\text{cl}_H \in \mathcal{O}(t^{q-2})$ .*

► **Lemma 4.4 (\*)**. *The graph  $G_x$  has a multicolored independent set of size  $k$  for some  $x \in [t]^q$  if and only if the graph  $H$  has an independent set  $I$  of size  $k'$ .*

For  $q = 2$ , we have a weak 2-composition from MULTICOLORED INDEPENDENT SET to INDEPENDENT SET on  $\mathcal{O}(t^{q-2}) = \mathcal{O}(1)$ -closed graphs by Lemmas 4.3 and 4.4. Since the constructed graph  $H$  has no clique of size  $k'$ , the construction also constitutes a weak 2-composition to RAMSEY on  $\mathcal{O}(1)$ -closed graphs. Thus, Lemma 4.2 implies the following:

► **Theorem 4.5**. *For any  $\varepsilon > 0$ , neither INDEPENDENT SET nor RAMSEY has a kernel of size  $k^{2-\varepsilon}$  on graphs of constant  $c$ -closure, unless  $\text{coNP} \subseteq \text{NP/poly}$ .*

By Theorem 4.5, neither INDEPENDENT SET nor RAMSEY admit a kernel of  $k^{1-\varepsilon}$  vertices. We improve this bound on the number of vertices, taking advantage of the fact that any  $n$ -vertex  $c$ -closed graph can be encoded using  $\mathcal{O}(cn^{1.5} \log n)$  bits in polynomial time [12]. Assume for a contradiction that INDEPENDENT SET or RAMSEY admit a kernel of  $k^{4/3-\varepsilon'}$  vertices for constant  $c$ . Using the above-mentioned encoding, we obtain a string with  $\mathcal{O}(k^{(4/3-\varepsilon')1.5} \log k) = \mathcal{O}(k^{2-\varepsilon})$  bits. So a kernel of  $k^{4/3-\varepsilon'}$  vertices implies that there is a compression of INDEPENDENT SET or RAMSEY with bitsize  $\mathcal{O}(k^{2-\varepsilon})$ , a contradiction. Thus, we have the following:

► **Theorem 4.6**. *For any  $\varepsilon > 0$ , neither INDEPENDENT SET nor RAMSEY has a compression of  $k^{4/3-\varepsilon}$  vertices on graphs of constant  $c$ -closure, unless  $\text{coNP} \subseteq \text{NP/poly}$ .*

We also obtain another kernel lower bound for INDEPENDENT SET; this bound excludes the existence of polynomial kernels (in terms of  $c+k$ ) whose dependence on  $c$  is sublinear.

► **Theorem 4.7**. *For any  $\varepsilon > 0$ , INDEPENDENT SET has no kernel of size  $c^{1-\varepsilon} k^{\mathcal{O}(1)}$  unless  $\text{coNP} \subseteq \text{NP/poly}$ .*

**Proof.** We show that INDEPENDENT SET admits no kernel of size  $c^{1-\varepsilon} k^i$  for any  $\varepsilon, i > 0$ , unless  $\text{coNP} \subseteq \text{NP/poly}$ . Let  $q$  be a sufficiently large integer with  $\frac{q-\varepsilon-i}{q-2} > 1-\varepsilon$  (that is,  $q > \frac{i+3\varepsilon-2}{\varepsilon}$ ). Recall, that in the constructed instance we have  $c \in \mathcal{O}(t^{q-2})$  and  $k' := qkt - qk + k$ . A straightforward calculation shows that  $\ell := c^{\frac{1}{q-2}(1-\frac{i}{q-\varepsilon})} k'^{\frac{i}{q-\varepsilon}} \in \mathcal{O}(t)$ , and hence INDEPENDENT SET admits a weak  $q$ -decomposition for the parameterization  $\ell$ . Thus, Lemma 4.2 implies that there is no kernel of size  $\ell^{q-\varepsilon} = c^{\frac{q-\varepsilon-i}{q-2}} k'^i > c^{1-\varepsilon} k^i$ . ◀

Finally, we show that  $\mathcal{G}$ -SUBGRAPH has a kernel of size  $k^{\mathcal{O}(\gamma)}$  for any graph class  $\mathcal{G}$  containing all complete graphs and empty graphs.



► **Proposition 4.8.** *Any graph  $G$  on at least  $R_\gamma(a, b) \in (a \cdot b)^{\gamma + \mathcal{O}(1)}$  vertices has a clique of size  $a$  or an independent set of size  $b$ .*

**Proof.** The Ramsey number  $R(a, b)$  denotes the smallest number such that every graph on  $R(a, b)$  vertices contains a clique of size  $a$  or an independent set of size  $b$ . It is known that  $R(a, b) \leq \binom{a+b}{b}$ . So whenever  $a \leq \gamma$  or  $b \leq \gamma$ , we have  $R_\gamma(a, b) \leq \binom{a+b}{\gamma} \in \mathcal{O}((a+b)^\gamma)$ . For  $a, b > \gamma$ , let  $R_\gamma(a, b)$  be some number greater than  $ab \binom{b}{\gamma} + b \binom{a}{\gamma} \sum_{b' \in [b]} \binom{b'-1}{\gamma}$ .

Let  $n = |V(G)|$  and let  $v_1, \dots, v_n$  be a weak closure ordering  $\sigma$  of  $G$ . Divide  $V(G)$  into  $b$  subsets  $V_1, \dots, V_b$  of equal size<sup>3</sup>: let  $V_i = \{v_{((b-i)n/b)+1}, \dots, v_{(b-i+1)n/b}\}$  for each  $i \in [b]$ . Notably,  $V_1$  is the set of  $n/b$  vertices occurring last in  $\sigma$  and  $V_b$  is the set of  $n/b$  vertices occurring first in  $\sigma$ . Moreover, let  $G_i := G[\{v_{(b-i)n/b+1}, \dots, v_n\}]$  be the subgraph induced by  $\bigcup_{i' \in [i]} V_{i'}$  for each  $i \in [b]$ . Suppose that  $G$  contains no clique of size  $a$ . We show that  $G$  contains an independent set of size  $b$ . More precisely, we prove by induction that  $G_i$  contains an independent set of size  $i$  for each  $i \in [b]$ .

This clearly holds for  $i = 1$ . For  $i > 1$ , assume that there is an independent set  $I$  of size  $i - 1$  in  $G_{i-1}$  by the induction hypothesis. In the following, we consider subsets  $X$  of size at most  $\gamma$  of  $I$  to obtain an independent set  $I'$  of size at least  $i$ .

First, consider vertex sets  $X \subseteq I$  of size  $\gamma$  and  $V_X := \{v \in V_i \mid N_G(v) \supseteq X\}$ . Note that  $X \subseteq Q(v)$  for each  $v \in V_X$ . Hence, since  $G$  is weakly  $\gamma$ -closed,  $V_X$  is a clique. It follows that  $|V_X| < a$ . Therefore, less than  $a \binom{b}{\gamma}$  vertices of  $V_i$  are adjacent to at least  $\gamma$  vertices in  $I$ .

Second, consider vertex sets  $X \subseteq I$  with  $X \neq \emptyset$  and  $|X| < \gamma$ . Furthermore, let  $V'_X = \{v \in V_i \mid N_G(v) \cap I = X\}$ . Since  $n > R_\gamma(a, b)$ , there exists  $X \subseteq I$  of size at most  $\gamma - 1$  such that  $|V'_X| > R(a, \gamma)$ . By Ramsey's theorem, we then find an independent set  $I' \subseteq V'_X$  of size  $\gamma$  (recall that  $G$  has no clique of size  $a$ ). It follows that  $(I \setminus X) \cup I'$  is an independent set of size at least  $i$  in  $G_i$ . ◀

Now, we directly obtain to a kernel for  $\mathcal{G}$ -SUBGRAPH where  $\mathcal{G}$  contains all cliques and all independent sets since each graph on  $k^{\mathcal{O}(\gamma)}$  vertices contains either a clique or an independent set of size at least  $k$  by the bound on the Ramsey number in weakly closed graphs shown in Proposition 4.8.

► **Corollary 4.9.** *Let  $\mathcal{G}$  be a class of graphs containing all cliques and independent sets.  $\mathcal{G}$ -SUBGRAPH has a kernel of size  $k^{\mathcal{O}(\gamma)}$ .*

## 5 Conclusion

We have provided several kernelization algorithms and kernelization lower bounds for classic graph problems on (weakly) closed graphs. How far can our results for CONNECTED VERTEX COVER and CAPACITATED VERTEX COVER be extended to other cases of connected or capacitated vertex deletion problems? We did show that CONNECTED  $\ell$ -COC admits a kernel of size  $k^{\mathcal{O}(\gamma)}$ . In contrast, CONNECTED FEEDBACK VERTEX SET does not admit a polynomial kernel for the solution size  $k$  even in 2-closed graphs [5]. Drawing a borderline between those desired graph properties where connected and capacitated vertex deletion problems do admit a kernel on (weakly) closed graphs and those where they do not would improve our understanding when (weak) closure can be exploited algorithmically. It is also open whether the Ramsey number of weakly closed graphs can be bounded by  $(a + b + \gamma)^{\mathcal{O}(1)}$ , such a bound would immediately improve some of our kernels. The most important open

<sup>3</sup> For ease of presentation we assume that  $|V(G)|$  is divisible by  $b$ .



problem is arguably whether DOMINATING SET parameterized by the solution size  $k$  admits a polynomial kernel on weakly closed graphs. We made partial progress by showing that DOMINATING SET admits a kernel on weakly closed bipartite graphs and on weakly closed split graphs. Answering this question positively would need further insights into the structure of weakly closed graphs, however.

---

## References

---

- 1 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, 049, 2003.
- 2 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.
- 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 4 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *ACM Transactions on Algorithms*, 13(3):43:1–43:22, 2017.
- 5 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Kernelization hardness of connectivity problems in  $d$ -degenerate graphs. *Discrete Applied Mathematics*, 160(15):2131–2141, 2012.
- 6 Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 68–81. SIAM, 2012.
- 7 Michael Dom, Daniel Lokshantov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms*, 11(2):13:1–13:20, 2014.
- 8 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 9 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013.
- 10 Paul Erdős and András Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1-2):37–52, 1989.
- 11 Rok Erman, Łukasz Kowalik, Matjaž Krnc, and Tomasz Waleń. Improved induced matchings in sparse graphs. *Discrete Applied Mathematics*, 158(18):1994–2003, 2010.
- 12 Elaine M. Eschen, Chinh T. Hoàng, Jeremy P. Spinrad, and R. Sritharan. On graphs without a  $C_4$  or a diamond. *Discrete Applied Mathematics*, 159(7):581–587, 2011.
- 13 Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020.
- 14 Peter Frankl and Richard M. Wilson. Intersection theorems with geometric consequences. *Combinatorica*, 1(4):357–368, 1981. doi:10.1007/BF02579457.
- 15 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 104–113. SIAM, 2012.
- 16 Edin Husic and Tim Roughgarden. FPT algorithms for finding dense subgraphs in  $c$ -closed graphs. *CoRR*, abs/2007.09768, 2020. arXiv:2007.09768.
- 17 Iyad A. Kanj, Michael J. Pelsmajer, Marcus Schaefer, and Ge Xia. On the induced matching problem. *Journal of Computer and System Sciences*, 77(6):1058–1070, 2011.
- 18 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002.

- 19 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. In *Proceedings of the 31st International Symposium on Algorithms and Computation, (ISAAC '20)*, volume 181 of *LIPICs*, pages 20:1–20:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 20 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting  $c$ -closure in kernelization algorithms for graph problems. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA '20)*, volume 173 of *LIPICs*, pages 65:1–65:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 21 Stefan Kratsch. Co-nondeterminism in compositions: A kernelization lower bound for a Ramsey-type problem. *ACM Transactions on Algorithms*, 10(4):19:1–19:16, 2014.
- 22 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113, 2014.
- 23 Stefan Kratsch, Marcin Pilipczuk, Ashutosh Rai, and Venkatesh Raman. Kernel lower bounds using co-nondeterminism: Finding induced hereditary subgraphs. *ACM Transactions on Computation Theory*, 7(1):4:1–4:18, 2014.
- 24 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):11:1–11:23, 2012.



# Filling Crosswords Is Very Hard

Laurent Gourvès ✉

Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Ararat Harutyunyan ✉

Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Michael Lampis ✉ 

Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

Nikolaos Melissinos ✉ 

Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

---

## Abstract

We revisit a classical crossword filling puzzle which already appeared in Garey&Jonhson’s book. We are given a grid with  $n$  vertical and horizontal slots and a dictionary with  $m$  words and are asked to place words from the dictionary in the slots so that shared cells are consistent. We attempt to pinpoint the source of intractability of this problem by carefully taking into account the structure of the grid graph, which contains a vertex for each slot and an edge if two slots intersect. Our main approach is to consider the case where this graph has a tree-like structure. Unfortunately, if we impose the common rule that words cannot be reused, we discover that the problem remains NP-hard under very severe structural restrictions, namely, if the grid graph is a union of stars and the alphabet has size 2, or the grid graph is a matching (so the crossword is a collection of disjoint crosses) and the alphabet has size 3. The problem does become slightly more tractable if word reuse is allowed, as we obtain an  $m^{\text{tw}}$  algorithm in this case, where  $\text{tw}$  is the treewidth of the grid graph. However, even in this case, we show that our algorithm cannot be improved to obtain fixed-parameter tractability. More strongly, we show that under the ETH the problem cannot be solved in time  $m^{o(k)}$ , where  $k$  is the number of horizontal slots of the instance (which trivially bounds  $\text{tw}$ ).

Motivated by these mostly negative results, we also consider the much more restricted case where the problem is parameterized by the number of slots  $n$ . Here, we show that the problem does become FPT (if the alphabet has constant size), but the parameter dependence is exponential in  $n^2$ . We show that this dependence is also justified: the existence of an algorithm with running time  $2^{o(n^2)}$ , even for binary alphabet, would contradict the randomized ETH. Finally, we consider an optimization version of the problem, where we seek to place as many words on the grid as possible. Here it is easy to obtain a  $\frac{1}{2}$ -approximation, even on weighted instances, simply by considering only horizontal or only vertical slots. We show that this trivial algorithm is also likely to be optimal, as obtaining a better approximation ratio in polynomial time would contradict the Unique Games Conjecture. The latter two results apply whether word reuse is allowed or not.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis

**Keywords and phrases** Crossword Puzzle, Treewidth, ETH

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.36

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.11203>

**Acknowledgements** We thank Dominique Taton for communicating the puzzle to us.

## 1 Introduction

Crossword puzzles are one-player games where the goal is to fill a (traditionally two-dimensional) grid with words. Since their first appearance more than 100 years ago, crossword puzzles have rapidly become popular. Nowadays, they can be found in many newspapers



© Laurent Gourvès, Ararat Harutyunyan, Michael Lampis, and Nikolaos Melissinos; licensed under Creative Commons License CC-BY 4.0

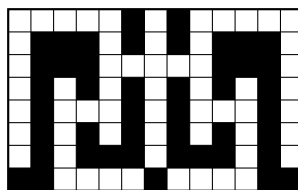
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 36; pp. 36:1–36:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Place valid words in this grid. In a possible instance, letters S, U, I, V, R, E, and T have weight 7, 5, 4, 2, 6, 1, and 3, respectively. Any other letter has null weight. Try to obtain at least 330 points.

and magazines around the world like the *New York Times* in the USA, or *Le Figaro* in France. Besides their obvious recreational interest, crossword puzzles are valued tools in education [2] and medicine. In particular, crossword puzzles participation seems to delay the onset of memory decline [14]. They are also helpful for developing and testing computational techniques; see for example [16]. In fact, both the design and the completion of a puzzle are challenging. In this article, we are interested in the task of solving a specific type of crossword puzzle.

There are different kinds of crossword puzzles. In the most famous ones, some clues are given together with the place where the answers should be located. A solution contains words that must be consistent with the given clues, and the intersecting pairs of words are constrained to agree on the letter they share. *Fill-in* crossword puzzles do not come with clues. Given a list of words and a grid in which some slots are identified, the objective is to fill all the slots with the given words. The list of words is typically succinct and provided explicitly.

In a variant of fill-in crossword puzzle currently proposed in a French TV magazine [12], one has to find up to 14 words and place them in a grid (the grid is the same for every instance, see Figure 1 for an illustration). The words are not explicitly listed but they must be *valid* (for instance, belong to the French language). In an instance of the game, some specified letters have a positive weight; the other letters have weight zero. The objective is to find a solution whose weight – defined as the total sum of the letters written in the grid – is at least a given threshold.

The present work deals with a theoretical study of this fill-in crossword puzzle (the grid is not limited to the one of Figure 1). We are mainly interested in two problems: Can the grid be entirely completed? How can the weight of a solution be maximized? Hereafter, these problems are called CROSSWORD PUZZLE DECISION and CROSSWORD PUZZLE OPTIMIZATION (CP-DEC and CP-OPT in short), respectively.

CP-DEC is not new; see GP14 in [5]. The proof of NP-completeness is credited to a personal communication with Lewis and Papadimitriou. Thereafter, an alternative NP-completeness proof appeared in [4] (see also [10]). Other articles on crossword puzzles exist and they are mostly empirically validated techniques coming from Artificial Intelligence and Machine Learning; see for example [6, 13, 11, 1, 16, 15] an references therein.

**Our Results.** Our goal in this paper is to pinpoint the relevant structural parameters that make filling crossword puzzles intractable. We begin by examining the structure of the given grid. It is natural to think that, if the structure of the grid is tree-like, then the problem should become easier, as the vast majority of problems are tractable on graphs of small treewidth. We only partially confirm this intuition: by taking into account the structure of a graph that encodes the intersections between slots (the grid-graph) we show in Section 3

that CP-OPT can be solved in polynomial time on instances of constant treewidth. However, our algorithm is not fixed-parameter tractable and, as we show, this cannot be avoided, even if one considers the much more restricted case where the problem is parameterized by the number of horizontal slots, which trivially bounds the grid-graph's treewidth (Theorem 4). More devastatingly, we show that if we also impose the natural rule that words cannot be reused, the problem already becomes NP-hard when the grid graph is a matching for alphabets of size 3 (Theorem 6), or a union of stars for a binary alphabet (Theorem 5). Hence, a tree-like structure does not seem to be of much help in rendering crosswords tractable.

We then go on to consider CP-OPT parameterized by the total number of slots  $n$ . This is arguably a very natural parameterization of the problem, as in real-life crosswords, the size of the grid can be expected to be significantly smaller than the size of the dictionary. We show that in this case the problem does become fixed-parameter tractable (Corollary 9), but the running time of our algorithm is exponential in  $n^2$ . Our main result is to show that this disappointing dependence is likely to be best possible: even for a binary alphabet, an algorithm solving CP-DEC in time  $2^{o(n^2)}$  would contradict the randomized ETH (Theorem 12). Note that all our positive results up to this point work for the more general CP-OPT, while our hardness results apply to CP-DEC.

Finally, in Section 5 we consider the approximability of CP-OPT. Here, it is easy to obtain a  $\frac{1}{2}$ -approximation by only considering horizontal or vertical slots. We are only able to slightly improve upon this, giving a polynomial-time algorithm with ratio  $\frac{1}{2} + O(\frac{1}{n})$ . Our main result in this direction is to show that this is essentially best possible: obtaining an algorithm with ratio  $\frac{1}{2} + \epsilon$  would falsify the Unique Games Conjecture (Theorem 15).

## 2 Problem Statement and Preliminaries

We are given a dictionary  $\mathcal{D} = \{d_1, \dots, d_m\}$  whose words are constructed on an alphabet  $\mathcal{L} = \{l_1, \dots, l_\ell\}$ , and a two-dimensional grid consisting of horizontal and vertical slots. A slot is composed of consecutive cells. Horizontal slots do not intersect each other; the same goes for vertical slots. However horizontal slots can intersect vertical slots. A cell is *shared* if it lies at the intersection of two slots. Unless specifically stated,  $n$ ,  $m$  and  $\ell$  denote the total number of slots, the size of  $\mathcal{D}$ , and the size of  $\mathcal{L}$ , respectively. Finally, let us mention that we consider only instances where the alphabet is of constant size, i.e.,  $\ell = O(1)$ .

In a feasible solution, each slot  $S$  receives either a word of  $\mathcal{D}$  of length  $|S|$ , or nothing (we sometimes say that a slot receiving nothing gets an *empty word*). Each cell gets at most one letter, and the words assigned to two intersecting slots must agree on the letter placed in the shared cell. All filled horizontal slots get words written from left to right (across) while all vertical slots get words written from top to bottom (down).

There is a weight function  $w : \mathcal{L} \rightarrow \mathbb{N}$ . The weight of a solution is the total sum of the weights of the letters placed in the grid. Observe that, for a given solution, the total weight of all filled-in words is not the same as the weight of this solution as, in the latter, the letters of the shared cells are counted only once.

The two main problems studied in this article are the following. Given a grid, a dictionary  $\mathcal{D}$  on alphabet  $\mathcal{L}$ , and a weight function  $w : \mathcal{L} \rightarrow \mathbb{N}$ , the objective of CROSSWORD PUZZLE OPTIMIZATION (CP-OPT in short) is to find a feasible solution of maximum weight. Given a grid and a dictionary  $\mathcal{D}$  on alphabet  $\mathcal{L}$ , the question posed by CROSSWORD PUZZLE DECISION (CP-DEC in short) is whether the grid can be completely filled or not?

Two cases will be considered: whether each word is used at most once, or if each word can be assigned multiple times. In this article, we will sometimes suppose that some cells are pre-filled with some elements of  $\mathcal{L}$ . In this case, a solution is feasible if it is consistent with the pre-filled cells. Below we propose a first result when all the shared cells are pre-filled.

► **Proposition 1.** *CP-DEC and CP-OPT can be solved in polynomial time if all the shared cells in the grid are pre-filled, whether word reuse is allowed or not.*

**Proof.** If word reuse is allowed, then for each combination of letters placed in these cells, we greedily fill out the rest of each slot with the maximum value word that can still be placed there. This is guaranteed to produce the optimal solution. On the other hand, if word reuse is not allowed, we construct a bipartite graph, with elements of  $\mathcal{D}$  on one side and the slots on the other, and place an edge between a word and a slot if the word can still be placed in the slot. If we give each edge weight equal to the value of its incident word reduced by the weight of the letters imposed by the shared cells of the slot, then an optimal solution corresponds to a maximum weight matching. ◀

One can associate a bipartite graph, hereafter called the *grid graph*, with each grid: each slot is a vertex and two vertices share an edge if the corresponding slots overlap. The grid (and then, the grid graph) is not necessarily connected.

Let us also note that so far we have been a bit vague about the encoding of the problem. Concretely, we could use a simple representation which lists for each slot the coordinates of its first cell, its size, and whether the slot is horizontal or vertical; and then supplies a list of all words in the dictionary and an encoding of the weight function. Such a representation would allow us to perform all the basic operations needed by our algorithms in polynomial time, such as deciding if it is possible to place a word  $d$  in a slot  $S$ , and which letter would then be placed in any particular cell of  $S$ . However, one drawback of this encoding is that its size may not be polynomially bounded in  $n + m$ , as some words may be exponentially long. We can work around this difficulty by using a more succinct representation: we are given the same information as above regarding the  $n$  slots; for each word we are given its total weight; and for each slot  $S$  and word  $d$ , we are told whether  $d$  fits exactly in  $S$ , and if yes, which letters are placed in the cells of  $S$  which are shared with other slots. Since the number of shared cells is  $O(n^2)$  this representation is polynomial in  $n + m$  and it is not hard to see that we are still able to perform any reasonable basic operation in polynomial time and that we can transform an instance given in the simple representation to this more succinct form. Hence, in the remainder, we will always assume that the size of the input is polynomially bounded in  $n + m$ .

We will rely on the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [8], which states the following:

► **Conjecture 2.** *Exponential Time Hypothesis: there exists an  $\epsilon > 0$ , such that 3-SAT on instances with  $n$  variables and  $m$  clauses cannot be solved in time  $2^{\epsilon(n+m)}$ .*

Note that it is common to use the slightly weaker formulation which states the ETH as the assumption that 3-SAT cannot be solved in time  $2^{o(n+m)}$ . This is known to imply that  $k$ -INDEPENDENT SET cannot be solved in time  $n^{o(k)}$ [3]. We use this fact in Theorem 4. In Section 4 we will rely on the randomized version of the ETH, which has the same statement as Conjecture 2 but for randomized algorithms with expected running time  $2^{\epsilon(n+m)}$ .

### 3 When the Grid Graph is Tree-like

In this section we are considering instances of CP-DEC and CP-OPT where the grid graph is similar to a tree. First, we give an algorithm for both problems in cases where the grid graph has bounded treewidth and we are allowed to reuse words and we show that this algorithm is essentially optimal. Then, we show that CP-DEC and CP-OPT are much harder to deal



with, in the case where we are not allowed to reuse words, by proving that the problems are NP-hard even for instances where the grid graph is just a matching. For the instances such that CP-DEC is NP-hard, we know that CP-OPT is NP-hard. That happens because we can assume that all the letters have weight equal to 1 hence a solution for CP-DEC is an optimal solution for CP-OPT.

### 3.1 Word Reuse

We propose a dynamic programming algorithm for CP-OPT and hence also for CP-DEC. Note that it can be extended to the case where some cells of the instance are pre-filled.

► **Theorem 3.** *If we allow word reuse, then CP-OPT can be solved in time  $(m+1)^{tw(n+m)}O(1)$  on inputs where  $tw$  is the treewidth of the grid graph.*

**Proof.** As the techniques we are going to use are standard we are sketching some details. For more details on tree decomposition (definition and terminology) see [3, Chap. 7]. Assuming that we have a rooted nice tree decomposition of the grid graph, we are going to perform dynamic programming on the nodes of this tree decomposition. For a node  $B_t$  of the given tree decomposition of the grid graph we denote by  $B_t^\downarrow$  the set of vertices of the grid graph that appears in the nodes of the subtree with  $B_t$  as a root. Since each vertex of the grid graph corresponds to a slot, we interchangeably mention a vertex of the grid graph and its corresponding slot. In particular, we say that a solution  $\sigma$  assigns words to the vertices of the grid graph, and  $\sigma(v)$  denotes the word assigned to  $v$ .

For each node  $B_t$  of the tree decomposition we are going to keep all the triplets  $(\sigma, W, W_t)$  such that:

- $\sigma$  is an assignment of words to the vertices of  $B_t$ ;
- $W$  is the weight of  $\sigma$  restricted to the vertices appearing in  $B_t$ ;
- and  $W_m$  is the maximum weight, restricted to the vertices appearing in  $B_t^\downarrow$ , of an assignment consistent with  $\sigma$ .

In order to create all the possible triplets for all the nodes of the tree decomposition we are going to explore the nodes from leaves to the root. Therefore, each time we visit a node we assume that we have already created the triplets for all its children. Let us explain how we deal with the different types of nodes.

In the Leaf nodes we have no vertices so we keep an empty assignment ( $\sigma$  does not assign any word) and the weights  $W$  and  $W_m$  are equal to 0.

For an Introduce node  $B_t$  we need to take in consideration its child node. Assume that  $u$  is the introduced vertex; for each triplet  $(\sigma, W, W_m)$  of the child node we are going to create all the triplets  $(\sigma', W', W'_m)$  for the new node as follows. First we find all the words  $d \in \mathcal{D}$  that fit in the corresponding slot of  $u$  and respect the assignment  $\sigma$  (i.e., if there are cells that are already filled under  $\sigma$  and  $d$  uses these cells then it must have the same letters). We create one triplet  $(\sigma', W', W'_m)$  for each such a  $d$  as follows:

- We set  $\sigma'(u) := d$  and  $\sigma'(v) := \sigma(v)$  for all  $v \in B_t \setminus \{u\}$ .
- We can easily calculate the total weight,  $W'$ , of the words in  $B_t$  where the shared letters are counted only once under the assignment  $\sigma'$ .
- For the maximum weight  $W'_m$  we know that it is increased by the same amount as  $W$ ; so we set  $W'_m = W_m + W' - W$ .

Observe that we do not need to consider the intersection with slots whose vertices appear in  $B_t^\downarrow \setminus B_t$  as each node of a tree decomposition is a cut set.

Finally, we need to take in consideration that we can leave a slot empty. For this case we create a new word  $d_*$  which, we assume that, fits in all slots and  $d_*$  has weight 0. Because the empty word has weight 0,  $W'$  and  $W'_m$  are identical to  $W$  and  $W_m$  so for each triplet of

the child node, we only need to extend  $\sigma$  by assigning  $d_*$  to  $u$ . In the case we assign the empty word somewhere we will consider that the cells of this slot are empty unless another word  $d \neq d_*$  uses them.

For the Forget nodes we need to restrict the assignments of the child node to the vertex set of the Forget node, as it has been reduced by one vertex (the forgotten vertex), and reduce the weight  $W$  (which we can calculate easily). The maximum weight is not changed by the deletion.

However, if we restrict the assignments we may end up with several triplets  $(\sigma, W, W_m)$  with identical assignments  $\sigma$ . In that case we are keeping only the triplet with maximum  $W_m$ . Observe that we are allowed to keep only triplets with the maximum  $W_m$  because each node of a tree decomposition is a cut set so the same holds for the Forget nodes. Specifically, the vertices that appear in the nodes higher than a Forget node  $B_t$  of the tree decomposition do not have edges incident to vertices in  $B_t^{\downarrow} \setminus B_t$  so we only care for the assignment in  $B_t$ .

Finally, we need to consider the Join nodes. Each Join node has exactly two children. For each possible assignment  $\sigma$  on the vertices of this Join node, we create a triplet iff this  $\sigma$  appears in a triplet of both children of the Join node.

Because  $W$  is related only to the assignment  $\sigma$ , it is easy to see that it will be the same as in the children of the Join node. So we need to find the maximum weight  $W_m$ . Observe that between the vertices that appear in the subtrees of two children of a Join node there are no edges except those incident to the vertices of the Join node. Therefore, we can calculate the maximum weight  $W_m$  as follows: first we consider the maximum weight of each child of the Join node reduced by  $W$ , we add all these weights and, in the end, we add again the  $W$ . It is easy to see that this way we consider the weight of the cells appearing in each subtree without those of the slots of the Join node and we add the weight of the words assigned to the vertices of the Join node in the end.

For the running time we need to observe that the number of nodes of a nice tree decomposition is  $O(\text{tw} \cdot n)$  and all the other calculations are polynomial in  $n + m$  so we only need to consider the different assignments for each node. Because for each vertex we have  $|\mathcal{D}| + 1$  choices, the number of different assignments for a node is at most  $(|\mathcal{D}| + 1)^{\text{tw}+1}$ . ◀

It seems that the algorithm we propose for CP-DEC is essentially optimal, even if we consider a much more restricted case.

► **Theorem 4.** *CP-DEC with word reuse is  $W[1]$ -hard parameterized by the number of horizontal slots of the grid, even for alphabets with two letters. Furthermore, under the ETH, no algorithm can solve this problem in time  $m^{o(k)}$ , where  $k$  is the number of horizontal slots.*

**Proof.** We perform a reduction from  $k$ -INDEPENDENT SET, where we are given a graph  $G = (V, E)$  with  $|V|$  vertices and  $|E|$  edges and are looking for an independent set of size  $k$ . This problem is well-known to be  $W[1]$ -hard and not solvable in  $|V|^{o(k)}$  time under the ETH [3]. We assume without loss of generality that  $|E| \neq k$ . Furthermore, we can safely assume that  $G$  has no isolated vertices.

We first describe the grid of our construction which fits within an area of  $2k - 1$  lines and  $2|E| - 1$  columns. We construct:

1.  $k$  horizontal slots, each of length  $2|E| - 1$  (so each of these slots is as long horizontally as the whole grid). We place these slots in the unique way so that no two of these slots are in consecutive lines. We number these horizontal slots  $1, \dots, k$  from top to bottom.
2.  $|E|$  vertical slots, each of length  $2k - 1$  (so each of these slots is long enough to cover the grid top to bottom). We place these slots in the unique way so that no two of them are in consecutive columns. We number them  $1, \dots, |E|$  from left to right.

Before we describe the dictionary, let us give some intuition about the grid. The main idea is that in the  $k$  horizontal slots we will place  $k$  words that signify which vertices are selected from the original graph. Each vertical slot represents an edge of  $E$ , and we will be able to place a word in it if and only if we have not placed words representing two of its endpoints in the horizontal slots.

Our alphabet has two letters, say 0, 1. In the remainder, we assume that the edges of the original graph are numbered, that is,  $E = \{e_1, \dots, e_{|E|}\}$ . The dictionary is as follows:

1. For each vertex  $v$  we construct a word of length  $2|E| - 1$ . For each  $i \in \{1, \dots, |E|\}$ , if the edge  $e_i$  is incident on  $v$ , then the letter at position  $2i - 1$  of the word representing  $v$  is 1. All other letters of the word representing  $v$  are 0. Observe that this means that if  $e_i$  is incident on  $v$  and we place the word representing  $v$  on a horizontal slot, the letter  $i$  will appear on the  $i$ -th vertical slot. Furthermore, the word representing  $v$  has a number of 1s equal to the degree of  $v$ .
2. We construct  $k + 1$  words of length  $2k - 1$ . One of them is simply  $0^{2k-1}$ . The remaining are  $0^{2j-2}10^{2k-2j}$ , for  $j \in \{1, \dots, k\}$ , that is, the words formed by placing a 1 in an odd-numbered position and 0s everywhere else. Observe that if we place one of these  $k$  words on a vertical slot, a 1 will be placed on exactly one horizontal slot.

This completes the construction. We now observe that the  $k$  horizontal slots correspond to a vertex cover of the grid-graph. Therefore, if the reduction preserves the answer, the hardness results for  $k$ -INDEPENDENT SET transfer to our problem, since we preserve the value of the parameter.

We claim that if there exists an independent set of size  $k$  in  $G$ , then it is possible to fill the grid. Indeed, take such a set  $S$  and for each  $v \in S$  we place the word representing  $v$  in a horizontal slot. Consider the  $i$ -th vertical slot. We will place in this slot one of the  $k + 1$  words of length  $2k - 1$ . We claim that the vertical slot at this moment contains the letter 1 at most once, and if 1 appears it must be at an odd position (since these are the positions shared with the horizontal slots). If this is true, clearly there is a word we can place. To see that the claim is true, recall that since  $S$  is an independent set of  $k$  distinct vertices, there exists at most one vertex in  $S$  incident on  $e_i$ .

For the converse direction, recall that  $|E| \neq k$ . This implies that if there is a way to fill out the whole grid, then words representing vertices must go into horizontal slots and words of length  $2k - 1$  must go into vertical slots. By looking at the words that have been placed in the horizontal slots we obtain a collection of  $k$  (not necessarily distinct) vertices of  $G$ . We will prove that these vertices must actually be an independent set of size exactly  $k$ . To see this, consider the  $i$ -th vertical slot. If our collection of vertices contained two vertices incident on  $e_i$ , it would have been impossible to fill out the  $i$ -th vertical slot, since we would need a word with two 1s. Observe that the same argument rules out the possibility that our collection contains the same vertex  $v$  twice, as the column corresponding to any edge  $e_i$  incident on  $v$  would have been impossible to fill. ◀

### 3.2 No Word Reuse

If a word cannot be reused, then CP-DEC looks more challenging. Indeed, in the following theorem we prove that if reusing words is not allowed, then the problem becomes NP-hard even if the grid graph is acyclic and the alphabet size is 2. (Note that if the alphabet size is 1, the problem is trivial, independent of the structure of the graph).

► **Theorem 5.** *CP-DEC is NP-hard, even for instances where all of the following restrictions apply: (i) the grid graph is a union of stars (ii) the alphabet contains only two letters (iii) words cannot be reused.*

**Proof.** We show a reduction from 3-PARTITION. Recall that in 3-PARTITION we are given a collection of  $3n$  distinct positive integers  $x_1, \dots, x_{3n}$  and are asked if it is possible to partition these integers into  $n$  sets of three integers (triples), such that all triples have the same sum. This problem has long been known to be strongly NP-hard [5] and NP-hardness when the integers are distinct was shown by Hulett et al. [7]. We can assume that  $\sum_{i=1}^{3n} x_i = nB$  and that if a partition exists each triple has sum  $B$ . Furthermore, we can assume without loss of generality that  $x_i > 6n$  for all  $i \in \{1, \dots, 3n\}$  (otherwise, we can simply add  $6n$  to all numbers and adjust  $B$  accordingly without changing the answer).

Given an instance of 3-PARTITION as above, we construct a crossword instance as follows. First, the alphabet only contains two letters, say the letters  $*$  and  $!$ . To construct our dictionary we do the following:

1. For each  $i \in \{1, \dots, 3n\}$ , we add to the dictionary one word of length  $x_i$  that begins with  $!$  and  $n - 1$  words of length  $x_i$  that begin with  $*$ . The remaining letters of these words are chosen in an arbitrary way so that all words remain distinct.
2. For each  $i, j, k \in \{1, \dots, 3n\}$  with  $i < j < k$  we check if  $x_i + x_j + x_k = B$ . If this is the case, we add to the dictionary the word  $*^{2i-2}!_*^{2j-2i-1}!_*^{2k-2j-1}!_*^{6n-2k}$ . In other words, we constructed a word that has  $*$  everywhere except in positions  $2i - 1, 2j - 1$ , and  $2k - 1$ . The length of this word is  $6n - 1$ . Let  $f$  be the number of words added to the dictionary in this step. We have  $f \leq \binom{3n}{3} = O(n^3)$ .

We now also need to specify our grid. We first construct  $f$  horizontal slots, each of length  $6n - 1$ . Among these  $f$  slots, we select  $n$ , which we call the “interesting” horizontal slots. For each interesting horizontal slot, we construct  $3n$  vertical slots, such that the  $i$ -th of these slots has length  $x_i$  and its first cell is the cell in position  $2i - 1$  of the interesting horizontal slot. This completes the construction, which can clearly be carried out in polynomial time. Observe that the first two promised restrictions are satisfied as we have an alphabet with two letters and each vertical slot intersects at most one horizontal slot (so the grid graph is a union of stars).

We claim that if there exists a partition of the original instance, then we can place all the words of the dictionary on the grid. Indeed, for each  $i, j, k \in \{1, \dots, 3n\}$  such that  $\{x_i, x_j, x_k\}$  is one of the triples of the partition, we have constructed a word of length  $6n - 1$  corresponding to the triple  $(i, j, k)$ , because  $x_i + x_j + x_k = B$ . We place each of these  $n$  words on an interesting horizontal slot and we place the remaining words of length  $6n - 1$  on the non-interesting horizontal slots. Now, for every  $i \in \{1, \dots, 3n\}$  we have constructed  $n$  words, one starting with  $!$  and  $n - 1$  starting with  $*$ . We observe that among the interesting horizontal slots, there is one that contains the letter  $!$  at position  $2i - 1$  (the one corresponding to the triple containing  $x_i$  in the partition) and  $n - 1$  containing the letter  $*$  at position  $2i - 1$ . By construction, the vertical slots that begin in these positions have length  $x_i$ . Therefore, we can place all  $n$  words corresponding to  $x_i$  on these vertical slots. Proceeding in this way we fill the whole grid, fulfilling the third condition.

For the converse direction, suppose that there is a way to fill the whole grid. Then, vertical slots must contain words that were constructed in the second step and represent integers  $x_i$ , while horizontal slots must contain words constructed in the first step (this is a consequence of the fact that  $x_i > 6n$  for all  $i \in \{1, \dots, 3n\}$ ). We consider the  $n$  interesting horizontal slots. Each such slot contains a word that represents a triple  $(i, j, k)$  with  $x_i + x_j + x_k = B$ .

We therefore collect these  $n$  triples and attempt to construct a partition from them. To do this, we must prove that each  $x_i$  must belong to exactly one of these triples. However, recall that we have exactly  $n$  words of length  $x_i$  (since all integers of our instance are distinct) and exactly  $n$  vertical slots of this length. We conclude that exactly one vertical slot must have  $!$  as its first letter, therefore  $x_i$  appears in exactly one triple and we have a proper partition. ◀

Actually, the problem remains NP-hard even in the case where the grid graph is a matching and the alphabet contains three letters. This is proved for grid graphs composed of  $\mathcal{T}$ s, where a  $\mathcal{T}$  is a horizontal slot solely intersected by the first cell of a vertical slot.

► **Theorem 6.** *CP-DEC is NP-hard, even for instances where all of the following restrictions apply: (i) each word can be used only once (ii) the grid is consisted only by  $\mathcal{T}$ s and (iii) the alphabet contains only three letters.*

► **Remark 7.** Theorem 4 can be adjusted to work also for the case where word reuse is not allowed. We simply need to add a suffix of length  $\log m$  to all words of length  $2k - 1$  and add rows to the grid accordingly. Hence, under the ETH, no algorithm can solve this problem in time  $m^{o(k)}$ , where  $k$  is the number of horizontal slots.

Finally, observe that by filling the slots of a vertex cover of the grid graph, all the shared cells are pre-filled. Since there are at most  $m^k$  (where  $k$  is the size of the vertex cover) ways to assign words to these slots, by Proposition 1, we get the following corollary.

► **Corollary 8.** *Given a vertex cover of size  $k$  of the grid graph we can solve CP-DEC and CP-OPT in time  $m^k(n + m)^{O(1)}$ . Furthermore, as vertex cover we can take the set of horizontal slots.*

Therefore, the bound given in Remark 7 for the parameter vertex cover is tight.

## 4 Parameterized by Total Number of Slots

In this section we consider a much more restrictive parameterization of the problem: we consider instances where the parameter is  $n$ , the total number of slots. Recall that in Theorem 4 (and Remark 7) we already considered the complexity of the problem parameterized by the number of *horizontal* slots of the instance. We showed that this case of the problem cannot be solved in  $m^{o(k)}$  and that an algorithm with running time roughly  $m^k$  is possible whether word reuse is allowed or not.

Since parameterizing by the number of horizontal slots is not sufficient to render the problem FPT, we therefore consider our parameter to be the total number of slots. This is, finally, sufficient to obtain a simple FPT algorithm.

► **Corollary 9.** *There is an algorithm that solves CP-DEC and CP-OPT in time  $O^*(\ell^{n^2/4})$ , where  $n$  is the total number of slots and  $\ell$  the size of the alphabet, whether word reuse is allowed or not.*

Even though the running time guaranteed by Corollary 9 is FPT for parameter  $n$ , we cannot help but observe that the dependence on  $n$  is rather disappointing, as our algorithm is exponential *in the square* of  $n$ . It is therefore a natural question whether an FPT algorithm for this problem can achieve complexity  $2^{o(n^2)}$ , assuming the alphabet size is bounded. The main result of this section is to establish that this is likely to be impossible.

## 36:10 Filling Crosswords Is Very Hard

**Overview.** Our hardness proof consists of two steps. In the first step we reduce 3-SAT to a version of the same problem where variables and clauses are partitioned into  $O(\sqrt{n+m})$  groups, which we call SPARSE 3-SAT. The key property of this intermediate problem is that interactions between groups of variables and groups of clauses are extremely limited. In particular, for each group of variables  $V_i$  and each group of clauses  $C_j$ , at most one variable of  $V_i$  appears in a clause of  $C_j$ . We obtain this rather severe restriction via a randomized reduction that runs in expected polynomial time. The second step is to reduce SPARSE 3-SAT to CP-DEC. Here, every horizontal slot will represent a group of variables and every vertical slot a group of clauses, giving  $O(\sqrt{n+m})$  slots in total. Hence, an algorithm for CP-DEC whose dependence on the total number of slots is subquadratic in the exponent will imply a sub-exponential time (randomized) algorithm for 3-SAT. The limited interactions between groups of clauses and variables will be key in allowing us to execute this reduction using a *binary* alphabet.

Let us now define our intermediate problem.

► **Definition 10.** *In SPARSE 3-SAT we are given an integer  $n$  which is a perfect square and a 3-SAT formula  $\phi$  with at most  $n$  variables and at most  $n$  clauses, such that each variable appears in at most 3 clauses. Furthermore, we are given a partition of the set of variables  $V$  and the set of clauses  $C$  into  $\sqrt{n}$  sets  $V_1, \dots, V_{\sqrt{n}}$  and  $C_1, \dots, C_{\sqrt{n}}$  of size at most  $\sqrt{n}$  each, such that for all  $i, j \in [\sqrt{n}]$  the number of variables of  $V_i$  which appear in at least one clause of  $C_j$  is at most one.*

Now, we are going to prove the hardness of SPARSE 3-SAT, which is the first step of our reduction.

► **Lemma 11.** *Suppose the randomized ETH is true. Then, there exists an  $\epsilon > 0$  such that SPARSE 3-SAT cannot be solved in time  $2^{\epsilon n}$ .*

We are now ready to prove the main theorem of this section.

► **Theorem 12.** *Suppose the randomized ETH is true. Then, there exists an  $\epsilon > 0$  such that CP-DEC on instances with a binary alphabet cannot be solved in time  $2^{\epsilon n^2} \cdot m^{O(1)}$ . This holds also for instances where all slots have distinct sizes (so words cannot be reused).*

**Proof.** Suppose for the sake of contradiction that for any fixed  $\epsilon > 0$ , CP-DEC on instances with a binary alphabet can be solved in time  $2^{\epsilon n^2} \cdot m^{O(1)}$ . We will then contradict Lemma 11. In particular, we will show that for any  $\epsilon'$  we can solve SPARSE 3-SAT in time  $2^{\epsilon' N}$ , where  $N$  is the upper bound on the number of variables and clauses. Fix some  $\epsilon' > 0$  and suppose that  $\phi$  is an instance of SPARSE 3-SAT with at most  $N$  variables and at most  $N$  clauses, where  $N$  is a perfect square. Recall that the variables are given partitioned into  $\sqrt{N}$  sets,  $V_1, \dots, V_{\sqrt{N}}$  and the clauses partitioned into  $\sqrt{N}$  sets  $C_1, \dots, C_{\sqrt{N}}$ . In the remainder, when we write  $V(C_j)$  we will denote the set of variables that appear in a clause of  $C_j$ . Recall that the partition satisfies the property that for all  $i, j \in [\sqrt{N}]$  we have  $|V_i \cap V(C_j)| \leq 1$ . Suppose that the variables of  $\phi$  are ordered  $x_1, x_2, \dots, x_N$ .

We construct a grid as follows: for each group  $V_i$  we construct a horizontal slot and for each group  $C_j$  we construct a vertical slot, in a way that all slots have distinct lengths. More precisely, the  $i$ -th horizontal slot, for  $i \in [\sqrt{N}]$  is placed on row  $2i - 1$ , starts in the first column and has length  $2\sqrt{N} + 2i$ . The  $j$ -th vertical slot is placed in column  $2j - 1$ , starts in the first row and has length  $5\sqrt{N} + 2j$ . (As usual, we number the rows and columns top-to-bottom and left-to-right). Observe that all horizontal slots intersect all vertical slots; in particular, the cell in row  $2i - 1$  and column  $2j - 1$  is shared between the  $i$ -th horizontal and  $j$ -th vertical slot, for  $i, j \in [\sqrt{N}]$ . We define  $\mathcal{L}$  to contain two letters  $\{0, 1\}$ .



What remains is to describe the dictionary.

- For each  $i \in [\sqrt{N}]$  and for each assignment function  $\sigma : V_i \rightarrow \{0, 1\}$  we construct a word  $w_\sigma$  of length  $2\sqrt{N} + 2i$ . The word  $w_\sigma$  has the letter 0 in all positions, except positions  $2j - 1$ , for  $j \in [\sqrt{N}]$ . For each such  $j$ , we consider  $\sigma$  restricted to  $V_i \cap V(C_j)$ . By the properties of SPARSE 3-SAT, we have  $|V_i \cap V(C_j)| \leq 1$ . If  $V_i \cap V(C_j) = \emptyset$  then we place letter 0 in position  $2j - 1$ ; otherwise we set in position  $2j - 1$  the letter that corresponds to the value assigned by  $\sigma$  to the unique variable of  $V_i \cap V(C_j)$ .
- For each  $j \in [\sqrt{N}]$  and for each *satisfying* assignment function  $\sigma : V(C_j) \rightarrow \{0, 1\}$ , that is, every assignment function that satisfies all clauses of  $C_j$ , we construct a word  $w'_\sigma$  of length  $5\sqrt{N} + 2j$ . The word  $w'_\sigma$  has the letter 0 in all positions, except positions  $2i - 1$ , for  $i \in [\sqrt{N}]$ . For each such  $i$ , we consider  $\sigma$  restricted to  $V_i \cap V(C_j)$ . If  $V_i \cap V(C_j) = \emptyset$  then we place letter 0 in position  $2i - 1$ ; otherwise we set in position  $2i - 1$  the letter that corresponds to the value assigned by  $\sigma$  to the unique variable of  $V_i \cap V(C_j)$ .

The construction is now complete. We claim that if  $\phi$  is satisfiable, then it is possible to fill out the grid we have constructed. Indeed, fix a satisfying assignment  $\sigma$  to the variables of  $\phi$ . For each  $i \in [\sqrt{N}]$  let  $\sigma_i$  be the restriction of  $\sigma$  to  $V_i$ . We place in the  $i$ -th horizontal slot the word  $w_{\sigma_i}$ . Similarly, for each  $j \in [\sqrt{N}]$  we let  $\sigma'_j$  be the restriction of  $\sigma$  to  $V(C_j)$  and place  $w'_{\sigma'_j}$  in the  $j$ -th vertical slot. Now if we examine the cell shared by the  $i$ -th horizontal and  $j$ -th vertical slot, we can see that it contains a letter that represents  $\sigma$  restricted to (the unique variable of)  $V_i \cap V(C_j)$  or 0 if  $V_i \cap V(C_j) = \emptyset$ , and both the horizontal and vertical word place the same letter in that cell.

For the converse direction, if the grid is filled, we can extract an assignment  $\sigma$  for the variables of  $\phi$  as follows: for each  $x \in V_i$  we find a  $C_j$  such that  $x$  appears in some clause of  $C_j$  (we can assume that every variable appears in some clause). We then look at the cell shared between the  $i$ -th horizontal and the  $j$ -th vertical slot. The letter we have placed in that cell gives an assignment for the variable contained  $V_i \cap V(C_j)$ , that is  $x$ . Having extracted an assignment to all the variables, we claim it must satisfy  $\phi$ . If not, there is a group  $C_j$  that contains an unsatisfied clause. Nevertheless, in the  $j$ -th vertical slot we have placed a word that corresponds to a *satisfying* assignment for the clauses of  $C_j$ , call it  $\sigma_j$ . Then  $\sigma_j$  must disagree with  $\sigma$  in a variable  $x$  that appears in  $C_j$ . Suppose this variable is part of  $V_i$ . Then, this would contradict the fact that we extracted an assignment for  $x$  from the word placed in the  $i$ -th horizontal slot.

Observe that the new instance has  $n = 2\sqrt{N}$  slots. If there exists an algorithm that solves CP-DEC in time  $2^{\epsilon n^2} m^{O(1)}$  for any  $\epsilon > 0$ , we set  $\epsilon = \epsilon'/8$  (so  $\epsilon$  only depends on  $\epsilon'$ ) and execute this algorithm on the constructed instance. We observe that  $m \leq 2\sqrt{N} \cdot 7^{\sqrt{N}}$ , and that  $2^{\epsilon n^2} \leq 2^{\epsilon' N/2}$ . Assuming that  $N$  is sufficiently large, using the supposed algorithm for CP-DEC we obtain an algorithm for SPARSE 3-SAT with complexity at most  $2^{\epsilon' N}$ . Since we can do this for arbitrary  $\epsilon'$ , this contradicts the randomized ETH. ◀

## 5 Approximability of CP-Opt

This section begins with a  $(\frac{1}{2} + O(\frac{1}{n}))$ -approximation algorithm which works when words can, or cannot, be reused. After that, we prove that under the unique games conjecture, an approximation algorithm with a significantly better ratio is unlikely.

► **Theorem 13.** *CP-OPT is  $(\frac{1}{2} + \frac{1}{2(\epsilon n + 1)})$ -approximable in polynomial time, for all  $\epsilon \in (0, 1]$ .*



## 36:12 Filling Crosswords Is Very Hard

**Proof.** Fix some  $\varepsilon \in (0, 1]$ . Let  $k_v := \min(\lceil \frac{1}{\varepsilon} \rceil, n - h)$  and  $r_v := \lceil \frac{n-h}{k_v} \rceil$ , where  $h$  is the number of horizontal slots in the grid. Create  $r_v$  groups of vertical slots  $G_1, \dots, G_{r_v}$  such that  $|G_i| \leq k_v$  for all  $i \in [r_v]$  and  $G_1 \cup \dots \cup G_{r_v}$  covers the entire set of vertical slots. For each  $G_i$ , guess an optimal choice of words, i.e., identical to a global optimum, and complete this partial solution by filling the horizontal slots (use the aforementioned matching technique where the words selected for  $G_i$  are excluded from  $\mathcal{D}$ ). Each slot of  $\bigcup_{j \neq i} G_j$  gets the empty word.

Since  $|G_i| \leq k_v$ , guessing an optimal choice of words for  $G_i$  by brute force requires at most  $m^{k_v}$  combinations. This is done  $r_v$  times (once for each  $G_i$ ). The maximum matching runs in time  $O((m+n)^2 \cdot mn)$ . In all, the time complexity of the algorithm is  $O(m^{k_v} \cdot r_v \cdot (m+n)^2 \cdot mn) \leq O(m^{1/\varepsilon} \cdot \varepsilon n \cdot (m+n)^2 \cdot mn)$ .

Assume that, given an optimal solution,  $W_H^*$  and  $W_V^*$  are the total weight of the words assigned to the horizontal and vertical slots, respectively, both including the shared cells. Furthermore, let  $W_S^*$  be the weight of the letters assigned to the shared cells in the optimal solution. Observe that the weight of the optimal solution is  $W_H^* + W_V^* - W_S^*$  and the weight of our solution is at least  $W_H^* + \frac{1}{r_v}(W_V^* - W_S^*)$ .

We repeat the same process, but the roles of vertical and horizontal slots are interchanged. Fix a parameter  $k_h := \min(\lceil \frac{1}{\varepsilon} \rceil, h)$ . Create  $r_h := \lceil \frac{h}{k_h} \rceil$  groups of horizontal slots  $G_1, \dots, G_{r_h}$  such that  $|G_i| \leq k_h$  for all  $i \in [r_h]$  and  $G_1 \cup \dots \cup G_{r_h}$  covers the entire set of horizontal slots. For each  $G_i$ , guess an optimal choice of words and complete this partial solution by filling the vertical slots. Each slot of  $\bigcup_{j \neq i} G_j$  gets the empty word.

Using the same arguments as above, we can conclude that the time complexity is  $O(m^{1/\varepsilon} \cdot \varepsilon n \cdot (m+n)^2 \cdot mn)$  and that we return a solution of weight at least  $W_V^* + \frac{1}{r_h}(W_H^* - W_S^*)$ .

Finally, between the two solutions, we return the one with the greater weight. It remains to argue about the approximation ratio. We need to consider two cases:  $W_H^* \geq W_V^*$  and  $W_V^* > W_H^*$ .

Suppose  $W_H^* \geq W_V^*$ . The first approximate solution has value  $W_H^* + \frac{1}{r_v}(W_V^* - W_S^*) \geq \frac{1+1/r_v}{2}(W_H^* + W_V^* - W_S^*)$ . If  $k_v = n - h$  then  $r_v = 1$  and our approximation ratio is 1. Otherwise,  $k_v = \lceil \frac{1}{\varepsilon} \rceil$  and  $r_v = \lceil \frac{n-h}{\lceil 1/\varepsilon \rceil} \rceil \leq \frac{n-h}{\lceil 1/\varepsilon \rceil} + 1 = \frac{n-h + \lceil 1/\varepsilon \rceil}{\lceil 1/\varepsilon \rceil}$ . It follows that  $\frac{1}{r_v} \geq \frac{\lceil 1/\varepsilon \rceil}{n-h + \lceil 1/\varepsilon \rceil}$ . Use  $n - h + \lceil 1/\varepsilon \rceil \leq n + \frac{1}{\varepsilon}$  and  $\lceil 1/\varepsilon \rceil \geq 1/\varepsilon$  to get that  $\frac{1}{r_v} \geq \frac{1/\varepsilon}{n+1/\varepsilon} = \frac{1}{\varepsilon n + 1}$ . Our approximation ratio is at least  $\frac{1+1/(\varepsilon n + 1)}{2}$ .

Suppose  $W_V^* > W_H^*$ . The second approximate solution has value  $W_V^* + \frac{1}{r_h}(W_H^* - W_S^*) > \frac{1+1/r_h}{2}(W_H^* + W_V^* - W_S^*)$ . If  $k_h = h$ , then our approximation ratio is 1. Otherwise,  $k_h = \lceil \frac{1}{\varepsilon} \rceil$  and, using the same arguments, our approximation ratio is at least  $\frac{1+1/(\varepsilon n + 1)}{2}$ .

Note that  $\frac{1+1/(\varepsilon n + 1)}{2} \leq 1$ . In all, we have a  $\frac{1+1/(\varepsilon n + 1)}{2}$ -approximate solution in  $O(m^{1/\varepsilon} \cdot \varepsilon n \cdot (m+n)^2 \cdot mn)$  for all  $\varepsilon \in (0, 1]$ . ◀

The previous approximation algorithm only achieves an approximation ratio of  $\frac{1}{2} + O(\frac{1}{n})$ , which tends to  $\frac{1}{2}$  as  $n$  increases. At first glance this is quite disappointing, as someone can observe that a ratio of  $\frac{1}{2}$  is achievable simply by placing words only on the horizontal or the vertical slots of the instance. Nevertheless, we are going to show that this performance is justified, as improving upon this trivial approximation ratio would falsify the Unique Games Conjecture (UGC).

Before we proceed, let us recall some relevant definitions regarding Unique Games. The UNIQUE LABEL COVER problem is defined as follows: we are given a graph  $G = (V, E)$ , with some arbitrary total ordering  $\prec$  of  $V$ , an integer  $R$ , and for each  $(u, v) \in E$  with  $u \prec v$  a 1-to-1 constraint  $\pi_{(u,v)}$  which can be seen as a permutation on  $[R]$ . The vertices of  $G$  are

considered as variables of a constraint satisfaction problem, which take values in  $[R]$ . Each constraint  $\pi_{(u,v)}$  defines for each value of  $u$  a unique value that must be given to  $v$  in order to satisfy the constraint. The goal is to find an assignment to the variables that satisfies as many constraints as possible. The Unique Games Conjecture states that for all  $\epsilon > 0$ , there exists  $R$ , such that distinguishing instances of UNIQUE LABEL COVER for which it is possible to satisfy a  $(1 - \epsilon)$ -fraction of the constraints from instances where no assignment satisfies more than an  $\epsilon$ -fraction of the constraints is NP-hard. In this section we will need a slightly different version of this conjecture, which was defined by Khot and Regev as the Strong Unique Games Conjecture. Despite the name, Khot and Regev showed that this version is implied by the standard UGC. The precise formulation is the following:

► **Theorem 14** (Theorem 3.2 of [9]). *If the Unique Games Conjecture is true, then for all  $\epsilon > 0$  it is NP-hard to distinguish between the following two cases of instances of UNIQUE LABEL COVER  $G = (V, E)$ :*

- (Yes case): *There exists a set  $V' \subseteq V$  with  $|V'| \geq (1 - \epsilon)|V|$  and an assignment for  $V'$  such that all constraints with both endpoints in  $V'$  are satisfied.*
- (No case): *For any assignment to  $V$ , for any set  $V' \subseteq V$  with  $|V'| \geq \epsilon|V|$ , there exists a constraint with both endpoints in  $V'$  that is violated by the assignment.*

Using the version of the UGC given in Theorem 14 we are ready to present our hardness of approximation argument for the crossword puzzle.

► **Theorem 15.** *Suppose that the Unique Games Conjecture is true. Then, for all  $\epsilon$  with  $\frac{1}{4} > \epsilon > 0$ , there exists an alphabet  $\Sigma_\epsilon$  such that it is NP-hard to distinguish between the following two cases of instances of the crossword problem on alphabet  $\Sigma_\epsilon$ :*

- (Yes case): *There exists a valid solution that fills a  $(1 - \epsilon)$ -fraction of all cells.*
- (No case): *No valid solution can fill more than a  $(\frac{1}{2} + \epsilon)$ -fraction of all cells.*

*Moreover, the above still holds if all slots have distinct lengths (and hence reusing words is trivially impossible).*

**Proof.** Fix an  $\epsilon > 0$ . We will later define an appropriately chosen value  $\epsilon' \in (0, \epsilon)$  whose value only depends on  $\epsilon$ . We present a reduction from a UNIQUE LABEL COVER instance, as described in Theorem 14. In particular, suppose we have an instance  $G = (V, E)$ , with  $|V| = n$ , alphabet  $[R]$ , such that (under UGC) it is NP-hard to distinguish if there exists a set  $V'$  of size  $(1 - \epsilon')n$  that satisfies all its induced constraints, or if all sets  $V'$  of size  $\epsilon'n$  induce at least one violated constraint for any assignment. Throughout this proof we assume that  $n$  is sufficiently large (otherwise the initial instance is easy). In particular, let  $n > \frac{20}{\epsilon}$ .

We construct an instance of the crossword puzzle that fits in an  $N \times N$  square, where  $N = 4n + n^2$ . We number the rows  $1, \dots, N$  from top to bottom and the columns  $1, \dots, N$  from left to right. The instance contains  $n$  horizontal and  $n$  vertical slots. For  $i \in [n]$ , the  $i$ -th horizontal slot is placed in row  $2i$ , starting at column 1, and has length  $2n + n^2 + i$ . For  $j \in [n]$ , the  $j$ -th vertical slot is placed in column  $2j$ , starts at row 1 and has length  $3n + n^2 + j$ . Observe that all horizontal slots intersect all vertical slots and in particular, for all  $i, j \in [n]$  the cell in row  $2i$ , column  $2j$  belongs to the  $i$ -th horizontal slot and the  $j$ -th vertical slot. Furthermore, each slot has a distinct length, as the longest horizontal slot has length  $3n + n^2$  while the shortest vertical slot has length  $3n + n^2 + 1$ .

We define the alphabet as  $\Sigma_\epsilon = [R] \cup \{*\}$ . Before we define our dictionary, let us give some intuition. Let  $V = \{v_1, \dots, v_n\}$ . The idea is that a variable  $v_i \in V$  of the original instance will be represented by both the  $i$ -th horizontal slot and the  $i$ -th vertical slot. In particular, we will define, for each  $\alpha \in [R]$  a pair of words that we can place in these slots to represent

## 36:14 Filling Crosswords Is Very Hard

the fact that  $v_i$  is assigned with the value  $\alpha$ . We will then ensure that if we place words on both the  $i$ -th horizontal slot and the  $j$ -th horizontal slot, where  $(v_i, v_j) \in E$ , then the assignment that can be extracted by reading these words will satisfy the constraint  $\pi_{(v_i, v_j)}$ . The extra letter  $*$  represents an indifferent assignment (which we need if  $(v_i, v_j) \notin E$ ).

Armed with this intuition, let us define our dictionary.

- For each  $i \in [n]$ , for each  $\alpha \in [R]$  we define a word  $d_{(i, \alpha)}$  of length  $2n + n^2 + i$ . The word  $d_{(i, \alpha)}$  has the character  $*$  everywhere except at position  $2i$  and at positions  $2j$  for  $j \in [n]$  and  $(v_i, v_j) \in E$ . In these positions the word  $d_{(i, \alpha)}$  has the character  $\alpha$ .
- For each  $j \in [n]$ , for each  $\alpha \in [R]$  we define a word  $d'_{(j, \alpha)}$  of length  $3n + n^2 + j$ . The word  $d'_{(j, \alpha)}$  has the character  $*$  everywhere except at position  $2j$  and at positions  $2i$  for  $i \in [n]$  and  $(v_i, v_j) \in E$ . In position  $2j$  we have the character  $\alpha$ . In position  $2i$  with  $(v_i, v_j) \in E$ , we place the character  $\beta \in [R]$  such that the constraint  $\pi_{(v_i, v_j)}$  is satisfied by assigning  $\beta$  to  $v_i$  and  $\alpha$  to  $v_j$ . (Note that  $\beta$  always exists and is unique, as the constraints are permutations on  $[R]$ , that is, for each value  $\alpha$  of  $v_j$  there exists a unique value  $\beta$  of  $v_i$  that satisfies the constraint).

This completes the construction. Suppose now that  $V = \{v_1, \dots, v_n\}$  and that we started from the Yes case of UNIQUE LABEL COVER, that is, there exists a set  $V' \subseteq V$  such that  $|V'| \geq (1 - \epsilon')n$  and all constraints induced by  $V'$  can be simultaneously satisfied. Fix an assignment  $\sigma : V' \rightarrow [R]$  that satisfies all constraints induced by  $V'$ . For each  $i \in [n]$  such that  $v_i \in V'$  we place in the  $i$ -th horizontal slot (that is, in row  $2i$ ) the word  $d_{(i, \sigma(v_i))}$ . For each  $j \in [n]$  such that  $v_j \in V'$  we place in the  $j$ -th vertical slot the word  $d'_{(j, \sigma(v_j))}$ . We leave all other slots empty. We claim that this solution is valid, that is, no shared cell is given different values from its horizontal and vertical slot. To see this, examine the cell in row  $2i$  and column  $2j$ . If both of the slots that contain it are filled, then  $v_i, v_j \in V'$ . If  $(v_i, v_j) \notin E$  and  $i \neq j$ , then the cell contains  $*$  from both words. If  $i = j$ , then the cell contains  $\sigma(v_i)$  from both words. If  $i \neq j$  and  $(v_i, v_j) \in E$ , then the cell contains  $\sigma(v_i)$ . This is consistent with the vertical word, as the constraint  $\pi_{(v_i, v_j)}$  is assumed to be satisfied by  $\sigma$ . We now observe that this solution covers at least  $2(1 - \epsilon')n^3$  cells, as we have placed  $2(1 - \epsilon')n$  words, each of length at least  $n^2 + 2n$ , that do not pairwise intersect beyond their first  $2n$  characters.

Suppose now we started our construction from a No instance of UNIQUE LABEL COVER. We claim that the optimal solution in the new instance cannot cover significantly more than half the cells. In particular, suppose a solution covers at least  $(1 + \epsilon')n^3 + 10n^2$  cells. We claim that the solution must have placed at least  $(1 + \epsilon')n$  words. Indeed, if we place at most  $(1 + \epsilon')n$  words, as the longest word has length  $n^2 + 4n$ , the maximum number of cells we can cover is  $(1 + \epsilon')n(n^2 + 4n) \leq (1 + \epsilon')n^3 + 4(1 + \epsilon')n^2 < (1 + \epsilon')n^3 + 10n^2$ . Let  $x$  be the number of indices  $i \in [n]$  such that the supposed solution has placed a word in both the  $i$ -th horizontal slot and the  $i$ -th vertical slot. We claim that  $x \geq \epsilon'n$ . Indeed, if  $x < \epsilon'n$ , then the total number of words we might have placed is at most  $(n - x) + 2x < (1 + \epsilon')n$ , which contradicts our previous observation that we placed at least  $(1 + \epsilon')n$  words. Let  $V' \subseteq V$  be defined as the set of  $v_i \in V$  such that the solution places words in the  $i$ -th horizontal and vertical slot. Then  $|V'| \geq \epsilon'n$ . We claim that it is possible to satisfy all the constraints induced by  $V'$  in the original instance, obtaining a contradiction. Indeed, we can extract an assignment for each  $v_i \in V'$  by assigning to  $v_i$  value  $\alpha$  if the  $i$ -th horizontal slot contains the word  $d_{(i, \alpha)}$ . Note that the  $i$ -th horizontal slot must contain such a word, as these words are the only ones that have an appropriate length. Observe that in this case the  $i$ -th vertical slot must also contain  $d'_{(i, \alpha)}$ . Now, for  $v_i, v_j \in V'$ , with  $(v_i, v_j) \in E$  we see that  $\pi_{(v_i, v_j)}$  is satisfied by our assignment, otherwise we would have a conflict in the cell in position  $(2i, 2j)$ . Therefore, in the No case, it must be impossible to fill more than  $(1 + \epsilon')n^3 + 10n^2$  cells.

The only thing that remains is to define  $\epsilon'$ . Let  $C$  be the total number of cells in the instance. Recall that we proved that in the Yes case we cover at least  $2(1 - \epsilon')n^3$  cells and in the No case at most  $(1 + \epsilon')n^3 + 10n^2$  cells. So we need to define  $\epsilon'$  such that  $2(1 - \epsilon')n^3 \geq (1 - \epsilon)C$  and  $(1 + \epsilon')n^3 + 10n^2 \leq (\frac{1}{2} + \epsilon)C$ . To avoid tedious calculations, we observe that  $2n^3 \leq C \leq 2n^3 + 8n^2$ . Therefore, it suffices to have  $2(1 - \epsilon')n^3 \geq 2(1 - \epsilon)(n^3 + 4n^2)$  and  $(1 + \epsilon')n^3 + 10n^2 \leq (1 + 2\epsilon)n^3$ . The first inequality is equivalent to  $(\epsilon - \epsilon')n \geq 4(1 - \epsilon)$  and the second inequality is equivalent to  $(2\epsilon - \epsilon')n \geq 10$ . Since we have assumed that  $n \geq 20/\epsilon$ , it is sufficient to set  $\epsilon' = \epsilon/2$ . ◀

## 6 Conclusion

We studied the parameterized complexity of some crossword puzzles under several different parameters and we gave some positive results followed by proofs which show that our algorithms are essentially optimal. Based on our results the most natural questions that arise are: What is the complexity of CP-DEC when the grid graph is a matching and the alphabet has size 2? Can Theorem 12 be strengthened by starting from ETH instead of randomized ETH? Can we beat the 1/2 approximation ratio of CP-OPT if we restrict our instances? Can Theorem 14 be strengthened by dropping the UGC? Furthermore, it would be interesting to investigate if there exist non trivial instances of the problem that can be solved in polynomial time. Finally, we could consider a variation of the crossword puzzle problems where each word can be used a given number of times. This would be an intermediate case between word reuse and no word reuse.

---

## References

- 1 Anbulagan and Adi Botea. Crossword puzzles as a constraint problem. In *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, pages 550–554, 2008.
- 2 Edward K. Crossman and Sharyn M. Crossman. The crossword puzzle as a teaching tool. *Teaching of Psychology*, 10(2):98–99, 1983.
- 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 4 Jakob Engel, Markus Holzer, Oliver Ruepp, and Frank Sehnke. On computer integrated rationalized crossword puzzle manufacturing. In *Fun with Algorithms – 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, pages 131–141, 2012.
- 5 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 6 Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search lessons learned from crossword puzzles. In *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, USA, July 29 – August 3, 1990, 2 Volumes*, pages 210–215, 1990.
- 7 Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Oper. Res. Lett.*, 36(5):594–596, 2008.
- 8 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- 9 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.
- 10 Michael Lampis, Valia Mitsou, and Karolina Soltys. Scrabble is PSPACE-complete. *J. Inf. Process.*, 23(3):284–292, 2015. doi:10.2197/ipsjjip.23.284.
- 11 Michael L. Littman, Greg A. Keim, and Noam M. Shazeer. Solving crosswords with PROVERB. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*, pages 914–915, 1999.

## 36:16 Filling Crosswords Is Very Hard

- 12 Télé Magazine. Publications Grand Public.
- 13 Gary Meehan and Peter Gray. Constructing crossword grids: Use of heuristics vs constraints. In *In: Proceedings of Expert Systems 97: Research and Development in Expert Systems XIV, SGES*, pages 159–174, 1997.
- 14 Jagan A. Pillai, Charles B. Hall, Dennis W. Dickson, Herman Buschke, Richard B. Lipton, and Joe Verghese. Association of crossword puzzle participation with memory decline in persons who develop dementia. *Journal of the International Neuropsychological Society*, 17(6):1006–1013, 2011.
- 15 Leonardo Rigutini, Michelangelo Diligenti, Marco Maggini, and Marco Gori. Automatic generation of crossword puzzles. *Int. J. Artif. Intell. Tools*, 21(3), 2012.
- 16 Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo tree search. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 649–654, 2011.

# Grid Recognition: Classical and Parameterized Computational Perspectives

Siddharth Gupta ✉

Ben Gurion University of the Negev, Beer Sheva, Israel

Guy Sa'ar ✉

Ben Gurion University of the Negev, Beer Sheva, Israel

Meirav Zehavi ✉

Ben Gurion University of the Negev, Beer Sheva, Israel

---

## Abstract

Grid graphs, and, more generally,  $k \times r$  grid graphs, form one of the most basic classes of geometric graphs. Over the past few decades, a large body of works studied the (in)tractability of various computational problems on grid graphs, which often yield substantially faster algorithms than general graphs. Unfortunately, the recognition of a grid graph (given a graph  $G$ , decide whether it can be embedded into a grid graph) is particularly hard – it was shown to be NP-hard even on trees of pathwidth 3 already in 1987. Yet, in this paper, we provide several positive results in this regard in the framework of parameterized complexity (additionally, we present new and complementary hardness results). Specifically, our contribution is threefold. First, we show that the problem is fixed-parameter tractable (FPT) parameterized by  $k + \text{mcc}$  where  $\text{mcc}$  is the maximum size of a connected component of  $G$ . This also implies that the problem is FPT parameterized by  $\text{td} + k$  where  $\text{td}$  is the treedepth of  $G$ , as  $\text{td} \leq \text{mcc}$  (to be compared with the hardness for pathwidth 2 where  $k = 3$ ). (We note that when  $k$  and  $r$  are unrestricted, the problem is trivially FPT parameterized by  $\text{td}$ .) Further, we derive as a corollary that strip packing is FPT with respect to the height of the strip plus the maximum of the dimensions of the packed rectangles, which was previously only known to be in XP. Second, we present a new parameterization, denoted  $a_G$ , relating graph distance to geometric distance, which may be of independent interest. We show that the problem is para-NP-hard parameterized by  $a_G$ , but FPT parameterized by  $a_G$  on trees, as well as FPT parameterized by  $k + a_G$ . Third, we show that the recognition of  $k \times r$  grid graphs is NP-hard on graphs of pathwidth 2 where  $k = 3$ . Moreover, when  $k$  and  $r$  are unrestricted, we show that the problem is NP-hard on trees of pathwidth 2, but trivially solvable in polynomial time on graphs of pathwidth 1.

**2012 ACM Subject Classification** Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

**Keywords and phrases** Grid Recognition, Grid Graph, Parameterized Complexity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.37

**Related Version** *Full Version:* <https://arxiv.org/abs/2106.16180>

**Funding** Supported in part by the United States – Israel Binational Science Foundation (BSF) grant no. 2018302 and Israel Science Foundation (ISF) individual research grant no. 1176/18.

*Siddharth Gupta:* Supported in part by the Zuckerman STEM Leadership Program.

*Guy Sa'ar:* Supported in part by the Israeli Smart Transportation Research Center (ISTRIC).



© Siddharth Gupta, Guy Sa'ar, and Meirav Zehavi;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 37; pp. 37:1–37:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



## 1 Introduction

Geometrically, a *grid graph* is a graph that can be drawn on the Euclidean plane so that all vertices are drawn on points having positive integer coordinates, and all edges are drawn as axis-parallel straight line segments of length 1;<sup>1</sup> when the maximum  $x$ -coordinate is at most  $r$  and the maximum  $y$ -coordinate is at most  $k$ , we may use the term  $k \times r$  *grid graph* (see Figure 2). Grid graphs form one of the simplest and most intuitive classes of geometric graphs. Over the past few decades, algorithmic research of grid graphs yielded a large body of works on the tractability or intractability of various computational problems when restricted to grid graphs (e.g., see [14, 15, 3, 37, 48, 17, 4] for a few examples). Even for problems that remain NP-hard on grid graphs, we know of practical algorithms for instances of moderate size (e.g., the STEINER TREE problem on grid graphs is NP-hard [27], but admits practical algorithms [25, 49]). Thus, the recognition of a graph as a grid graph unlocks highly efficient tools for its analysis. In practice, grid graphs can represent layouts or environments, and have found applications in several fields, such as VLSI design [45], motion planning [32] and routing [46]. Indeed, grid graphs naturally arise to represent entities and the connections between them in *existing* layouts or environments. However, often we are given just a (combinatorial) graph  $G$  – i.e., we are given entities and the connections desired to have between them, and we are to construct the layout or environment; specifically, we wish to test whether  $G$  can be embedded into a grid graph (where if it can so, realize it as such a graph). Equivalently, the recognition of a grid graph can be viewed as an embedding problem, where a given graph is to be embedded within a rectangular solid grid.

Accordingly, the problem of recognizing (as well as realizing) grid graphs is a basic recognition problem in Graph Drawing. In what follows, we discuss only recognition – however, it would be clear that all of our results hold also for realization (with the same time complexity in case of algorithms). Formally, in the GRID EMBEDDING problem, we are given a (simple, undirected)  $n$ -vertex graph  $G$ , and need to decide whether it can be embedded into a grid graph. In many cases, taking into account physical constraints, compactness or visual clarity, we would like to not only have a grid graph, but also restrict its dimensions. This yields the  $k \times r$ -GRID EMBEDDING problem, where given an  $n$ -vertex graph  $G$  and positive integers  $k, r \in \mathbb{N}$ , we need to decide whether  $G$  is a  $k \times r$  grid graph. Notice that GRID EMBEDDING is the special case of  $k \times r$ -GRID EMBEDDING where  $k = r = n$  (which virtually means that no dimension restriction is posed).

The GRID EMBEDDING problem has been proven to be NP-hard already in 1987, even on trees of pathwidth 3 [9]. Shortly afterwards, it has been proven to be NP-hard even on binary trees [29]. On the positive side, there is research on practical algorithms for this problem [7]. The related upward planarity testing and rectilinear planarity testing problems are also known to be NP-hard [28], as well as HV-planarity testing even on graphs of maximum degree 3 [20]. We remark that when the embedding is fixed, i.e., the clockwise order of the edges is given for each vertex, the situation becomes drastically easier computationally; then, for example, a rectangular drawing of a plane graph of maximum degree 3, as well as an orthogonal drawing without bends of a plane graph of maximum degree 3, were shown to be computable in linear time in [43] and [44], respectively.

In this paper, we study the classical and parameterized complexity of the GRID EMBEDDING and  $k \times r$ -GRID EMBEDDING problems. To the best of our knowledge, this is the first time that these problems are studied from the perspective of parameterized complexity. Let

---

<sup>1</sup> Some papers in the literature use the term grid graphs to refer to *induced* grid graphs, where we require also that every pair of vertices at distance 1 from each other have an edge between them.



$\Pi$  be an NP-hard problem. In the framework of parameterized complexity, each instance of  $\Pi$  is associated with a *parameter*  $k$ . We say that  $\Pi$  is *fixed-parameter tractable (FPT)* if any instance  $(I, k)$  of  $\Pi$  is solvable in time  $f(k) \cdot |I|^{\mathcal{O}(1)}$ , where  $f$  is an arbitrary computable function of  $k$ . Nowadays, Parameterized Complexity supplies a rich toolkit to design FPT algorithms as well as to prove that some problems are unlikely to be FPT [22, 16, 24]. In particular, the term para-NP-hard refers to problems that are NP-hard even when the parameter is fixed, which implies that they are not FPT unless  $P=NP$ .

Research at the intersection of graph drawing and parameterized complexity (and parameterized algorithms in particular) is in its infancy. Most (in particular, the early efforts) have been directed at variants of the classic CROSSING MINIMIZATION problem, introduced by Turán in 1940 [47], parameterized by the number of crossings (see, e.g., [30, 38, 23, 35, 36, 39]). However, in the past few years, there is an increasing interest in the analysis of a variety of other problems in graph drawing from the perspective of parameterized complexity (see, e.g., [1, 10, 2, 31, 13, 34, 6, 19, 18, 11, 21, 41, 40]).

## Our Contribution and Main Proof Ideas

**I. Parameterized Complexity: Maximum Connected Component Size.** Our contribution is threefold. First, we prove that  $k \times r$ -GRID EMBEDDING is FPT parameterized by  $\text{mcc} + k$ . Here, the idea of the proof is first to recognize all possible embeddings of any choice of connected components or parts of connected components of  $G$  into  $k \times \text{mcc}(G)$  grids, called blocks. These blocks then serve as vertices of a new digraph, where there is an arc from one vertex to another if and only if the corresponding blocks can be placed one after the other. After that, we also guess which blocks should occur at least once in the solution, as well as a spanning tree of the underlying undirected graph of the graph induced on them. This then leads us to a formulation of an Integer Linear Program (ILP), where we ensure that each connected component is used as many times as it is in the input, and that overall we get an Eulerian trail in the graph – having such a trail allows us to place the blocks one after the other, so that every pair of consecutive blocks are compatible. The ILP can then be solved using known tools.

► **Theorem 1.1.**  $k \times r$ -GRID EMBEDDING is FPT parameterized by  $\text{mcc} + k$  where  $\text{mcc}$  is the maximum size of a connected component in the input graph.

One almost immediate corollary of this theorem concerns the 2-STRIP PACKING problem. In this problem, we are given a set of  $n$  rectangles  $S$ , and positive integers  $k, W \in \mathbb{N}$ , and the objective is to decide whether all the rectangles in  $S$  can be packed in a rectangle (called a *strip*) of dimensions  $k \times W$ . In [5], it was shown that if the maximum of the dimensions of the input rectangles, denoted by  $\ell$ , is fixed, then the problem is FPT by  $k$ . Thus, the question whether the problem is FPT parameterized by  $k + \ell$  remained open. By a straightforward reduction, we resolve this question as a corollary of our theorem.

► **Corollary 1.2.** 2-STRIP PACKING is FPT parameterized by  $\ell + k$  where  $\ell$  is the maximum of the dimensions of the input rectangles.

We remark that in case  $k$  and  $r$  are unrestricted, the problem is trivially FPT with respect to  $\text{mcc}$ , since one can embed each connected component (using brute-force) individually. This implies that GRID EMBEDDING is FPT parameterized by  $\text{mcc}$ .

As a corollary of Theorem 1.1 and the above observation, we obtain that  $k \times r$ -GRID EMBEDDING is FPT parameterized by  $\text{td} + k$ , and GRID EMBEDDING is FPT parameterized by  $\text{td}$ , where  $\text{td}$  is the treedepth of the input graph. This finding is of interest when contrasted with the hardness of these problems when pathwidth equals 2 and  $k = 3$  or unrestricted. Thus, this also charts a tractability border between pathwidth and treedepth.

## II. Parameterized Complexity: Difference Between Graph and Geometric Distances.

Secondly, we introduce a new parameterization that relates graph distance to geometric distance, and may be of independent interest. Roughly speaking, the rationale behind this parameterization is to bound the difference between them, so that graph distances may act as approximate indicators to geometric distances. In particular, vertices that are close in the graph, are to be close in the embedding, and vertices that are distant in the graph, are to be distant in the embedding as well. Specifically, with respect to an embedding  $f$  of  $G$  in a grid, we define the *grid distance* between any two vertices as the distance between them in  $f$  in L1 norm. Then, we define the measure of *distance approximation* of  $f$  as the maximum of the difference between the graph distance (in  $G$ ) and the grid distance of two vertices, taken over all pairs of vertices in  $G$ . Here, it is implicitly assumed that  $G$  is connected. Then, the parameter  $a_G$  is the minimum distance approximation  $a_f$  of any embedding  $f$  of  $G$  in a (possibly  $k \times r$ ) grid, defined as  $|V(G)|$  if no such embedding exists. A more formal definition as well as motivation is given in Section 2.

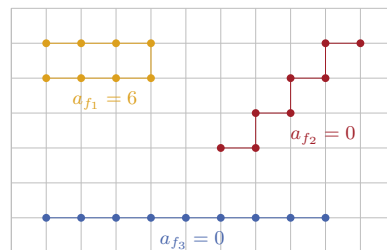
We first prove that the problems are para-NP-hard parameterized by  $a_G$ . This reduction is quite technical. On a high level, we present a construction of “blocks” that are embedded in a grid-like fashion, where we place an outer “frame” of the form of a grid to guarantee that the boundary (which is a cycle) of each of these blocks must be embedded as a square. Each variable is associated with a column of blocks, and each clause is associated with a row of blocks. Within each block, we place two gadgets, one which transmits information in a row-like fashion, to ensure that the clause corresponding to the row has at least one literal that is satisfied, and the other (which is very different than the first) transmits information in a column-like fashion, to ensure consistency between all blocks corresponding to the same variable (i.e., that all of them will be embedded internally in a way that represents only truth, or only false). For clarity, in the full version we split the reduction into two, and use as an intermediate problem a new problem that we call the BATTERIES problem.

► **Theorem 1.3.** GRID EMBEDDING (and hence also  $k \times r$ -GRID EMBEDDING) is para-NP-hard parameterized by  $a_G$ .

When we enrich the parameterization by  $k$ , then  $k \times r$ -GRID EMBEDDING problem becomes FPT. (Recall that parameterized by  $k$  alone, the problem is para-NP-hard). The idea of the proof is to partition a rectangular solid  $k \times r$  grid in which we embed our graph into blocks of size  $k \times (a_G + k)$ , and “guess” one vertex that is to be embedded in the leftmost column of the leftmost block. Then, the crux is in the observation that, for every vertex, the block in which it should be placed is “almost” fixed – that is, we can determine two consecutive blocks in which the vertex may be placed, and then we only have a choice of one among them. This, in turn, leads us to the design of an iterative procedure that traverses the blocks from left to right, and stores, among other information, which vertices were used in the previous block.

► **Theorem 1.4.**  $k \times r$ -GRID EMBEDDING is FPT parameterized by  $a_G + k$ .

Lastly, we prove that when restricted to trees, the problems become FPT parameterized by  $a_G$  alone. Here, a crucial ingredient is to understand the structure of the tree, including a bound on the number of vertices of degree at least 3 in the tree that split it to “large” subtrees. For this, one of the central insights is that, with respect to an internal vertex  $v$  and any two “large” subtrees attached to it (there can be up to four subtrees attached to it), in order not to exceed the allowed difference between the graph and geometric distances, one of the subtrees must be embedded in the “opposite” direction of the other (so, both



■ **Figure 1** Example of a path  $P$  on 8 vertices with three different grid graph embeddings  $f_1$ ,  $f_2$  and  $f_3$ . Since  $a_{f_2} = a_{f_3} = 0$ , we get that  $a_P = 0$ .

are embedded roughly on the same vertical or horizontal line in opposite sides). Now, for an internal vertex of degree at least 3, there must be two attached subtrees that are not embedded in this fashion (as a line can only accommodate two subtrees), which leads us to the conclusion that all but two of the attached subtrees are small. Making use of this ingredient, we argue that a dynamic programming procedure (somewhat similar to the one mentioned for the previous theorem but much more involved) can be used.

► **Theorem 1.5.**  $k \times r$ -GRID EMBEDDING (and hence also GRID EMBEDDING) on trees is FPT parameterized by  $a_G$ .

**III. Classical Complexity.** Lastly, we extend current knowledge of the classical complexity of GRID EMBEDDING and  $k \times r$ -GRID EMBEDDING at several fronts. Here, we begin by developing a refinement of the classic reduction from NOT-ALL-EQUAL 3SAT in [9] (which asserted hardness on trees of pathwidth 3) to derive the following result. While the reduction itself is similar, our proof is more involved and requires, in particular, new inductive arguments.

► **Theorem 1.6.** GRID EMBEDDING is NP-hard even on trees of pathwidth 2. Thus, it is para-NP-hard parameterized by  $\text{pw}$ , where  $\text{pw}$  is the pathwidth of the input graph.

In particular, now the hardness result is *tight* with respect to pathwidth due to the simple observation that GRID EMBEDDING is polynomial time solvable on graphs of pathwidth 1. Because GRID EMBEDDING is a special case of  $k \times r$ -GRID EMBEDDING, the above theorem has the following result as an immediate corollary:  $k \times r$ -GRID EMBEDDING is NP-hard even on trees of pathwidth 2.

Additionally, we show that  $k \times r$ -GRID EMBEDDING is NP-hard on graphs of pathwidth 2 even when  $k = 3$ . Here, we give a relatively simple reduction from 3-PARTITION (whose objective is to partition a set of numbers encoded in unary into sets of size 3 that sum up to the same number), where the idea is to encode “containers” by special identical connected components whose embedding is essentially fixed, and then each number as a simple path on a corresponding number of vertices.

► **Theorem 1.7.**  $k \times r$ -GRID EMBEDDING is NP-hard even on graphs of pathwidth 2 when  $k = 3$ . Thus, it is para-NP-hard parameterized by  $k + \text{pw}$ , where  $\text{pw}$  is the pathwidth of the input graph.

## 2 Preliminaries

Definitions of various standard concepts can be found in the full version. For every  $k \in \mathbb{N}$ , denote  $[k] = \{1, 2, \dots, k\}$ , and for  $i, j \in \mathbb{N}$ , denote  $[i, j] = \{i, i + 1, \dots, j\}$ . Given a set  $W$  of integers,  $\sum W$  denotes the sum of its elements. Given a function  $g$  defined on a set  $W$ , we denote the set of images of its elements by  $g(W)$ . Given a graph  $G$ , we denote its vertex set and edge set by  $V(G)$  and  $E(G)$ , respectively. For a vertex  $v \in V(G)$ , we denote the degree of  $v$  in  $G$  by  $\deg_G(v)$ . Given a set  $V' \subseteq V(G)$ , the subgraph of  $G$  induced by  $V'$  is denoted by  $G[V']$ . Given  $u, v \in V(G)$ , the distance  $d(u, v)$  between  $u$  and  $v$  in  $G$  is the length of a shortest path between them. The *pathwidth* of a graph  $G$  is denoted by  $\text{pw}(G)$ .

We now define basic notions related to grids and grid embeddings. Let  $f : V(G) \rightarrow \mathbb{N} \times \mathbb{N}$  be a function that maps each vertex  $v$  of  $G$  to a point  $(i, j)$  of an integer grid; then,  $i$  and  $j$  are also denoted as  $f_{\text{row}}(v)$  and  $f_{\text{col}}(v)$ , respectively, that is,  $f(v) = (f_{\text{row}}(v), f_{\text{col}}(v))$ . Let  $u, v \in V(G)$  be two vertices. The *grid graph distance of  $u$  and  $v$  induced by  $f$* , denoted by  $d_f(u, v)$ , is defined to be  $d_f(u, v) = |f_{\text{row}}(u) - f_{\text{row}}(v)| + |f_{\text{col}}(u) - f_{\text{col}}(v)|$ . A  $k \times r$  *grid graph embedding* of  $G$  is an injection  $f : V(G) \rightarrow [k] \times [r]$  such that for every  $\{u, v\} \in E(G)$  it follows that  $d_f(u, v) = 1$ . Moreover, a *grid graph embedding* of  $G$  is a  $|V(G)| \times |V(G)|$  grid graph embedding of  $G$ . We say that  $G$  is a (*resp.*  $k \times r$ ) *grid graph* if there exists a (*resp.*  $k \times r$ ) grid graph embedding of  $G$ . Now, the  $k \times r$ -GRID EMBEDDING and GRID EMBEDDING problems are defined as follows. Given a graph  $G$  and two positive integers  $k, r$ , the  $k \times r$ -GRID EMBEDDING and GRID EMBEDDING problems ask whether  $G$  is a  $k \times r$  grid graph or a grid graph, respectively.

We now define the distance approximation parameter formally and discuss some of the motivation behind it. Towards this, we first present the following simple observation. Let  $G$  be a connected grid graph with a grid embedding  $f$ , and let  $u, v \in V(G)$ . Then,  $d_f(u, v) \leq d(u, v)$ . Keeping this in mind, we drop the absolute value notation from the following definition: For any  $k \times r$  grid graph embedding  $f$  of  $G$ , define  $a_f = \max_{u, v \in V(G)} (d(u, v) - d_f(u, v))$ . Then, if  $G$  is a  $k \times r$  grid graph, let  $a_G(k, r) = \min\{a_f \mid f \text{ is a } k \times r \text{ grid graph embedding of } G\}$ ; otherwise,  $a_G(k, r) = |V(G)|$ . When  $k$  and  $r$  are clear from context, we write “distance approximation parameter” and  $a_G$  rather than “ $k \times r$  distance approximation parameter” and  $a_G(k, r)$ , respectively. When  $k$  and  $r$  are unrestricted,  $a_G(k, r) = a_G(|V(G)|, |V(G)|)$ . See Figure 1. We also remark that whenever  $G$  is a  $k \times r$  grid graph, then  $a_G(k, r) \leq |V(G)| - 2$  (because for any grid graph embedding  $f$  of  $G$  and two different vertices  $u, v \in V(G)$ ,  $d(u, v) \leq |V(G)| - 1$  and  $d_f(u, v) \geq 1$ ).

This rationale behind this parameter makes sense in various scenarios. Suppose that vertices represent utilities, factories or organizations, or, very differently, components to be placed on a chip. On the one hand, those that are closer to each other in the graph might need to cooperate more often: they have direct and indirect (through other entities on the path) connections between them; the more “links on the chain”, the less is directed interaction required. On the other hand, we may have a competitive constraint – we may want these entities to also be “as far as possible”. In particular, if they are far in the graph, we will take advantage of this to place them far in the embedding (proportionally). For example, these entities may cause pollution, radiation or heat [8, 26]. Alternatively, in the case of utilities, we may want to cover as large area as we can. Recently, due to the COVID-19 pandemic, many governments around the world have introduced social distancing. Briefly, social distancing means that people should be physically away from each other, if possible. According to experts, one of the most effective ways to reduce the spread of coronavirus is social distancing [12, 33, 42]. Suppose that the vertices represent people, the edges represent

social (or other) relations between them, and we want to find a seating arrangement. In order to preserve the social distancing, we would like that people who do not need to be close to each other, to be relatively far away from each other. In another example, suppose that the vertices represent some facilities that “attract” people, like stores. Placing the stores far away from each other, if possible, contributes to social distancing. More intuition is given in the full version. We remark that the embeddings that our algorithms compute satisfy the conditions of being a grid graph embedding, in particular, the embeddings are planar. Furthermore, we do not need to know the value of  $a_G$  in advance, in order to use our algorithm, as we iterate over all the potential values for  $a_G$ .

### 3 FPT Algorithm on General Graphs

In this section, we show that  $k \times r$ -GRID EMBEDDING is FPT parameterized by  $\text{mcc}(G) + k$ . We first give the definition of a  $k \times r$  rectangular grid graph and some related terms. An undirected graph  $H$  is a  $k \times r$  rectangular grid graph if there exists a bijection  $f : V \rightarrow [k] \times [r]$ , such that for every pair of vertices  $u, v \in V(H)$ ,  $\{v, u\} \in E(H)$  if and only if  $d_f(u, v) = 1$ . Given a  $k \times r$  rectangular grid graph  $H$  and a corresponding bijective function  $f$ , we define the columns of  $H$  as follows: For every  $j \in [r]$ , let  $C_j(H) = \{u \in V(H) \mid f_{\text{col}}(u) = j\}$ . Clearly,  $V(H) = \bigcup_{j=1}^r C_j(H)$ . We refer to  $C_1(H)$  and  $C_r(H)$  as the left boundary column and right boundary column, respectively, of  $H$ .

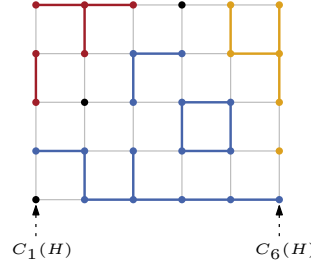
Given a subgraph  $S$  of a  $k \times r$  rectangular grid graph  $H$ , we denote by  $\mathcal{FC}(S)$  the set of fully contained connected components of  $S$  defined as all the connected components of  $S$  that either do not intersect the boundary columns of  $H$  or intersect both boundary columns of  $H$ . Moreover, we denote by  $\mathcal{LC}(S)$  ( $\mathcal{RC}(S)$ ) the set of left contained (right contained) connected components of  $S$  defined as all the connected components of  $S$  that intersect the left (right) boundary column of  $H$  but do not intersect the right (left) boundary column of  $H$ . Note that the three sets  $\mathcal{FC}(S)$ ,  $\mathcal{LC}(S)$  and  $\mathcal{RC}(S)$  are pairwise disjoint and  $S = \mathcal{FC}(S) \cup \mathcal{LC}(S) \cup \mathcal{RC}(S)$ . See Figure 2. We now prove that  $k \times r$ -GRID EMBEDDING is FPT.

**Proof Sketch of Theorem 1.1.** The FPT algorithm is based on ILP. To this end, let  $G$  be an instance of the  $k \times r$ -GRID EMBEDDING problem. Due to lack of space, we only give a sketch of the algorithm. For completeness, please refer to the full version.

**Algorithm.** Let  $H$  be a  $k \times r$  rectangular grid graph. Let  $\mathcal{B} = \{B_1, B_2, \dots, B_p\}$  be an almost partition of  $H$  into blocks of size  $k \times \text{mcc}(G)$  such that  $V(B_i) = \bigcup_{j=(i-1)(\text{mcc}(G)-1)+1}^{i(\text{mcc}(G)-1)+1} C_j(H)$ , for each  $i \in [p]$  where  $p = (r-1)/(\text{mcc}(G)-1)$ . Here, we consider the case where  $r-1$  is a multiple of  $\text{mcc}(G)-1$ . Note that each block  $B_i$  is a  $k \times \text{mcc}(G)$  rectangular grid graph, and for all  $i \in [p-1]$ ,  $B_i$  and  $B_{i+1}$  share a boundary column.

We can restate the  $k \times r$ -GRID EMBEDDING problem as follows: is  $G$  a subgraph of  $H$ ? As  $H$  is a planar graph,  $G$  must be planar. So, we first check if  $G$  is planar. Let  $\text{Comp}(G) = \{G_1, G_2, \dots, G_t\}$  be the set of all non-isomorphic connected components of  $G$ . For every  $i \in [t]$ , let  $\text{num}(G_i)$  be the number of times  $G_i$  appears in  $G$ . As the size of any connected component of  $G$  is at most  $\text{mcc}(G)$ , any connected component  $C$  of  $G$  (when embedded as a subgraph of  $H$ , if possible) intersects (i) only one block  $B_i$  (in particular, it does not intersect either the right or the left boundary of  $B_i$ ), or (ii) exactly two consecutive blocks  $B_i$  and  $B_{i+1}$  through the right boundary column of  $B_i$ , or (iii) exactly three consecutive blocks  $B_{i-1}$ ,  $B_i$  and  $B_{i+1}$  through the left and right boundary columns of  $B_i$ .

Based on the above observation, we compute the set  $\mathcal{S}$  of all the possible snapshots of a  $k \times \text{mcc}(G)$  rectangular grid graph  $R$ , i.e. the set of all the subgraphs of  $R$ , and the left and right adjacencies between snapshots. For every subgraph  $S$  of  $R$ , if  $\mathcal{FC}(S) \subseteq \text{Comp}(G)$ ,



■ **Figure 2** A  $5 \times 6$  rectangular grid graph  $H$ , its boundary columns and a subgraph  $S$  of  $H$  shown by colored vertices and thick colored edges. The blue, red and orange colored connected components belong to  $\mathcal{FC}(S)$ ,  $\mathcal{LC}(S)$  and  $\mathcal{RC}(S)$ , respectively.

then we add  $S$  to  $\mathcal{S}$ . Note that, if there exists a connected component of  $S$  in  $\mathcal{FC}(S)$  that does not belong to  $\text{Comp}(G)$ , then  $S$  cannot “contribute to” a valid solution. We also find the set, denoted **Source**, of snapshots that may correspond to  $B_1$  and the set, denoted **Sink**, of snapshots that may correspond to  $B_p$ . Note that we make this distinction, as except for blocks  $B_1$  and  $B_p$ , all the other blocks share both boundary columns, but  $B_1$  ( $B_p$ ) only share their right (left) boundary column. So for every  $S \in \mathcal{S}$ , if  $\mathcal{LC}(S) \subseteq \text{Comp}(G)$ , then add  $S$  to **Source**, and if  $\mathcal{RC}(S) \subseteq \text{Comp}(G)$ , then add  $S$  to **Sink**. For every snapshot  $S \in \mathcal{S}$  and  $i \in [t]$ , let  $\text{freq}_{\text{cen}}(G_i, S)$  be the number of times  $G_i$  appears in  $\mathcal{FC}(S)$ . Similarly, for every snapshot  $S \in \text{Source}$  ( $S \in \text{Sink}$ ) and  $i \in [t]$ , let  $\text{freq}_{\text{left}}(G_i, S)$  ( $\text{freq}_{\text{right}}(G_i, S)$ ) be the number of times  $G_i$  appears in  $\mathcal{LC}(S)$  ( $\mathcal{RC}(S)$ ).

We now find the set  $\text{Adj} \subseteq \mathcal{S} \times \mathcal{S}$  of all possible adjacencies between pairs of snapshots in  $\mathcal{S}$ . Let  $R'$  be a  $k \times (2\text{mcc}(G) - 1)$  rectangular grid graph. We divide  $R'$  into two blocks  $B'_1$  and  $B'_2$  of size  $k \times \text{mcc}(G)$  such that  $V(B'_1) = \bigcup_{i=1}^{\text{mcc}(G)} C_i(R')$  and  $V(B'_2) = \bigcup_{i=\text{mcc}(G)}^{2\text{mcc}(G)-1} C_i(R')$ . For every  $i \in \{1, 2\}$  and subgraph  $S'$  of  $R'$ , let  $S'_i = S'[V(S') \cap V(B'_i)]$  be the subgraph of  $S'$  in block  $B'_i$ . We look only at those subgraphs  $S'$  for which both  $S'_1$  and  $S'_2$  belong to  $\mathcal{S}$ . For every such  $S'$ , we add the pair  $(S'_1, S'_2)$  to  $\text{Adj}$  if all the connected components of  $S'_1 \cup S'_2 = S'$  that intersect both  $B'_1$  and  $B'_2$  (i.e., intersect column  $C_{\text{mcc}(G)}(R')$ ) belong to  $\text{Comp}(G)$ . Let  $\mathcal{BC}(S'_1, S'_2)$  be the set of all the connected components of  $S'_1 \cup S'_2 = S'$  that intersect the column  $C_{\text{mcc}(G)}(R')$  but intersect neither  $C_1(R')$  nor  $C_{2\text{mcc}(G)-1}(R')$ . For every  $i \in [t]$ , we denote the number of times  $G_i$  appears in  $\mathcal{BC}(S'_1, S'_2)$ , by  $\text{freq}_{\text{boun}}(G_i, (S'_1, S'_2))$ .

For every pair of snapshots  $(start, end)$  such that  $start \in \text{Source}$  and  $end \in \text{Sink}$  and a set  $\mathcal{S}' \subseteq \mathcal{S}$  of snapshots, we create a directed graph  $D$  as follows. We add all the snapshots in  $\mathcal{S}'$  as vertices of  $D$ , and for every pair of snapshots  $S, S' \in \mathcal{S}'$ , if  $(S, S') \in \text{Adj}$ , then add an arc from  $S$  to  $S'$  in  $D$ . We then add both  $start$  and  $end$  as vertices of  $D$  and for every snapshot  $S \in \mathcal{S}$ , if  $(start, S) \in \text{Adj}$  ( $(S, end) \in \text{Adj}$ ), add an arc from  $start$  to  $S$  ( $S$  to  $end$ ) in  $D$ . We then find the number of times  $X(S, S')$ , each arc  $(S, S')$  should be duplicated in  $D$  to get a new multidigraph  $D'$  such that we get a (connected) Eulerian trail in  $D'$  from  $start$  to  $end$  of length  $p$  and all the connected components of  $G$  are covered by the Eulerian trail. Finally, we use the path to get the correspondence between the blocks of  $H$  and the snapshots in  $\mathcal{S}'$  with a correct placement from left to right. The algorithm to find  $D'$  proceeds as follows.

- Find the set  $\mathcal{T}$  of all spanning trees of the underlying undirected graph of  $D$ .
- For every spanning tree  $T \in \mathcal{T}$ , solve the following ILP to find  $X(S, S')$  for every edge  $(S, S') \in E(D)$ .

$$\forall S \in V(D) \setminus \{start, end\} : \sum_{(S, S') \in E(D)} X(S, S') = \sum_{(S'', S) \in E(D)} X(S'', S). \quad (1a)$$



$$\sum_{(start, S) \in E(D)} X(start, S) = 1. \quad (1b)$$

$$\sum_{(S, end) \in E(D)} X(S, end) = 1. \quad (1c)$$

$$\sum_{(S, S') \in E(D)} X(S, S') = p - 1. \quad (1d)$$

$$\forall i \in [t] : \text{freq}_{\text{left}}(G_i, start) + \sum_{(S, S') \in E(D)} X(S, S') \cdot \text{freq}_{\text{boun}}(G_i, (S, S')) + \sum_{S \in V(D)} \left( \sum_{(S, S') \in E(D)} X(S, S') \right) \cdot \text{freq}_{\text{cen}}(G_i, S) + \text{freq}_{\text{right}}(G_i, end) = \text{num}(G_i). \quad (1e)$$

$$\forall (S, S') \in E(T) : X(S, S') + X(S', S) \geq 1. \quad (1f)$$

$$\forall (S, S') \in E(D) \setminus E(T) : X(S, S') \geq 0. \quad (1g)$$

- If the ILP returns a feasible solution, then return **Yes**.

Recall that we run the algorithm for every possible  $D$ . If for none of them we return **Yes**, we eventually return **No**. Equation 1f ensures that the digraph  $D'$  is connected, and, in this context, recall that we go over all the possible spanning trees to check all the different possible connectivities between the vertices of  $D'$ . Equations 1a, 1b and 1c ensure that there exists an Eulerian trail in  $D$  from  $start$  to  $end$ . Equation 1d ensures that the total number of edges in  $D'$  is  $p - 1$ , which in turn means that the Eulerian trail from  $start$  to  $end$  in  $D'$  is of length  $p$ , which is equal to the number of required blocks. Given a multidigraph  $D'$ , each connected component of  $G$  can “contribute to” only one set out of  $\mathcal{LC}(start)$ ,  $\mathcal{BC}(S', S'')$ ,  $\mathcal{FC}(S)$  and  $\mathcal{RC}(end)$ , for  $S, S', S'' \in \mathcal{S}'$  such that  $(S', S'') \in E(D)$ , as there exists no  $S \in \mathcal{S}'$  such that  $(S, start) \in E(D)$  or  $(end, S) \in E(D)$ . So, Equation 1e ensures that all the connected components of  $G$  are covered by the path exactly once. ◀

## 4 Distance Approximation Parameter

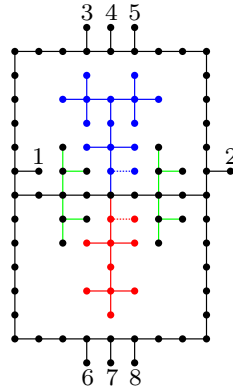
In this section, we consider the distance approximation parameter, and sketch the proofs of Theorems 1.3 and 1.4. The proof of Theorem 1.5 is deferred to the full version.

### 4.1 Para-NP-hardness with Respect to $a_G$ on General Graphs

We show a reduction from SAT to GRID EMBEDDING where  $a_G$  is upper bounded by a constant if the output is a **Yes** instance. For this purpose, we present the *battery gadget* (see Figure 3), composed of a  $13 \times 9$  rectangle. It has a *positive side* and a *negative side*, two *wire vertices* and six *synchronization vertices* attached to the top and bottom sides of the rectangle. On the top and bottom halves of the gadget, it has an optional extra edge, called the *positive voltage* and the *negative voltage*, respectively. We describe the battery gadget by a boolean pair  $H = (x_1, x_2)$ ,  $x_1, x_2 \in \{0, 1\}$  where  $x_1 = 1$  (resp.  $x_2 = 1$ ) if and only if we added the positive voltage edge (resp. negative voltage edge).

Given an instance  $\pi$  of SAT with variables  $x_1, \dots, x_n$  and clauses  $\mu_1, \dots, \mu_m$ , the reduction output is  $\text{reduc}(\pi) = G_\pi$  where  $G_\pi$  defined as follows.  $G_\pi$  is composed of  $m \cdot n$  battery gadgets ordered in a “matrix shape”, i.e. the battery gadget  $H_{i,j}$  is located at the  $i$ -th “row” and  $j$ -th “column” (see Figure 4). Every column  $j \in [n]$  of gadgets corresponds to the variable  $x_j$ , and every row  $i \in [m]$  of gadgets corresponds to the clause  $\mu_i$ . Now, for each  $i \in [m]$  and  $j \in [n]$  we set  $H_{i,j} = (x_1^{i,j}, x_2^{i,j})$  where  $x_1^{i,j} = 0$  (resp.  $x_2^{i,j} = 0$ ) if and only if





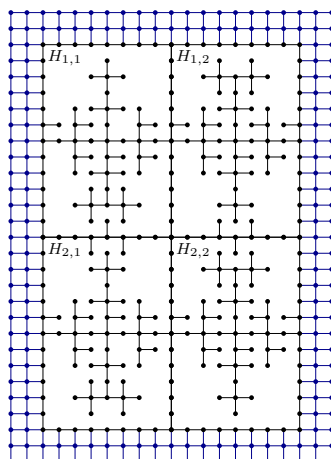
■ **Figure 3** The battery gadget. The positive side is in blue and the negative side is in red. The positive voltage is in dashed blue and the negative voltage is in dashed red. The wire vertices are numbered 1, 2. The synchronization vertices are numbered 3 to 8.

the literal  $x_j$  (resp.  $\bar{x}_j$ ) appears in  $\mu_i$ . We add the positive (resp. negative) voltage edge to the gadget  $H_{i,j}$  if and only if the literal  $x_j$  (resp.  $\bar{x}_j$ ) does not appear in  $\mu_i$ . In addition, the “matrix” of battery gadgets is encircled by an  $m \times n$ -grid frame (see Figure 4). Lastly, we delete the “redundant” topmost and bottommost synchronization edges, i.e. those not attached to a side shared by two rectangles. More precisely, for every  $j \in [n]$ , we delete the three edges attached to the top side of the rectangle of  $H_{1,j}$  and the three edges attached to the bottom side of the rectangle of  $H_{m,j}$ . Observe that each synchronization vertex is common to two battery gadgets, that is, for every  $i \in [m-1]$  and  $j \in [n]$ , the synchronization vertices 3, 4, 5 of  $H_{i+1,j}$  are the synchronization vertices 6, 7, 8 of  $H_{i,j}$ .

For the correctness of the reduction, we distinguish between “parts” (of the graph) that must have a fixed embedding and “parts” that might have several embeddings. We show that the embedding of the  $m \times n$ -grid frame is “almost fixed”. More precisely, the embedding is fixed once we choose an embedding of three specific vertices of it. Intuitively, the “shape” of the  $m \times n$ -grid frame is fixed in every embedding, up to “rotation” of the frame in 90, 180 or 270 degrees, or “movement” (shifting) to another “location”, which are immaterial for our purposes. In addition to the  $m \times n$ -grid frame, the embeddings of the sides of the rectangles, as well as the “middle crossing lines”, of the battery gadgets are also fixed once we choose an embedding for the aforementioned three vertices.

Now, observe that given a battery gadget, we might be able to choose to embed the positive or the negative side on the top of the gadget. For a battery gadget  $H$  with a grid graph embedding  $f$ , we set  $p_f(H) = +$  if the positive side of the gadget is embedded to the top of the gadget; otherwise the negative side is embedded to the top of the gadget, and we set  $p_f(H) = -$ . In addition, we denote by  $V_f(H)$  the voltage of the side of the battery gadget that  $f$  embeds at the top of  $H$ . That is, given  $H = (x_1, x_2)$ , if  $p_f(H) = +$ , then  $V_f(H) = x_1$ , and if  $p_f(H) = -$ , then  $V_f(H) = x_2$ .

Next, we show that for any  $j \in [n]$ , the gadgets in the  $j$ -th column are “synchronized”: for every  $1 \leq i, i' \leq m$ ,  $p_f(H_{i,j}) = p_f(H_{i',j})$ . For intuition, observe that the six synchronization vertices in the battery gadget maybe embedded inside or outside the rectangle and consider two adjacent battery gadget in the same column,  $H_{i,j}$  and  $H_{i+1,j}$ . If  $p_f(H_{i+1,j}) = +$ , then vertices 1 and 3 of  $H_{i+1,j}$  (see Figure 3) must be embedded outside  $H_{i+1,j}$ , so they are embedded inside  $H_{i,j}$ . Then, in  $H_{i,j}$  the positive side cannot be embedded at the bottom, and hence  $p_f(H_{i,j}) = +$ . Similarly, if  $p_f(H_{i+1,j}) = -$ , then vertex 4 must be embedded outside  $H_{i+1,j}$ , so it is embedded inside  $H_{i,j}$ . Then, in  $H_{i,j}$  the negative side cannot be embedded at the bottom, and hence  $p_f(H_{i,j}) = -$ . Formally, we prove the following.



■ **Figure 4** Construction of  $G_\pi$  where  $\pi = (\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_2)$ . The  $2 \times 2$ -grid frame is in blue.

► **Lemma 4.1 (\*)**. *Let  $\pi$  be an instance of SAT with  $n$  variables and  $m$  clauses. Let  $f$  be a grid graph embedding of  $G_\pi$ . For every  $j \in [n]$ , there exists  $p_j \in \{+, -\}$  such that for every  $i \in [m]$  it follows that  $p_f(H_{i,j}) = p_j$ .*

Next, we show that for every  $i \in [m]$ , there exists  $j \in [n]$  such that  $V_f(H_{i,j}) = 0$ . As intuition, note that if vertex 1 of  $H_{i,j}$  is embedded inside  $G_{i,j}$  and  $V_f(G_{i,j}) = 1$ , then it must be that vertex 2 is embedded outside  $G_{i,j}$ . So, since vertex 1 must be embedded inside  $G_{i,1}$  and vertex 2 must be embedded inside  $G_{i,n}$ , there must be a  $j$  such that  $V_f(H_{i,j}) = 0$ .

► **Lemma 4.2 (\*)**. *Let  $\pi$  be an instance of SAT with  $n$  variables and  $m$  clauses. Let  $f$  be a grid graph embedding of  $G_\pi$ . For every  $i \in [m]$ , there exists  $j \in [n]$  such that  $V_f(H_{i,j}) = 0$ .*

We are ready to prove the reverse direction of the correctness of the reduction. Due to lack of space, we omit proof of the forward direction.

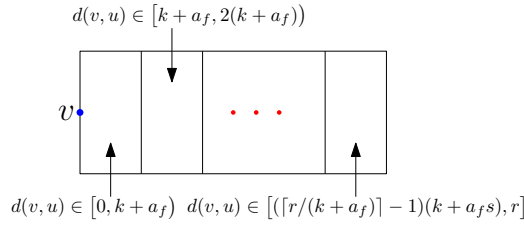
► **Lemma 4.3**. *Let  $\pi$  be an instance of SAT with  $n$  variables and  $m$  clauses. Then  $G_\pi$  is a grid graph if and only if  $\pi$  is a Yes instance of SAT.*

**Partial proof.** Let  $f$  be a grid graph embedding of  $G_\pi$ . By Lemma 4.1, for every  $j \in [n]$ , there exists  $p_j \in \{+, -\}$  such that for every  $i \in [m]$ ,  $p_f(H_{i,j}) = p_j$ . We define  $s : \{x_1, \dots, x_n\} \rightarrow \{T, F\}$  as follows. For  $j \in [n]$ ,  $s(x_j) = T$  if  $p_j = +$ ; otherwise,  $s(x_j) = F$ . We show that  $s$  is a satisfying assignment for  $\pi$ . Let  $i \in [m]$ . By Lemma 4.2, there exists  $j \in [n]$  such that  $V_p(i, j) = 0$ . If  $p(i, j) = p_j = +$ , then  $x_1^{(i,j)} = 0$ , and by the definition of  $G_\pi$ ,  $x_j$  appears in  $\mu_i$ . By the definition of  $s$ , since  $p_j = +$ , we get that  $s(x_j) = T$ , therefore  $\mu_i$  is satisfied. Similarly, if  $p(i, j) = p_j = -$ , then  $x_2^{(i,j)} = 0$ , and by the definition of  $G_\pi$ ,  $\bar{x}_j$  appears in  $\mu_i$ . By the definition of  $s$ , since  $p_j = -$ , we get that  $s(x_j) = F$ , therefore  $\mu_i$  is satisfied. So,  $\pi$  is a Yes instance of SAT. ◀

It only remains to show is that the distance approximation of  $G_\pi$  is bounded by a constant if  $G_\pi$  is a grid graph. Due to lack of space, we omit this proof.

► **Lemma 4.4 (\*)**. *Let  $\pi$  be an instance of SAT. If  $G_\pi$  is a grid graph, then for every grid graph embedding  $f$  of  $G_\pi$  it follows that  $a_f \leq 234$ .*

Using Lemma 4.3 and Lemma 4.4, we can conclude Theorem 1.3.



■ **Figure 5** Sorting the vertices  $u$  of a  $k \times r$  grid into small rectangles.

### 4.2 $k \times r$ -GRID EMBEDDING IS FPT WITH RESPECT TO $k + a_G$

We present an FPT algorithm with respect to  $k + a_G$  for  $k \times r$ -GRID EMBEDDING. The idea is as follows. We guess a leftmost vertex  $v$  in the  $k \times r$  grid (i.e. satisfying  $f_{\text{col}}(v) = 0$ ). Then, going from left to right, we divide the  $k \times r$  grid into  $\lceil \frac{r}{k+a_G} \rceil$  “small rectangles” of size  $k \times (a_G + k)$ , except the last which might be smaller. After this, we sort the vertices into the small rectangles: We put each vertex  $u$ , having graph distance  $d(v, u)$  from  $v$ , in the  $\lceil \frac{d(v, u)}{k+a_G} \rceil$ -th rectangle; if no such rectangle exists, we put  $u$  in the  $(\lceil \frac{d(v, u)}{k+a_G} \rceil - 1)$ -th rectangle, if this rectangle does not exist as well (as  $\lceil \frac{d(v, u)}{k+a_G} \rceil - 1$  is too large), we can conclude that we have a no-instance. In particular, we show that in every  $k \times r$  grid graph embedding  $f$  of  $G$  with  $a_f = a_G$  where  $v$  is a leftmost vertex, every  $u$  is embedded either in its sorted rectangle or the previous one. For intuition, observe that  $u$  cannot be embedded into a farther rectangle as then its shortest path(s) from  $v$  cannot be embedded. On the other hand, if  $u$  is embedded into a closer rectangle, then the embedding does not respect the distance approximation. After we know the “approximate location” of each vertex, we try to find a  $k \times r$  grid graph embedding of  $G$  using an iterative algorithm.

We start by proving the correctness of the “location approximation” of each vertex. To this end, for any  $0 \leq s \leq t$ , we define  $C_f(s, t) = \{u \in V \mid s \leq f_{\text{col}}(u) \leq t\}$  and  $D_v(s, t) = \{u \in V \mid s \leq d(v, u) \leq t\}$ . See Figure 5.

► **Lemma 4.5 (\*)**. *Let  $G = (V, E)$  be a  $k \times r$  grid graph, and let  $f$  be a  $k \times r$  grid graph embedding of  $G$ . Let  $v \in V$  such that  $f_{\text{col}}(v) = 0$ . Let  $u \in V$  be a vertex. Then  $u \in C_f(d(v, u) - a_f - k, d(v, u))$ , and  $u \in D_v(f_{\text{col}}(u), f_{\text{col}}(u) + a_f + k)$ .*

Therefore, every vertex must be embedded either in its sorted rectangle or in the one to its left. In light of this, we try to find a  $k \times r$  embedding of  $G$  by iteration on the small rectangles from left to right. At each step, we seek  $k \times (a_f + k)$  embeddings for the vertices in the current rectangle and for a subset  $U$  of the vertices of the next rectangles. For each such embedding, we only need to store the following information: the subset  $U$  and the “right column” of the embedding (so as to “glue” embeddings of adjacent rectangles properly). In the next rectangle, we try to embed (using brute-force) the vertices we did not embed in the previous rectangle (those outside  $U$ ) and some of the vertices of its next rectangle, such that the left column of the current embedding “agrees” with the right column of the embedding of the previous rectangle. Note that at each step we only store “FPT amount” of information, and so we use only “FPT runtime”. A formal description of the algorithm can be found in the full version. Here, we directly proceed to state the correctness.

► **Lemma 4.6 (\*)**. *There exists an algorithm that given a graph and  $k, r \in \mathbb{N}$  runs in time  $\mathcal{O}(|V|^2 (ka_G)^{\mathcal{O}(ka_G + k^2)})$  and returns “Yes instance” if and only if  $G$  is a  $k \times r$  grid graph.*

Using Lemma 4.6, we can conclude Theorem 1.4.

## References

- 1 Parameterized complexity in graph drawing (Dagstuhl Seminar 21062). URL: <https://www.dagstuhl.de/21062>.
- 2 Akanksha Agrawal, Grzegorz Guspel, Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Connecting the Dots (with Minimum Crossings). In Gill Barequet and Yusu Wang, editors, *Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2019.7.
- 3 Eric Allender, Tanmoy Chakraborty, David A Mix Barrington, Samir Datta, and Sambuddha Roy. Grid graph reachability problems. In *21st Annual IEEE Conference on Computational Complexity (CCC'06)*, pages 15–pp. IEEE, 2006.
- 4 Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM J. Comput.*, 35(3):531–566, 2005.
- 5 Pradeesha Ashok, Sudeshna Kolay, Syed Mohammad Meesum, and Saket Saurabh. Parameterized complexity of strip packing and minimum volume packing. *Theor. Comput. Sci.*, 661:56–64, 2017.
- 6 Michael J. Bannister, Sergio Cabello, and David Eppstein. Parameterized complexity of 1-planarity. *J. Graph Algorithms Appl.*, 22(1):23–49, 2018.
- 7 Moritz Beck and Sabine Storandt. Puzzling grid embeddings. In *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 94–105. SIAM, 2020.
- 8 Arnold D Bergstra, Bert Brunekreef, and Alex Burdorf. The effect of industry-related air pollution on lung function and respiratory symptoms in school children. *Environmental Health*, 17(1):1–9, 2018.
- 9 Sandeep N Bhatt and Stavros S Cosmadakis. The complexity of minimizing wire lengths in vlsi layouts. *Information Processing Letters*, 25(4):263–267, 1987.
- 10 Sujoy Bhore, Robert Ganian, Fabrizio Montecchiani, and Martin Nöllenburg. Parameterized algorithms for book embedding problems. In *Graph Drawing and Network Visualization - 27th International Symposium, GD 2019, Prague, Czech Republic, September 17-20, 2019, Proceedings*, pages 365–378, 2019.
- 11 Thomas Bläsius, Marcus Krug, Ignaz Rutter, and Dorothea Wagner. Orthogonal graph drawing with flexibility constraints. *Algorithmica*, 68(4):859–885, 2014.
- 12 CDC. Social distancing. URL: <https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/social-distancing.html>, November 2020.
- 13 Hubert Y. Chan. A parameterized algorithm for upward planarity testing. In *European Symposium on Algorithms (ESA 2004)*, volume 3221 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 2004. doi:10.1007/978-3-540-30140-0\_16.
- 14 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Almost polynomial hardness of node-disjoint paths in grids. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1220–1233, 2018.
- 15 Julia Chuzhoy, David H. K. Kim, and Rachit Nimavat. Improved approximation for node-disjoint paths in grids with sources on the boundary. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 38:1–38:14, 2018.
- 16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 17 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. A framework for eth-tight algorithms and lower bounds in geometric intersection graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 574–586, 2018.

- 18 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Sketched representations and orthogonal planarity of bounded treewidth graphs. In *Graph Drawing and Network Visualization (GD 2019)*, Lecture Notes in Computer Science. Springer, 2019. To appear. [arXiv:1908.05015](https://arxiv.org/abs/1908.05015).
- 19 Walter Didimo and Giuseppe Liotta. Computing orthogonal drawings in a variable embedding setting. In *Algorithms and Computation (ISAAC 1998)*, volume 1533 of *Lecture Notes in Computer Science*, pages 79–88. Springer, 1998.
- 20 Walter Didimo, Giuseppe Liotta, and Maurizio Patrignani. On the complexity of hv-rectilinear planarity testing. In *International Symposium on Graph Drawing*, pages 343–354. Springer, 2014.
- 21 Walter Didimo, Giuseppe Liotta, and Maurizio Patrignani. HV-planarity: Algorithms and complexity. *J. Comput. Syst. Sci.*, 99:72–90, 2019.
- 22 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.
- 23 Vida Dujmovic, Michael R. Fellows, Matthew Kitching, Giuseppe Liotta, Catherine McCartin, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Sue Whitesides, and David R. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2):267–292, 2008. doi:10.1007/s00453-007-9151-1.
- 24 Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: theory of parameterized preprocessing*. Cambridge University Press, 2019.
- 25 Joseph L Ganley. Computing optimal rectilinear steiner trees: A survey and experimental evaluation. *Discrete Applied Mathematics*, 90(1-3):161–171, 1999.
- 26 Javier García-Pérez, Nerea Fernández de Larrea-Baz, Virginia Lope, Antonio J Molina, Cristina O’Callaghan-Gordo, María Henar Alonso, Marta María Rodríguez-Suárez, Benito Mirón-Pozo, Juan Alguacil, Inés Gómez-Acebo, et al. Residential proximity to industrial pollution sources and colorectal cancer risk: A multicase-control study (mcc-spain). *Environment International*, 144:106055, 2020.
- 27 Michael R Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.
- 28 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.
- 29 Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, 31(4):167–173, 1989.
- 30 Martin Grohe. Computing crossing numbers in quadratic time. *J. Comput. Syst. Sci.*, 68(2):285–302, 2004. doi:10.1016/j.jcss.2003.07.008.
- 31 Magnús M. Halldórsson, Christian Knauer, Andreas Spillner, and Takeshi Tokuyama. Fixed-parameter tractability for non-crossing spanning trees. In *Algorithms and Data Structures (WADS 2007)*, volume 4619 of *Lecture Notes in Computer Science*, pages 410–421. Springer, 2007.
- 32 Dan Halperin, Oren Salzman, and Micha Sharir. Handbook of discrete and computational geometry. In Jacob E. Goodman, Joseph O’Rourke, and Csaba D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, chapter 50, pages 1311–1342. CRC Press LLC, Boca Raton, FL, 2017.
- 33 Cone Health. Social distancing faq: How it helps prevent covid-19 (coronavirus) and steps we can take to protect ourselves. <https://www.conehealth.com/services/primary-care/social-distancing-faq-how-it-helps-prevent-covid-19-coronavirus-/>, May 2020.
- 34 Patrick Healy and Karol Lynch. Two fixed-parameter tractable algorithms for testing upward planarity. *Int. J. Found. Comput. Sci.*, 17(5):1095–1114, 2006. doi:10.1142/S0129054106004285.
- 35 Petr Hlinený and Marek Dernár. Crossing number is hard for kernelization. In *Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *LIPICs*, pages 42:1–42:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

- 36 Petr Hlinený and Abhisekh Sankaran. Exact crossing number parameterized by vertex cover. In *Graph Drawing and Network Visualization (GD 2019)*, Lecture Notes in Computer Science. Springer, 2019. To appear. [arXiv:1906.06048](https://arxiv.org/abs/1906.06048).
- 37 Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.
- 38 Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In *Symposium on Theory of Computing (STOC 2007)*, pages 382–390. ACM, 2007.
- 39 Fabian Klute and Martin Nöllenburg. Minimizing crossings in constrained two-sided circular graph layouts. In *Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *LIPICs*, pages 53:1–53:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 40 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. Subexponential-time and FPT algorithms for embedded flat clustered planarity. In *Graph-Theoretic Concepts in Computer Science – 44th International Workshop, WG 2018, Cottbus, Germany, June 27-29, 2018, Proceedings*, pages 111–124, 2018. doi:10.1007/978-3-030-00256-5\_10.
- 41 Giordano Da Lozzo, David Eppstein, Michael T. Goodrich, and Siddharth Gupta. C-planarity testing of embedded clustered graphs with bounded dual carving-width. In *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, pages 9:1–9:17, 2019. doi:10.4230/LIPICs.IPEC.2019.9.
- 42 Lisa Lockerd Maragakis. Coronavirus, social and physical distancing and self-quarantine. URL: <https://www.hopkinsmedicine.org/health/conditions-and-diseases/coronavirus/coronavirus-social-distancing-and-self-quarantine>, July 2020.
- 43 Md Saidur Rahman, Shin-ichi Nakano, and Takao Nishizeki. Rectangular grid drawings of plane graphs. *Computational Geometry*, 10(3):203–220, 1998.
- 44 Md. Saidur Rahman, Takao Nishizeki, and Mahmuda Naznin. Orthogonal drawings of plane graphs without bends. *J. Graph Algorithms Appl.*, 7(4):335–362, 2003. doi:10.7155/jgaa.00074.
- 45 Sadiq M Sait and Habib Youssef. *VLSI physical design automation: theory and practice*, volume 6. World Scientific Publishing Company, 1999.
- 46 Nathan R Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.
- 47 Paul Turán. A note of welcome. *Journal of Graph Theory*, 1(1):7–9, 1977. doi:10.1002/jgt.3190010105.
- 48 Christopher Umans and William Lenhart. Hamiltonian cycles in solid grid graphs. In *Proceedings 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 496–505. IEEE, 1997.
- 49 Martin Zachariasen. A catalog of hanan grid problems. *Networks: An International Journal*, 38(2):76–83, 2001.





# An Improved Approximation Algorithm for the Matching Augmentation Problem

Joseph Cheriyan ✉ 🏠

Comb. & Opt. Dept., University of Waterloo, Canada

Robert Cummings

Comb. & Opt. Dept., University of Waterloo, Canada

Jack Dippel

Mathematics & Statistics, McGill University, Montreal, Canada

Jasper Zhu

Comb. & Opt. Dept., University of Waterloo, Canada

---

## Abstract

---

We present a  $\frac{5}{3}$ -approximation algorithm for the matching augmentation problem (MAP): given a multi-graph with edges of cost either zero or one such that the edges of cost zero form a matching, find a 2-edge connected spanning subgraph (2-ECSS) of minimum cost.

A  $\frac{7}{4}$ -approximation algorithm for the same problem was presented recently, see Cheriyan, et al., “The matching augmentation problem: a  $\frac{7}{4}$ -approximation algorithm,” *Math. Program.*, 182(1):315–354, 2020.

Our improvement is based on new algorithmic techniques, and some of these may lead to advances on related problems.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis

**Keywords and phrases** 2-Edge connected graph, 2-edge covers, approximation algorithms, connectivity augmentation, forest augmentation problem, matching augmentation problem, network design

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.38

**Related Version** *Full Version:* <https://arxiv.org/abs/2007.11559>

**Funding** *Joseph Cheriyan:* NSERC, RGPIN-2019-04197

**Acknowledgements** We are grateful to several colleagues for their careful reading of preliminary drafts and for their comments.

## 1 Introduction

The design and analysis of algorithms for problems in network design is a core topic in Theoretical Computer Science and Combinatorial Optimization. Algorithmic research on problems such as the minimum spanning tree problem and the Traveling Salesman Problem (TSP) started decades ago and is a thriving area even today. One of the key problems in this area is the minimum-cost 2-ECSS (2-edge connected spanning subgraph) problem: Given an undirected graph  $G = (V, E)$  and a nonnegative cost for each edge  $e \in E$ , denoted  $\text{cost}(e)$ , find a minimum-cost spanning subgraph  $H = (V, F)$ ,  $F \subseteq E$ , that is 2-edge connected. Throughout, we use  $n := |V|$  to denote the number of nodes of  $G$ . (Recall that a graph is 2-edge connected if it is connected and has no “cut edges”, or equivalently, each of its nontrivial cuts has  $\geq 2$  edges.) This problem is NP-hard, and the best approximation guarantee known, due to [22], is 2.

On the other hand, the best “hardness of approximation threshold” known is much smaller; for example, it is  $(1 + \frac{\rho_{VC3}}{104})$  for the unweighted problem, where  $1 + \rho_{VC3}$  is the “hardness of approximation threshold” for the minimum vertex cover problem on a graph with



© Joseph Cheriyan, Robert Cummings, Jack Dippel, and Jasper Zhu;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 38; pp. 38:1–38:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

maximum degree 3, [10, Theorem 5.2]. Also, the best lower bound known on the integrality ratio of the standard LP relaxation (for minimum-cost 2-ECSS) is around 1.5 (thus, well below 2), see [5].

## 1.1 FAP, TAP and MAP

Given this significant gap between the lower bounds and the upper bounds, research in this area over the last two decades has focused on the case of zero-one cost functions (every edge has a cost of zero or one). Let us call an edge  $e \in E$  with  $\text{cost}(e) = 0$  a zero-edge, and let us call an edge  $e \in E$  with  $\text{cost}(e) = 1$  a unit-edge. Intuitively, the zero-edges define some existing network that we wish to augment (with unit-edges) such that the augmented network is resilient to the failure of any one edge. We may assume that the zero-edges form a forest; otherwise, there is at least one cycle  $C$  formed by the zero edges, and in that case, we may contract  $C$ , solve the problem on the resulting graph  $G/C$ , find a solution (edge set)  $F$ , and return  $F \cup C$  as a solution of the original problem. Consequently, the minimum-cost 2-ECSS problem with a zero-one cost function is called the *Forest Augmentation Problem* or FAP. The challenge is to design an approximation algorithm with guarantee strictly less than 2 for FAP.

A well known special case of FAP is TAP, the *Tree Augmentation Problem*: the set of zero-edges forms a spanning tree. The first publication to break the “2-approximation barrier” for TAP is [14] (2003), and since then there have been several important advances, including recent work, see [11, 17, 1, 19, 4, 9, 13].

Recently, see [2], there has been progress on another important (in our opinion) special case of FAP called the *Matching Augmentation Problem* or MAP: Given a multi-graph with edges of cost either zero or one such that the zero-edges form a matching, find a 2-ECSS of minimum cost. From the view-point of approximation algorithms, MAP is “complementary” to TAP, in the sense that the forest formed on  $V(G)$  by the zero-edges has many connected components, each with one node or two nodes, whereas this forest has only one connected component in TAP.

## 1.2 Previous literature and possible approaches for attacking MAP

There is a large body of work on network design and the design of algorithms (for finding optimal solutions, as well as for finding approximately optimal solutions); see the books in the area [20, 25, 18]. Unfortunately, none of these results and methods has helped to break the “2-approximation barrier” for FAP (to the best of our knowledge).

Powerful and versatile methods such as the *primal-dual method* (see [25, 12]) and the *iterative rounding method* (see [18, 16]) have been developed for problems in network design, but the provable approximation guarantees for these methods are  $\geq 2$ . (These methods work by rounding LP relaxations; informally speaking, the approximation guarantee is proved via an upper bound of 2 per iteration on the “integral cost incurred” versus the “chargeable LP cost”, and it is plausible that the factor of 2 cannot be improved for this type of analysis.)

Combinatorial methods that may also exploit lower-bounds from LP relaxations have been developed for approximation algorithms for unweighted minimum-cost 2-ECSS, e.g.,  $\frac{4}{3}$ -approximation algorithms are presented in [24, 21, 15]. For the unweighted problem, there is a key lower bound of  $n$  on  $\text{opt}$  (since any solution must have  $\geq n$  edges, each of cost one). This fails to hold for MAP; indeed, the analogous lower bound on  $\text{opt}$  is  $\frac{1}{2}n$  for MAP. This rules out any direct extension of these combinatorial methods (for the unweighted problem) to prove approximation guarantees below 2 for MAP.

Recently, Traub and Zenklusen [23] presented an approximation algorithm for Weighted TAP with guarantee  $1 + (\ln 2) + \epsilon < 1.7$ , via a so-called relative greedy algorithm, based on previous work by Cohen and Nutov [6] that, in turn, is based on earlier work by Zelikovsky on the Steiner Tree problem [26]. While the result of Traub and Zenklusen is a major advance in the area, the core ideas of the relative greedy algorithm are well known, and, as yet, there is no advance on FAP via relative greedy algorithms. (Informally speaking, Weighted TAP is a special case of SCP (the Set Covering Problem), and the method of Cohen and Nutov exploits special properties of the SCP instances associated with TAP, whereas, to the best of our knowledge, this paradigm does not extend to FAP.)

### 1.3 Our results and techniques

Our main contribution is a  $\frac{5}{3}$ -approximation algorithm for MAP, improving on the  $\frac{7}{4}$  approximation guarantee of [2], see Theorems 12, 13.

At a high level (hiding many important points), our algorithm is based on a “discharging scheme” where we compute a lower bound on  $opt$  (the optimal value) and fix a “budget” of  $\alpha$  times this lower bound (where  $\alpha > 1$  is a constant), “scatter” this budget over the graph  $G$ , use the budget to buy some edges to obtain a “base graph”, then traverse the “base graph” and buy more edges to augment the “base graph”, so that (eventually) we have a 2-ECSS whose cost is within the budget of  $\alpha$  times our lower bound. We mention that several of the results cited above are based on discharging schemes, e.g., [24, 11, 17, 15, 2]. In some more detail, but still at a high level, we follow the method of [2]. (Our presentation can be read independently of [2]; we have repeated a few definitions and statements of results from [2].) We first pre-process the input instance  $G$ , with the goal of removing all “obstructions” (e.g., cut nodes), and we decompose  $G$  into a list of “well structured” sub-instances  $G_1, G_2, \dots$  that are pairwise edge-disjoint. Now, consider one of these sub-instances  $G_i$  (it has none of the “obstructions”). We compute a subgraph  $H_i$  whose cost is a lower bound on  $opt(G_i)$ . Finally, we augment  $H_i$  to make it 2-edge connected, and use a credit-based analysis to prove an approximation guarantee.

A 2-edge cover is a subgraph that has at least two edges incident to every node. The minimum-cost 2-edge cover is the key subgraph used as a lower bound in our algorithm; we refer to it as D2. (D2 can be computed in polynomial time via extensions of Edmonds’ algorithm for computing a minimum-cost perfect matching.) Since every 2-ECSS is a 2-edge cover, we have  $\text{cost}(\text{D2}) \leq opt$ . So, by transforming D2 to a 2-ECSS of cost  $\leq \frac{5}{3}\text{cost}(\text{D2})$ , we achieve our claimed approximation guarantee.

Our pre-processing includes several new ideas, and moreover, it is essential to handle new “obstructions” that are not handled in [2]; indeed, [2] has tight examples such that  $opt/\text{cost}(\text{D2}) \geq \frac{7}{4} - \epsilon$  (for some  $\epsilon > 0$ ). Although our algorithm handles several new “obstructions”, our analysis and proofs for the pre-processing are simple. One of our key tools (for our pre-processing analysis) is to prove a stronger guarantee of  $\max(opt, \frac{5}{3}opt - 2)$  rather than just  $\frac{5}{3}opt$ . When we analyze our decomposition of an instance into sub-instance(s), then this additive term of  $-2$  is useful in combining solutions back together at the end of the algorithm (when we “undo” the decomposition of  $G$  into sub-instances  $G_1, G_2, \dots$ ). (Our analysis of pre-processing is omitted in this paper, due to space constraints; it is given in the full paper, [3].)

Our main algorithm (following [2]) has two key subroutines for transforming a D2 of a “well structured” sub-instance  $G_i$  to a 2-ECSS of  $G_i$  while ensuring that the total cost is  $\leq \frac{5}{3}\text{cost}(\text{D2})$ .

- (i) **Bridge covering step:** The goal is to augment edges such that each connected component of our “current solution graph”  $H_i$  is 2-edge-connected; we start with  $H_i := D2(G_i)$ . Our analysis is based on a new and simple credit scheme that bypasses some difficulties in the credit scheme of [2].
- (ii) **Gluing step:** Finally, this step merges the (already 2-edge connected) connected components of  $H_i$  to form a 2-ECSS of the sub-instance  $G_i$ . A key part of this step handles so-called “small 2ec-blocks”; these are cycles of cost 2 that occur as connected components of  $D2(G_i)$  and stay unchanged through the bridge covering step. Observe that a “small 2ec-block” has only  $\frac{4}{3}$  credits (it has a “budget” of  $(\frac{5}{3})(2)$ , and after paying for its two unit-edges, there is only  $\frac{4}{3}$  credits available). Our gluing step applies a careful swapping of unit-edges for the “small 2ec-blocks” while it merges the connected components of  $H_i$  into a 2-ECSS, and ensures that the net augmentation cost does not exceed the available credit.

There are well-known polynomial time algorithms for implementing all of the basic computations in this paper, see [20]. We state this explicitly in all relevant results (e.g., Theorem 12), but we do not elaborate on this elsewhere.

## 2 Preliminaries

This section has definitions and preliminary results. Our notation and terms are consistent with [7], and readers are referred to that text for further information.

Let  $G = (V, E)$  be a (loop-free) multi-graph with edges of cost either zero or one such that the edges of cost zero form a matching. We take  $G$  to be the input graph, and we use  $n$  to denote  $|V(G)|$ . Let  $M$  denote the set of edges of cost zero. Throughout, the reader should keep in mind that  $M$  is a matching; this fact is used in many of our proofs without explicit reminders. We call an edge of  $M$  a *zero-edge* and we call an edge of  $E - M$  a *unit-edge*.

We denote the cost of an edge  $e$  of  $G$  by  $\text{cost}(e)$ . For a set of edges  $F \subseteq E(G)$ ,  $\text{cost}(F) := \sum_{e \in F} \text{cost}(e)$ , and for a subgraph  $G'$  of  $G$ ,  $\text{cost}(G') := \sum_{e \in E(G')} \text{cost}(e)$ .

For ease of exposition, we often denote an instance  $G, M$  by  $G$ ; then, we do not have explicit notation for the edge costs of the instance, but the edge costs are given implicitly by  $\text{cost} : E(G) \rightarrow \{0, 1\}$ , and  $M$  is given implicitly by  $\{e \in E(G) : \text{cost}(e) = 0\}$ .

For a positive integer  $k$ , we use  $[k]$  to denote the set  $\{1, \dots, k\}$ .

We use the standard notion of contraction of an edge, see [20, p.25]. For a graph  $H$  and a set of its nodes  $S$ ,  $\Gamma_H(S) := \{w \in V(H) - S : v \in S, vw \in E(H)\}$ , thus,  $\Gamma_H(S)$  denotes the set of neighbours of  $S$ . For a graph  $H$  and a set of nodes  $S \subseteq V(H)$ ,  $\delta_H(S)$  denotes the set of edges that have one end node in  $S$  and one end node in  $V(H) - S$ . Moreover,  $H[S]$  denotes the subgraph of  $H$  induced by  $S$ , and  $H - S$  denotes the subgraph of  $H$  induced by  $V(H) - S$ . For a graph  $H$  and a set of edges  $F \subseteq E(H)$ ,  $H - F$  denotes the graph  $(V(H), E(H) - F)$ . For any subgraph  $K$  of a graph  $H$  with  $V(K) \subsetneq V(H)$ , an *attachment* of  $K$  is a node of  $K$  that has a neighbour in  $V(H) - V(K)$ .

We may use relaxed notation for singleton sets, and, we may not distinguish between a subgraph and its node set; for example, given a graph  $H$  and a set  $S$  of its nodes, we use  $E(S)$  to denote the edge set of the subgraph of  $H$  induced by  $S$ .

## 2.1 2EC, 2NC, bridges and D2

A multi-graph  $H$  is called  $k$ -edge connected if  $|V(H)| \geq 2$  and for every  $F \subseteq E(H)$  of size  $< k$ ,  $H - F$  is connected. Thus,  $H$  is 2-edge connected if it has  $\geq 2$  nodes and the deletion of any one edge results in a connected graph. A multi-graph  $H$  is called  $k$ -node connected if  $|V(H)| > k$  and for every  $S \subseteq V(H)$  of size  $< k$ ,  $H - S$  is connected. We use the abbreviations *2EC* for “2-edge connected,” and *2NC* for “2-node connected.”

We assume w.l.o.g. that the input  $G$  is 2EC. Moreover, for some (but not all) of our discussions, we assume that there are  $\leq 2$  copies of each edge (in the multi-graph under consideration); this is justified since an edge-minimal 2-ECSS cannot have three or more copies of any edge (see Proposition 1 below).

For any instance  $H$ , let  $opt(H)$  denote the minimum cost of a 2-ECSS of  $H$ . When there is no danger of ambiguity, we use  $opt$  rather than  $opt(H)$ .

By a *bridge* we mean an edge of a connected (sub)graph whose removal results in two connected components, and by a *cut node* we mean a node of a connected (sub)graph whose deletion results in two or more connected components. We call a bridge of cost zero a *zero-bridge* and we call a bridge of cost one a *unit-bridge*.

By a *2ec-block* we mean a maximal connected subgraph with two or more nodes that has no bridges. We call a 2ec-block *pendant* if it is incident to exactly one bridge. We call a 2ec-block *small* if it has  $\leq 2$  unit-edges, and we call it *large* otherwise.

For a 2EC graph  $G$  and a cut node  $v$  of  $G$ , a *2ec- $v$ -block* means the subgraph of  $G$  induced by  $\{v\} \cup V(C)$  where  $C$  is one of the connected components of  $G - v$ .

The next result characterizes edges that are not essential for 2-edge connectivity.

► **Proposition 1.** *Let  $H$  be a 2EC graph and let  $e = vw$  be an edge of  $H$ . If  $H - e$  has two edge-disjoint  $v, w$  paths, then  $H - e$  is 2EC.*

The next lemma partially characterizes the cuts of size  $\leq 2$  in a graph obtained by “uncontracting” a set of nodes of a 2EC graph.

► **Lemma 2.** *Let  $H$  be a 2EC graph and let  $C \subsetneq V(H)$  be a set of nodes such that the induced subgraph  $H[C]$  is connected. Suppose that  $H^*$  is a 2-ECSS of  $H/C$ . Let  $H'$  be the spanning subgraph of  $H$  with edge set  $E(C) \cup E(H^*)$ . Then  $H'$  is a connected graph such that each of its bridges (if any) is in  $E(C)$ .*

By a *2-edge cover* (of  $G$ ) we mean a set of edges  $F$  of  $G$  such that each node  $v$  is incident to at least two edges of  $F$  (i.e.,  $F \subseteq E(G) : |\delta_F(v)| \geq 2, \forall v \in V(G)$ ). By  $D2(G)$  we mean any minimum-cost 2-edge cover of  $G$  ( $G$  may have several minimum-cost 2-edge covers, and  $D2(G)$  may refer to any one of them); when there is no danger of ambiguity, we use  $D2$  rather than  $D2(G)$ .

By a *bridgeless 2-edge cover* (of  $G$ ) we mean a 2-edge cover (of  $G$ ) that has no bridges.

The next result follows from Theorem 34.15 in [20, Chapter 34].

► **Proposition 3.** *There is a polynomial-time algorithm for computing  $D2$ .*

The next result states the key lower bound used by our approximation algorithm.

► **Lemma 4.** *Let  $H$  be any 2EC graph. Then we have  $opt(H) \geq cost(D2(H))$ .*

For any fixed positive integer  $z$  (thus,  $z = O(1)$ ) and any instance of MAP, in time  $O(1)$ , we can determine whether the instance has  $opt > z$ , and if not, then we can find an optimal 2-ECSS of the instance.

► **Lemma 5.** *Let  $H$  be an instance of MAP, and let  $z$  be a fixed positive integer. There is an  $O(1)$ -time algorithm to determine whether  $\text{opt}(H) \geq z$ . Moreover, if  $\text{opt}(H) \leq z$ , then a minimum-cost 2-ECSS of  $H$  can be found in  $O(1)$  time.*

## 2.2 Obstructions for the approximation guarantee

There are several obstructions (e.g., cut nodes) that prevent our algorithm (and analysis) from achieving our target approximation factor of  $\frac{5}{3}$ . We eliminate all such obstructions in a pre-processing step that takes the given instance  $G$  of MAP (the input) and replaces it by a list of sub-instances  $G_1, G_2, \dots$ , such that (a) none of the obstructions occurs in a sub-instance  $G_i$ , (b) the edge-sets of the sub-instances are pairwise-disjoint, and (c) given a 2-ECSS of each sub-instance  $G_i$  of approximately optimal cost, we can construct a 2-ECSS of  $G$  of cost  $\leq \frac{5}{3}\text{opt}(G)$ . (Precise statements are given later.) The obstructions for our algorithm are:

- |                      |              |
|----------------------|--------------|
| (i) cut nodes,       | (v) S{3, 4}, |
| (ii) parallel edges, | (vi) R4,     |
| (iii) zero-cost S2,  | (vii) R8.    |
| (iv) unit-cost S2,   |              |

Below, we formally define each of these obstructions. Four of these obstructions were introduced in [2], and readers interested in a deeper understanding may refer to that paper, in particular, see the remark after [2, Theorem 6] and see [2, Figure 2] for instances  $G$  of MAP that contain cut nodes, parallel edges, zero-cost S2s, or R4s such that  $\text{opt}(G)/\text{cost}(\text{D2}(G)) \approx 2$ ; informally speaking, an approximation algorithm based on the lower bound  $\text{cost}(\text{D2}(G))$  on  $\text{opt}(G)$  fails to beat the approximation threshold of 2 in the presence of any of these four obstructions.

► **Definition 6.** *By a zero-cost S2 (also called a bad-pair), we mean a zero-edge  $e$  and its end nodes,  $u, v$ , such that  $G - \{u, v\}$  has  $\geq 2$  connected components.*

► **Definition 7.** *By a unit-cost S2, we mean a unit-edge  $e$  and its end nodes,  $u, v$ , such that  $G - \{u, v\}$  has  $\geq 2$  connected components; moreover, in the graph  $G/\{u, v\}$ , there exist two distinct 2ec- $\hat{v}$ -blocks  $B_1, B_2$  incident to the contracted node  $\hat{v}$  such that  $\text{opt}(B_i) \geq 3$  and  $B_i$  has a zero-edge incident to the contracted node,  $\forall i \in [2]$ .*

► **Definition 8.** *By an S{3, 4}, we mean an induced 2NC subgraph  $C$  of  $G$  with  $|V(C)| \in \{3, 4\}$  that has a spanning cycle of cost two such that  $G - V(C)$  has  $\geq 2$  connected components, and the cut  $\delta(V(C))$  has no zero-edges; moreover, in the graph  $G/C$ , there exist two distinct 2ec- $\hat{v}$ -blocks  $B_1, B_2$  incident to the contracted node  $\hat{v}$  that have  $\text{opt}(B_1) \geq 3$  and  $\text{opt}(B_2) \geq 3$ .*

► **Definition 9.** *By an R4 (also called a redundant 4-cycle), we mean an induced subgraph  $C$  of  $G$  with four nodes such that  $V(C) \neq V(G)$ ,  $C$  contains a 4-cycle of cost two, and  $C$  contains a pair of nonadjacent nodes that each have degree two in  $G$ .*

► **Definition 10.** *By an R8, we mean an induced subgraph  $C$  of  $G$  with eight nodes such that  $V(C) \neq V(G)$ ,  $C$  contains two disjoint 4-cycles  $C_1, C_2$  with  $\text{cost}(C_i) = 2, \forall i \in [2]$ ,  $C$  has exactly two attachments  $a_1, a_2$  where  $a_i \in C_i, \forall i \in [2]$ , and both end nodes of the (unique) unit-edge of  $C_i - a_i$  are adjacent to  $C_{3-i}, \forall i \in [2]$ .*

### 3 Outline of the algorithm

This section has an outline of our algorithm. We start by defining an instance of MAP $\star$ .

► **Definition 11.** *An instance of MAP $\star$  is an instance of MAP with  $\geq 12$  nodes that contains*

- no cut nodes,
- no parallel edges,
- no zero-cost S2,
- no unit-cost S2,
- no S{3, 4},
- no R4, and
- no R8.

In this section and Section 4, we sketch how to “decompose” any instance of MAP  $G$  with  $|V(G)| \geq 12$  into a collection of instances  $G_1, \dots, G_k$  of MAP such that (a) either  $|V(G_i)| < 12$  or  $G_i$  is an instance of MAP $\star$ ,  $\forall i \in [k]$ , (b) the edge sets  $E(G_1), \dots, E(G_k)$  are pairwise disjoint (thus  $E(G_1), \dots, E(G_k)$  forms a subpartition of  $E(G)$ ), and (c) a 2-ECSS  $H$  of  $G$  can be obtained by computing 2-ECSSes  $H_1, \dots, H_k$  of  $G_1, \dots, G_k$ . Moreover, the approximation guarantee is preserved, meaning that  $\text{cost}(H) \leq \frac{5}{3} \text{opt}(G) - 2$  provided  $\text{cost}(H_i) \leq \max(\text{opt}(G_i), \frac{5}{3} \text{opt}(G_i) - 2), \forall i \in [k]$ .

---

#### Algorithm (outline).

- (0) apply the pre-processing steps (see below and see Section 4) to obtain a collection of instances  $G_1, \dots, G_k$  such that either  $|V(G_i)| < 12$  or  $G_i$  is an instance of MAP $\star$ ,  $\forall i \in [k]$ ;  
     **for** each  $G_i$  ( $i = 1, \dots, k$ ),  
     **if**  $|V(G_i)| < 12$
  - (1) exhaustively compute an optimum 2-ECSS  $H_i$  of  $G_i$  via Lemma 5;  
     **else**
  - (2.1) compute  $D2(G_i)$  in polynomial time (w.l.o.g. assume  $D2(G_i)$  contains all zero-edges of  $G_i$ );
  - (2.2) then apply “bridge covering” from Section 5 to  $D2(G_i)$  to obtain a bridgeless 2-edge cover  $\tilde{H}_i$  of  $G_i$ ;
  - (2.3) then apply the “gluing step” from Section 6 to  $\tilde{H}_i$  to obtain a 2-ECSS  $H_i$  of  $G_i$ ;  
     **endif**;
  - endfor**;
  - (3) finally, output a 2-ECSS  $H$  of  $G$  from the union of  $H_1, \dots, H_k$  by undoing the transformations applied in step (0).
- 

The pre-processing of step (0) consists of several reductions; most of these reductions are straightforward, but we have to prove that the approximation guarantee is preserved when we “undo” each of these reductions. These proofs are given in the full paper, [3].

---

#### Pre-processing – Step (0) of Algorithm.

**While** the current list of sub-instances  $G_1, G_2, \dots$  has a sub-instance  $G_i$  that has  $\geq 12$  nodes and is not an instance of MAP $\star$  (assume that  $G_i$  is 2EC):

**if**  $G_i$  is not 2NC:

- (i) (handle a cut-node)  
     let  $v$  be a cut node of  $G_i$ , and let  $B_1, \dots, B_k$  be the 2ec- $v$ -blocks of  $G_i$ ; replace  $G_i$  by  $B_1, \dots, B_k$  in the current list;
- else** apply exactly one of the following steps to  $G_i$ :



- (ii) (handle a pair of parallel edges)
    - let  $\{e, f\}$  be a pair of parallel edges of  $G_i$  (one of the edges in  $\{e, f\}$  is a unit-edge);
    - discard a unit-edge of  $\{e, f\}$  from  $G_i$ ;
  - (iii) (handle an “S obstruction”)
    - (a) (handle a unit-cost S2)
    - (b) (handle a zero-cost S2)
    - (c) (handle an  $S\{3, 4\}$ )

let  $C$  denote a subgraph of  $G_i$  that is, respectively, (a) a unit-cost S2, (b) a zero-cost S2, or (c) an  $S\{3, 4\}$ ;

contract  $C$  to obtain  $G_i/C$  and let  $\hat{v}$  denote the contracted node; let  $B_1, \dots, B_k$  be the  $2\text{ec-}\hat{v}$ -blocks of  $G_i/C$ ; replace  $G_i$  by  $B_1, \dots, B_k$  in the current list;
  - (iv) (handle an “R obstruction”)
    - (a) (handle an R4)
    - (b) (handle an R8)

let  $C$  denote a subgraph of  $G_i$  that is, respectively, (a) an R4, or (b) an R8;

contract  $C$  to obtain  $G_i/C$ , and replace  $G_i$  by  $G_i/C$  in the current list;
- 

Our  $\frac{5}{3}$  approximation algorithm for MAP follows from the following theorem.

► **Theorem 12.** *Given an instance of  $\text{MAP}\star$   $G'$ , there is a polynomial-time algorithm that obtains a 2-ECSS  $H'$  such that  $\text{cost}(H') \leq \max(\text{opt}(G'), \frac{5}{3}\text{opt}(G') - 2)$ .*

We use a credit scheme to prove this theorem; the details are presented in Sections 5 and 6. The algorithm starts with  $\text{D2}(G')$  as the current graph, and assigns  $\frac{5}{3}$  tokens to each unit-edge of  $\text{D2}(G')$ ; each such edge keeps one unit to pay for itself and the other  $\frac{2}{3}$  is taken to be credit of the edge; thus, the algorithm has  $\frac{2}{3}\text{cost}(\text{D2}(G'))$  credits at the start; the algorithm uses the credits to pay for the augmenting edges “bought” in steps (2.2) or (2.3) (see the outline); also, the algorithm may “sell” unit-edges of the current graph (i.e., such an edge is permanently discarded and is not contained in the 2-ECSS output by the algorithm).

The factor  $\frac{5}{3}$  in our approximation guarantee is tight in the sense that there exists an instance  $G$  of  $\text{MAP}\star$  such that  $\text{opt}(G)/\text{cost}(\text{D2}(G)) \geq \frac{5}{3} - \epsilon$ , for any small positive number  $\epsilon$ . The instance  $G$  consists of a root 2ec-block  $B_0$ , say a 6-cycle of cost 6,  $v_1, \dots, v_6, v_1$ , and  $\ell \gg 1$  copies of the following gadget that are attached to  $B_0$ . The gadget consists of a 6-cycle  $C = u_1, \dots, u_6, u_1$  of cost 3 that has alternating zero-edges and unit-edges; moreover, there are three unit-edges between  $C$  and  $B_0$ :  $v_1u_1, v_3u_3, v_5u_5$ . Observe that a (feasible) 2-edge cover of this instance consists of  $B_0$  and the 6-cycle  $C$  of each copy of the gadget, and it has cost  $6 + 3\ell$ . Observe that for any 2-ECSS and for each copy of the gadget, the six edges of  $C$  as well as (at least) two of the edges between  $C$  and  $B_0$  are contained in the 2-ECSS. Thus,  $\text{opt}(G) \geq 6 + 5\ell$ , whereas  $\text{cost}(\text{D2}(G)) \leq 6 + 3\ell$ .

#### 4 Pre-processing

Due to space constraints, we only state our main result for pre-processing and skip several lemmas that are needed to prove this result. These lemmas and their proofs are given in the full paper, [3].

► **Theorem 13.** *Suppose that there is an approximation algorithm that given an instance  $H$  of  $\text{MAP}\star$ , finds a 2-ECSS of cost  $\leq \max(\text{opt}(H), \alpha \text{opt}(H) - 2)$ . Then, given an instance  $G$  of MAP, there is a polynomial-time algorithm to find a 2-ECSS of cost  $\leq \max(\text{opt}(G), \alpha \text{opt}(G) - 2)$ .*

**5 Bridge covering**

The results in this section are based on the prior results and methods of [2, 8], but the goal in these previous papers is to obtain an approximation guarantee of  $\frac{7}{4}$  for MAP, whereas our goal is an approximation guarantee of  $\frac{5}{3}$ . Our credit invariant is presented in Section 5.1 below, and it is based on the credit invariant in [8].

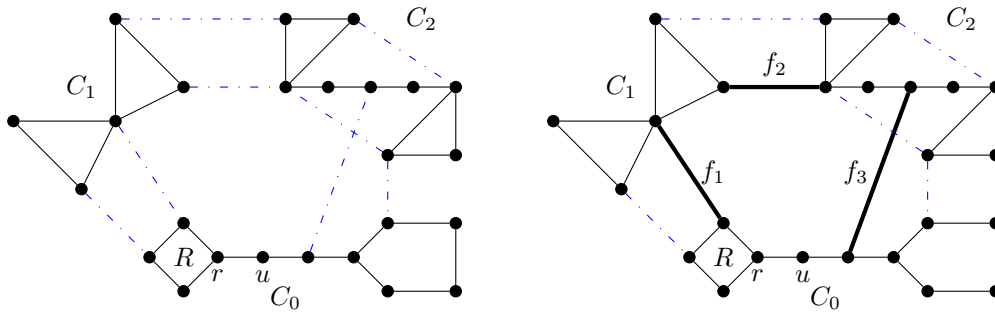
In this section and in Section 6, we assume that the input is an instance of MAP $\star$ . For notational convenience, we denote the input by  $G$ . Recall that  $G$  is a simple, 2NC graph on  $\geq 12$  nodes, and  $G$  has no zero-cost S2, no unit-cost S2, no  $S\{3, 4\}$ , no R4, and no R8. Recall that a 2ec-block is called small if it has  $\leq 2$  unit-edges, and is called large otherwise. Since  $G$  is 2NC and simple, a small 2ec-block is either a 3-cycle with one zero-edge and two unit-edges, or a 4-cycle with alternating zero-edges and unit-edges.

Each unit-edge  $e$  of D2 starts with  $\frac{5}{3}$  tokens, and from this, one unit is kept aside (to pay for  $e$ ), and the other  $\frac{2}{3}$  is defined to be the credit of  $e$ . Our overall goal is to find a 2-ECSS  $H'$  of  $G$  of cost  $\leq \frac{5}{3}\text{cost}(D2)$ , and we keep  $\frac{2}{3}\text{cost}(D2)$  from our budget in the form of credit while using the rest of our budget for “buying” the unit-edges of D2. We use the credit for “buying” unit-edges that are added to our current graph during the bridge covering step or the gluing step. (In the gluing step, we may “sell” unit-edges of our current graph, that is, we may permanently discard some unit-edges of our current graph; thus, our overall budgeting scheme does not rely solely on credits.)

We use  $H$  to denote the current graph of the bridge covering step; initially,  $H = D2$ .

The outcome of the bridge covering step is stated in the following result.

► **Proposition 14.** *At the termination of the bridge covering step,  $H$  is a bridgeless 2-edge cover; moreover, every small 2ec-block of  $H$  has  $\geq \frac{4}{3}$  credits and every large 2ec-block of  $H$  has  $\geq 2$  credits. The bridge covering step can be implemented in polynomial time.*



■ **Figure 1** Illustration of an iteration of our bridge-covering step. Solid lines indicate edges of the graph  $H$ , and (blue) dash-dotted lines indicate edges of  $E(G) - E(H)$ . The pseudo-ear  $R, f_1, C_1, f_2, C_2, f_3$  covers the bridge  $ru$  of  $C_0$  (right subfigure). Thick lines indicate the edges  $f_1, f_2, f_3$  of the pseudo-ear.

A brief overview of the bridge covering step follows: The goal is to add “new” edges to  $H$  to obtain a bridgeless 2-edge cover, and to pay for these “new” edges from credits available in  $H$  while preserving a credit invariant (stated below). In each iteration, we pick a connected component  $C_0$  of  $H$  such that  $C_0$  has a bridge, then we pick any pendant 2ec-block  $R$  of  $C_0$ , then we add a set of edges  $\{f_1, \dots, f_k\} \subseteq E(G) - E(H)$  that “covers” the unique bridge of  $C_0$  incident to  $R$  (possibly,  $k = 1$ ). Informally speaking, this step merges  $k - 1$  connected components  $C_1, C_2, \dots, C_{k-1}$  of  $H$  with  $C_0$  (see the discussion below). Each

connected component of  $H$  has one unit of so-called c-credit (by the credit invariant stated below), and we take this credit from each of  $C_1, C_2, \dots, C_{k-1}$  and use that to pay for  $k-1$  of the newly added edges. The challenge is to find one more unit of credit (since we added  $k$  edges), and this is the focus of our analysis given below.

By [2, Section 5.1, Proposition 5.20], we may assume without loss of generality that D2 has the following properties:

(\*) D2 contains all the zero-edges. Every pendant 2ec-block of D2 that is incident to a zero-bridge is a large 2ec-block.

Recall that  $H$  denotes the current graph, and initially,  $H = \text{D2}$ . We call a node  $v$  of  $H$  a *white* node if  $v$  belongs to a 2ec-block of  $H$ , otherwise, we call  $v$  a *black* node.

It is convenient to define the following multi-graphs: let  $\tilde{H}$  be the multi-graph obtained from  $H$  by contracting each 2ec-block  $B_i$  of  $H$  into a single node that we will denote by  $B_i$ . Observe that each connected component of  $\tilde{H}$  is a tree (possibly, an isolated node). We call a node  $v$  of the multigraph  $\tilde{H}$  black if it is the image of a black node of  $H$ , otherwise, we call  $v$  a white node. Each 2ec-block of  $H$  maps to a white node of  $\tilde{H}$ . Each bridge of  $H$  maps to a bridge of  $\tilde{H}$ . Clearly, each black node of  $\tilde{H}$  is incident to  $\geq 2$  bridges of  $\tilde{H}$ .

Similarly, let  $\tilde{G}$  be the multi-graph obtained from  $G$  by contracting each 2ec-block  $B_i$  of  $H$  into a single node.

## 5.1 Credit invariant

We re-assign the credits of D2 such that the following credit invariant holds for  $H$  at the start/end of every iteration in the bridge covering step.

For a black node  $v$  of  $H$ , we use  $\deg_H^{(1)}(v)$  to denote the number of unit-bridges incident to  $v$  in  $H$ .

### Credit invariant for $H$

- (a) each connected component is assigned at least one credit (called c-credit);
- (b) each connected component that is a small 2ec-block is assigned  $\frac{1}{3}$  credits (called b-credit);
- (c) every other 2ec-block is assigned at least one credit (called b-credit);
- (d) each black node  $v$  is assigned  $\frac{1}{3} \deg_H^{(1)}(v)$  credits (called n-credit).

Note that the four types of credit are distinct, and the invariant gives lower bounds. For example, a connected component that is a large 2ec-block has one c-credit and at least one b-credit.

► **Lemma 15.** *The initial credits of D2 can be re-assigned such that (the initial)  $H = \text{D2}$  satisfies the credit invariant.*

## 5.2 Analysis of a pseudo-ear augmentation

In this subsection, our goal is to show that a so-called pseudo-ear augmentation can be applied to  $H$  whenever a connected component of  $H$  has a bridge, such that the cost of the newly added unit-edges is paid from the credits released by the pseudo-ear augmentation, and moreover, the credit invariant is preserved.

In the graph  $H$ , let  $C_0$  be a connected component that has a bridge, let  $R$  be a pendant 2ec-block of  $C_0$ , and let  $ru$  be the unique bridge (of  $C_0$ ) incident to  $R$ , where  $r \in V(R)$ .

► **Definition 16.** A pseudo-ear of  $H$  w.r.t.  $C_0$  starting at  $R$  is a sequence  $R, f_1, C_1, f_2, C_2, \dots, f_{k-1}, C_{k-1}, f_k$ , where  $C_0, C_1, \dots, C_{k-1}$  are distinct connected components of  $H$ ,  $f_1, \dots, f_k \in E(G) - E(H)$ , each  $f_i, i \in [k-1]$ , has one end node in  $C_{i-1}$  and the other end node in  $C_i$ ,  $f_1$  has an end node in  $R$ , and  $f_k$  has one end node in  $C_{k-1}$  and one end node in  $C_0 - V(R)$ . The end node of  $f_k$  in  $C_0 - V(R)$  is called the head node of the pseudo-ear.

Any shortest (w.r.t. the number of edges) path of  $C_0$  between  $r$  and the head node of the pseudo-ear is called the witness path of the pseudo-ear.

Our plan is to find a pseudo-ear (as above) such that for any witness path  $Q$ , there is at least one unit of credit in  $Q - r$ . Let  $R^{new}$  denote the 2ec-block that results from the addition of the pseudo-ear; thus,  $R^{new}$  contains  $R \cup Q$ . The b-credit of  $R$  is transferred to  $R^{new}$ ; thus,  $R^{new}$  satisfies part (c) of the credit invariant; see Proposition 19 below. After we add the pseudo-ear to  $H$ , the credits of  $Q - r$  are released (they are no longer needed for preserving the credit invariant, because  $Q \cup R$  is merged into  $R^{new}$ ). Informally speaking, we use the credits released from  $Q - r$  to pay for the cost of the last unit-edge added by the pseudo-ear augmentation.

In the graph  $\tilde{G}$ , let  $\tilde{C}_0$  denote the tree corresponding to  $C_0$  and let  $\tilde{R}$  denote the leaf of  $\tilde{C}_0$  corresponding to  $R$ . Let  $\tilde{P}$  be a shortest (w.r.t. the number of edges) path of  $\tilde{G} - E(\tilde{C}_0)$  that has one end node at  $\tilde{R}$  and the other end node at another node of  $\tilde{C}_0$ . Then  $\tilde{P}$  corresponds to a pseudo-ear  $R, f_1, C_1, \dots, C_{k-1}, f_k$ ; the sequence of edges of  $E(\tilde{G}) - E(\tilde{H})$  of  $\tilde{P}$  corresponds to  $f_1, \dots, f_k$  and the sequence of trees  $\tilde{C}_1, \dots, \tilde{C}_{k-1}$  of  $\tilde{P}$  corresponds to  $C_1, \dots, C_{k-1}$ .

It is easy to find a pseudo-ear such that any witness path  $Q$  has  $\geq 2$  edges. To see this, observe that  $G - u$  is connected (since  $G$  is 2NC); let  $P$  be a shortest (w.r.t. the number of edges) path between  $R$  and  $C_0 - V(R)$  in  $G - u$ ; then  $P$  corresponds to our desired pseudo-ear, and the head node is the end node of  $P$  in  $C_0 - u - V(R)$ . Clearly, any path of  $C_0$  between  $r$  and the head node has  $\geq 2$  edges, hence, any witness path of the pseudo-ear has  $\geq 2$  edges.

In each iteration (of bridge covering), we compute a pseudo-ear using a polynomial-time algorithm that is presented in the proof of Proposition 18, see below.

The next lemma is used to lower bound the credit of a witness path.

► **Lemma 17.** Let  $\Psi$  be a pseudo-ear of  $H$  w.r.t.  $C_0$  starting at  $R$ , let  $Q$  be a witness path of  $\Psi$ , and let  $ru$  be unique bridge of  $C_0$  incident to  $R$ . Suppose that  $Q$  satisfies one of the following:

- (a)  $Q$  contains a white node distinct from  $r$ , or
- (b)  $Q$  contains exactly one white node and  $\geq 3$  bridges, or
- (c)  $Q$  contains exactly one white node, exactly two bridges, and a black node  $v$  such that  $\deg_H^{(1)}(v) \geq 2$ .

Then  $Q - r$  has at least one credit, and that credit is not needed for the credit invariant of the graph resulting from the pseudo-ear augmentation that adds  $\Psi$  to  $H$ .

► **Proposition 18.** There is a polynomial-time algorithm for finding a pseudo-ear (of  $H$  w.r.t.  $C_0$  starting at  $R$ ) such that any witness path  $Q$  of the pseudo-ear satisfies one of the three conditions of Lemma 17.

► **Proposition 19.** Suppose that  $H$  satisfies the credit invariant, and a pseudo-ear augmentation is applied to  $H$ . Then the resulting graph  $H^{new}$  satisfies the credit invariant.

**Proof of Proposition 14.** The proof follows from Lemmas 15, 17, and Propositions 18, 19, and the preceding discussion.

Each iteration, i.e., each pseudo-ear augmentation, can be implemented in polynomial time, and the number of iterations is  $\leq |E(D2)|$ .

At the termination of bridge covering, each connected component of  $H$  is a 2ec-block that has one c-credit and either one b-credit, or (in the case of a small 2ec-block)  $\frac{1}{3}$  b-credits. By summing the two types of credit, it follows that each small 2ec-block has  $\frac{4}{3}$  credits and each large 2ec-block has  $\geq 2$  credits. ◀

## 6 The gluing step

In this section, we focus on the gluing step, and we assume that the input is an instance of MAP $\star$ . For notational convenience, we denote the input by  $G$ . Recall that  $G$  is a simple, 2NC graph on  $\geq 12$  nodes, and  $G$  has no zero-cost S2, no unit-cost S2, no S{3, 4}, no R4, and no R8. (In this section, we use all the properties of  $G$  except the absence of unit-cost S2s.)

There are important differences between our gluing step and the gluing step of [2]. Our gluing step (and overall algorithm) beats the  $\frac{7}{4}$  approximation threshold because our pre-processing step eliminates the S{3, 4} obstruction and the R8 obstruction (these obstructions are not relevant to other parts of our algorithm). In the full version of the paper, see [3, Appendix], we present instances  $G$  of MAP that contain S{3, 4}s (respectively, R8s) and contain none of the other six obstructions such that  $\text{opt}(G)/\text{cost}(D2(G)) \approx \frac{7}{4}$ .

We use  $H$  to denote the current graph of the gluing step. At the start of the gluing step,  $H$  is a simple, bridgeless graph of minimum degree two; thus, each connected component of  $H$  is 2EC; clearly, the 2ec-blocks of  $H$  correspond to the connected components of  $H$ . Recall that a 2ec-block of  $H$  is called small if it has  $\leq 2$  unit-edges, and is called large otherwise. Observe that a small 2ec-block of  $H$  is either a 3-cycle with one zero-edge and two unit-edges, or a 4-cycle with alternating zero-edges and unit-edges.

The following result summarizes this section:

► **Proposition 20.** *At the termination of the bridge-covering step, let  $H$  denote the bridgeless 2-edge cover computed by the algorithm and suppose that each small 2ec-block of  $H$  has  $\frac{4}{3}$  credits and each large 2ec-block of  $H$  has  $\geq 2$  credits. Let  $\gamma$  denote  $\text{credit}(H)$ . Assume that  $H$  contains all zero-edges. Then the gluing step augments  $H$  to a 2-ECSS  $H'$  of  $G$  (by adding edges and deleting edges) such that  $\text{cost}(H') \leq \text{cost}(H) + \gamma - 2$ . The gluing step can be implemented in polynomial time.*

Our gluing step applies a number of iterations. Each iteration picks two or more 2ec-blocks of  $H$ , and merges them into a new large 2ec-block by adding some unit-edges and possibly deleting some unit-edges such that the following invariant holds for  $H$  at the start/end of every iteration of the gluing step.

### Invariants for the gluing step:

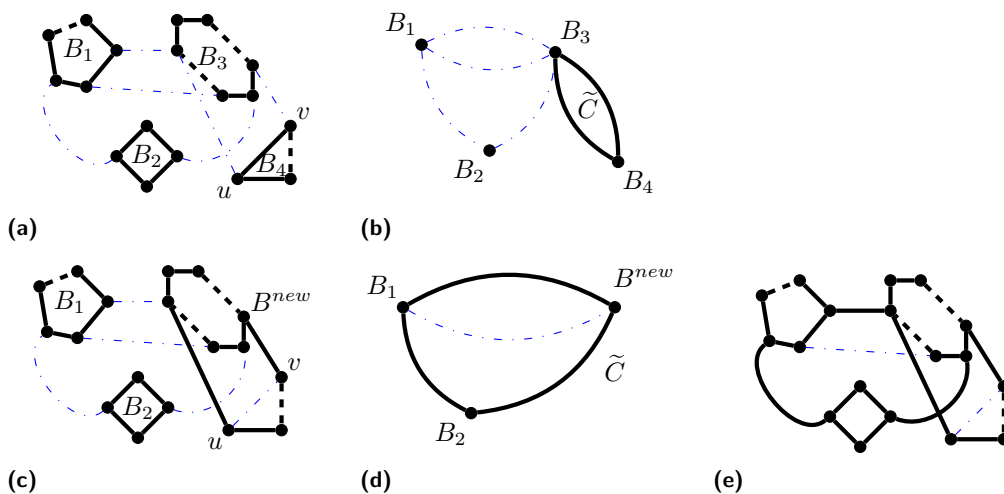
- $H$  is a simple, bridgeless graph of minimum degree two (hence, the 2ec-blocks of  $H$  correspond to the connected components of  $H$ );
- (credit invariant) each small 2ec-block of  $H$  has  $\frac{4}{3}$  credits and each large 2ec-block of  $H$  has  $\geq 2$  credits.

It is convenient to define the following multi-graph: let  $\tilde{G}$  be the multi-graph obtained from  $G$  by contracting each 2ec-block  $B_i$  of  $H$  into a single node that we will denote by  $B_i$  (thus, the notation  $B_i$  refers to either a 2ec-block of  $H$  or a node of  $\tilde{G}$ ). Observe that  $\tilde{G}$  is 2EC. We call a node of  $\tilde{G}$  small (respectively, large) if the corresponding 2ec-block of  $H$  is

small (respectively, large). The gluing step “operates” on  $G$  and never refers to  $\tilde{G}$ ; but, for our discussions and analysis, it is convenient to refer to  $\tilde{G}$ . (Note that  $\tilde{G}$  changes in each iteration, since the current graph  $H$  changes in each iteration.)

Suppose that  $\tilde{G}$  has  $\geq 2$  nodes and has no small nodes. Then, we pick any (large) node  $\tilde{v}$  of  $\tilde{G}$ . Since  $\tilde{G}$  is 2EC, it has a cycle  $\tilde{C}$  incident to  $\tilde{v}$ . Let  $|\tilde{C}|$  denote the number of edges of  $\tilde{C}$ ; note that  $|\tilde{C}| \geq 2$ . Our iteration adds to  $H$  the unit-edges corresponding to  $\tilde{C}$ . The credit available in  $H$  for the 2ec-blocks incident to  $\tilde{C}$  is  $\geq 2|\tilde{C}|$  and the cost of the augmentation is  $|\tilde{C}|$ ; hence, we have surplus credit of  $2|\tilde{C}| - |\tilde{C}| \geq 2$ . The surplus credit is given to the new large 2ec-block. Clearly, the credit invariant is preserved.

In general, small nodes may be present in  $\tilde{G}$ . If we apply the above scheme and find a cycle  $\tilde{C}$  incident only to small nodes with  $|\tilde{C}| \leq 5$ , then we fail to maintain the credit invariant (since only  $|\tilde{C}|/3$  credits are available for the new large 2ec-block). Consider a special case when  $\tilde{G}$  has a small node  $\mathcal{A}$  that has a unique neighbour  $B$  and  $B$  is large; clearly, there are  $\geq 2$  parallel edges between  $\mathcal{A}$  and  $B$ . Below, we show that  $\mathcal{A}$  and  $B$  can be merged to form a new large 2ec-block using an augmentation of net cost one, rather than two, by deleting one or more unit-edges of  $\mathcal{A}$ ; then we have surplus credit  $\geq 2$  for the new large 2ec-block. For example, if  $\mathcal{A}$  is a 3-cycle of  $H$ , then there exists a unit-edge  $uw$  of  $\mathcal{A}$  such that  $G$  has edges  $uv_1$  and  $wv_2$  where  $v_1, v_2 \in B$ ; so the augmentation adds the unit-edges  $uv_1$  and  $wv_2$  to  $H$  and discards  $uw$  from  $H$ .

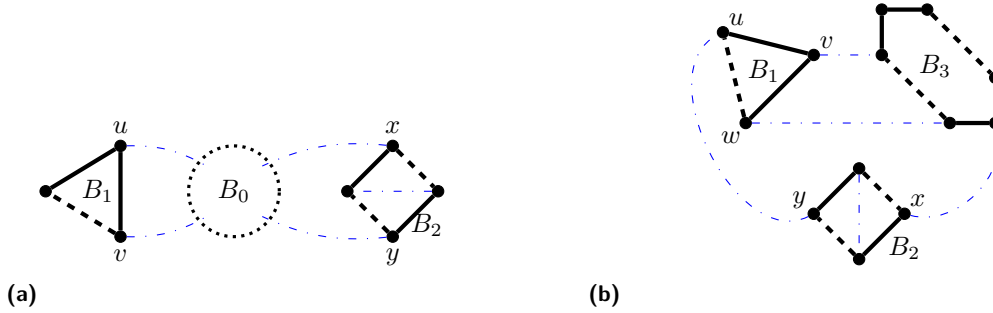


■ **Figure 2** Two iterations of our gluing step are illustrated. At each iteration, solid lines indicate unit-edges of  $H$ , dashed lines indicate zero-edges of  $H$ , and (blue) dash-dotted lines indicate edges of  $E(G) - E(H)$ . Subfigures (a), (b) show Iteration 1. (a) The input graph  $G$  and a bridgeless 2-edge cover  $H$  of  $G$ . (b) The graph  $\tilde{G}$  of the graphs  $G, H$  in (a). The small node  $B_4$  has a unique neighbour  $B_3$  which is large. The first iteration augments via  $\tilde{C} = B_4, B_3, B_4$ . Subfigures (c), (d) show Iteration 2. (c) The graphs  $G$  and  $H$  after the first iteration.  $B_3$  and  $B_4$  have been merged to form  $B^{new}$  by adding two unit-edges to  $H$  and deleting the unit-edge  $uw$  from  $H$ . (d) The graph  $\tilde{G}$  of the graphs  $G, H$  in (c). All nodes of  $\tilde{G}$  are large. The second iteration augments via  $\tilde{C} = B_1, B_2, B^{new}, B_1$ . Subfigure (e) shows the output 2-ECSS of our gluing step.

We present key definitions and results on small 2ec-blocks in Section 6.1. Our algorithm for the gluing step is presented in Section 6.2.

### 6.1 Analysis of small 2ec-blocks

In this subsection, we focus on the small 2ec-blocks of  $H$  and we present the definitions and results that underlie our algorithm for the gluing step. Recall that  $G$  has  $\geq 12$  nodes.



■ **Figure 3** Illustrations of swappable edges and swappable pairs of small 2ec-blocks. Solid lines indicate unit-edges of  $H$ , dashed lines indicate zero-edges of  $H$ , and (blue) dash-dotted lines indicate edges of  $E(G) - E(H)$ . (a) The 2ec-block  $B_1$  has a swappable edge  $uv$ , and the 2ec-block  $B_2$  has a swappable pair  $\{x, y\}$ . (b) The 2ec-block  $B_1$  has two swappable edges:  $uv$  is good, and  $vw$  is bad. The swappable pair  $\{x, y\}$  of the 2ec-block  $B_2$  is good.

► **Definition 21.** Let  $\mathcal{A}$  be a small 2ec-block of  $H$ . A unit-edge  $uw$  of  $\mathcal{A}$  is called *swappable* if both  $u$  and  $w$  are attachments of  $\mathcal{A}$  in  $G$  (that is,  $G$  has an edge  $ux$  where  $x \in V(G) - \mathcal{A}$  and  $G$  has an edge  $wy$  where  $y \in V(G) - \mathcal{A}$ ).

► **Definition 22.** Let  $\mathcal{A}$  be a small 2ec-block of  $H$ . A pair of nodes  $\{u, w\}$  of  $\mathcal{A}$  is called a *swappable pair* if either (i)  $uw$  is a swappable edge of  $\mathcal{A}$ , or (ii) both  $u$  and  $w$  are attachments of  $\mathcal{A}$  in  $G$ ,  $u, w$  are not adjacent in  $\mathcal{A}$  (note that  $\mathcal{A}$  is a 4-cycle in this case), and the other two nodes of  $\mathcal{A}$  are adjacent in  $G$  (that is,  $E(G) - E(H)$  has a “diagonal edge” between the other two nodes of  $\mathcal{A}$ ).

► **Definition 23.** Let  $\mathcal{A}$  be a small 2ec-block of  $H$ . A swappable pair  $\{u, w\}$  of  $\mathcal{A}$  is called *good* if there are distinct 2ec-blocks  $B_u$  and  $B_w$  ( $\mathcal{A} \neq B_u \neq B_w \neq \mathcal{A}$ ) such that  $G$  has an edge  $ux$  where  $x \in B_u$  and  $G$  has an edge  $wy$  where  $y \in B_w$ ; otherwise,  $\{u, w\}$  is called a *bad swappable pair* of  $\mathcal{A}$ . A *good* (respectively, *bad*) *swappable edge* of  $\mathcal{A}$  is defined similarly.

► **Remark 24.** Observe that each iteration merges two or more 2ec-blocks of  $H$  (see the discussion following Proposition 20). Consider a small 2ec-block  $\mathcal{A}$  of  $H$  that stays unchanged over several iterations. After one of these iterations, a swappable pair  $\{u, w\}$  of  $\mathcal{A}$  may change from good to bad, but  $\{u, w\}$  cannot change from bad to good.

► **Lemma 25.** Let  $\mathcal{A}$  be a small 2ec-block of  $H$ . If  $\mathcal{A}$  is adjacent (in  $G$ ) to a unique 2ec-block  $B$ , then  $B$  is large. (That is, if there is 2ec-block  $B$  such that  $\Gamma_G(V(\mathcal{A})) \subseteq V(B)$ , then  $B$  is large.)

► **Lemma 26.** Let  $\mathcal{A}$  be a small 2ec-block of  $H$ . Then  $\mathcal{A}$  has at least one swappable pair. Moreover, if  $\mathcal{A}$  is a 3-cycle, then  $\mathcal{A}$  has at least one swappable edge.

► **Lemma 27.** Let  $\mathcal{A}$  be a small 2ec-block of  $H$ . If  $\mathcal{A}$  is a 3-cycle, and  $\mathcal{A}$  is adjacent (in  $G$ ) to at least two other 2ec-blocks, then it has a good swappable edge.



Suppose that the current graph  $H$  has no good swappable pairs, that is, for every small 2ec-block  $\mathcal{A}$  of  $H$ , every swappable pair of  $\mathcal{A}$  is bad. To “merge away” the remaining small 2ec-blocks of  $H$ , we construct the following auxiliary digraph  $D^{aux}$ : there is a node for each 2ec-block of  $H$ , and we call the nodes corresponding to the small 2ec-blocks the *red* nodes, and the other nodes the *green* nodes; for each small 2ec-block  $\mathcal{A}$  of  $H$  and each of its swappable pairs  $\{u, w\}$ ,  $D^{aux}$  has an arc  $(\mathcal{A}, B)$  where  $B$  corresponds to the unique 2ec-block  $B$  of  $H$  such that  $\Gamma_G(\{u, w\}) \subseteq V(B) \cup V(\mathcal{A})$ . Observe that each red node of  $D^{aux}$  has at least one outgoing arc.

► **Lemma 28.** *Suppose that there exist no good swappable pairs. Then,  $D^{aux}$  does not have a pair of red nodes  $\mathcal{A}_1, \mathcal{A}_2$  such that  $(\mathcal{A}_1, \mathcal{A}_2)$  is the unique outgoing arc of  $\mathcal{A}_1$  and  $(\mathcal{A}_2, \mathcal{A}_1)$  is the unique outgoing arc of  $\mathcal{A}_2$  (that is, if  $D^{aux}$  has a directed 2-cycle  $C$  on the red nodes, then one of the red nodes incident to  $C$  has  $\geq 2$  outgoing arcs).*

By the above lemma,  $D^{aux}$  either has an arc  $(\mathcal{A}, B)$  from a red node  $\mathcal{A}$  to a green node  $B$ , or it has a directed path  $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$  on three red nodes. In both cases, we can apply a merge step to obtain a new large 2ec-block (i.e., a green node) while preserving the credit invariant. More details are presented in the next subsection.

## 6.2 Algorithm for the gluing step

In this subsection, we explain the working of the algorithm for the gluing step, based on the results in the previous subsection.

Consider any small 2ec-block  $\mathcal{A}$  that has a good swappable pair  $\{u, w\}$  such that  $u$  is adjacent (in  $G$ ) to another 2ec-block  $B_u$ , and  $w$  is adjacent (in  $G$ ) to another 2ec-block  $B_w$ , and  $\mathcal{A} \neq B_u \neq B_w \neq \mathcal{A}$ . Observe that  $G - V(\mathcal{A})$  is connected, otherwise,  $\mathcal{A}$  would be an  $S\{3, 4\}$  of  $G$  (the arguments in the proof of Lemma 25 can be used to verify this statement). Hence,  $\tilde{G} - \mathcal{A}$  has a path between  $B_u$  and  $B_w$ ; adding the edges  $\mathcal{A}B_u$  and  $\mathcal{A}B_w$  to this path gives a cycle  $\tilde{C}$  of  $\tilde{G}$ . We merge the 2ec-blocks incident to  $\tilde{C}$  into a new large 2ec-block by adding the unit-edges corresponding  $\tilde{C}$  to  $H$ . Moreover, if  $uw \in E(\mathcal{A})$ , then we discard  $uw$  from  $H$ , otherwise,  $\mathcal{A}$  is a 4-cycle (with two zero-edges) and  $E(G) - E(H)$  has a unit-edge  $f$  between the two nodes of  $\mathcal{A} - \{u, w\}$ , and in this case, we add the edge  $f$  to  $H$  and we discard the two unit-edges of  $\mathcal{A}$  from  $H$ . The credit available in  $H$  for  $\tilde{C}$  is  $\geq \frac{4}{3}|\tilde{C}|$  and the net cost of the augmentation is  $|\tilde{C}| - 1$ ; hence, we have surplus credit of  $\frac{1}{3}|\tilde{C}| + 1 \geq 2$  (since  $|\tilde{C}| \geq 3$ ). The surplus credit is given to the new large 2ec-block.

The gluing step applies the above iteration until there are no good swappable pairs in the current graph  $H$ . Then the auxiliary digraph  $D^{aux}$  is constructed. By Lemma 28,  $D^{aux}$  has either (i) an arc  $(\mathcal{A}, B)$  from a red node  $\mathcal{A}$  to a green node  $B$ , or (ii) a directed path  $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$  on three red nodes.

In the first case,  $\mathcal{A}$  is a small 2ec-block,  $B$  is a large 2ec-block, and  $\mathcal{A}$  has a swappable pair  $\{u, w\}$  such that  $\mathcal{A} \cup B$  contains all neighbours (in  $G$ ) of  $\{u, w\}$ . We merge  $\mathcal{A}$  and  $B$  into a new large 2ec-block as follows. We add two unit-edges between  $\mathcal{A}$  and  $B$  to  $H$  (one edge is incident to  $u$  and the other edge is incident to  $w$ ). Moreover, if  $uw \in E(\mathcal{A})$ , then we discard  $uw$  from  $H$ , otherwise,  $\mathcal{A}$  is a 4-cycle (with two zero-edges) and  $E(G) - E(H)$  has a unit-edge  $f$  between the two nodes of  $\mathcal{A} - \{u, w\}$ , and in this case, we add the edge  $f$  to  $H$  and we discard the two unit-edges of  $\mathcal{A}$  from  $H$ . The credit available in  $H$  for  $\mathcal{A} \cup B$  is  $\geq \frac{4}{3} + 2$  and the net cost of the augmentation is one; hence, we have surplus credit of  $\frac{1}{3} + 2 \geq 2$ . The surplus credit is given to the new large 2ec-block. Consider the second case. Then  $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$  are small 2ec-blocks such that  $\mathcal{A}_1$  has a swappable pair  $u_1 w_1$  such that  $\Gamma_G(\{u_1, w_1\}) \subseteq V(\mathcal{A}_1) \cup V(\mathcal{A}_2)$ , and  $\mathcal{A}_2$  has a swappable pair  $u_2 w_2$  such that  $\Gamma_G(\{u_2, w_2\}) \subseteq V(\mathcal{A}_2) \cup V(\mathcal{A}_3)$ . We add two unit-edges between  $\mathcal{A}_1$  and  $\mathcal{A}_2$  to  $H$  (one edge is incident to  $u_1$  and the other edge is incident to  $w_1$ ), and then we either discard one unit-edge from  $H$  (if  $u_1 w_1 \in E(\mathcal{A}_1)$ ) or we add another

edge to  $H$  and discard two unit-edges of  $\mathcal{A}_1$  from  $H$  (if  $u_1w_1 \notin E(\mathcal{A}_1)$ ). We apply a similar augmentation to  $\mathcal{A}_2$  and  $\mathcal{A}_3$  using the swappable pair  $\{u_2, w_2\}$ . The credit available in  $H$  for  $\mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3$  is  $\geq (3 \cdot \frac{4}{3}) = 4$  and the net cost of the augmentation is two; hence, we have surplus credit of  $\geq 4 - 2$ . The surplus credit is given to the new large 2ec-block.

By repeatedly applying the above iteration (that merges red nodes of  $D^{aux}$  into green nodes), we obtain a current graph  $H$  that has no small 2ec-blocks. As discussed above, the merge step is straightforward when all 2ec-blocks of  $H$  are large.

► **Lemma 29.** *After every merge step, the subgraph  $B^{new}$  constructed by that step (that is a so-called large 2ec-block) is 2EC.*

---

## References

- 1 David Adjiashvili. Beating approximation factor two for weighted tree augmentation with bounded costs. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2384–2399. SIAM, 2017. doi:10.1137/1.9781611974782.157.
- 2 Joe Cheriyan, Jack Dippel, Fabrizio Grandoni, Arindam Khan, and Vishnu V. Narayan. The matching augmentation problem: a  $\frac{7}{4}$ -approximation algorithm. *Math. Program.*, 182(1):315–354, 2020. doi:10.1007/s10107-019-01394-z.
- 3 Joseph Cheriyan, Robert Cummings, Jack Dippel, and Jasper Zhu. An improved approximation algorithm for the matching augmentation problem. *CoRR*, abs/2007.11559, 2020. arXiv:2007.11559.
- 4 Joseph Cheriyan and Zhihan Gao. Approximating (unweighted) tree augmentation via lift-and-project, part II. *Algorithmica*, 80(2):608–651, 2018. doi:10.1007/s00453-017-0275-7.
- 5 Joseph Cheriyan, Howard J. Karloff, Rohit Khandekar, and Jochen Könemann. On the integrality ratio for tree augmentation. *Oper. Res. Lett.*, 36(4):399–401, 2008. doi:10.1016/j.orl.2008.01.009.
- 6 Nachshon Cohen and Zeev Nutov. A  $(1 + \ln 2)$ -approximation algorithm for minimum-cost 2-edge-connectivity augmentation of trees with constant radius. *Theor. Comput. Sci.*, 489-490:67–74, 2013. doi:10.1016/j.tcs.2013.04.004.
- 7 R. Diestel. *Graph Theory (4th ed.)*. Graduate Texts in Mathematics, Volume 173. Springer-Verlag, Heidelberg, 2010. URL: <http://diestel-graph-theory.com/>.
- 8 Dippel, Jack. *The Matching Augmentation Problem: A  $\frac{7}{4}$ -Approximation Algorithm*. M.Math. Thesis, C&O Department. UWSpace (University of Waterloo), 2019. URL: <http://hdl.handle.net/10012/14700>.
- 9 Samuel Fiorini, Martin Groß, Jochen Könemann, and Laura Sanità. Approximating weighted tree augmentation via Chvátal-Gomory cuts. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 817–831. SIAM, 2018. doi:10.1137/1.9781611975031.53.
- 10 Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest  $k$ -edge connected spanning subgraph by LP-rounding. *Networks*, 53(4):345–357, 2009. doi:10.1002/net.20289.
- 11 G.Even, J.Feldman, G.Kortsarz, and Z.Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Trans. Algorithms*, 5(2):21:1–17, 2009.
- 12 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.
- 13 Fabrizio Grandoni, Christos Kalaitzis, and Rico Zenklusen. Improved approximation for tree augmentation: saving by rewiring. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 632–645. ACM, 2018. doi:10.1145/3188745.3188898.

- 14 H.Nagamochi. An approximation for finding a smallest 2-edge connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126:83–113, 2003.
- 15 Christoph Hunkenschroder, Santosh S. Vempala, and Adrian Vetta. A  $4/3$ -approximation algorithm for the minimum 2-edge connected subgraph problem. *ACM Trans. Algorithms*, 15(4):55:1–55:28, 2019. doi:10.1145/3341599.
- 16 K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.
- 17 Guy Kortsarz and Zeev Nutov. A simplified 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Trans. Algorithms*, 12(2):23:1–20, 2016. doi:10.1145/2786981.
- 18 Lap Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics (No. 46). Cambridge University Press, 2011. URL: <http://www.cambridge.org/catalogue/catalogue.asp?ISBN=9780521189439>.
- 19 Zeev Nutov. On the tree augmentation problem. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPICs*, pages 61:1–61:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ESA.2017.61.
- 20 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics, Volume 24. Springer-Verlag, Berlin Heidelberg, 2003. URL: <http://www.springer.com/gp/book/9783540443896>.
- 21 András Sebő and Jens Vygen. Shorter tours by nicer ears:  $7/5$ -approximation for the graph-TSP,  $3/2$  for the path version, and  $4/3$  for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014. doi:10.1007/s00493-014-2960-3.
- 22 S.Khuller and U.Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.
- 23 Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. *CoRR*, abs/2104.07114, 2021. arXiv:2104.07114.
- 24 Santosh Vempala and Adrian Vetta. Factor  $4/3$  approximations for minimum 2-connected subgraphs. In Klaus Jansen and Samir Khuller, editors, *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1913 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 2000. doi:10.1007/3-540-44436-X\_26.
- 25 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: [http://www.cambridge.org/de/knowledge/isbn/item5759340/?site\\_locale=de\\_DE](http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE).
- 26 Alexander Zelikovsky. Better approximation bounds for the network and Euclidean Steiner tree problems. Technical report, CS-96-06, Department of Computer Science, University of Virginia, USA, 1996.



# Multimodal Transportation with Ridesharing of Personal Vehicles

Qian-Ping Gu ✉

School of Computing Science, Simon Fraser University, Burnaby, Canada

JiaJian Liang ✉

School of Computing Science, Simon Fraser University, Burnaby, Canada

---

## Abstract

Many public transportation systems are unable to keep up with growing passenger demand as the population grows in urban areas. The slow or lack of improvement for public transportation pushes people to use private transportation modes, such as carpooling and ridesharing. However, the occupancy rate of personal vehicles has been dropping in many cities. In this paper, we describe a centralized transit system that integrates public transit and ridesharing, which matches drivers and transit riders such that the riders would result in shorter travel time using both transit and ridesharing. The optimization goal of the system is to assign as many riders to drivers as possible for ridesharing. We give an exact approach and approximation algorithms to achieve the optimization goal. As a case study, we conduct an extensive computational study to show the effectiveness of the transit system for different approximation algorithms, based on the real-world traffic data in Chicago City; the data sets include both public transit and ridesharing trip information. The experiment results show that our system is able to assign more than 60% of riders to drivers, leading to a substantial increase in occupancy rate of personal vehicles and reducing riders' travel time.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Applied computing → Transportation

**Keywords and phrases** Multimodal transportation, ridesharing, approximation algorithms, computational study

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.39

**Related Version** *Full Version*: <https://arxiv.org/abs/2106.00232> [13]

**Funding** This work was partially supported by Canada NSERC Discovery Grant 253500.

**Acknowledgements** The authors thank anonymous reviewers for their constructive comments.

## 1 Introduction

As the population grows in urban areas, commuting between and within large cities is time-consuming and resource-demanding. Due to growing passenger demand, the number of vehicles on the road for both public and private transportation has increased to handle the demand. Public transportation systems are unable to keep up with the demand in terms of service quality. This pushes people to use personal vehicles for work commute. In the United States, personal vehicles are the main transportation mode [6]. However, the occupancy rate of personal vehicles in the U.S. is 1.6 persons per vehicle in 2011 [12, 24] (and decreased to 1.5 persons per vehicle in 2017 [6]), which can be a major cause for congestion and pollution. This is the reason municipal governments encourage the use of public transit; the major drawback of public transit is the inconvenience of last mile and/or first mile transportation compared to personal vehicles [28]. With the increasing popularity in ridesharing/ridehailing service, there may be potential in integrating private and public transportation. From the research report of [9], it is recommended that public transit agencies should build on mobility innovations to allow public-private engagement in ridesharing because the use of shared



© Qian-Ping Gu and JiaJian Liang;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 39; pp. 39:1–39:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

modes increases the likelihood of using public transit. As pointed out by Ma et al. [20], some basic form of collaboration between MoD (mobility-on-demand) services and public transit already exists (for first and last mile transportation). There is an increasing interest for collaboration between private companies and public sector entities [22].

The sparseness of transit networks usually is the main cause of the inconvenience in public transit. Such transit networks have infrequent transit schedule and can cause customers to have multiple transfers. In this paper, we investigate the potential effectiveness of integrating public transit with ridesharing to increase ridership in such sparse transit networks and reduce traffic congestion for work commute (not very short trips). For example, people who drive their vehicles to work can pick-up *riders*, who use public transit regularly, at designated locations and drop-off them at some transit stops, and then those riders can take public transit to their destinations. In this way, riders are presented with a cheaper alternative than ridesharing for the entire trip, and it is more convenient than using public transit only. The transit system also gets a higher ridership, which matches the recommendation of [9] for a more sustainable transportation system. Our research focuses on a centralized system that is capable of matching drivers and riders satisfying their trips' requirements while achieving some optimization goal; the requirements of a trip may include an origin and a destination, time constraints, capacity of a vehicle, and so on. When a rider is assigned a driver, we call this *ridesharing route*, and it is compared with the fastest *public transit route* for this rider which uses only public transit. If the ridesharing route is faster than the public transit route, the ridesharing route is provided to both the rider and driver. To increase the number of rider participants, our system-wide optimization goal is to maximize the number of riders, each of whom is assigned a ridesharing route. We call this the *maximization problem* (formal definition in Section 2).

In the literature, there are many papers about standalone ridesharing/carpooling, from theoretical to empirical studies (e.g., [1, 4, 14, 29]). For literature reviews on ridesharing, readers are referred to [2, 10, 21, 27]. On the other hand, there are only few papers study the integration of public transit with dynamic ridesharing. Aissat and Varone [3] and Huang et al. [17] proposed approaches which find a route with ridesharing that substitutes part of a public transit route for each rider in the first-come first-serve basis (system-wide optimization goal is not considered). Ma [19] and Stiglic et al. [26] proposed models to integrate ridesharing and public transit as graph matching problems to achieve system-wide optimization goals; their approaches are similar, except the work in [26] supports more rideshare match types. The graph matching problems in [19, 26] are formulated as integer linear program (ILP) and solved by standard branch and bound (CPLEX). The optimization goal in [19] is to minimize the cost related to waiting time and travel time, but ridesharing routes are not guarantee to be better than transit route. Although the optimization goal in [26] aligns with ours, there are some limitations in their approach; they limit at most two riders for each rideshare match, each rider must travel to the transit stop that is closest to the rider's destination, and more importantly, ridesharing routes assigned to riders can be longer than public transit routes.

In this paper, we use a similar model as in [19, 26]. We extend the work in [26] to eliminate the limitations described above and give approximation algorithms for the optimization problem to ensure solution quality. Our discrete algorithms allow to control the trade-off between quality and computational time. Our main contributions are summarized as follows:

1. We give an exact algorithm approach (an ILP formulation based on a hypergraph representation) for integrating public transit and ridesharing.
2. We prove our maximization problem is NP-hard and give a 2-approximation algorithm for the problem. We show that previous  $O(k)$ -approximation algorithms [5, 7] for the  $k$ -set packing problem are 2-approximation algorithms for our maximization problem. Our algorithm is more time and space efficient than previous algorithms.

3. As a case study, we conduct an extensive numerical study based on real-life data in Chicago City to evaluate the potential of having an integrated transit system and the effectiveness of different approximation algorithms.

The rest of the paper is organized as follows. In Section 2, we give the preliminaries of the paper, describe a centralized system that integrates public transit and ridesharing, and define the maximization problem. In Section 3, we describe our exact algorithm approach. We then propose approximation algorithms in Section 4. We discuss our numerical experiments and results in Section 5. Finally, Section 6 concludes the paper.

## 2 Problem definition and preliminaries

In the problem *multimodal transportation with ridesharing* (MTR), we have a centralized system, and for every fixed time interval, the system receives a set  $\mathcal{A} = D \cup R$  of trips with  $D \cap R = \emptyset$ , where  $D$  is the set of driver trips and  $R$  is the set of rider trips. Each trip is expressed by an integer label  $i$  and consists of an individual, a vehicle (for driver trip) and some requirements. A connected public transit network with a fixed timetable  $T$  is given. We assume that for any source  $o$  and destination  $d$  in the public transit network,  $T$  gives the fastest travel time from  $o$  to  $d$ . A *ridesharing route*  $\pi_i$  for a rider  $i \in R$  is a travel plan using a combination of public transportation and ridesharing to reach  $i$ 's destination satisfying  $i$ 's requirements, whereas a *public transit route*  $\hat{\pi}_i$  for a rider  $i$  is a travel plan using only public transportation. The multimodal transportation with ridesharing problem asks to provide at least one feasible route ( $\pi_i$  or  $\hat{\pi}_i$ ) for every rider  $i \in R$ . We denote an instance of multimodal transportation with ridesharing problem by  $(N, \mathcal{A}, T)$ , where  $N$  is an edge-weighted directed graph (network) for both private and public transportation. We call a public transit station or stop just *station*. The terms rider and passenger are used interchangeably (although passenger emphasizes a rider has been provided with a ridesharing route).

The requirements of each trip  $i$  in  $\mathcal{A}$  are specified by  $i$ 's parameters submitted by the individual. The parameters of a trip  $i$  contain an origin location  $o_i$ , a destination location  $d_i$ , an earliest departure time  $\alpha_i$ , a latest arrival time  $\beta_i$  and a maximum trip time  $\gamma_i$ . A driver trip  $i$  also contains a capacity  $n_i$  of the vehicle, a limit  $\delta_i$  on the number of stops a driver wants to make to pick-up/drop-off passengers, and an optional path to reach its destination. The maximum trip time  $\gamma_i$  of a driver  $i$  includes a travel time from  $o_i$  to  $d_i$  and a detour time limit  $i$  can spend for offering ridesharing service. A rider trip  $i$  also contains an acceptance rate  $\theta_i$  for a ridesharing route  $\pi_i$ , that is,  $\pi_i$  is given to rider  $i$  if  $t(\pi_i) \leq \theta_i \cdot t(\hat{\pi}_i)$  for every public transit route  $\hat{\pi}_i$  and  $0 < \theta_i \leq 1$ , where  $t(\cdot)$  is the travel time. Such a route  $\pi_i$  is called an *acceptable ridesharing route* (acceptable route for brevity). For example, suppose the best public transit route  $\hat{\pi}_i$  takes 100 minutes for  $i$  and  $\theta_i = 0.9$ . An acceptable route  $\pi_i$  implies that  $t(\pi_i) \leq \theta_i \cdot t(\hat{\pi}_i) = 90$  minutes. We consider two match types for practical reasons.

- **Type 1 (rideshare-transit):** a driver may make multiple stops to pick-up different passengers, but makes only one stop to drop-off all passengers. In this case, the *pick-up locations* are the passengers' origin locations, and the *drop-off location* is a public station.
- **Type 2 (transit-rideshare):** a driver makes only one stop to pick-up passengers and may make multiple stops to drop-off all passengers. In this case, the *pick-up location* is a public station and the *drop-off locations* are the passengers' destination locations.

Riders and drivers specify one of the match types to participate in; they are allowed to choose both in hope to increase the chance being selected, but the system will assign them only one of the match types such that the optimization goal of the MTR problem is achieved, which is to assign acceptable routes to as many riders as possible. Formally, the **maximization problem** we consider is to maximize the number of passengers, each of whom is assigned an acceptable route  $\pi_i$  for every  $i \in R$ .



For a driver  $i$  and a set  $J \subseteq R$  of riders,  $\sigma(i) = \{i\} \cup J$  is called a *feasible match* if the routes for all trips of  $\sigma(i)$  satisfy the requirements (constraints) specified by the parameters of the trips collectively as listed below (a summary of notation and constraints can be found in [13], Section 3.2):

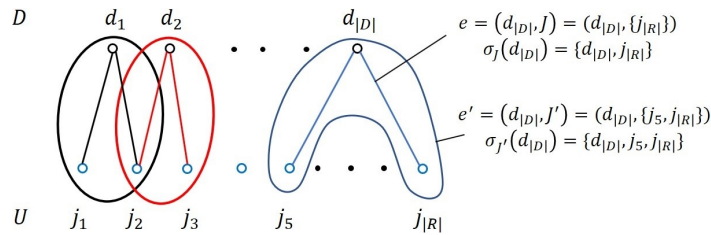
1. *Ridesharing route constraint*: for  $J = \{j_1, \dots, j_k\}$ , there is a path  $(o_i, o_{j_1}, \dots, o_{j_k}, s, d_i)$  in  $N$ , where  $s$  is the drop-off location for Type 1 match; or there is a path  $(o_i, s, d_{j_1}, \dots, d_{j_k}, d_i)$  in  $N$ , where  $s$  is the pick-up location for Type 2 match.
2. *Capacity constraint*:  $1 \leq |J| \leq n_i$ .
3. *Acceptable constraint*: each passenger  $j \in J$  is given an acceptable route  $\pi_j$  offered by  $i$ .
4. *Travel time constraint*: each trip  $j \in \sigma(i)$  departs from  $o_j$  no earlier than  $\alpha_j$ , arrives at  $d_j$  no later than  $\beta_j$ , and the total travel duration of  $j$  is at most  $\gamma_j$ .
5. *Stop constraint*: the number of unique locations visited by driver  $i$  to pick-up (for Type 1) or drop-off (for Type 2) all passengers of  $\sigma(i)$  is at most  $\delta_i$ .

Two feasible matches  $\sigma(i), \sigma(i')$  are *disjoint* if  $\sigma(i) \cap \sigma(i') = \emptyset$ . Then, the maximization problem considered is to find a set of pairwise disjoint feasible matches such that the number of passengers included in the feasible matches is maximized.

Intuitively, a rideshare-transit (Type 1) feasible match  $\sigma(i)$  is that all passengers in  $\sigma(i)$  are picked-up at their origins and dropped-off at a station, and then  $i$  drives to destination  $d_i$  while each passenger  $j$  of  $\sigma(i)$  takes transit to destination  $d_j$ . A transit-rideshare (Type 2) feasible match  $\sigma(i)$  is that all passengers in  $\sigma(i)$  are picked-up at a station and dropped-off at their destinations, and then  $i$  drives to destination  $d_i$  after dropping the last passenger. We give algorithms to find pairwise disjoint feasible matches to maximize the number of passengers included in the matches. We describe our algorithms for Type 1 only. Algorithms for Type 2 can be described with the constraints on the drop-off location and pick-up location of a driver exchanged, and we omit the description. Further, it is not difficult to extend to other match types, such as rideshare only and park-and-ride, as described in [26].

### 3 Exact algorithm

An exact algorithm for the maximization problem is presented in this section, which is similar to the matching approach described in [4, 23] for ridesharing and in [19, 26] for MTR. The exact algorithm is summarized as follows. First, we compute all feasible matches for each driver  $i$ . Then, we create a bipartite (hyper)graph  $H(D, R, E)$ , where  $D(H)$  is the set of drivers, and  $R(H)$  is the set of riders. There is a hyperedge  $e = (i, J)$  in  $E(H)$  between  $i \in D(H)$  and a non-empty subset  $J \subseteq R(H)$  if  $\{i\} \cup J$  is a feasible match, denoted by  $\sigma_J(i)$ , for driver  $i$ . An example is given in Figure 1. Any driver  $i$  and rider  $j$  with no feasible match is removed from  $D(H)$  and  $R(H)$  respectively, namely, no isolated vertex (such riders must use public transit routes). For an edge  $e = (i, J)$ , let  $A(e) = \{i\} \cup J$  and  $p(e) = |J|$  be the



■ **Figure 1** A bipartite hypergraph for all possible matches of an instance  $(N, \mathcal{A}, T)$ .

number of riders represented by  $e$ . For a trip  $j \in \mathcal{A}$ , define  $E_j = \{e \in E \mid j \in A(e)\}$  to be the set of edges in  $E$  associated with  $j$ . To solve the maximization problem, we give an integer program (ILP) formulation:

$$\text{maximize} \quad \sum_{e \in E(H)} p(e) \cdot x_e \quad (1)$$

$$\text{subject to} \quad \sum_{e \in E_j} x_e \leq 1, \quad \forall j \in \mathcal{A} \quad (2)$$

$$x_e \in \{0, 1\}, \quad \forall e \in E(H) \quad (3)$$

The binary variable  $x_e$  indicates whether the edge  $e = (i, J)$  is in the solution ( $x_e = 1$ ) or not ( $x_e = 0$ ). If  $x_e = 1$ , it means that all passengers in  $J$  are served by  $i$ . Inequality (2) in the ILP formulation guarantees that each driver serves at most one feasible set of passengers and each passenger is served by one driver. Note that the ILP (1)-(3) is similar to a set packing formulation. An advantage of this ILP formulation is that the number of constraints is substantially decreased, compared to traditional ridesharing formulation. From Observation 1 in [25], it is not difficult to see that the following result holds (a proof of Theorem 1 is given in [13], Sections 3.1).

► **Theorem 1.** *Given a bipartite graph  $H(D, R, E)$  representing an instance of the multimodal transportation with ridesharing maximization problem, an optimal solution to the ILP (1)-(3) is an optimal solution to the maximization problem and vice versa.*

**Computing feasible matches.** Let  $i$  be a driver in  $D$  and  $n_i$  be the capacity of  $i$  (maximum number of riders  $i$  can serve). The maximum number of feasible matches for  $i$  is  $\sum_{p=1}^{n_i} \binom{|R|}{p}$ . Assuming the capacity  $n_i$  is a small constant (which is reasonable in practice), the above summation is polynomial in  $R$ , that is,  $O((|R|+1)^{n_i})$ . Let  $K = \max_{i \in D} n_i$  be the maximum capacity among all vehicles (driver trips). Then, in the worst case,  $|E(H)| = O(|D| \cdot (|R|+1)^K)$ . We compute all feasible matches for each trip in two phases. In phase one, for each driver  $i$ , we find all feasible matches  $\sigma(i) = \{i, j\}$  with one rider  $j$ . In phase two, for each driver  $i$ , we compute all feasible matches  $\sigma(i) = \{i, j_1, \dots, j_p\}$  with  $p$  riders, based on the feasible matches  $\sigma(i)$  with  $p-1$  riders computed previously, for  $p=2$  and upto the number of passengers  $i$  can serve. Complete description and algorithms (Algorithm 1 for phase one and Algorithm 2 for phase two) for computing the feasible matches can be found in Sections 3.2.1 and 3.2.2 of [13]. We make two simplifications in our algorithms:

- Given a source station  $s_o$  and a destination  $d_i$  of trip  $i$  with departure time  $t$  at  $s_o$ , we use a simplified transit system in our experiments to calculate the fastest public transit route from  $s_o$  to  $d_i$ .
- We use a simplified model for the transit waiting time and ridesharing service time (time it takes to pick-up and drop-off riders, walking time between locations and stations).

As shown in [13](Section 3.2.2), we compute a feasible path with minimum travel time for driver  $i$  to pick-up  $p$  passengers in each feasible match  $\sigma(i)$ .

## 4 NP-hardness and approximation algorithms

We show that the maximization problem is NP-hard and give approximation algorithms for the problem. When every edge in  $H(D, R, E)$  consists of only two vertices (one driver and one passenger), the maximization problem is equivalent to the maximum matching, which can be solved in polynomial time (e.g., [16]). However, if the edges consist of more than two

vertices, they become hyperedges. In this case, the ILP (1)–(3) becomes a formulation of the maximum weighted set packing problem (MWSP), which is NP-hard [11, 18]. In fact, ILP (1)–(3) formulation gives a special case of MWSP (due to the structure of  $H(D, R, E)$ ). We prove that this special case is also NP-hard, and by Theorem 1, the maximization problem is NP-hard (a proof of Theorem 2 is in [13], Section 4.1).

► **Theorem 2.** *The maximization problem is NP-hard.*

Next, we describe approximation algorithms for the maximization problem. For consistency, we follow the convention in [5, 7] that a  $\rho$ -approximation algorithm for a maximization problem is defined as  $\rho \cdot w(\mathcal{C}) \geq OPT$  for  $\rho > 1$ , where  $w(\mathcal{C})$  and  $OPT$  are the values of approximation and optimal solutions respectively.

#### 4.1 2-Approximation algorithm

We first give a simple 2-approximation algorithm for our maximization problem. For a maximization problem instance  $H(D, R, E)$ , we use  $\Gamma$  to denote a current partial solution, which consists of a set of matches represented by the hyperedges in  $E(H)$ . Let  $P(\Gamma) = \bigcup_{e \in \Gamma} J_e$  (called *covered passengers*). Initially,  $\Gamma = \emptyset$ . In each iteration, we add a match with the most number of uncovered passengers to  $\Gamma$ , that is, select an edge  $e = (i, J_e)$  such that  $|J_e \setminus P(\Gamma)|$  is maximum, and then add  $e$  to  $\Gamma$ . Remove  $E_e = \bigcup_{j \in A(e)} E_j$  from  $E(H)$  ( $E_j$  is defined in Section 3). Repeat until  $P(\Gamma) = R$  or  $|\Gamma| = |D|$ . The pseudo code of ImpGreedy is shown in Algorithm 3. In the ImpGreedy algorithm, when an edge  $e$  is added to  $\Gamma$ ,  $E_e$  is removed from  $E(H)$ , so Property 3 holds for  $\Gamma$ .

► **Property 3.** *For every  $i \in D$ , at most one edge  $e$  from  $E_i$  can be selected in any solution.*

■ **Algorithm 3** ImpGreedy Algorithm.

---

**input** : The hypergraph  $H(D, R, E)$  for problem instance  $(N, \mathcal{A}, T)$   
**output** : A solution  $\Gamma$  to  $(N, \mathcal{A}, T)$  with 2-approximation ratio

- 1  $\Gamma = \emptyset$ ;  $P(\Gamma) = \emptyset$ ;
- 2 **while** ( $P(\Gamma) \neq R$  and  $|V(\Gamma)| < |D|$ ) **do**
- 3     compute  $e = \operatorname{argmax}_{e \in E(H)} |J_e \setminus P(\Gamma)|$ ,  $\Gamma = \Gamma \cup \{e\}$ , and remove  $E_e$  from  $E(H)$ ;
- 4     update  $P(\Gamma)$ ;
- 5 **end**

---

##### 4.1.1 Analysis of ImpGreedy Algorithm

Let  $\Gamma = \{x_1, x_2, \dots, x_a\}$  be a solution found by Algorithm 3, where  $x_i$  is the  $i^{\text{th}}$  edge added to  $\Gamma$ . Throughout the analysis, we use  $OPT$  to denote an optimal solution, that is,  $P(OPT) \geq P(\Gamma)$ . Further,  $\Gamma_i = \bigcup_{1 \leq b \leq i} x_b$  for  $1 \leq i \leq a$ ,  $\Gamma_0 = \emptyset$  and  $\Gamma_a = \Gamma$ . The driver of match  $x_i$  is denoted by  $d(x_i)$ . The main idea of our analysis is to add up the maximum difference between the number of covered passengers by selecting  $x_i$  in  $\Gamma$  and not selecting  $x_i$  in  $OPT$ . For each  $x_i \in \Gamma$ , by Property 3, there is at most one  $y \in OPT$  with  $d(y) = d(x_i)$ . We order  $OPT$  and introduce dummy edges to  $OPT$  such that  $d(y_i) = d(x_i)$  for  $1 \leq i \leq a$ . Formally, for  $1 \leq i \leq a$ , define

$$OPT(i) = \{y_1, \dots, y_i \mid 1 \leq b \leq i, d(y_b) = d(x_b) \text{ if } y_b \in OPT, \text{ otherwise } y_b \text{ a dummy edge}\}.$$

A dummy edge  $y_b \in OPT(i)$  is defined as  $d(y_b) = d(x_b)$  with  $J_{y_b} = \emptyset$ . The gap of an edge  $x_i \in \Gamma$  is defined as

$$\text{gap}(x_i) = |J_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}|,$$

where  $J'_{x_i} = (J_{x_i} \setminus P(\Gamma_{i-1})) \cap P(OPT \setminus \Gamma)$  is the maximum subset of passengers in  $J_{x_i} \setminus P(\Gamma_{i-1})$  that are also covered in  $OPT \setminus \Gamma$ . The intuition is that the sum of  $\text{gap}(x_i)$  for all  $x_i \in \Gamma$  states the maximum possible number of passengers may not be covered by  $\Gamma$ . Let  $P(OPT(i)) = \bigcup_{1 \leq b \leq i} J_{y_b}$  and  $P(OPT'(i)) = \bigcup_{1 \leq b \leq i} J'_{x_b}$  for any  $i \in [1, \dots, a]$ . Then the maximum gap between  $\Gamma$  and  $OPT$  can be calculated as  $\sum_{x \in \Gamma_a} \text{gap}(x) = |P(OPT(a))| + |P(OPT'(a))| - |P(\Gamma_a)|$ . First, we show that  $P(OPT) = P(OPT(a)) \cup P(OPT'(a))$ .

► **Proposition 4.** *Let  $\Gamma = \{x_1, \dots, x_a\}$ ,  $P(OPT(a)) = \bigcup_{1 \leq i \leq a} J_{y_i}$  and  $P(OPT'(a)) = \bigcup_{1 \leq i \leq a} J'_{x_i}$ . Then,  $P(OPT) = P(OPT(a)) \cup P(OPT'(a))$ .*

**Proof.** By definition,  $P(OPT) = P(OPT(a)) \cup P(OPT \setminus OPT(a))$ . For any  $z$  in  $OPT \setminus OPT(a)$ ,  $d(z) \neq d(x)$  for every  $x \in \Gamma$ . If  $J_z \setminus P(\Gamma) \neq \emptyset$ , then  $z$  would have been found and added to  $\Gamma$  by Algorithm 3. Hence,  $J_z \setminus P(\Gamma) = \emptyset$ , implying  $J_z \subseteq P(OPT'(a))$  and  $P(OPT \setminus OPT(a)) \subseteq P(OPT'(a))$ . ◀

► **Lemma 5.** *Let  $OPT$  be an optimal solution and  $\Gamma = \{x_1, x_2, \dots, x_a\}$  be a solution found by the algorithm. For any  $1 \leq i \leq a$ ,  $\sum_{x \in \Gamma_i} \text{gap}(x) = |P(OPT(i))| - |P(\Gamma_i)| + |P(OPT'(i))| \leq |P(\Gamma_i)|$ .*

**Proof.** Recall that  $OPT(i) = \{y_1, \dots, y_i\}$  as defined above. For  $y_b \in OPT(i)$ ,  $1 \leq b \leq i$ ,  $d(y_b) = d(x_b)$ . We prove the lemma by induction on  $i$ . Base case  $i = 1$ :  $|P(OPT(1))| - |P(\Gamma_1)| + |P(OPT'(1))| \leq |P(\Gamma_1)|$ . By definition,  $\text{gap}(x_1) = |J_{y_1}| - |J_{x_1} \setminus \Gamma_0| + |J'_{x_1}|$ . Since  $x_1$  is selected by the algorithm, it must be that  $|J_{x_1}| \geq |J_u|$  for all  $u \in V(G')$ , so  $|J_{y_1}| \leq |J_{x_1}|$ . Thus,

$$\begin{aligned} \text{gap}(x_1) &= |J_{y_1}| - |J_{x_1} \setminus \Gamma_0| + |J'_{x_1}| \\ &\leq |J'_{x_1}| \leq |J_{x_1}|. \end{aligned}$$

Assume the statement is true for  $i-1 \geq 1$ , that is,  $\sum_{x \in \Gamma_{i-1}} \text{gap}(x) \leq |P(\Gamma_{i-1})|$ , and we prove for  $i \leq a$ . By the induction hypothesis, both  $P(OPT(i-1))$  and  $P(OPT'(i-1))$  are included in the calculation of  $\sum_{x \in \Gamma_{i-1}} \text{gap}(x)$ . More precisely,  $\sum_{x \in \Gamma_{i-1}} \text{gap}(x) = |P(OPT(i-1))| - |P(\Gamma_{i-1})| + |P(OPT'(i-1))| \leq |P(\Gamma_{i-1})|$ . If  $|J_{y_i}| \leq |J_{x_i} \setminus P(\Gamma_{i-1})|$ , the lemma is true since we can assume  $|J'_{x_i}| \leq |J_{x_i}|$ . Suppose  $|J_{y_i}| > |J_{x_i} \setminus P(\Gamma_{i-1})|$ . Before  $x_i$  is selected, the algorithm must have considered  $y_i$  and found that  $|J_{x_i} \setminus P(\Gamma_{i-1})| \geq |J_{y_i} \setminus P(\Gamma_{i-1})|$ . Then,  $|J_{y_i}| > |J_{x_i} \setminus P(\Gamma_{i-1})| \geq |J_{y_i} \setminus P(\Gamma_{i-1})|$ , implying  $J_{y_i} \cap P(\Gamma_{i-1}) \neq \emptyset$ . We have

$$|J_{x_i} \setminus P(\Gamma_{i-1})| + |J_{y_i} \cap P(\Gamma_{i-1})| \geq |J_{y_i} \setminus P(\Gamma_{i-1})| + |J_{y_i} \cap P(\Gamma_{i-1})| = |J_{y_i}|. \quad (4)$$

Let  $J''_{y_i} \subseteq (J_{y_i} \cap P(\Gamma_{i-1}))$  be the set of passengers covered by  $P(OPT(i-1)) \cup P(OPT'(i-1))$ , namely  $J''_{y_i} \subseteq (P(OPT(i-1)) \cup P(OPT'(i-1)))$ . Then by the induction hypothesis,

$$\sum_{x \in \Gamma_{i-1}} \text{gap}(x) \leq |P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J''_{y_i}|. \quad (5)$$

Adding  $\sum_{x \in \Gamma_{i-1}} \text{gap}(x)$  and  $\text{gap}(x_i)$  together:

$$\begin{aligned}
 & \left( \sum_{x \in \Gamma_{i-1}} \text{gap}(x) \right) + (\text{gap}(x_i)) \\
 &= |P(\text{OPT}(i-1))| - |P(\Gamma_{i-1})| + |P(\text{OPT}'(i-1))| + |J_{y_i} \setminus J''_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}| \\
 &\leq (|P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J''_{y_i}|) + |J_{y_i} \setminus J''_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}| \quad \text{by (5)} \\
 &= |P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J_{y_i}| - |J_{x_i} \setminus P(\Gamma_{i-1})| + |J'_{x_i}| \\
 &\leq |P(\Gamma_{i-1})| - |J_{y_i} \cap P(\Gamma_{i-1})| + |J_{y_i} \cap P(\Gamma_{i-1})| + |J'_{x_i}| \quad \text{by (4)} \\
 &= |P(\Gamma_{i-1})| + |J'_{x_i}| \leq |P(\Gamma_{i-1})| + |J_{x_i} \setminus P(\Gamma_{i-1})| \quad \text{by definition of } J'_{x_i} \\
 &= P(\Gamma_i)
 \end{aligned}$$

Therefore, by the property of induction, the lemma holds.  $\blacktriangleleft$

► **Theorem 6.** *Given the hypergraph instance  $H(D, R, E)$ . Algorithm 3 computes a solution  $\Gamma$  to  $H$  such that  $2|P(\Gamma)| \geq |P(\text{OPT})|$ , where  $\text{OPT}$  is an optimal solution, with running time  $O(|D| \cdot |E|)$  and  $|E| \leq |D| \cdot (|R| + 1)^K$ .*

**Proof.** Let  $\Gamma = \{x_1, \dots, x_a\}$ ,  $P(\text{OPT}(a)) = \bigcup_{1 \leq i \leq a} J_{y_i}$  and  $P(\text{OPT}'(a)) = \bigcup_{1 \leq i \leq a} J'_{x_i}$ . By Proposition 4,  $P(\text{OPT}) = P(\text{OPT}(a)) \cup P(\text{OPT}'(a))$ , and by Lemma 5,  $|P(\text{OPT}(a))| + |P(\text{OPT}'(a))| - |P(\Gamma_a)| \leq |P(\Gamma_a)|$ . We have

$$|P(\text{OPT})| \leq |P(\text{OPT}(a))| + |P(\text{OPT}'(a))| \leq 2|P(\Gamma)|.$$

In each iteration of the while-loop, it takes  $O(E)$  to find an edge  $x$  with maximum  $|J_x \setminus P(\Gamma)|$ , and there are at most  $|D|$  iterations. Hence, Algorithm 3 runs in  $O(|D| \cdot |E|)$  time.  $\blacktriangleleft$

## 4.2 Approximation algorithms for maximum weighted set packing

We briefly explain the algorithms for the maximum weighted set packing problem, which solve our maximization problem. Given a universe  $\mathcal{U}$  and a family  $\mathcal{S}$  of subsets of  $\mathcal{U}$ , a *packing* is a subfamily  $\mathcal{C} \subseteq \mathcal{S}$  of sets such that all sets in  $\mathcal{C}$  are pairwise disjoint. Every subset  $S \in \mathcal{S}$  has at most  $k$  elements and is given a real weight. The maximum weighted  $k$ -set packing problem (MWSP) asks to find a packing  $\mathcal{C}$  with the largest total weight. We can see that the maximization problem on  $H(D, R, E)$  is a special case of the maximum weighted  $k$ -set packing problem, where the trips of  $D \cup R$  is the universe  $\mathcal{U}$  and  $E(H)$  is the family  $\mathcal{S}$  of subsets, and every  $e \in E(H)$  represents at most  $k = K + 1$  trips ( $K$  is the maximum capacity of all vehicles). Hence, solving MWSP also solves our maximization problem. Hazan et al. [15] showed that the  $k$ -set packing problem cannot be approximated to within  $O(\frac{k}{\ln k})$  in general unless  $P = NP$ . Chandra and Halldórsson [7] presented a  $\frac{2(k+1)}{3}$ -approximation and a  $\frac{2(2k+1)}{5}$ -approximation algorithms (refer to as *BestImp* and *AnyImp* respectively), and Berman [5] presented a  $(\frac{k+1}{2} + \epsilon)$ -approximation algorithm (refer to as *SquareImp*) for the weighted  $k$ -set packing problem (here,  $k = K + 1$ ), where the latter still has the best approximation ratio. These three algorithms in [5, 7] (*AnyImp*, *BestImp* and *SquareImp*) solve the weighted  $k$ -set packing problem by first transferring it into a weighted independent set problem, which consists of a vertex weighted graph  $G(V, E)$  and asks to find a maximum weighted independent set in  $G(V, E)$ . These algorithms use a greedy algorithm (refer to as *Greedy*) to find an initial independent set solution  $I$  and then use local searches to improve the weight of the solution. Algorithm *Greedy* can be summarized as follows: Select a vertex  $u \in V(G)$  with largest weight and add  $u$  to  $I$ . Eliminate  $u$  and all  $u$ 's neighbors from being selected. Repeatedly select the largest weight vertex until all vertices are eliminated from  $G$ .

To apply these algorithms to our maximization problem, we need to convert the bipartite hypergraph  $H(D, R, E)$  to a weighted independent set instance  $G(V, E)$ . The details of the conversion can be found in [13], Section 4.3. Notice that Algorithm 3 is a simplified version of algorithm Greedy, and Greedy is used to get an initial solution in algorithms AnyImp, BestImp and SquareImp. From Theorem 6, we have Corollary 7.

► **Corollary 7.** *Greedy, AnyImp, BestImp and SquareImp algorithms compute a solution to  $H(D, R, E)$  with 2-approximation ratio.*

Since Algorithm 3 finds a solution directly on  $H(D, R, E)$  without converting it to  $G(V, E)$  and solving the independent set problem of  $G(V, E)$ , it is more time and space efficient than the algorithms for MWSP. In the rest of this paper, Algorithm 3 is referred to as ImpGreedy.

## 5 Numerical experiments

We create a simulation environment consists of a centralized system that integrates public transit and ridesharing. We implement our proposed approximation algorithm (ImpGreedy) and Greedy, AnyImp and BestImp algorithms for the  $k$ -set packing problem to evaluate the benefits of having an integrated transportation system supporting public transit and ridesharing. The exact algorithm, ILP (1)-(3), is not evaluated because it takes too long to complete for the instances in our study. The results of SquareImp are not discussed because its performance is same as AnyImp; this is due to the implementation of the search/enumeration order of the vertices and edges in the independent set instance  $G(V, E)$  being fixed, and each vertex in  $V(G)$  has integer weight. We use a simplified transit network of Chicago to simulate the public transit and ridesharing.

### 5.1 Description and characteristics of datasets

We built a simplified transit network of Chicago to simulate practical scenarios of public transit and ridesharing. The roadmap data of Chicago is retrieved from OpenStreetMap<sup>1</sup>. We used the GraphHopper<sup>2</sup> library to construct the logical graph data structure of the roadmap. The Chicago city is divided into 77 officially community areas, each of which is assigned an area code. We examined two different dataset in Chicago to reveal some basic traffic pattern (the datasets are provided by the Chicago Data Portal (CDP) and Chicago Transit Authority (CTA)<sup>3</sup>, maintained by the City of Chicago). The first dataset is bus and rail ridership, which shows the monthly averages and monthly totals for all CTA bus routes and train station entries. We denote this dataset as *PTR*, *public transit ridership*. The PTR dataset contains data for the month June, 2019. The second dataset is rideshare trips reported by Transportation Network Providers (sometimes called rideshare companies) to the City of Chicago. We denote this dataset as *TNP*. The TNP dataset range is chosen from June 3rd, 2019 to June 30th, 2019, total of 4 weeks of data. Table 1 and Table 2 show some basic stats of both datasets.

We examined the 12 busiest bus routes based on the total ridership and selected 7 out of the 12 routes as listed in Table 1 to build the transit network. We also selected all major trains/metro lines within Chicago. Each record in the TNP dataset describes a passenger trip

<sup>1</sup> Planet OSM. <https://planet.osm.org>

<sup>2</sup> GraphHopper 1.0. <https://www.graphhopper.com>

<sup>3</sup> CDP. <https://data.cityofchicago.org>. CTA. <https://www.transitchicago.com>



## 39:10 Multimodal Transportation with Ridesharing of Personal Vehicles

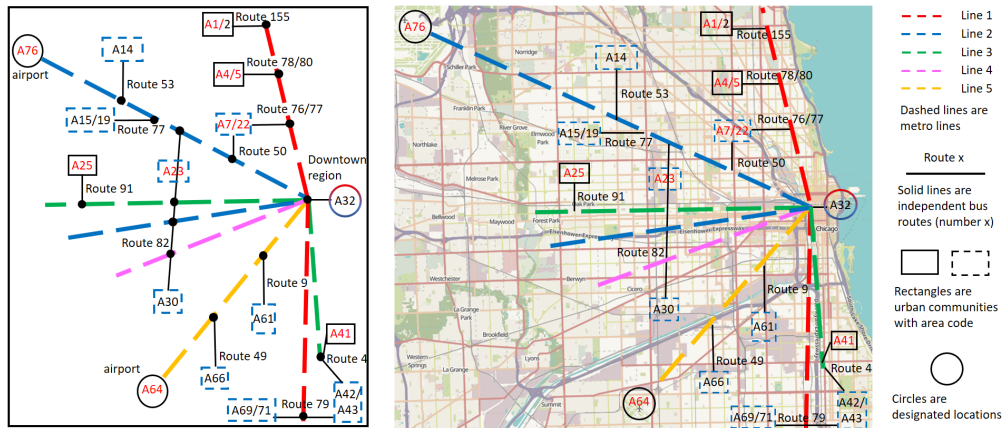
■ **Table 1** Basic stats of the PTR dataset.

Total Bus Ridership	20,300,416
Total Rail Ridership	19,282,992
12 busiest bus routes	3, 4, 8, 9, 22, 49, 53, 66, 77, 79, 82, 151
The busiest bus routes selected	4, 9, 49, 53, 77, 79, 82

■ **Table 2** Basic stats of the TNP dataset.

# of original records	8,820,037
# of records considered	7,427,716
# of shared trips	1,015,329
# of non-shared trips	6,412,387
The most visited community areas selected	1, 4, 5, 7, 22, 23, 25, 32, 41, 64, 76

served by a driver who provides the rideshare service; a trip record consists of the pick-up and drop-off time and the pick-up and drop-off community area of the trip, and exact locations are provided sometimes. We selected 11 of the 20 most visited areas as listed in Table 2 (area 32 is Chicago downtown, areas 64 and 76 are airports) to build the transit network for our simulation. From the selected bus routes, trains and community areas, we create a simplified public transit network connecting the selected areas, depicted in Figure 2. More details on selecting areas and the public transit network are included in [13] (Section 5.1).



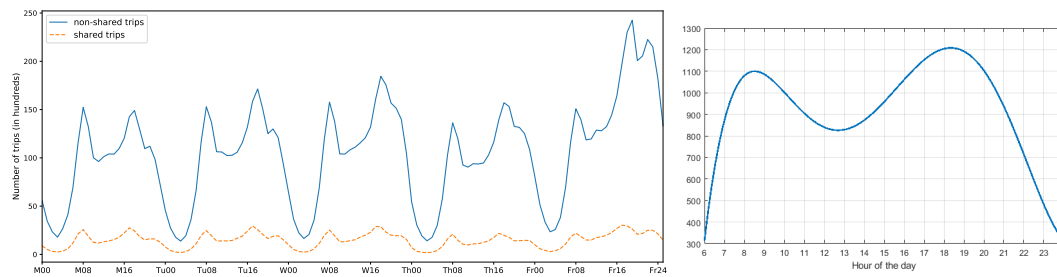
■ **Figure 2** Simplified public transit network of Chicago with 13 urban communities and 3 designated locations. Figure on the right has the Chicago city map overlay for scale.

The travel time between two locations uses the fastest/shortest route computed by GraphHopper, based on personal cars. The shortest paths are **computed in real-time**, unlike many previous simulations where the shortest paths are precomputed and stored. As stated in Section 3, waiting time and service time are considered in a simplified model; we multiply a small constant  $\epsilon > 1$  to the fastest route to mimic waiting time and service time.

### 5.2 Generating instances

In our simulation, we partition each day from 6:00 to 23:59 into 72 time intervals (each has 15 minutes), and we only focus on weekdays. To see ridesharing traffic pattern, we calculated the average number of served trips per hour for each day of the week using the TNP dataset. The dashed (orange) line and solid (blue) line of the plot in Figure (3a) represent shared trips and non-shared trips respectively. A set of trips are called *shared trips* if this set of trips are matched for the same vehicle consecutively such that their trips may potentially overlap (one or more passengers are in the same vehicle). For all other trips, we call them *non-shared trips*. The number of trips generated for each interval is plotted in Figure (3b), which is a





(a) Average numbers of shared and non-shared trips in TNP dataset. (b) Total number of driver and rider trips generated for each time interval.

■ **Figure 3** Plots for the number of trips for every hour from data and generated.

scaled down and smoothed version of the TNP dataset for weekdays. The ratio between the number of drivers and riders generated is roughly 1:3 (1 driver and 3 riders) for each interval. Such a ratio is chosen because it should reflect the system's potential as capacity of 3 is common for most vehicles. For each time interval, we first generate a set  $R$  of riders and then a set  $D$  of drivers. We do not generate a trip where its origin and destination are close.

The main idea of generating rider trips is described as follows. Each day is divided into 6 different consecutive time periods (each consists of multiple time intervals): morning rush, morning normal, noon, afternoon normal, afternoon rush, and evening time periods. Each time period determines the probability and distribution of origins and destinations. Based on the PTR dataset and Rail Capacity Study [8], many riders are going into downtown in the morning and leaving downtown in the afternoon. To generate a rider trip  $j$  during a time period, we first select a *pick-up area* and a *drop-off area* randomly following the probability distribution for the time period (e.g., downtown is selected with higher probability as drop-off area for the morning rush period). The origin  $o_j$  and destination  $d_j$  are random points within the pick-up and drop-off areas respectively. The above is repeated until  $a_t$  riders are generated, where  $a_t + a_t/3$  (riders + drivers) is the total number of trips for time interval  $t$  shown in Figure (3b). The probability distribution for each time period and detailed description of generating rider trips can be found in Section 5.2 of [13].

The main idea of generating driver trips is described as follows. We examined the TNP dataset to create a grid heatmap (Figure 8 in [13]) for traffic for each hour. Each cell  $(c, r)$  in the heatmap represents the the average number of trips per hour originated from area  $c$  to destination area  $r$  in the transit network (Figure 2). Let  $d(c, r, h)$  be the value at the cell  $(c, r)$  for origin  $c$ , destination  $r$  and hour  $h$  in the heatmap. Let  $P(c, h) = \sum_r d(c, r, h)$  be the sum of the values of the whole column  $c$  for hour  $h$ . Given a time interval  $t$  in hour  $h$ , let  $c_t$  be the number of generated riders with origin in area  $c$ ; and for each area  $c$ , we generate  $c_t/3$  drivers such that each driver  $i$  has origin  $o_i = c$  and destination  $d_i = r$  with probability  $d(c, r, h)/P(c, h)$ . The probability of selecting an airport as destination is fixed at 5%.

After the origin and destination of a rider/driver trip have been determined, we decide other parameters of the trip (e.g., the vehicle capacity of a driver  $i$  is at most 6). Details of these parameters are specified in Section 5.2 of [13].

When the number of trips increases, the running time for Algorithm 2 and the time needed to construct the  $k$ -set packing instance also increase. In a practical setup, we may restrict the number of feasible matches a driver can have. Each match produced by Algorithm 1 is called a *base match*. To make the simulation feasible, we heuristically limit the numbers of base matches for each driver and each rider and the number of total feasible matches for each

driver. We use  $(x\%, y, z)$ , called *reduction configuration* (*Config* for short), to denote that for each driver  $i$ , the number of base matches of  $i$  is reduced to  $x$  percentage and at most  $y$  total feasible matches are computed for  $i$ ; and for each rider  $j$ , at most  $z$  base matches containing  $j$  are used. Details of the reduction procedure can be found in the end of Section 5.2 of [13].

### 5.3 Computational results

We use the same transit network and same set of generated trip data for all algorithms. All experiments were implemented in Java and conducted on Intel Core i7-2600 processor with 1333 MHz of 8 GB RAM available to JVM. Since the optimization goal is to assign feasible acceptable routes to as many riders as possible, the performance measure is focused on the number of riders served by ridesharing routes, followed by the total time saved for the riders as a whole. The base case instance uses the parameter setting described in Section 5.2 and Config (30%, 600, 20). The experiment results are shown in Table 3. The results of

■ **Table 3** Base case solution comparison between the approximation algorithms.

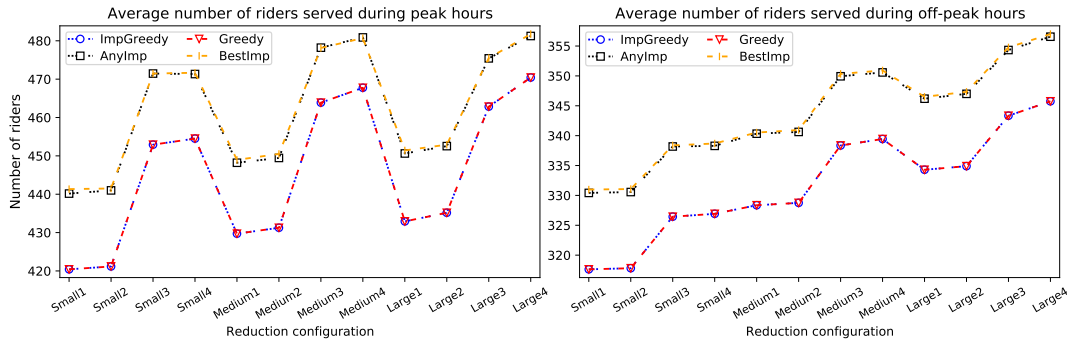
	ImpGreedy	Greedy	AnyImp	BestImp
Total number of riders served	27413	27413	28248	28258
Avg number of riders served per interval	380.736	380.736	392.333	392.472
Total time saved of all riders (minute)	354568.2	354568.2	365860.6	365945.8
Avg time saved of riders per interval (minute)	4924.56	4924.56	5,081.40	5082.58
Total number of riders and public transit duration	45314 and 1383743.97 minutes			

ImpGreedy and Greedy are aligned since they are essentially the same algorithm – 60.5% of total passengers are assigned ridesharing routes and 25.6% of total time are saved. The results of AnyImp and BestImp are similar because of the density of the graph  $G(V, E)$ . For AnyImp and BestImp, roughly 62.4% of total passengers are assigned ridesharing routes and 26.4% of total time are saved. On average, passengers are able to reduce their travel duration from 30.5 minutes to 22.5 minutes by using public transit plus ridesharing. The results of these four algorithms are not too far apart. However, it takes too long for AnyImp and BestImp to run to completion. A 10-second limit is set for both algorithms in each iteration for finding an independent set improvement. With this time limit, AnyImp and BestImp run to completion within 15 minutes for almost all intervals. We also recorded the mean occupancy rate of drivers. The mean occupancy rate is calculated as, in each interval, the number of passengers served divided by the number of drivers who serve them. The results are depicted in Figure 9 in [13], which show that mean occupancy rate of a personal vehicle is 2.9–3 (including the driver) on average. Further discussions can be found in [13] (Section 5.3).

Another major component of the experiment is to measure the computational time of the algorithms, which is highly affected by the base match reduction configurations. By reducing more matches, we are able to improve the running time of AnyImp and BestImp significantly, but sacrifice performance slightly. We tested 12 different Configs:

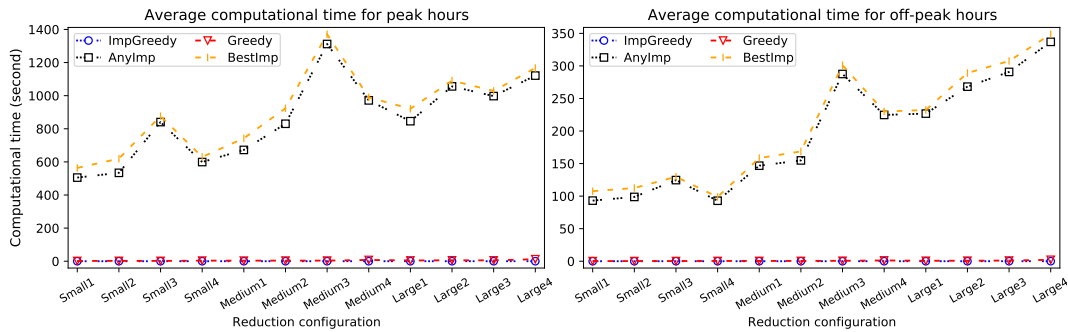
- *Small1* (20%,300,10), *Small2* (20%,600,10), *Small3* (20%,300,20), *Small4-10* (20%,600,20), *Medium1* (30%,300,10), *Medium2* (30%,600,10), *Medium3* (30%,300,20), *Medium4-10* (30%,600,20), *Large1* (40%,300,10), *Large2* (40%,600,10), *Large3-10* (40%,300,20), and *Large4-10* (40%,600,20).

Configs with label “-10” have a 10-second limit to find an independent set improvement, and all other Configs have 20-second limit. Notice that all 12 Configs have the same sets of driver/rider trips and base match sets but generate different feasible match sets. The



■ **Figure 4** Average performance of peak and off-peak hours for different configurations.

performance and running time results of all 12 Configs are depicted in Figures 4 and 5 respectively. The results are divided into peak and off-peak hours for each Config (averaging all intervals of peak hours and off-peak hours). The running time of ImpGreedy and Greedy are within seconds for all Configs for each interval. On the other hand, it may not be practical to use AnyImp and BestImp for peak hours since they require around 15 minutes for most Configs. Since AnyImp and BestImp provide better performance than ImpGreedy/Greedy when each Config is compared side-by-side, one can use ImpGreedy/Greedy for peak hours and AnyImp/BestImp for off-peak hours so that it becomes practical. The increase in



■ **Figure 5** Average running time of peak and off-peak hours for different configurations.

performance from Small1 to Small3 is much larger than that from Small1 to Small2 (same for Medium and Large), implying any parameter in a Config should not be too small. The increase in performance from Large1 to Large4 is higher than that from Medium1 to Medium4 (similarly for Small). Therefore, a balanced configuration is more important than a configuration emphasizes only one or two parameters.

Because ImpGreedy does not create the independent set instance, it runs quicker than Greedy. More importantly, ImpGreedy uses less memory space than Greedy does. We tested ImpGreedy and Greedy with the following Configs: *Huge1* (100%,600,10), *Huge2* (100%,2500,20) and *Huge3* (100%,10000,30) (these Configs have the same sets of driver/ rider trips and base match sets as those in the previous 12 Configs). The focus of these Configs is to see if Greedy can handle large number of feasible matches. The results are shown in Table 4. Greedy cannot run to completion for all Configs because in many intervals, the whole graph  $G(V, E)$  of the independent set instance is too large to hold in memory. The average number of edges for afternoon peak hours is 0.02, 0.38 and 5.47 billion for *Huge1*, *Huge2* and *Huge3* respectively. Further, the time it takes to create  $G(V, E)$  excess practicality. Hence, using Greedy (AnyImp/BestImp) for large instances may not be practical. In addition, the performance of ImpGreedy with *Huge3* is better than that of AnyImp/BestImp with Large4.

■ **Table 4** The results of ImpGreedy and Greedy using Unlimited reduction configurations.

ImpGreedy	Huge1	Huge2	Huge3
Avg running time for peak/off-peak hours (sec)	0.08 / 0.03	0.43 / 0.12	1.2 / 0.29
Avg number of riders served for peak/off-peak hours	406.9 / 339.0	458.8 / 355.4	484.1 / 361.9
Avg time saved of riders per interval (sec)	284891.8	302774.1	310636.9
Greedy	Huge1	Huge2	Huge3
Avg running time	N/A	N/A	N/A
Avg instance size $G(V, E)$ of afternoon peak ( $ E(G) $ )	0.02 billion	0.38 billion	5.47 billion
Avg time creating $G(V, E)$ of afternoon peak (sec)	14.6	320.9	3726.79

We also looked at the total running times of the approximation algorithms including the time for computing feasible matches (Algorithms 1 and 2). The running time of Algorithm 1 solely depends on computing the shortest paths between the trips and stations. Table 5 shows that Algorithm 1 runs to completion within 500 seconds for each interval on average for peak hours. As for Algorithm 2, when many trips' origins/destinations are concentrated

■ **Table 5** Average computational time (in seconds) of peak hours for all algorithms.

	Alg1	Alg2	ImpGreedy	Greedy	AnyImp	BestImp	Total computational time			
							ImpGreedy	Greedy	AnyImp	BestImp
Small3	485.2	26.8	0.021	2.0	840.5	876.4	512.1	514.1	1352.5	1388.5
Small4	485.2	28.2	0.029	3.6	599.1	629.9	513.4	517.0	1112.5	1143.3
Medium3	485.2	43.6	0.031	3.7	1312.1	1371.0	532.5	543.0	1840.9	1899.9
Medium4	485.2	50.1	0.048	7.7	971.5	990.0	535.3	543.0	1506.8	1525.3
Large4	485.2	72.0	0.076	12.2	1121.3	1167.2	557.3	569.5	1678.6	1724.4
Huge3	485.2	339.4	1.2	N/A	N/A	N/A	825.8	N/A	N/A	N/A

in one area, the running time increases significantly, especially for drivers with high capacity. Combining the results of this and previous (Table 4) experiments, ImpGreedy is capable of handling large instances while providing quality solution compared to other approximation algorithms.

From the experiment results in Figures 4 and 5, it is beneficial to dynamically select different algorithms and reduction configurations for each interval depending on the number of trips. With large problem instances, previous approximation algorithms are not efficient (time and memory consuming), so they require aggressive reduction to reduce the instance size. On the other hand, ImpGreedy is much faster and capable of handling large instances. The running time of ImpGreedy can also be an advantage to improve the quality of solutions. For example, as shown in Figures 4 and 5, for the same set of drivers and riders, ImpGreedy assigns more riders when taking Medium/Medium4 as inputs than AnyImp/BestImp on Small1/Small2, and uses less time than AnyImp/BestImp. When the size of an instance is not small and a solution must be computed within some time-limit, ImpGreedy has a distinct advantage over the previous approximation algorithms.

## 6 Conclusion

Based on real-world transit datasets in Chicago, our study has shown that integrating public and private transportation can benefit the transit system as a whole, more than 60% of the passenger are assigned ridesharing routes and able to save 25% of travel time. Majority of the drivers are matched with at least one passenger, and vehicle occupancy rate has improved

close to 3 (including the driver) on average. These results suggest that ridesharing can be a complement to public transit. Our experiments show that the whole system is capable of handling more than 1100 trip requests in real-time using ordinary computer hardware. The performance results of ImpGreedy may be further improved by extending it with the local search strategy. Perhaps the biggest challenge for scalability comes from computing the base matches (Algorithm 1) since it has to compute many shortest paths; it may be worth to apply heuristics to speed-up Algorithm 1. To better understand practicality, a more sophisticated simulation incorporating traffic and transit schedule and demand may be needed.

---




## References

- 1 Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological*, 45(9):1450–1464, 2011. doi:10.1016/j.trb.2011.05.017.
- 2 Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012. doi:10.1016/j.ejor.2012.05.028.
- 3 Kamel Aissat and Sacha Varone. Carpooling as complement to multi-modal transportation. In *Enterprise Information Systems*, pages 236–255, January 2015. doi:10.1007/978-3-319-29133-8\_12.
- 4 Javier Alonso-Mora, Samitha Samaranyake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017. doi:10.1073/pnas.1611675114.
- 5 Piotr Berman. A  $d/2$  approximation for maximum weight independent set in  $d$ -claw free graphs. In *Algorithm Theory - SWAT 2000*, pages 214–219. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-44985-X\_19.
- 6 Center for Sustainable Systems, University of Michigan. Personal transportation factsheet, 2020. URL: <http://css.umich.edu/factsheets/personal-transportation-factsheet>.
- 7 Barun Chandra and Magnús M Halldórsson. Greedy local improvement and weighted set packing approximation. *Journal of Algorithms*, 39(2):223–240, 2001. doi:10.1006/jagm.2000.1155.
- 8 Chicago Transit Authority. System-wide rail capacity study. Published June 2017; Revised February 2019. URL: [https://www.transitchicago.com/assets/1/6/RP\\_CDMSMITH\\_RCM\\_Task2AExecutiveSummary\\_20170628\\_FINAL.pdf](https://www.transitchicago.com/assets/1/6/RP_CDMSMITH_RCM_Task2AExecutiveSummary_20170628_FINAL.pdf).
- 9 Sharon Feigon and Colin Murphy. Shared mobility and the transformation of public transit. TCRP Research Report, Transportation Research Board, 2016. doi:10.17226/23578.
- 10 Masabumi Furuhashi, Maged Dessouky, Fernando Ordóñez, Marc-Etienne Brunet, Xiaoqing Wang, and Sven Koenig. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, 57:28–46, 2013. doi:10.1016/j.trb.2013.08.012.
- 11 Michael R. Garey and David S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.
- 12 Keivan Ghoseiri, Ali Haghani, and Masoud Hamedi. Real-time rideshare matching problem. Final Report of UMD-2009-05, U.S. Department of Transportation, 2011.
- 13 Qian-Ping Gu and Jiajian Leo Liang. Multimodal transportation with ridesharing of personal vehicles, 2021. arXiv:2106.00232.
- 14 Qian-Ping Gu, Jiajian Leo Liang, and Guochuan Zhang. Approximate ridesharing of personal vehicles problem. *Theoretical Computer Science*, 871:30–50, 2021. doi:10.1016/j.tcs.2021.04.009.
- 15 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating  $k$ -set packing. *Computational Complexity*, 15(1):20–39, 2006. doi:10.1007/s00037-006-0205-6.
- 16 John Hopcroft and Richard Karp. An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

- 17 Haosheng Huang, Dominik Bucher, Julian Kissling, Robert Weibel, and Martin Raubal. Multimodal route planning with public transport and carpooling. *IEEE Transactions on Intelligent Transportation Systems*, 20(9):3513–3525, 2019. doi:10.1109/TITS.2018.2876570.
- 18 Richard M. Karp. *Reducibility among combinatorial problems*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2\_9.
- 19 Tai-Yu Ma. On-demand dynamic bi-/multi-modal ride-sharing using optimal passenger-vehicle assignments. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*, pages 1–5, 2017. doi:10.1109/EEEIC.2017.7977646.
- 20 Tai-Yu Ma, Saeid Rasulkhani, Joseph Y.J. Chow, and Sylvain Klein. A dynamic ridesharing dispatch and idle vehicle repositioning strategy with integrated transit transfers. *Transportation Research Part E: Logistics and Transportation Review*, 128:417–442, 2019. doi:10.1016/j.tre.2019.07.002.
- 21 Abood Mourad, Jakob Puchinger, and Chengbin Chu. A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological*, 123:323–346, 2019. doi:10.1016/j.trb.2019.02.003.
- 22 Arvind U Raghunathan, David Bergman, John Hooker, Thiago Serra, and Shingo Kobori. Seamless multimodal transportation scheduling, 2018. arXiv:1807.09676.
- 23 Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–13294, 2014. doi:10.1073/pnas.1403657111.
- 24 A. Santos, N. McGuckin, H.Y. Nakamoto, D. Gray, and S. Liss. Summary of travel trends: 2009 national household travel survey. Technical report, US Department of Transportation Federal Highway Administration, 2011.
- 25 Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, 82:36–53, 2015. doi:10.1016/j.trb.2015.07.025.
- 26 Mitja Stiglic, Niels Agatz, Martin Savelsbergh, and Mirko Gradisar. Enhancing urban mobility: Integrating ride-sharing and public transit. *Computers & Operations Research*, 90:12–21, 2018. doi:10.1016/j.cor.2017.08.016.
- 27 Amirmahdi Tafreshian, Neda Masoud, and Yafeng Yin. Frontiers in service science: Ride matching for reer-to-reer ride sharing: A review and future directions. *Service Science*, 12(2-3):41–60, 2020. doi:10.1287/serv.2020.0258.
- 28 Hai Wang and Amedeo Odoni. Approximating the performance of a “last mile” transportation system. *Transportation Science*, 50(2):659–675, 2014. doi:10.1287/trsc.2014.0553.
- 29 Zhengtian Xu, Yafeng Yin, and Jieping Ye. On the supply curve of ride-hailing systems. *Transportation Research Part B: Methodological*, 132:29–43, 2020. 23rd International Symposium on Transportation and Traffic Theory (ISTTT 23). doi:10.1016/j.trb.2019.02.011.



# Algorithms for Normalized Multiple Sequence Alignments

Eloi Araujo<sup>1</sup>   

Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brasil

Luiz C. Rozante   

Centro de Matemática Computação e Cognição, Universidade Federal do ABC, Santo André, MS, Brasil

Diego P. Rubert   

Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brasil

Fábio V. Martinez   

Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brasil

---

## Abstract

---

Sequence alignment supports numerous tasks in bioinformatics, natural language processing, pattern recognition, social sciences, and other fields. While the alignment of two sequences may be performed swiftly in many applications, the simultaneous alignment of multiple sequences proved to be naturally more intricate. Although most multiple sequence alignment (MSA) formulations are NP-hard, several approaches have been developed, as they can outperform pairwise alignment methods or are necessary for some applications. Taking into account not only similarities but also the lengths of the compared sequences (i.e. normalization) can provide better alignment results than both unnormalized or post-normalized approaches. While some normalized methods have been developed for pairwise sequence alignment, none have been proposed for MSA. This work is a first effort towards the development of normalized methods for MSA. We discuss multiple aspects of normalized multiple sequence alignment (NMSA). We define three new criteria for computing normalized scores when aligning multiple sequences, showing the NP-hardness and exact algorithms for solving the NMSA using those criteria. In addition, we provide approximation algorithms for MSA and NMSA for some classes of scoring matrices.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness; Theory of computation → Approximation algorithms analysis; Applied computing → Molecular sequence analysis

**Keywords and phrases** Multiple sequence alignment, Normalized multiple sequence alignment, Algorithms and complexity

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.40

**Related Version** *Full Version:* <https://arxiv.org/abs/2107.01607> [4]

**Acknowledgements** We thank the three anonymous reviewers for their valuable comments and suggestions on our manuscript.

## 1 Introduction

Sequence alignment lies at the foundation of bioinformatics. Several procedures rely on alignment methods for a range of distinct purposes, such as detection of sequence homology, secondary structure prediction, phylogenetic analysis, identification of conserved motifs or genome assembly. On the other hand, alignment techniques have also been reshaped and found applications in other fields, such as natural language processing, pattern recognition, or social sciences [1, 3, 8, 18].

---

<sup>1</sup> Corresponding author





Given its range of applications in bioinformatics, extensive efforts have been made to improve existing or developing novel methods for sequence alignment. The simpler ones compare a pair of sequences in polynomial time on their lengths, usually trying to find editing operations (insertions, deletions, and substitutions of symbols) that transform one sequence into another while maximizing or minimizing some objective function called edit distance [16]. This concept can naturally be generalized to align multiple sequences [28], adding another new layer of algorithmic complexity, though. In this case, most multiple sequence alignment (MSA) formulations lead to NP-hard problems [13]. Nevertheless, a variety of methods suitable for aligning multiple sequences have been developed, as they can outperform pairwise alignment methods on tasks such as phylogenetic inference [21], secondary structure prediction [12] or identification of conserved regions [24].

In order to overcome the cost of exact solutions, a number of MSA heuristics have been developed in recent years, most of them using the so-called progressive or iterative methods [17, 23, 25, 27]. Experimental data suggest that the robustness and accuracy of heuristics can still be improved, however [28].

Most approaches for pairwise sequence alignment define edit distances as absolute values, lacking some normalization that would result in edit distances relative to the lengths of the sequences. However, some applications may require sequence lengths to be taken into account. For instance, a difference of one symbol between sequences of length 5 is more significant than between sequences of length 1000. In addition, experiments suggest that normalized edit distances can provide better results than both unnormalized or post-normalized edit distances [18]. While normalized edit distances have been developed for pairwise sequence alignment [6, 18], none have been proposed for MSA to the best of our knowledge.

In this work, we propose exact and approximation algorithms for normalized MSA (NMSA). This is a first step towards the development of methods that take into account the lengths of sequences for computing edit distances when multiple sequences are compared.

The remainder of this paper is organized as follows. Section 2 introduces concepts related to sequence alignment and presents normalized scores for NMSA, followed by the complexity analysis of NMSA using those scores in Section 3. Next, Sections 4 and 5 describe exact and approximation algorithms, respectively. Section 6 closes the paper with the conclusion and prospects for future work.

## 2 Preliminaries

An *alphabet*  $\Sigma$  is a finite non-empty set of *symbols*. A finite sequence  $s$  with  $n$  symbols in  $\Sigma$  is seen as  $s(1) \cdots s(n)$ . We say that the *length* of  $s$ , denoted by  $|s|$ , is  $n$ . The (sub)sequence  $s(p) \cdots s(q)$  of  $s$ , with  $1 \leq p \leq q \leq n$ , is denoted by  $s(p:q)$ . If  $p > q$ ,  $s(p:q)$  is the *empty* sequence, whose length is zero, and it is denoted by  $\varepsilon$ . We denote the sequence resulting from the concatenation of sequences  $s$  and  $t$  by  $st$ . A sequence of  $n$  symbols  $\mathbf{a}$  is denoted by  $\mathbf{a}^n$ . A  $k$ -tuple  $S$  over  $\Sigma^*$  is called a  $k$ -*sequence* and we write  $s_1, \dots, s_k$  to refer to  $S$ , where  $s_i$  is the  $i$ -th sequence in  $S$ . Let  $\Sigma_- := \Sigma \cup \{-\}$ , where  $- \notin \Sigma$  and the symbol  $-$  is called a *space*. Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence. An *alignment* of  $S$  is a  $k$ -tuple  $A = [s'_1, \dots, s'_k]$  over  $\Sigma_-^*$ , where (a) each sequence  $s'_h$  is obtained by inserting spaces in  $s_h$ ; (b)  $|s'_h| = |s'_i|$  for each pair  $h, i$ , with  $1 \leq h, i \leq k$ ; and (c) there is no  $j$  in  $\{1, \dots, k\}$  such that  $s'_1(j) = \dots = s'_k(j) = -$ . Notice that  $k$ -tuples over  $\Sigma_-^*$  are written enclosed by square brackets “[ ]”. The sequence  $[s'_1(j), \dots, s'_k(j)]$  is the *column*  $j$  of the alignment  $[s'_1, \dots, s'_k]$ . We denote the column  $j$  of the alignment  $A$  by  $A(j)$  and by  $A[j_1:j_2]$  the columns  $j_1, j_1 + 1, \dots, j_2$  of  $A$ . We say that the pair  $[s'_h(j), s'_i(j)]$  *aligns* in  $A$  or, simply, that  $s'_h(j)$  and  $s'_i(j)$  are *aligned* in  $A$ , and  $|A| = |s'_i|$  is the *length* of the alignment  $A$ . It is easy to check that  $\max_i \{|s_i|\} \leq |A| \leq \sum_i |s_i|$ .

An alignment can be used to represent *editing operations* of *insertions*, *deletions* and *substitutions* of symbols in sequences, where the symbol  $-$  represents insertions or deletions. An alignment can also be represented in the matrix format. Thus, alignments  $[aaa-, ab--, -cac]$  and  $[-aaa-, ab---, -ca-c]$  of  $aaa, ab, cac$  can be represented respectively as

$$\begin{bmatrix} a & a & a & - \\ a & b & - & - \\ - & c & a & c \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} - & a & a & a & - \\ a & b & - & - & - \\ - & c & a & - & c \end{bmatrix}.$$

Let  $I = \{i_1, \dots, i_m\} \subseteq \{1, \dots, k\}$  be a set of indices such that  $i_1 < \dots < i_m$  and let  $A = [s'_1, \dots, s'_k]$  be an alignment of  $S = s_1, \dots, s_k$ . We write  $S_I$  to denote the  $m$ -tuple  $s_{i_1}, \dots, s_{i_m}$ . The alignment of  $S_I$  induced by  $A$  is the alignment  $A_I$  obtained from the alignment  $A$ , considering only the corresponding sequences in  $S_I$  and, from the resulting structure, removing columns where all symbols are  $-$ . In the following example,  $A = [aaa-, ab--, -cac]$  is an alignment of  $aaa, ab, cac$  and

$$\begin{bmatrix} a & a & a \\ a & b & - \end{bmatrix}$$

is an alignment of  $aaa, ab$  induced by  $A$ . We denote by  $\mathcal{A}_S$  the set of all alignments of  $S$ .

For a problem  $P$ , we call  $\mathbb{I}_P$  the set of instances of  $P$ . If  $P$  is a decision problem, then  $P(I) \in \{\text{Yes}, \text{No}\}$  is the image of an instance  $I$ . If  $P$  is an optimization (minimization) problem, there is a set  $\text{Sol}(I)$  for each instance  $I$ , a function  $v$  defining a non-negative rational number for each  $X \in \text{Sol}(I)$ , and a function  $\text{opt}_v(I) = \min_{X \in \text{Sol}(I)} \{v(X)\}$ . We use  $\text{opt}$  instead of  $\text{opt}_v$  if  $v$  is obvious. Let  $\mathbf{A}(I)$  be a solution computed by an algorithm  $\mathbf{A}$  with input  $I$ , and  $\mathbf{A}(I) \geq \text{opt}(I)$ . We say that  $\mathbf{A}$  is an  $\alpha$ -approximation for  $P$  if  $\mathbf{A}(I) \leq \alpha \text{opt}(I)$ , for each  $I \in \mathbb{I}_P$ , with  $\alpha \geq 1$ . We say that  $\alpha$  is an *approximation factor* for  $P$ .

The *alignment problem* is a collection of decision and optimization problems whose instances are finite subsets of  $\Sigma^*$  and  $\text{Sol}(S) = \mathcal{A}_S$ , for each instance  $S$ . Function  $v$ , used for scoring alignments, is called *criterion* for  $P$  and we call  $v[A]$  the *cost of the alignment*  $A$ . The  *$v$ -optimal alignment*  $A$  of  $S$  is such that  $v[A] = \text{opt}(S)$ . Thus, we state the following general optimization problems using the criterion  $v$ :

► **Problem 1 (Alignment with criterion  $v$ ).** Given a  $k$ -sequence  $S$ , with  $k \in \mathbb{N}$ , find a  $v$ -optimal alignment of  $S$ .

We also need the decision version of the alignment problem with criterion  $v$ , where we are given a  $k$ -sequence  $S$  and a number  $d \in \mathbb{Q}_{\geq}$ , and we want to decide whether there exists an alignment  $A$  of  $S$  such that  $v[A] \leq d$ .

Usually the cost of an alignment  $v$  is defined from a scoring matrix. A *scoring matrix*  $\gamma$  is a rational matrix such that the elements in  $\Sigma_-$  are indices of its rows and columns. For  $\mathbf{a}, \mathbf{b} \in \Sigma_-$  and a scoring matrix  $\gamma$ , we denote by  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}}$  the entry of  $\gamma$  in line  $\mathbf{a}$  and column  $\mathbf{b}$ . The value  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}}$  defines the score for a substitution if  $\mathbf{a}, \mathbf{b} \in \Sigma$ , for an insertion if  $\mathbf{a} = -$ , and for a deletion if  $\mathbf{b} = -$ . The entry  $\gamma_{- \rightarrow -}$  is not defined.

Given a scoring function  $v_\gamma$  for alignments that depends on a scoring matrix  $\gamma$ , we say that two scoring matrices  $\gamma$  and  $\rho$  are *equivalent* considering  $v$  when  $v_\gamma[A] \leq v_\gamma[B]$  if and only if  $v_\rho[A] \leq v_\rho[B]$  for any pair of alignments  $A, B$  of sequences  $s, t$ . If  $\rho$  is a matrix obtained from  $\gamma$  by multiplying each entry of  $\gamma$  by a constant  $c > 0$ , then  $v_\rho[A] = c v_\gamma[A]$  and  $vN_\rho[A] = c vN_\gamma[A]$ , which implies that  $\gamma$  and  $\rho$  are equivalent. As a consequence, when the scoring function is  $vA_\gamma$  or  $vN_\gamma$  and it is convenient, we can suppose that all entries of  $\gamma$  are integers instead of rationals, according to the definition.

A  $k$ -vector  $\vec{j} = [j_1, \dots, j_k]$  is a  $k$ -tuple, where  $j_i \in \mathbb{N} = \{0, 1, 2, \dots\}$ . We say that  $j_i$  is the  $i$ -th element of  $\vec{j}$ . The  $k$ -vector  $\vec{0}$  is such that all its elements are zero. If  $\vec{j}$  and  $\vec{h}$  are  $k$ -vectors, we write  $\vec{j} \leq \vec{h}$  if  $j_i \leq h_i$  for each  $i$ ; and  $\vec{j} < \vec{h}$  if  $\vec{j} \leq \vec{h}$  and  $\vec{j} \neq \vec{h}$ . A sequence of  $k$ -vectors  $\vec{j}_1, \vec{j}_2, \dots$  is in *lexicographical order* if  $\vec{j}_i \leq \vec{j}_{i+1}$  for each  $i$ .

Consider  $S = s_1, \dots, s_k$  a  $k$ -sequence with  $n_i = |s_i|$  for each  $i$  and  $\vec{n} = [n_1, \dots, n_k]$ . Let  $V_{\vec{n}} = \{\vec{j} : \vec{j} \leq \vec{n}\}$  be the set of all  $k$ -vectors  $\vec{j}$  such that  $\vec{j} \leq \vec{n}$ . For example, if  $k = 3$  and  $\vec{n} = [1, 2, 1]$ , then  $V_{\vec{n}} = \{[x, y, z] : x, y, z \in \mathbb{N}, x \leq 1, y \leq 2, z \leq 1\}$ . Notice that if  $n_i = n$  for all  $i$ , then  $|V_S| = (n+1)^k$ . Define  $S(\vec{j}) = s_1(j_1), \dots, s_k(j_k)$  a column  $\vec{j}$  in  $S$  and we say that  $S(1:\vec{j}) = s_1(1:j_1), \dots, s_k(1:j_k)$  is the *prefix of  $S$  ending in  $\vec{j}$* . Thus,  $S = S(1:\vec{n})$ . Besides that, if  $A$  is an alignment and  $\vec{v} = [j, j, \dots, j]$ , then  $A[\vec{v}] = A(j)$ .

Denote by  $\mathcal{B}^k$  the set of  $k$ -bit vectors  $[b_1, \dots, b_k]$ , where  $b_i \in \{0, 1\}$  for each  $i$ . Now, for  $\vec{b} \leq \vec{j}$ , define  $\vec{b} \cdot S(\vec{j}) = [x_1, \dots, x_k] \in \Sigma^k$  such that  $x_i = s_i(j_i)$  if  $b_i = 1$  and  $x_i = -$  otherwise. Therefore, given an alignment  $A$  of  $S(1:\vec{j})$ , there exists  $\vec{b} \in \mathcal{B}^k$ , with  $\vec{b} \leq \vec{j}$ , such that  $A(|A|) = \vec{b} \cdot S(\vec{n})$ . In other words, if  $\vec{n} = [n_1, \dots, n_k]$  and  $\vec{b} = [b_1, \dots, b_k]$ , we have  $b_i = 1$  if and only if  $s_i(n_i)$  is in the  $i$ -th row of the last column of  $A$ . We also define the operation  $\vec{j} - \vec{b} = [j_1 - b_1, \dots, j_k - b_k]$ . Notice that  $|\mathcal{B}_k| = 2^k$ .

Consider a scoring matrix  $\gamma$ . Let  $s, t \in \Sigma^*$ , with  $n = |s|, m = |t|$ . A simple criterion for scoring alignments using the function  $vA_\gamma$  follows. For an alignment  $[s', t']$  of  $s, t$  we define

$$vA_\gamma[s', t'] = \sum_{j=1}^{|[s', t']|} \gamma_{s'(j) \rightarrow t'(j)}.$$

We say that  $vA_\gamma[s', t']$  is a  $vA_\gamma$ -score of  $s, t$ . The optimal function for this criterion is denoted by  $\text{opt}A_\gamma$  and an alignment  $A$  of  $s, t$  is called an *A-optimal alignment* of  $s, t$  if  $vA_\gamma[A] = \text{opt}A_\gamma(s, t)$ .

Now, suppose that  $n \geq m$ . Needleman and Wunch [20] proposed an  $O(n^2)$ -time algorithm for computing  $\text{opt}A_\gamma(s, t)$ . If  $\text{opt}A_\gamma$  is a Levenstein distance, Masek and Paterson [19] presented an  $O(n^2/\log n)$ -time algorithm using the “*Four Russian’s Method*”. Crochemore, Landau and Ziv-Ukelson [11] extended this result for real arrays, describing an  $O(n^2/\log n)$ -time algorithm. Indeed, there is no algorithm to determine  $\text{opt}A_\gamma(s, t)$  in  $O(n^{2-\delta})$ -time for any  $\delta > 0$ , unless SETH is false [7]. Andoni, Krauthgamer and Onak [2] described a nearly linear time algorithm approximating the edit distance within an approximation factor  $\text{poly}(\log n)$ . Later, Chakraborty et al. [10] presented an  $O(n^{2-2/7})$ -time  $\alpha$ -approximation for this problem, where  $\alpha$  is constant.

Marzal and Vidal [18] defined another criterion for scoring alignments of two sequences called  $vN_\gamma$ -score, which is a normalization of  $vA_\gamma$ -score, as follows:

$$vN_\gamma[A] = \begin{cases} 0, & \text{if } |A| = 0, \\ vA_\gamma[A]/|A|, & \text{otherwise.} \end{cases}$$

The optimal function for this criterion is  $\text{opt}N_\gamma(s, t) = \min_{A \in \mathcal{A}_{s,t}} \{vN_\gamma[A]\}$ , and an *N-optimal alignment*  $A$  of  $s, t$  is such that  $vN_\gamma[A] = \text{opt}N_\gamma(s, t)$ .

A naive dynamic programming algorithm was proposed by Marzal and Vidal [18] to obtain an N-optimal alignment of two sequences in  $O(n^3)$ -time. Using fractional programming, Vidal, Marzal and Aibar [26] presented an algorithm with running time  $O(n^3)$ , requiring  $O(n^2)$ -time in practice, similarly to the classical (unnormalized) edit distance algorithm. Further, Arslan and Egecioglu [6] described an  $O(n^2 \log n)$ -time algorithm to solve this problem.

Let  $A$  be an A-optimal alignment of maximum length of 2-sequence  $S = s, t$ , with  $|s| = n$  and  $|t| = m$ . Considering  $\vec{n} = [n, m]$  and  $\vec{b}$  a bit vector such that  $A(|A|) = \vec{b} \cdot S(\vec{n})$ , the length of a maximum length A-optimal alignment of  $S(1:\vec{n} - \vec{b})$  must be  $|A| - 1$ . Thus, the maximum length  $L(n, m)$  can be found by a dynamic programming formula as following:

$$L(0, 0) = 0, \quad L(0, j) = j, \quad L(i, 0) = i,$$

$$L(i, j) = \max \left\{ \begin{array}{ll} L(i-1, j), & \text{if } d(i, j) = d(i-1, j) + \gamma_{s(i) \rightarrow -} \\ L(i, j-1), & \text{if } d(i, j) = d(i, j-1) + \gamma_{-\rightarrow t(j)} \\ L(i-1, j-1), & \text{if } d(i, j) = d(i-1, j-1) + \gamma_{s(i) \rightarrow s(j)} \end{array} \right\} + 1, \quad i, j > 0,$$

where  $d(i, j) = \text{opt}A_\gamma(s(1:i), t(1:j))$ . Therefore, the maximum length A-optimal alignment of  $s, t$  can be obtained in  $O(nm)$ -time. The following theorem shows that we can propose a simple approximation algorithm to find an A-optimal alignment of maximum length.

► **Theorem 2.1.** *Let  $s, t$  be sequences of lengths  $n, m$ , respectively, and let  $L(n, m)$  be the maximum length of an A-optimal alignment of  $s, t$ . Then,*

$$\text{opt}A_\gamma(s, t)/L(n, m) \leq 2 \text{opt}N_\gamma(s, t),$$

and it can be computed in  $O(n^2)$ -time if  $n = m$ . Moreover, this ratio is tight, i.e., for any positive rational  $\varepsilon$ , there exists a scoring matrix  $\gamma$ , sequences  $s, t$  and an A-optimal alignment of  $s, t$  with maximum length  $A$  such that  $\text{opt}A_\gamma(s, t)/|A| = vA_\gamma[A]/|A| = (2 - \varepsilon) \text{opt}N_\gamma(s, t)$ .

**Proof.** Let  $A$  be an A-optimal alignment with maximum length computed by the heuristic above in  $O(nm)$ -time and space. Let  $B$  be an N-optimal alignment. Thus,  $vA_\gamma[A] \leq vA_\gamma[B]$ . Moreover,  $|B| \leq n + m \leq 2 \max\{n, m\} \leq 2|A|$ , that is,  $|A| \geq |B|/2$ . Therefore,  $vN_\gamma[A] = \frac{vA_\gamma[A]}{|A|} \leq \frac{vA_\gamma[B]}{|A|} \leq \frac{vA_\gamma[B]}{|B|/2} = 2 \text{opt}N_\gamma(s, t)$ .

We present now two sequences and a scoring matrix  $\gamma$  such that the solution given by the heuristic is at least  $2 - \varepsilon$  times the  $vN_\gamma$ -score of an N-optimal alignment, for any  $\varepsilon$  in  $\mathbb{Q}_>$ . Let  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ ,  $\gamma$  be a scoring matrix such that  $\gamma_{\mathbf{a} \rightarrow -} = \gamma_{\mathbf{b} \rightarrow -} = 1/\varepsilon$  and  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = 2/\varepsilon - 1$  and  $\mathbf{a}^n, \mathbf{b}^n \in \Sigma^*$ , with  $n$  in  $\mathbb{N}^*$ . Observe that the  $vA_\gamma$ -score of any alignment of  $(\mathbf{a}^n, \mathbf{b}^n)$ , where  $[\mathbf{a}, \mathbf{b}]$  is aligned in  $k$  columns, is  $2n/\varepsilon - k$ . Thus,  $\text{opt}A_\gamma(\mathbf{a}^n, \mathbf{b}^n) = \min_{0 \leq k \leq n} \{2n/\varepsilon - k\} = 2n/\varepsilon - n = (2/\varepsilon - 1)n$  which implies  $[\mathbf{a}^n, \mathbf{b}^n]$  is the A-optimal alignment with maximum length. Since  $\text{opt}N_\gamma(\mathbf{a}^n, \mathbf{b}^n) \leq vN_\gamma([\mathbf{a}^n, \mathbf{b}^n]) = 1/\varepsilon$ , it follows

$$\frac{\text{opt}A_\gamma(s, t)}{|[\mathbf{a}^n, \mathbf{b}^n]|} = \frac{vA_\gamma(s, t)}{|[\mathbf{a}^n, \mathbf{b}^n]|} = \frac{(2/\varepsilon - 1)n}{n} = (2 - \varepsilon)/\varepsilon \geq (2 - \varepsilon) \text{opt}N_\gamma(\mathbf{a}^n, \mathbf{b}^n). \quad \blacktriangleleft$$

We define now classes of scoring matrices. The usual class of scoring matrices  $\mathbb{M}^C$  has the following properties: for all symbols  $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \Sigma_-$ , we have (a)  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} > 0$  if  $\mathbf{a} \neq \mathbf{b}$ , and  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = 0$  if  $\mathbf{a} = \mathbf{b}$ ; (b)  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = \gamma_{\mathbf{b} \rightarrow \mathbf{a}}$ ; and (c)  $\gamma_{\mathbf{a} \rightarrow \mathbf{c}} \leq \gamma_{\mathbf{a} \rightarrow \mathbf{b}} + \gamma_{\mathbf{b} \rightarrow \mathbf{c}}$ . The class  $\mathbb{M}^A$  of scoring matrices is such that, for all symbols  $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \Sigma$ , we have (a)  $\gamma_{\mathbf{a} \rightarrow -} = \gamma_{-\rightarrow \mathbf{a}} > 0$ ; (b)  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} > 0$  if  $\mathbf{a} \neq \mathbf{b}$ , and  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = 0$  if  $\mathbf{a} = \mathbf{b}$ ; (c) if  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} < \gamma_{\mathbf{a} \rightarrow -} + \gamma_{-\rightarrow \mathbf{b}}$ , then  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = \gamma_{\mathbf{b} \rightarrow \mathbf{a}}$ ; (d)  $\gamma_{\mathbf{a} \rightarrow -} \leq \gamma_{\mathbf{a} \rightarrow \mathbf{b}} + \gamma_{\mathbf{b} \rightarrow -}$ ; and (e)  $\min\{\gamma_{\mathbf{a} \rightarrow \mathbf{c}}, \gamma_{\mathbf{a} \rightarrow -} + \gamma_{-\rightarrow \mathbf{c}}\} \leq \gamma_{\mathbf{a} \rightarrow \mathbf{b}} + \gamma_{\mathbf{b} \rightarrow \mathbf{c}}$ . Moreover, the class  $\mathbb{M}^N$  is such that (a)  $\mathbb{M}^N \subseteq \mathbb{M}^A$  and (b)  $\gamma_{\mathbf{a} \rightarrow -} \leq 2\gamma_{\mathbf{b} \rightarrow -}$  for each  $\mathbf{a}, \mathbf{b} \in \Sigma$ .

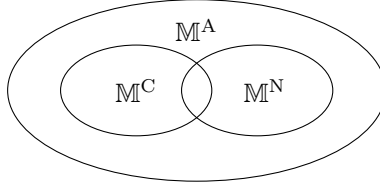
For a set  $S$ , we say that the (distance) function  $f : S \times S \rightarrow \mathbb{R}$  is a *metric* on  $S$  if, for all  $s, t, u \in S$ , the distance  $f$  satisfies: (1)  $f(s, s) = 0$  (*reflexive*); (2)  $f(s, t) > 0$  if  $s \neq t$  (*positive*); (3)  $f(s, t) = f(t, s)$  (*symmetry*); and (4)  $f(s, u) \leq f(s, t) + f(t, u)$  (*triangle inequality*).

If a given criterion  $v$  depends on a scoring matrix  $\gamma$  and it is a metric on  $\Sigma^*$ , we say that the scoring matrix  $\gamma$  *induces* a  $v$ -distance on  $\Sigma^*$ . Sellers [22] showed that matrices in  $\mathbb{M}^C$  induce an  $\text{opt}A_\gamma$ -distance on  $\Sigma^*$  and Araujo and Soares [5] showed that  $\gamma \in \mathbb{M}^A$  if and only if  $\gamma$  induces an  $\text{opt}A_\gamma$ -distance on  $\Sigma^*$ . Furthermore,  $\gamma \in \mathbb{M}^N$  if and only if  $\gamma$  induces an  $\text{opt}N_\gamma$ -distance on  $\Sigma^*$ . Figure 1 shows the relationship between these classes.

## 2.1 $v\text{SP}_\gamma$ -score for $k$ sequences

Consider a scoring matrix  $\gamma$ . Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence and  $A = [s'_1, \dots, s'_k]$  be an alignment of  $S$ . The criterion  $v\text{SP}_\gamma$ , also called *SP-score*, for scoring the alignment  $A$  is

$$v\text{SP}_\gamma[A] = \sum_{h=1}^{k-1} \sum_{i=h+1}^k vA_\gamma[A_{\{h, i\}}]. \quad (1)$$



■ **Figure 1** Relationship between scoring matrices. Araujo and Soares [5] showed that  $M^C \subseteq M^A$ ,  $M^N \subseteq M^A$ ,  $M^C \not\subseteq M^N$  and  $M^N \not\subseteq M^C$ . Moreover, the scoring matrix  $\gamma$  such that  $\gamma_{a \rightarrow a} = 0$  for each  $a$  and  $\gamma_{a \rightarrow b} = 1$  for each  $a \neq b$  is in  $M^C \cap M^N$ , which implies that  $M^C \cap M^N \neq \emptyset$ .

We define  $\text{optSP}_\gamma$  as the optimal function for the criterion  $v\text{SP}_\gamma$ . An alignment  $A$  of  $S$  such that  $v\text{SP}_\gamma[A] = \text{optSP}_\gamma(S)$  is called  *$v\text{SP}_\gamma$ -optimal alignment*. Regardless its decision or optimization version, we call this the *multiple sequence alignment problem (MSA)*. Formally,

► **Problem 2 (Multiple sequence alignment)**. Let  $\gamma$  be a fixed scoring matrix. Given a  $k$ -sequence  $S$ , find a  $v\text{SP}_\gamma$ -optimal alignment of  $S$ .

In order to compute  $\text{optSP}_\gamma$ , we extend the definition of  $v\text{SP}_\gamma$  considering a column of an alignment  $A = [s'_1, \dots, s'_k]$  as its parameter. Thus,  $v\text{SP}_\gamma(A(j)) = \sum_{i < h} \gamma_{s'_i(j) \rightarrow s'_h(j)}$  assuming that  $\gamma_{- \rightarrow -} = 0$  and

$$\text{optSP}_\gamma(S) = \text{optSP}_\gamma(S(1:\vec{n})) = \min_{\vec{b} \in \mathcal{B}^k, \vec{b} \leq \vec{j}} \{ \text{optSP}_\gamma(S(1:\vec{n} - \vec{b})) + v\text{SP}_\gamma[\vec{b} \cdot S(\vec{n})] \}. \quad (2)$$

Recurrence (2) can be computed using a dynamic programming algorithm, obtaining  $D(\vec{j}) = \text{optSP}_\gamma(S(1:\vec{j}))$  for all  $\vec{j} \leq \vec{n}$ . This task can be performed by generating all indexes of  $D$  in lexicographical order, starting with  $D(\vec{0}) = 0$ , as presented in Algorithm 1.

■ **Algorithm 1**  $v\text{SP}_\gamma$ -optimal alignment of  $S$ .

---

**Input:**  $S = s_1, \dots, s_k \in (\Sigma^*)^k$

**Output:**  $\text{optSP}_\gamma(S)$

- 1:  $D(\vec{0}) \leftarrow 0$
  - 2: **for** each  $\vec{j} \leq \vec{n}$  in lexicographical order **do**
  - 3:    $D(\vec{j}) \leftarrow \min_{\vec{b} \in \mathcal{B}^k, \vec{b} \leq \vec{j}} \{ D(\vec{j} - \vec{b}) + v\text{SP}_\gamma[\vec{b} \cdot S(\vec{j})] \}$
  - 4: **return**  $D(\vec{n})$
- 

Suppose that  $|s_i| = n$  for each  $i$ . Notice that the space to store the matrix  $D$  is  $\Theta((n+1)^k)$  and thus Algorithm 1 uses  $\Theta((n+1)^k)$ -space. Besides that, Algorithm 1 checks, in the worst case,  $\Theta(2^k)$  entries for computing all entries in the matrix  $D$  and each computation spends  $\Theta(k^2)$ -time. Therefore, its running time is  $O(2^k k^2 (n+1)^k)$ . Observe that when the distances between sequences are small, not all entries in  $D$  need to be computed, such as in the Carrillo and Lipman's algorithm [9].

## 2.2 $V_\gamma^i$ -score for $k$ sequences

In this section we define a new criteria to normalize the  $v\text{SP}_\gamma$ -score of a multiple alignment. The symbol - aligned to the same symbol - does not contribute to the definition of scoring, and thus this entry is not defined. However, as all the criteria are additive, it is convenient to consider  $\gamma_{- \rightarrow -} = 0$ . The new criteria for aligning sequences takes into account the length of the alignments according to the following:

$$V_\gamma^1[A] = \begin{cases} 0, & \text{if } |A| = 0, \\ v\text{SP}_\gamma[A]/|A|, & \text{otherwise,} \end{cases} \quad (3)$$

$$V_\gamma^2[A] = \sum_{h=1}^{k-1} \sum_{i=h+1}^k vN_\gamma[A_{\{h,i\}}], \quad (4)$$

$$V_\gamma^3[A] = \begin{cases} 0, & \text{if } |A| = 0, \\ vSP_\gamma[A] / (\sum_{h=1}^{k-1} \sum_{i=h+1}^k |A_{\{h,i\}}|), & \text{otherwise.} \end{cases} \quad (5)$$

We define  $\text{optNSP}_\gamma^z$  as the optimal function for the criterion  $V_\gamma^z$ . An alignment  $A$  of  $S$  such that  $V_\gamma^z[A] = \text{optNSP}_\gamma^z(S)$  is called  $V_\gamma^z$ -*optimal alignment*. Moreover, regardless its decision or optimization version, we establish the *criterion  $V_\gamma^z$  for the normalized multiple sequence alignment problem (NMSA- $z$ )*, for  $z = 1, 2, 3$ . Formally,

► **Problem 3** (Normalized multiple sequence alignment with score  $V_\gamma^z$ ). Let  $\gamma$  be a fixed scoring matrix and  $z \in \{1, 2, 3\}$ . Given a  $k$ -sequence  $S$ , find a  $V_\gamma^z$ -optimal alignment of  $S$ .

### 3 Complexity

We study now the complexity of the multiple sequence alignment problem for each new criterion defined in Section 2. We consider the decision version of the computational problems and we prove **NMSA- $z$**  is NP-complete for each  $z$  when the following additional restrictions for the scoring matrix  $\gamma$  hold:  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = \gamma_{\mathbf{b} \rightarrow \mathbf{a}}$  and  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = 0$  if and only if  $\mathbf{a} = \mathbf{b}$  for each pair  $\mathbf{a}, \mathbf{b} \in \Sigma_-$ . Elias [13] shows that, even considering such restrictions, **MSA** is NP-complete. We show a polynomial time reduction from **MSA** to **NMSA- $z$** .

Consider a fixed alphabet  $\Sigma$  and a scoring matrix  $\gamma$  with the restrictions above. Let  $\sigma \notin \Sigma_-$  be a new symbol and  $\Sigma^\sigma = \Sigma \cup \{\sigma\}$ . Let  $G$  be a fixed (constant) positive integer such that each entry in  $\gamma$  is at most  $G$ . We define a scoring matrix  $\gamma^\sigma$  such that  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}}^\sigma = \gamma_{\mathbf{a} \rightarrow \mathbf{b}}, \gamma_{\mathbf{a} \rightarrow \sigma}^\sigma = \gamma_{\sigma \rightarrow \mathbf{a}}^\sigma = G$  and  $\gamma_{\sigma \rightarrow \sigma}^\sigma = 0$ , for each pair  $\mathbf{a}, \mathbf{b} \in \Sigma_-$ .

For an instance  $(S = s_1, \dots, s_k, C)$  of **MSA**, let  $S^L = s_1\sigma^L, \dots, s_k\sigma^L$ , where  $L = Nk^2MG$ ,  $M = \max_i\{|s_i|\}$  and  $N = \binom{k}{2}M$ . Define  $l$  as the *tail length* of an alignment  $A$  if  $A[i, j] = \sigma$  for each  $i = 1, \dots, k$  and  $j = l + 1, l + 2, \dots, |A|$ , i.e., every symbol in the last  $l$  columns of  $A$  is  $\sigma$ . We say that an alignment of  $S^L$  is *canonical* if its tail length is  $L$ . If  $A = [s''_1, \dots, s''_k]$  is an alignment of  $S$ , we denote by  $A^L$  the canonical alignment  $[s''_1\sigma^L, \dots, s''_k\sigma^L]$  of  $S^L$ . The two following results are useful to prove Theorem 3.3, which is the main result of this section. The proofs of these two lemmas can be found in [4].

► **Lemma 3.1.** *There exists a canonical alignment of  $S^L$  which is  $V_{\gamma^\sigma}^z$ -optimal for each  $z$ .*

► **Lemma 3.2.** *If  $C \geq k^2MG$ , then  $\text{MSA}(S, C) = \text{NMSA-}z(S^L, C^z) = \text{Yes}$ , for each  $z$ .*

► **Theorem 3.3.** *NMSA- $z$  is NP-complete for each  $z$ .*

**Proof.** Given a  $k$ -sequence  $S$  over  $\Sigma^*$ , an alignment  $A$  of  $S$  and a integer  $C$ , it is easy to check in polynomial time on the length of  $A$  that  $V_\gamma^z[A] \leq C$ , for  $z \in \{1, 2, 3\}$ , and then **NMSA- $z$**  is in NP.

Consider now  $C^1 := C^2 := C/L, C^3 := C/(\binom{k}{2}L)$  and  $L := Nk^2MG$  and then we prove that  $\text{MSA}(S, C) = \text{Yes}$  if and only if  $\text{NMSA-}z(S^L, C^z) = \text{Yes}$  for each  $z \in \{1, 2, 3\}$ . If  $C \geq k^2MG$ , the Lemma 3.2 holds trivially. Thus, we assume  $C < k^2MG$ . Suppose that  $\text{MSA}(S, C) = \text{Yes}$  and, hence, there exists an alignment  $A$  such that  $vSP_\gamma[A] \leq C$ :

$$V_{\gamma^\sigma}^1[A^L] = \frac{vSP_{\gamma^\sigma}[A^L]}{|A^L|} \leq \frac{vSP_{\gamma^\sigma}[A^L]}{L} = \frac{vSP_\gamma[A]}{L} \leq \frac{C}{L} = C^1,$$

$$V_{\gamma^\sigma}^2[A^L] = \sum_{h=1}^{k-1} \sum_{i=h+1}^k \frac{vA_{\gamma^\sigma}[A^L_{\{h,i\}}]}{|A^L_{\{h,i\}}|} \leq \sum_{h=1}^{k-1} \sum_{i=h+1}^k \frac{vA_{\gamma^\sigma}[A^L_{\{h,i\}}]}{L} = \frac{vSP_{\gamma^\sigma}[A^L]}{L} = \frac{vSP_\gamma[A]}{L} \leq \frac{C}{L} = C^2,$$

$$V_{\gamma\sigma}^3[A^L] = \frac{vSP_{\gamma\sigma}[A^L]}{\sum_{h=1}^{k-1} \sum_{i=k+1}^k |A_{\{h,i\}}^L|} \leq \frac{vSP_{\gamma\sigma}[A^L]}{\sum_{h=1}^{k-1} \sum_{i=k+1}^k L} = \frac{vSP_{\gamma\sigma}[A^L]}{\binom{k}{2}L} = \frac{vSP_{\gamma\sigma}[A]}{\binom{k}{2}L} \leq \frac{C}{\binom{k}{2}L} = C^3,$$

where the first inequality in each equation follows since either  $A^L$  or each alignment induced by  $A^L$  has length at least  $L$  and the second inequality follows since  $vSP_{\gamma\sigma}[A] \leq C$ . Thus, if  $\mathbf{MSA}(S, C) = \text{Yes}$  then  $\mathbf{NMSA}\text{-}z(S^L, C^z) = \text{Yes}$ .

Suppose that  $\mathbf{NMSA}\text{-}z(S^L, C^z) = \text{Yes}$ . It follows from Lemma 3.1 that, for each  $z$ , there exists a canonical alignment  $A^L$  such that  $V_{\gamma\sigma}^z[A^L] \leq C^z$ . Thus, considering  $V_{\gamma\sigma}^1[A^L] \leq C^1$ , we have

$$\begin{aligned} vSP_{\gamma}[A] &= vSP_{\gamma\sigma}[A^L] = (N+L) \frac{vSP_{\gamma\sigma}[A^L]}{N+L} \leq (N+L) \frac{vSP_{\gamma\sigma}[A^L]}{|A^L|} = (N+L) V_{\gamma\sigma}^1[A^L] \\ &\leq (N+L) C^1 = (N+L) \frac{C}{L} = \frac{NC}{L} + C < \frac{Nk^2MG}{L} + C = 1 + C, \end{aligned}$$

where the first equality holds since  $A^L$  is canonical, the first inequality holds since  $|A^L| \leq N+L$  and the second and the third inequalities hold by hypothesis. Considering  $V_{\gamma\sigma}^2[A^L] \leq C^2$ , we have

$$\begin{aligned} vSP_{\gamma}[A] &= vSP_{\gamma\sigma}[A^L] = (N+L) \frac{vSP_{\gamma\sigma}[A^L]}{N+L} = (N+L) \sum_{h=1}^{k-1} \sum_{i=h+1}^k \frac{vA_{\gamma\sigma}[A_{\{h,i\}}^L]}{N+L} \\ &\leq (N+L) \sum_{h=1}^{k-1} \sum_{i=h+1}^k \frac{vA_{\gamma\sigma}[A_{\{h,i\}}^L]}{|A_{\{h,i\}}^L|} = (N+L) V_{\gamma\sigma}^2[A^L] \\ &\leq (N+L) C^2 = (N+L) \frac{C}{L} = \frac{NC}{L} + C < \frac{Nk^2MG}{L} + C = 1 + C, \end{aligned}$$

where the first equality holds since  $A^L$  is canonical, the first inequality holds since, for each  $h, i$ ,  $|A_{\{h,i\}}^L| \leq N+L$ , and the second and the third inequalities hold by hypothesis. And finally, considering  $V_{\gamma\sigma}^3[A^L] \leq C^3$ , we have

$$\begin{aligned} vSP_{\gamma}[A] &= vSP_{\gamma\sigma}[A^L] = (N + \binom{k}{2}L) \frac{vSP_{\gamma\sigma}[A^L]}{N + \binom{k}{2}L} \\ &\leq (N + \binom{k}{2}L) \frac{vSP_{\gamma\sigma}[A^L]}{\sum_{h=1}^{k-1} \sum_{i=h+1}^k |A_{\{h,i\}}^L|} = (N + \binom{k}{2}L) V_{\gamma\sigma}^3[A^L] \leq (N + \binom{k}{2}L) C^3 \\ &= (N + \binom{k}{2}L) \frac{C}{\binom{k}{2}L} = \frac{NC}{\binom{k}{2}L} + C < \frac{Nk^2MG}{\binom{k}{2}L} + C = \frac{1}{\binom{k}{2}} + C \leq 1 + C, \end{aligned}$$

where the first equality holds since  $A^L$  is canonical, the first inequality holds since the sum of lengths of two sequences induced by a canonical alignment is at most  $N + \binom{k}{2}L$  and the second and the third inequalities hold by hypothesis.

Therefore, if  $\mathbf{NMSA}\text{-}z(S^L, C^z) = \text{Yes}$  then  $vSP_{\gamma}[A] < 1 + C$ , for any  $z \in \{1, 2, 3\}$ . Since the entries in the scoring matrix are integers, we have that  $vSP_{\gamma}[A]$  is an integer. And since  $C$  is an integer, it follows that  $vSP_{\gamma}[A] \leq C$ .  $\blacktriangleleft$



## 4 Exact algorithms

### 4.1 NMSA-1

Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence and  $A = [s'_1, \dots, s'_k]$  be an alignment of  $S$ . As defined in Equation (3),  $V_\gamma^1[A]$  takes into account the length of  $A$ , and the optimal function is given by  $\text{opt}V_\gamma^1(S) = \min_{A \in \mathcal{A}_S} \{V_\gamma^1[A]\}$ . The  $V_\gamma^1$ -optimal alignment of  $S$  is an alignment  $A$  such that  $V_\gamma^1[A] = \text{opt}V_\gamma^1(S)$ . Thus, in **NMSA-1** we are given a  $k$ -sequence  $S$  and we want to compute  $\text{opt}V_\gamma^1(S)$  for a fixed matrix  $\gamma$ . We can solve **NMSA-1** by calculating the minimum SP-score considering every possible length of an alignment. That is, we compute the entries of a table  $D$  indexed by  $V_S \times \{0, 1, \dots, N\}$ , where  $N = \sum_{i=1}^k |s_i|$ . The entry  $D(\vec{v}, L)$  stores the score of an alignment of  $S(\vec{v})$  of length  $L$  with lowest SP-score. Notice that  $D(\vec{0}, 0) = 0$ ,  $D(\vec{v} \neq \vec{0}, 0) = D(\vec{0}, L \neq 0) = \infty$ . Therefore, the table entries can be calculated as:

$$D(\vec{v}, L) = \begin{cases} 0, & \text{if } \vec{v} = \vec{0}, L = 0, \\ \infty, & \text{if } \vec{v} = \vec{0}, L \neq 0 \text{ or } \vec{v} \neq \vec{0}, L = 0, \\ \min_{\vec{b} \in \mathcal{B}_k, \vec{b} \leq \vec{v}} \{D(\vec{v} - \vec{b}, L - 1) + v\text{SP}_\gamma[\vec{b} \cdot S(\vec{v})]\}, & \text{otherwise.} \end{cases}$$

Table  $D$  is computed for all possible values of  $L = 0, \dots, N$ . Consequently,  $\text{opt}V_\gamma^1(S) = \min_L \{D(\vec{n}, L)/L\}$  is returned. Algorithm 2 describes this procedure more precisely.

■ **Algorithm 2**  $V_\gamma^1$ -optimal alignment of  $S$ .

---

**Input:**  $k$ -sequence  $S = s_1, \dots, s_k$  such that  $n_i = |s_i|$

**Output:**  $\text{opt}V_\gamma^1(S)$

- 1:  $D(\vec{0}, 0) \leftarrow 0$
  - 2: **for** each  $L \neq 0$  **do**  $D(\vec{0}, L) \leftarrow \infty$
  - 3: **for** each  $\vec{v} \neq \vec{0}$  **do**  $D(\vec{v}, 0) \leftarrow \infty$
  - 4: **for** each  $\vec{0} < \vec{v} \leq \vec{n}$  in lexicographical order **do**
  - 5:   **for** each  $L \leftarrow 1, 2, \dots, N$  **do**
  - 6:      $D(\vec{v}, L) \leftarrow \min_{\vec{b} \in \mathcal{B}_k, \vec{b} \leq \vec{v}} \{D(\vec{v} - \vec{b}, L - 1) + v\text{SP}_\gamma[\vec{b} \cdot S(\vec{v})]\}$
  - 7: **return**  $\min_L \{D(\vec{n}, L)/L\}$
- 

Suppose that  $n_i = |s_i| = n$  for each  $i$ . Notice that the space to store the matrix  $D$  is  $\Theta(N(n+1)^k)$ . The time consumption of Algorithm 2 corresponds to the time needed to fill the table  $D$  up, plus the running time of line 7. Each entry of  $D$  can be computed in  $O(2^k k^2)$ -time. Therefore, the algorithm spends  $O(2^k k^2 \cdot N(n+1)^k)$ -time to compute the entire table  $D$ , since  $D$  has  $\Theta(N(n+1)^k)$  entries. Line 7 is computed in  $\Theta(N)$ -time. Therefore, the running time of Algorithm 2 is  $O(2^k k^2 \cdot N(n+1)^k) + \Theta(N) = O(2^k k^2 \cdot N(n+1)^k)$ . If  $N = kn$ , it follows that the total running time is  $O(2^k k^3 (n+1)^{k+1})$ .

### 4.2 NMSA-2

Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence and  $A = [s'_1, \dots, s'_k]$  be an alignment of  $S$ . As defined in Equation (4),  $V_\gamma^2[A]$  takes into account the length of the induced alignment  $A$ , and the optimal function is given by  $\text{opt}V_\gamma^2(S) = \min_{A \in \mathcal{A}_S} \{V_\gamma^2[A]\}$ . The  $V_\gamma^2$ -optimal alignment of  $S$  is an alignment  $A$  such that  $V_\gamma^2[A] = \text{opt}V_\gamma^2(S)$ . Then, in **NMSA-2** we are given a  $k$ -sequence  $S$  and we want to compute  $\text{opt}V_\gamma^2(S)$  for a fixed matrix  $\gamma$ .

Let  $\vec{L} = [L_{12}, L_{13}, \dots, L_{1k}, L_{23}, \dots, L_{2k}, \dots, L_{(k-1)k}]$  be a  $\binom{k}{2}$ -vector indexed by sets of two integers  $\{h, i\}$  such that  $1 \leq h < i \leq k$  and  $L_{hi}$  denotes the element of  $\vec{L}$  of index  $\{h, i\}$ . The lengths of the induced alignments by an alignment can be represented by a vector

$\vec{L}$ . Thus, if  $A$  is an alignment and  $|A_{\{h,i\}}| = L_{hi}$  for each pair  $h, i$ , we say that  $\vec{L}$  is the *induced length* of  $A$ . For a  $k$ -sequence  $S = s_1, \dots, s_k$ , where  $n_i = |s_i|$  for each  $i$ , we define  $\mathbb{L} = \{\vec{L} = [L_{12}, L_{13}, \dots, L_{1k}, L_{23}, \dots, L_{2k}, \dots, L_{(k-1)k}] : 0 \leq L_{hi} \leq n_h + n_i\}$ . Note that if  $n$  is the length of each sequence in  $S$ , then  $|\mathbb{L}| = (2n+1)^{\binom{k}{2}}$ . Let  $\vec{b} = [b_1, \dots, b_k]$  be a  $k$ -bit vector. Overloading the minus operator “ $-$ ”, we define  $\vec{L} - \vec{b}$  to be a  $\binom{k}{2}$ -vector  $\vec{L}'$  which is obtained from  $\vec{L}$  and from  $\vec{b}$  such that, for each pair  $h, i$ , we have  $L'_{hi} = L_{hi}$  if  $b_h = b_i = 0$ , and  $L'_{hi} = L_{hi} - 1$ , otherwise. Observe that if  $\vec{L}$  is the induced length of an alignment  $A$  of  $S(\vec{v})$  and  $\vec{b}$  is a  $k$ -bit vector such that  $\vec{b} \cdot S(\vec{v})$  is the last column of  $A$ , then  $\vec{L}' = \vec{L} - \vec{b}$  is the induced length of the alignment  $A(1:|A| - 1)$ .

Let  $\vec{\gamma}$  be a vector of  $\binom{k}{2}$  scoring matrices indexed by two integers  $\{h, i\}$ , with  $1 \leq h < i \leq k$ . We denote by  $\gamma^{(hi)}$  the element of  $\vec{\gamma}$  with index  $\{h, i\}$ . Then, we have  $\vec{\gamma} = [\gamma^{(12)}, \gamma^{(13)}, \dots, \gamma^{(1k)}, \gamma^{(23)}, \dots, \gamma^{(2k)}, \dots, \gamma^{((k-1)k)}]$ , and define the  $\vec{\gamma}$ -SP-score of  $A$  as  $vSP_{\vec{\gamma}}[A] = \sum_{h=1}^{k-1} \sum_{i=h+1}^k vA_{\gamma^{(hi)}}[A_{\{h,i\}}]$ . If we define the  $\vec{\gamma}$ -SP-score of a vector  $\vec{\sigma} = [\sigma_1, \dots, \sigma_k]$  in  $\Sigma$  as  $vSP_{\vec{\gamma}}[\vec{\sigma}] = \sum_{h=1}^{k-1} \sum_{i=h+1}^k \gamma_{\sigma_h \rightarrow \sigma_i}^{(hi)}$ , then we can alternatively calculate the  $\vec{\gamma}$ -SP-score of the alignment  $A$  as  $vSP_{\vec{\gamma}}[A] = \sum_j vSP_{\vec{\gamma}}[A(j)]$ .

### 4.2.1 Computing $\text{optV}_{\vec{\gamma}}^2$

In this section we describe an algorithm in two steps for computing  $\text{optV}_{\vec{\gamma}}^2$  for a given  $k$ -sequence  $S$ : in Step 1 we consider the particular case where we have three sequences, and in Step 2 we treat the general case.

#### Step 1: $k = 3$

Let  $S = s_1, s_2, s_3$  be a 3-sequence. Suppose that we have induced lengths  $\vec{\mathcal{L}} = [\mathcal{L}_{12}, \mathcal{L}_{13}, \mathcal{L}_{23}]$  of a  $V_{\vec{\gamma}}^2$ -optimal alignment  $A$  of  $S$ . In consequence, we have that  $\mathcal{L}_{hi} = |A_{\{h,i\}}|$  for each pair  $h, i$ . Notice that knowing the lengths  $\mathcal{L}_{12}, \mathcal{L}_{13}$  and  $\mathcal{L}_{23}$  does not imply knowing the  $V_{\vec{\gamma}}^2$ -optimal alignment  $A$ . In general, we cannot even infer what  $|A|$  is. For example, the alignments

$$\begin{bmatrix} s_1(1) & s_1(2) & - \\ s_2(1) & - & s_2(2) \\ - & s_3(1) & s_3(2) \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} s_1(1) & s_1(2) & - & - \\ s_2(1) & - & s_2(2) & - \\ s_3(1) & - & - & s_3(2) \end{bmatrix}$$

have different lengths but same induced lengths for  $s_1, s_2, s_3$ , where  $|s_1| = |s_2| = |s_3| = 2$  and  $\mathcal{L}_{12} = \mathcal{L}_{13} = \mathcal{L}_{23} = 3$ . However, if we know  $\vec{\mathcal{L}} = [\mathcal{L}_{12}, \mathcal{L}_{13}, \mathcal{L}_{23}]$ , we have

$$\begin{aligned} \text{optV}_{\vec{\gamma}}^2(S) &= \min_{A \in \mathcal{A}_S : \mathcal{L}_{hi} = |A_{\{h,i\}}|, \forall h,i} \left\{ \sum_{h=1}^{k-1} \sum_{i=h+1}^k vA_{\gamma^{(hi)}}[A_{\{h,i\}}] / \mathcal{L}_{hi} \right\} \\ &= \min_{A \in \mathcal{A}_S} \left\{ \sum_{h=1}^{k-1} \sum_{i=h+1}^k vA_{\gamma^{(hi)}}[A_{\{h,i\}}] \right\}, \end{aligned}$$

where  $\gamma^{(hi)}$  is a scoring matrix obtained by multiplying the elements of  $\gamma$  by  $1/\mathcal{L}_{hi}$ . Since we guarantee it is the induced length of an alignment  $V_{\vec{\gamma}}^2$ -optimal, we fix  $\vec{\mathcal{L}}$  and compute  $\vec{\gamma}$  in order to calculate the entries of a table  $D_{\vec{\mathcal{L}}}$ , such that

$$D_{\vec{\mathcal{L}}}(\vec{v} = [v_1, v_2, v_3], \vec{L} = [L_{12}, L_{13}, L_{23}]) = \min_{A \in \mathcal{A}_{S(\vec{v})}, \mathcal{L}_{hi} = |A_{\{h,i\}}|, \forall h,i} \left\{ \sum_{h < i} vA_{\gamma^{(hi)}}[A_{\{h,i\}}] \right\}$$

corresponds to the score of an alignment with the lowest  $\vec{\gamma}$ -SP-score when the induced length is  $\vec{L}$ . The table  $D_{\vec{\mathcal{L}}}$  can then be computed using the following recurrence

$$D_{\vec{\mathcal{L}}}(\vec{v}, \vec{L}) = \begin{cases} 0, & \text{if } \vec{v} = \vec{0}, \vec{L} = \vec{0}, \\ \infty, & \text{if } \vec{v} = \vec{0}, \vec{L} \neq \vec{0} \text{ or } \vec{v} \neq \vec{0}, \vec{L} = \vec{0}, \\ \min_{\vec{b} \in \mathcal{B}_k, \vec{b} \leq \vec{v}, \vec{b} \leq \vec{L}} \{ D_{\vec{\mathcal{L}}}(\vec{v} - \vec{b}, \vec{L} - \vec{b}) + vSP_{\vec{\gamma}}[\vec{b} \cdot S(\vec{v})] \}, & \text{otherwise,} \end{cases}$$

where  $\vec{b} \leq \vec{L}$  is also an overloading, meaning that  $\vec{L} - \vec{b} \geq \vec{0}$ .

In this case, if  $\vec{\mathcal{L}}$  is the induced length of a  $V_\gamma^2$ -optimal alignment of  $S$ , then  $\text{opt}V_\gamma^2(S) = D_{\vec{\mathcal{L}}}(\vec{n}, \vec{\mathcal{L}})$ . If each sequence has length  $n$ , then the total space to store the table  $D_{\vec{\mathcal{L}}}$  is  $(2n+1)^{\binom{3}{2}} \cdot (n+1)^3 = \Theta(n^6)$ . When  $\vec{\mathcal{L}}$  is unknown, the computation must be repeated for each  $\vec{\mathcal{L}} \in \mathbb{L}$ , but the space can be reused and no additional space required. If  $\vec{\mathcal{L}}$  is known, the algorithm runs to compute all entries of  $D_{\vec{\mathcal{L}}}$ . As  $D_{\vec{\mathcal{L}}}$  has  $\Theta(n^6)$  entries and each entry takes  $O(1)$ -time to be computed, the total time spent is  $O(n^6)$ . If  $\vec{\mathcal{L}}$  is unknown, the time needed to compute  $\vec{\mathcal{L}}$  must be multiplied by the total of elements in  $\mathbb{L}$  which is  $(2n+1)^3$ . Therefore, in the latter case, the total time is  $O(n^6 \cdot (2n+1)^3) = O(n^9)$ .

## Step 2: $k > 3$

Algorithm 3 is a natural extension of the algorithm described in Step 1. Given a scoring matrix  $\gamma$  and an induced length  $\vec{\mathcal{L}}$ , let  $\gamma \times \vec{\mathcal{L}} = [\gamma^{(12)}, \dots, \gamma^{(k(k-1))}]$  be the vector of  $\binom{k}{2}$  scoring matrices, where  $\gamma^{(hi)}$  is obtained dividing each entry of  $\gamma$  by  $\mathcal{L}_{hi}$  for each  $h < i$ .

### Algorithm 3 $V_\gamma^2$ -optimal alignment of $S$ .

**Input:** A  $k$ -sequence  $S = s_1, \dots, s_k$  such that  $n_i = |s_i|$

**Output:**  $\text{opt}V_\gamma^2(S)$

---

```

1: for each  $\vec{\mathcal{L}} \in \vec{\mathbb{L}}$  do
2:    $D_{\vec{\mathcal{L}}}(\vec{0}, \vec{0}) \leftarrow 0$ 
3:   for each  $\vec{L} \neq \vec{0}$  do  $D_{\vec{\mathcal{L}}}(\vec{0}, \vec{L}) \leftarrow \infty$ 
4:   for each  $\vec{v} \neq \vec{0}$  do  $D_{\vec{\mathcal{L}}}(\vec{v}, \vec{0}) \leftarrow \infty$ 
5:    $\vec{\gamma} \leftarrow \gamma \times \vec{\mathcal{L}}$ 
6:   for each  $\vec{0} < \vec{v} \leq \vec{n}$  in lexicographical order do
7:     for each  $\vec{L} \neq \vec{0}$  in lexicographical order do
8:        $D_{\vec{\mathcal{L}}}(\vec{v}, \vec{L}) = \min_{\vec{b} \in \mathcal{B}^k, \vec{b} \leq \vec{v}, \vec{b} \leq \vec{L}} \{D_{\vec{\mathcal{L}}}(\vec{v} - \vec{b}, \vec{L} - \vec{b}) + v\text{SP}_{\vec{\gamma}}[\vec{b} \cdot S(\vec{v})]\}$ 
9: return  $\min_{\vec{\mathcal{L}} \in \vec{\mathbb{L}}} \{D_{\vec{\mathcal{L}}}(\vec{n}, \vec{\mathcal{L}})\}$ 

```

---

For  $k$  sequences of length  $n$ , Algorithm 3 needs  $(2n+1)^{\binom{k}{2}} \cdot (n+1)^k$  space to store the table  $D_{\vec{\mathcal{L}}}$ . For each of the  $(2n+1)^{\binom{k}{2}}$  values  $\vec{\mathcal{L}} \in \vec{\mathbb{L}}$ , table  $D_{\vec{\mathcal{L}}}$  is recalculated. Since the computation of each entry takes  $O(2^k k^2)$ -time, the total time is

$$O(2^k k^2 \cdot (2n+1)^{\binom{k}{2}} \cdot (2n+1)^{\binom{k}{2}} (n+1)^k) = O((1 + 1/(2n+1))^k (2n+1)^{k^2} k^2).$$

If  $k \leq 2n+1$ , the total time can be written as  $O((2n+1)^{k^2} k^2)$ , since  $(1 + 1/k)^k \leq e = 2.718281828\dots$ . Notice that  $(1 + 1/(2n+1))^k \leq (1 + 1/k)^k \leq e$  is also constant.

## 4.3 NMSA-3

Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence and  $A = [s'_1, \dots, s'_k]$  be an alignment of  $S$ . As defined in Equation (5),  $V_\gamma^3[A]$  takes into account the length of  $A$ , and the optimal function is given by  $\text{opt}V_\gamma^3(S) = \min_{A \in \mathcal{A}_S} \{V_\gamma^3[A]\}$ . The  $V_\gamma^3$ -optimal alignment of  $S$  is an alignment  $A$  such that  $V_\gamma^3[A] = \text{opt}V_\gamma^3(S)$ . Then NMSA-3 is defined as follows: for a fixed matrix  $\gamma$ , given a  $k$ -tuple  $S$ , determine  $\text{opt}V_\gamma^3(S)$ .

### 4.3.1 Computing $\text{opt}V_\gamma^3$

Here, each entry  $D(\vec{v}, L)$  of  $D$  stores the SP-score of an alignment  $A$  of the prefix  $S(\vec{v})$  with the lowest SP-score, such that  $\sum_{i < h} |A_{\{i, h\}}| = L$ . If  $\vec{b}$  is a  $k$ -vector, define  $\|\vec{b}\| = \sum_{h < i} b_h b_i$ . Notice that if  $\vec{v} - \vec{b}$  is the last column of an alignment  $A$  and  $L = \sum_{h=1}^{k-1} \sum_{i=h+1}^k |A_{\{h, i\}}|$

## 40:12 Algorithms for Normalized Multiple Sequence Alignments

is the sum of lengths of the alignments induced by  $A$ . Thus, the sum of lengths of the alignments induced by  $A(1:|A| - 1)$  is  $L - \|\vec{b}\|$ . Therefore,

$$D(\vec{v}, L) = \begin{cases} 0, & \text{if } \vec{v} = \vec{0}, L = 0, \\ \infty, & \text{if } \vec{v} = \vec{0}, L \neq 0 \text{ or } \vec{v} \neq \vec{0}, L = 0, \\ \{D(\vec{v} - \vec{b}, L - \|\vec{b}\|) + v\text{SP}_\gamma[\vec{b} \cdot S(\vec{v})]\}, & \text{otherwise.} \end{cases}$$

Algorithm 4 provides more details about the procedure for computing  $\text{optV}_\gamma^3$ .

■ **Algorithm 4**  $V_\gamma^3$ -optimal alignment of  $S$ .

**Input:** a  $k$ -sequence  $S = s_1, \dots, s_k$  such that  $n_i = |s_i|$

**Output:**  $\text{optV}_\gamma^3(S)$

---

```

1:  $D(\vec{0}, 0) \leftarrow 0$ 
2: for each  $L \neq 0$  do  $D(\vec{0}, L) \leftarrow \infty$ 
3: for each  $\vec{v} \neq \vec{0}$  do  $D(\vec{v}, 0) \leftarrow \infty$ 
4: for each  $\vec{0} < \vec{v} \leq \vec{n}$  in lexicographical order do
5:   for  $L \leftarrow 1, 2, \dots, N(k-1)$  do
6:      $D(\vec{v}, L) \leftarrow \min_{\vec{b} \in \mathcal{B}^k, \vec{b} \leq \vec{v}, \|\vec{b}\| \leq L} \{D(\vec{v} - \vec{b}, L - \|\vec{b}\|) + v\text{SP}_\gamma[\vec{b} \cdot S(\vec{v})]\}$ 
7: return  $\min_L \{D(\vec{n}, L)/L\}$ 

```

---

Assume that all sequences in  $S$  have length  $n$ . The table  $D$  is computed for all possible values of  $L = 1, \dots, \binom{k}{2}(2n) (= nk^2 - nk)$  and, after this, we determine  $\text{optSP}_\gamma(S) = \min_L \{D(\vec{n}, L)/L\}$ . Thus, table  $D$  needs space equivalent to  $(nk^2 - nk + 1) \cdot (n + 1)^k = \Theta(k^2(n + 1)^{k+1})$ . Since the time required to determine each entry of  $D$  is  $O(2^k k^2)$ , the running time of Algorithm 4 is  $O(2^k k^4 (n + 1)^{k+1})$ .

## 5 Approximation algorithms for MSA and NMSA-2

Gusfield [15] described a 2-approximation algorithm for **MSA**. It assumes that  $\gamma \in \mathbb{M}^{\mathbb{C}}$ . In this section, we adapt Gusfield's algorithm, proposing a 6-approximation algorithm for **MSA** when  $\gamma \in \mathbb{M}^{\mathbb{A}}$  and a 12-approximation algorithm for **NMSA-2** problem when  $\gamma \in \mathbb{M}^{\mathbb{N}}$ .

We consider here a generic function  $v$  to score an alignment of a 2-sequence such that  $\text{opt}(s, s) = 0$  (identity) and  $\text{opt}(s, t) = \text{opt}(t, s)$  (symmetry), where  $\text{opt}(s, t)$  is the score of a  $v$ -optimal alignment of a 2-sequence  $s, t$ . Notice that  $vA_\gamma$  and  $vN_\gamma$  have these properties when  $\gamma \in \mathbb{M}^{\mathbb{A}}$  and  $\gamma \in \mathbb{M}^{\mathbb{N}}$ , respectively. Let  $S$  be a  $k$ -sequence and  $A$  in  $\mathcal{A}_S$  be an alignment. We define  $V$  and  $\text{OPT}$  as functions such that  $V[A] = \sum_{h=1}^{k-1} \sum_{i=h+1}^k v[A_{\{h,i\}}]$  and  $\text{OPT}(S) = \min_{A \in \mathcal{A}_S} V(A)$ . Thus, a  $V$ -optimal alignment is an alignment  $A$  such that  $V[A] = \text{OPT}(S)$ .

Let  $c$  be an integer with  $1 \leq c \leq k$ . A *star*  $X$  with center  $c$  of  $S = s_1, \dots, s_k$ , also called a  $c$ -star, is a collection of  $k - 1$  alignments: alignment  $X_h = [s'_h, s_c^h]$  of  $s_h, s_c$ , for each  $h < c$ , where  $v[s'_h, s_c^h] = v[s_c^h, s'_h]$ , and alignment  $X_h = [s_c^h, s'_h]$  of  $s_c, s_h$ , for each  $h > c$ , where  $v[s_c^h, s'_h] = v[s'_h, s_c^h]$ . The set of all  $c$ -stars is denoted by  $\mathcal{X}_c$ . The score of the  $c$ -star  $X$  is  $\text{cStar}(X) = \sum_{h \neq c} v[X_h]$  and a  $v$ -optimal star is one whose score is  $\text{optStar}(S) = \min_{X \in \mathcal{X}_c, c \in \mathbb{N}} \{\text{cStar}(X)\}$ . Notice that  $\text{optStar}(S) = \min_c \{\sum_{h \neq c} \text{opt}(s_h, s_c)\}$ , and if  $v = vA_\gamma$  and  $\gamma \in \mathbb{M}^{\mathbb{A}}$ ,  $\text{optStar}(S)$  can be computed in  $O(k^2 n^2)$ -time, and if  $v = vN_\gamma$  and  $\gamma \in \mathbb{M}^{\mathbb{N}}$ ,  $\text{optStar}(S)$  can be computed in  $O(k^2 n^3)$ -time when  $|s_i| \leq n$ , for each  $s_i$  in  $S$ .

We say that an alignment  $A$  of a  $k$ -sequence  $S$  and a  $c$ -star  $X$  of  $S$  are *compatible* ( $A$  is compatible with  $X$  or  $X$  is compatible with  $A$ ) in  $S$  when either  $A_{\{h,c\}}$  or  $A_{\{c,h\}}$  is equal to  $X_h$ , for each  $h$ . It is easy to obtain, from an alignment  $A$  and  $c$  in  $\mathbb{N}$ , the unique  $c$ -star  $X$

that is compatible with  $A$ . On the other hand, it is known from Feng and Doolittle [14] that one can find an alignment  $A$  compatible with a given  $c$ -star  $X$  in  $O(kn)$ -time, where  $|s_i| \leq n$  for each sequence  $s_i$  in the  $k$ -sequence  $S$ . In this case, there exists one or more compatible alignments with  $X$ .

The next result is a straightforward consequence of a result from Gusfield [15], where  $v$  is the scoring function. See the proof in [4].

► **Lemma 5.1.** *Given a  $k$ -sequence  $S$ ,  $\text{optStar}(S) \leq (2/k) \text{OPT}(S)$ .*

From now on, we consider  $v = vA_\gamma$ , or  $v = vN_\gamma$  and  $\gamma = \mathbb{M}^A$  or  $\gamma = \mathbb{M}^N$ , respectively. Let  $s, t \in \Sigma^*$  be sequences. Suppose that  $A = [s', t']$  is an alignment of  $s, t$ . We say that a column  $j$  is *splittable in  $A$*  if  $s'(j) \neq -, t'(j) \neq -$  and  $\min\{\gamma_{t'(j) \rightarrow -}, \gamma_{s'(j) \rightarrow -}\} \leq \gamma_{s'(j) \rightarrow t'(j)}$ . Let  $J := \{j_i \in \mathbb{N} : 1 \leq j_1 < \dots < j_m \leq |A| \text{ and } j_i \text{ is splittable in } A\}$ . An  $A$ -*splitting* is the alignment

$$\begin{bmatrix} s'(1:j_1-1) & s'(j_1) & - & s'(j_1+1:j_2-1) & s'(j_2) & - & \dots & s'(j_m+1:|A|) \\ t'(1:j_1-1) & - & t'(j_1) & t'(j_1+1:j_2-1) & - & t'(j_2) & \dots & t'(j_m+1:|A|) \end{bmatrix}.$$

We say that  $J$  is *required to split  $A$* . The following proposition is used to check properties of an  $A$ -splitting. See its proof in [4].

► **Proposition 5.2.** *Consider  $\gamma \in \mathbb{M}^A$  and  $\mathbf{a}, \mathbf{b} \in \Sigma$ . If  $\gamma_{\mathbf{a} \rightarrow -} > \gamma_{\mathbf{a} \rightarrow \mathbf{b}}$  or  $\gamma_{\mathbf{a} \rightarrow -} > \gamma_{\mathbf{b} \rightarrow \mathbf{a}}$ , then  $\gamma_{\mathbf{a} \rightarrow \mathbf{b}} = \gamma_{\mathbf{b} \rightarrow \mathbf{a}}$ .*

Let  $X = \{X_1, \dots, X_{c-1}, X_{c+1}, X_k\}$  be a  $c$ -star. The  $X$ -*starsplitting* is the  $c$ -star  $Y = \{Y_1, \dots, Y_{c-1}, Y_{c+1}, Y_k\}$ , where  $Y_j$  is the  $X_j$ -splitting for each  $j$ . The next result shows that the  $v$ -score of the star  $Y$  is bounded by the  $v$ -score of the star  $X$  when  $\gamma \in \mathbb{M}^A$  and  $v = vA_\gamma$ , or  $\gamma \in \mathbb{M}^N$  and  $v = vN_\gamma$ . Thus, as a consequence of Proposition 5.2, we have the following lemma. The formal proof can be seen in [4].

► **Lemma 5.3.** *Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence,  $X$  be a  $c$ -star of  $S$ ,  $Y$  be the  $X$ -starsplitting and  $v$  be a function to score alignments. Consider  $\gamma \in \mathbb{M}^A$  and  $v = vA_\gamma$ , or  $\gamma \in \mathbb{M}^A$  and  $v = vN_\gamma$ . Then,  $Y$  is also a  $c$ -star and  $c\text{Star}(Y) \leq 3 c\text{Star}(X)$ .*

**Proof sketch.** As a consequence of  $\gamma \in \mathbb{M}^A$  and Proposition 5.2, we have that  $Y$  is also a  $c$ -star. Let  $h \in \{1, \dots, k\}$  and  $J$  be a set required to split  $X$ . We prove that  $vA_\gamma[Y_h] \leq 3vA_\gamma[X_h]$  and  $vN_\gamma[Y_h] \leq 3vN_\gamma[X_h]$  hold since  $\gamma \in \mathbb{M}^A$  when  $v = vA_\gamma$  and  $\gamma \in \mathbb{M}^N$  when  $v = vN_\gamma$ , and  $J \subseteq \{1, 2, \dots, |A|\}$ . Therefore, in these cases, we have

$$c\text{Star}(Y) = \sum_{h \neq c} v[Y_h] = \sum_{h < c} v[Y_h] + \sum_{h > c} v[Y_h] \leq 3 \sum_{h \neq c} v[X_h] = 3c\text{Star}(X). \quad \blacktriangleleft$$

Notice that the time consumption for computing an  $X$ -splitting from  $X$  is  $O(kn)$  when  $|s_i| \leq n$ , for each  $s_i \in S$ . Considering a star  $X$  of  $S = s_1, \dots, s_k$ , there can exist many compatible alignments with a  $v$ -star  $Y$  which is a  $X$ -splitting. Let `COMPATIBLEALIGN` be a subroutine that receives the  $c$ -star  $Y$  and returns an alignment  $A$  compatible with  $Y$ . It is quite simple: if symbols  $s_h(j_1)$  and  $s_c(j_2)$  are aligned in  $X_h$ , they are also aligned in  $A$ ; otherwise,  $s_h(j)$  aligns only with  $-$  in  $A$ . This property is enough to guarantee the approximation factor of `MSA` and `NMSA-2`.

Let  $Q_{\max} := \max_{\mathbf{a} \in \Sigma} \{\gamma_{\mathbf{a} \rightarrow -}, \gamma_{- \rightarrow \mathbf{a}}\}$  and consider the following result, whose proof can be found in [4].

► **Proposition 5.4.** *Let  $S$  be a  $k$ -sequence,  $X$  be a  $c$ -star of  $S$  and  $Y$  be a  $X$ -starsplitting. Assume that  $\gamma \in \mathbb{M}^A$  and that  $\text{COMPATIBLEALIGN}(Y)$  returns  $A = [s'_1, \dots, s'_k]$ . If  $h \neq c$  and  $i \neq c$ , we have that*

- (i)  $\gamma_{s'_h(j) \rightarrow s'_i(j)} \leq \gamma_{s'_h(j) \rightarrow s'_c(j)} + \gamma_{s'_c(j) \rightarrow s'_i(j)}$  for each  $j = 1, \dots, |A|$ , and
- (ii)  $vN_\gamma[A_{\{h,i\}}] \leq 2Q_{\max}$  when  $\gamma \in \mathbb{M}^N$ .

Proposition 5.4 is an auxiliary result to show the following lemma.

► **Lemma 5.5.** *Let  $S$  be a  $k$ -sequence,  $X$  be a  $c$ -star of  $S$ ,  $Y$  be a  $X$ -starsplitting and  $\text{COMPATIBLEALIGN}(Y) = A$ . Then, for each  $h < i$ ,  $h \neq c$  and  $i \neq c$ ,*

- (i)  $vA_\gamma[A_{\{h,i\}}] \leq vA_\gamma[A_{\{h,c\}}] + vA_\gamma[A_{\{c,i\}}]$  when  $\gamma \in \mathbb{M}^A$ , and
- (ii)  $vN_\gamma[A_{\{h,i\}}] \leq 2(vN_\gamma[A_{\{h,c\}}] + vN_\gamma[A_{\{c,i\}}])$  when  $\gamma \in \mathbb{M}^N$ .

**Proof.** Let  $A = [s'_1, \dots, s'_k]$  and  $Z = \{j : s'_c(j) \neq - \text{ and } s'_h(j) = s'_i(j) = -\}$ . We have  $vA_\gamma[A_{\{h,i\}}] \leq vA_\gamma[A_{\{h,c\}}] + vA_\gamma[A_{\{c,i\}}] - \sum_{j \in Z} (\gamma_{- \rightarrow s'_c(j)} + \gamma_{s'_c(j) \rightarrow -})$  from a consequence of Proposition 5.4. Besides, since  $\gamma \in \mathbb{M}^A$ , we have that  $\gamma_{- \rightarrow s'_c(j)}, \gamma_{s'_c(j) \rightarrow -} > 0$ . It implies that (i) is proven.

For proving (ii), observe first that, by definition of  $\mathbb{M}^N$ , we have that  $Q_{\max} \leq \gamma_{- \rightarrow s'_c(j)} + \gamma_{s'_c(j) \rightarrow -}$  for every  $j$ . Furthermore, following these statements, we have that

$$\begin{aligned} vN_\gamma[A_{\{h,i\}}] &= \frac{vA_\gamma[A_{\{h,i\}}]}{|A_{\{h,i\}}|} \\ &\leq \frac{vA_\gamma[A_{\{h,i\}}] + 2 \cdot Q_{\max} |Z|}{|A_{\{h,i\}}| + |Z|} \\ &\leq 2 \cdot \frac{vA_\gamma[A_{\{h,i\}}] + Q_{\max} |Z|}{|A_{\{h,i\}}| + |Z|} \end{aligned} \quad (6)$$

$$\leq 2 \cdot \frac{vA_\gamma[A_{\{h,c\}}] + vA_\gamma[A_{\{c,i\}}] - \sum_{j \in Z} (\gamma_{- \rightarrow s'_c(j)} + \gamma_{s'_c(j) \rightarrow -}) + Q_{\max} |Z|}{|A_{\{h,i,c\}}| - |Z| + |Z|} \quad (7)$$

$$\begin{aligned} &\leq 2 \cdot \frac{vA_\gamma[A_{\{h,c\}}] + vA_\gamma[A_{\{c,i\}}] - Q_{\max} |Z| + Q_{\max} |Z|}{|A_{\{h,i,c\}}| - |Z| + |Z|} \\ &= 2 \cdot \left( \frac{vA_\gamma[A_{\{h,c\}}]}{|A_{\{h,i,c\}}|} + \frac{vA_\gamma[A_{\{c,i\}}]}{|A_{\{h,i,c\}}|} \right) \end{aligned} \quad (8)$$

$$\leq 2 \cdot (vN_\gamma[A_{\{h,c\}}] + vN_\gamma[A_{\{c,i\}}]), \quad (9)$$

where the first inequality of (6) is a consequence of Proposition 5.4 and the second inequality follows since every entry of  $\gamma$  is nonnegative, (7) follows from the result in the first paragraph and from  $|A_{\{h,i\}}| = |A_{\{h,i,c\}}| - |Z|$ , (8–9) follow as a consequence of the definition of  $Q_{\max}$ , and as a consequence of  $|A_{\{h,c\}}| \leq |A_{\{h,i,c\}}|$  and  $|A_{\{c,i\}}| \leq |A_{\{h,i,c\}}|$ . ◀

We can now describe the approximation algorithm (Algorithm 5).

■ **Algorithm 5** Approximation algorithm for MSA and NMSA-2.

**Input:**  $k$ -sequence  $S = s_1, \dots, s_k$

**Output:**  $v[A]$ , where  $A$  is an alignment of  $S$ , and  $vSP_\gamma[A] \leq 6 \text{optSP}_\gamma(S)$  if  $v = vA_\gamma$  and  $\gamma \in \mathbb{M}^A$ , and  $V_\gamma^2[A] \leq 12 \text{optNSP}_\gamma^2(S)$  if  $v = vN_\gamma$  and  $\gamma \in \mathbb{M}^N$ .

- 1: Let  $X$  be a  $v$ -optimal star of  $S$  with center  $c$
- 2: Compute the  $X$ -splitting  $Y$
- 3:  $A \leftarrow \text{COMPATIBLEALIGN}(Y)$
- 4: **return**  $v[A]$

Clearly, Algorithm 5 is correct. Furthermore, Lemmas 5.1, 5.3 and 5.5 are auxiliary to prove its approximation factor. Since the running time of COMPATIBLEALIGN is  $O(k^2n)$ , a straightforward running time analysis allows us to state the following theorem. The detailed proof is presented in [4].

► **Theorem 5.6.** *Let  $S = s_1, \dots, s_k$  be a  $k$ -sequence and  $\gamma$  be a scoring matrix. Then, Algorithm 5 computes  $v[A]$  correctly:*

- (i) *in  $O(k^2n^2)$ -time such that  $v\text{SP}_\gamma[A] \leq 6 \text{optSP}_\gamma(S)$ , if  $v = vA_\gamma$  and  $\gamma = \mathbb{M}^A$ , or*
- (ii) *in  $O(k^2n^3)$ -time such that  $V_\gamma^2[A] \leq 12 \text{optNSP}_\gamma^2(S)$ , if  $v = vN_\gamma$  and  $\gamma \in \mathbb{M}^N$ , where  $A$  is the alignment of  $S$  computed by the algorithm.*

## 6 Conclusion and future work

We presented and discussed several aspects of normalized multiple sequence alignment (NMSA). We defined three new criteria for computing normalized scores when aligning multiple sequences, showing the NP-hardness and exact algorithms for solving the NMSA- $z$  given criterion  $V_\gamma^z$  for each  $z$ . In addition, we adapted an existing 2-approximation algorithm for MSA when the scoring matrix  $\gamma$  is in the common class  $\mathbb{M}^C$ , leading to a 6-approximation algorithm for MSA when  $\gamma$  is in the broader class  $\mathbb{M}^A \supseteq \mathbb{M}^C$ , and to a 12-approximation for NMSA-2 when  $\gamma$  is in  $\mathbb{M}^N \subseteq \mathbb{M}^A$ , a slightly more restricted class compared to  $\mathbb{M}^A$  such that the cost of a deletion for any symbol is at most twice the cost for any other.

This work is an effort to expand the boundaries of multiple sequence alignment algorithms towards normalization, an unexplored domain that can produce results with higher accuracy in some applications. In future work, we will implement our algorithms in order to verify how large are the sequences that our algorithms are able to handle. Also, we plan to perform practical experiments, measuring how well alignments provided by our algorithms and other MSA algorithms agree with multiple alignment benchmarks. In addition, we intend to measure the accuracy of phylogenetic tree reconstruction based on our alignments for simulated and real genomes. Finally, we will work on heuristics and parallel versions of our algorithms in order to faster process large datasets.

---




## References

- 1 A. Abbott and A. Tsay. Sequence analysis and optimal matching methods in sociology: Review and prospect. *Sociol Method Res*, 29(1):3–33, 2000. doi:10.1177/0049124100029001001.
- 2 A. Andoni, R. Krauthgamer, and K. Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proc. of FOCS*, pages 377–386. IEEE, 2010. doi:10.1109/FOCS.2010.43.
- 3 A. Apostolico and Z. Galil. *Pattern Matching Algorithms*. Oxford University Press, 1997.
- 4 E. Araujo, L. C. Rozante, D. P. Rubert, and F. V. Martinez. Algorithms for normalized multiple sequence alignments, 2021. arXiv:2107.01607.
- 5 E. Araujo and J. Soares. Scoring matrices that induce metrics on sequences. In *Proc. of LATIN*, pages 68–79, 2006. doi:10.1007/11682462\_11.
- 6 A. N. Arslan and Ö. Egecioglu. An efficient uniform-cost normalized edit distance algorithm. In *Proc. of SPIRE*, pages 9–15. IEEE, 1999. doi:10.1109/SPIRE.1999.796572.
- 7 A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J Comput*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.
- 8 R. Barzilay and L. Lee. Bootstrapping lexical choice via multiple-sequence alignment. In *Proc. of EMNLP*, pages 164–171. ACL, 2002. doi:10.3115/1118693.1118715.
- 9 H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J Appl Math*, 48(5):1073–1082, 1988. doi:10.1137/0148063.



- 10 D. Chakraborty, D. Das, E. Goldenberg, M. Koucký, and M. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *J ACM*, 67(6):1–22, 2020. doi:10.1145/3422823.
- 11 M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. In *Proc. of SODA*, pages 679–688. SIAM, 2002. doi:10.5555/545381.545472.
- 12 J. A. Cuff and G. J. Barton. Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. *Proteins*, 34(4):508–519, 1999. doi:10.1002/(SICI)1097-0134(19990301)34:4<508::AID-PROT10>3.0.CO;2-4.
- 13 I. Elias. Settling the intractability of multiple alignment. *J Comput Biol*, 13(7):1323–1339, 2006. doi:10.1089/cmb.2006.13.1323.
- 14 D.-F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol*, 25(4):351–360, 1987. doi:10.1007/BF02603120.
- 15 D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull Math Biol*, 55(1):141–154, 1993. doi:10.1007/BF02460299.
- 16 W. Haque, A. Aravind, and B. Reddy. Pairwise sequence alignment algorithms: A survey. In *Proc. of ISTA*, pages 96–103. ACM, 2009. doi:10.1145/1551950.1551980.
- 17 M. Hirose, Y. Totoki, M. Hoshida, and M. Ishikawa. Comprehensive study on iterative algorithms of multiple sequence alignment. *Bioinformatics*, 11(1):13–18, 1995. doi:10.1093/bioinformatics/11.1.13.
- 18 A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE T Pattern Anal*, 15(9):926–932, 1993. doi:10.1109/34.232078.
- 19 W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *J Comput Syst Sci*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.
- 20 S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, 1970. doi:10.1016/0022-2836(70)90057-4.
- 21 T. H. Ogden and M. S. Rosenberg. Multiple sequence alignment accuracy and phylogenetic inference. *Syst Biol*, 55(2):314–328, 2006. doi:10.1080/10635150500541730.
- 22 P. H. Sellers. On the theory and computation of evolutionary distances. *SIAM J Appl Math*, 26(4):787–793, 1974. doi:10.1137/0126070.
- 23 F. Sievers and D. G. Higgins. Clustal Omega. *Curr Protoc Bioinfo*, 48(1):3.13.1–3.13.16, 2014. doi:10.1002/0471250953.bi0313s48.
- 24 F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Söding, J. D. Thompson, and D. G. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Mol Syst Biol*, 7(1):539, 2011. doi:10.1038/msb.2011.75.
- 25 J. D. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Res*, 27(13):2682–2690, 1999. doi:10.1093/nar/27.13.2682.
- 26 E. Vidal, A. Marzal, and P. Aibar. Fast computation of normalized edit distances. *IEEE T Pattern Anal*, 17(9):899–902, 1995. doi:10.1109/34.406656.
- 27 I. M. Wallace, O. O’Sullivan, D. G. Higgins, and C. Notredame. M-Coffee: combining multiple sequence alignment methods with T-Coffee. *Nucleic Acids Res*, 34(6):1692–1699, 2006. doi:10.1093/nar/gkl091.
- 28 X.-D. Wang, J.-X. Liu, Y. Xu, and J. Zhang. A survey of multiple sequence alignment techniques. In *Proc. of ICIC*, pages 529–538. Springer, 2015. doi:10.1007/978-3-319-22180-9\_52.

# Separated Red Blue Center Clustering

Marzieh Eskandari   

Department of Computer Science, Alzahra University, Tehran, Iran

Bhavika Khare  

Department of Computer Science, University of Memphis, TN, USA

Nirman Kumar   

Department of Computer Science, University of Memphis, TN, USA

---

## Abstract

We study a generalization of  $k$ -center clustering, first introduced by Kavand et. al., where instead of one set of centers, we have two types of centers,  $p$  red and  $q$  blue, and where each red center is at least  $\alpha$  distant from each blue center. The goal is to minimize the covering radius. We provide an approximation algorithm for this problem, and a polynomial-time algorithm for the constrained problem, where all the centers must lie on a line  $\ell$ .

**2012 ACM Subject Classification** Mathematics of computing → Approximation algorithms; Theory of computation → Facility location and clustering; Theory of computation → Computational geometry

**Keywords and phrases** Algorithms, Facility Location, Clustering, Approximation Algorithms, Computational Geometry

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.41

**Related Version** *Full Version:* <https://arxiv.org/abs/2107.07914>

## 1 Introduction

The  $k$ -center problem is a well-known geometric location problem, where we are given a set  $P$  of  $n$  points in a metric space and a positive integer  $k$ , and the task is to find  $k$  balls of minimum radius whose union covers  $P$ . This problem can be used to model the following facility location scenario. Suppose we want to open  $k$  facilities (such as supermarkets) to serve the customers in a city. It is common to assume that a customer shops at the facility closest to their residence. Thus, we want to locate  $k$  locations to open the facilities, so that the maximum distance between a customer and their nearest facility is minimized. The problem was first shown to be NP-hard by Megiddo and Supowit [12] for Euclidean spaces. We consider a variation of this classic problem where instead of just one set of centers, we consider two sets of centers, one of size  $p$ , and the other of size  $q$ , but with the constraint that each center of the first set is separated by a distance of at least some given  $\alpha$  from each center of the second set. This follows from a more practical facility location scenario, where we want to open two types of facilities (say “Costco’s” and “Sam’s club”). Each facility type wants to cover all the customers within the minimum possible distance (similar to the  $k$ -center clustering objective), but the facilities want to be separated from each other to avoid crowding or getting unfavorably affected by competition from the other.

The  $k$ -center problem has a long history. In 1857 Sylvester [15] presented the 1-center problem for the first time, and Megiddo [11] gave a linear time algorithm for solving this problem, also known as the minimum enclosing ball problem, in 1983, using linear programming. Hwang et al. [9] showed that the Euclidean  $k$ -center problem in the plane can be solved in  $n^{O(\sqrt{k})}$ . Agarwal and Procopiuc [1] presented an  $n^{O(k^{1-1/d})}$ -time algorithm for solving the  $k$ -center problem in  $\mathbb{R}^d$  and a  $(1 + \epsilon)$ -approximation algorithm with running time  $O(n \log k) + (k/\epsilon)^{O(k^{1-1/d})}$ .



© Marzieh Eskandari, Bhavika Khare, and Nirman Kumar;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 41; pp. 41:1–41:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Due to the importance of this problem, many researchers have considered variations of the basic problem to model different situations. Brass et al. studied the constrained version of the  $k$ -center problem in which the centers are constrained to be co-linear [4], also considered previously for  $k = 1$  by Megiddo [11]. They gave an  $O(n \log^2 n)$ -time algorithm when the line is fixed in advance. Also, they solved the general case where the line has an arbitrary orientation in  $O(n^4 \log^2 n)$  expected time. They presented an application of the constrained  $k$ -center in wireless network design: For a given set of  $n$  sensors (which are modeled as points), we want to locate  $k$  base servers (centers of balls) for receiving the signal from the sensors. The servers should be connected to a power line, so they have to lie on a straight line which models the power line. Other variations have also been considered [8, 2, 3] for  $k = 1$ . For  $k \geq 2$ , variants have been studied as this has applications to the placement of base stations in wireless sensor networks [5, 13, 14].

Hwang et al. [9] studied a variant somewhat opposite to our variant. In their variant, for a given constant  $0 \leq \alpha \leq 1$ , the  $\alpha$ -connected two-center problem is to find two balls of minimum radius  $r$  whose union covers the points, and the distance of the two centers is at most  $2(1 - \alpha)r$ , i.e., any two of those balls intersect such that each ball penetrates the other to a distance of at least  $2\alpha r$ . They presented an  $O(n^2 \log^2 n)$  expected-time algorithm.

The variant we consider was first considered by Kavand et. al. [10]. They termed it as the  $(n, 1, 1, \alpha)$ -center problem. They aimed to find two balls each of which covers the entire point set, the radius of the bigger one is minimized, and the distance of the two centers is at least  $\alpha$ . They presented an  $O(n \log n)$ -time algorithm for this problem and a linear time algorithm for its constrained version using the furthest point Voronoi diagram.

This paper considers the generalization of the problem defined by [10], and we denote it by  $(n, p \wedge q, \alpha)$  problem. We explain our choice of notation, particularly the  $\wedge$  sign, in Section 2. For a given set  $P$  of  $n$  points in a metric space and integers  $p, q \geq 1$ , we want to find  $p + q$  balls of two different types, called **red** and **blue** with the minimum radius such that  $P$  is covered by the  $p$  red balls and also covered by the second type of  $q$  blue balls, and the distance of the centers of each red ball from the centers of the blue balls is at least  $\alpha$ . In addition to one example mentioned before, another motivating application of the  $(n, p \wedge q, \alpha)$  problem would be to locate  $p$  police stations and  $q$  hospitals in an area such that the distance between each police station and a hospital is not smaller than a predefined distance  $\alpha$  for the convenience of patients. By locating hospitals and police stations at an admissible distance from each other, patients stay away from crowd and noise while the clients have access to hospitals and police stations that are close enough to them. Moreover, it is obviously desirable that the maximum distance between a client and its nearest police station as well as its nearest hospital is minimized. In addition to this general problem we also consider the constrained version due to its applicability in many situations, where the centers are constrained to lie on a given line.

**Paper organization.** In Section 2, we present the formal problem statement and the definitions required in the sequel. In Section 3, we present an  $O(1)$  factor approximation algorithm for the problem in Euclidean spaces. Then, in Section 4, we present a polynomial-time algorithm for the constrained problem. We conclude in Section 5.

## 2 Problem and Definitions

Let  $\mathcal{M}$  denote a metric space. Let  $\text{dist}(p, q)$  denote the distance between points  $p, q$  in  $\mathcal{M}$ . For a point  $x \in \mathcal{M}$  and a number  $r \geq 0$  the ball  $\mathbb{B}(x, r)$  is the set of points with distance at most  $r$  from  $x$ , i.e.,  $\mathbb{B}(x, r) = \{p \in \mathcal{M} \mid \text{dist}(x, p) \leq r\}$  is the *closed* ball of radius  $r$  with center  $x$ .

In the  $\alpha$ -separated red-blue clustering problem we are given a set  $P$  with  $n$  points in some metric space  $\mathcal{M}$ , integers  $p > 0, q > 0$ , and a real number  $\alpha > 0$ . For a given number  $r \geq 0$ ,  $p$  points  $c_1, \dots, c_p$  in  $\mathcal{M}$  (with possibly repeating points) called the red centers and  $q$  points  $d_1, \dots, d_q$  in  $\mathcal{M}$  (with possibly repeating points) called the blue centers are said to be a **feasible solution** for the problem, with **radius of covering**  $r$  if they satisfy,

- **Covering constraints:** The union of the balls  $\bigcup_{i=1}^p \mathbb{B}(c_i, r)$  (called the red balls) covers  $P$ , and the union of the balls  $\bigcup_{j=1}^q \mathbb{B}(d_j, r)$  (called the blue balls) covers  $P$ .
- **Separation constraint:** For each  $1 \leq i \leq p, 1 \leq j \leq q$  we have  $\text{dist}(c_i, d_j) \geq \alpha$ , i.e., the red and blue centers are separated by at least a distance of  $\alpha$ .

If there exists a feasible solution for a certain value of  $r$ , such an  $r$  is said to be feasible for the problem. The goal of the problem is to find the minimum possible value of  $r$  that is feasible.

We denote this problem as the  $(n, p \wedge q, \alpha)$ -problem. The  $\wedge$  in the notation stresses the fact that *both* the red balls *and* the blue balls cover  $P$ . Let  $r_{p \wedge q, \alpha}(P)$  denote the optimal radius for this problem. When  $P, p, q, \alpha$  are clear from context sometimes we will also denote this by  $r^*$ . Also, let  $r_k(P)$  denote the optimal  $k$ -center clustering radius, for all  $k \geq 1$ . To be clear, the centers in the  $k$ -center clustering problem can be any points in  $\mathcal{M}$ , not necessarily belonging to  $P$ . If that is the requirement, the problem is the *discrete*  $k$ -center clustering problem.

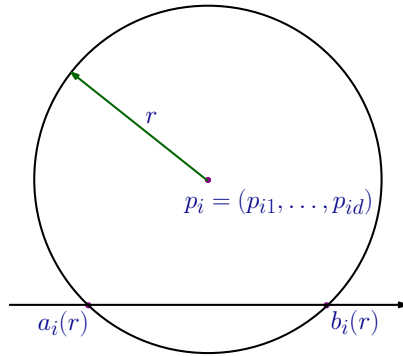
For this paper, we will always be concerned with  $\mathcal{M} = \mathbb{R}^d$ , but we will use the notations as defined above without qualifying the metric space. We let  $P = \{p_1, \dots, p_n\}$  where  $p_i = (p_{i1}, p_{i2}, \dots, p_{id})$ . We also consider the *constrained*  $\alpha$ -separated red-blue clustering problem (when  $\mathcal{M} = \mathbb{R}^d$ ) we are given a line  $\ell$  and all the red and blue centers are constrained to lie on the line  $\ell$ . Without loss of generality, we will assume that  $\ell$  is the  $x$ -axis since this can be achieved by an appropriate affine transformation of space. Moreover, we will use the same notation for the optimal radii and centers. For the constrained problem we need some additional definitions and notations. We assume that no two points in  $P$  have the same distance from  $\ell$ . (This general position assumption can however be removed.) For each point  $p_i$ , we consider the set of points on the line ( $x$ -axis) such that the ball of radius  $r$  centered at one of those points can cover  $p_i$ . This is the intersection of  $\mathbb{B}(p_i, r)$  with the  $x$ -axis, see Figure 1. Assuming this intersection is not empty, let the interval be  $I_i(r) = [a_i(r), b_i(r)]$ . Denote the set of all intervals as  $\mathcal{I}(r) = \{I_1(r), \dots, I_n(r)\}$  where we assume that the numbering is in the sorted order of intervals: those with earlier left endpoints are before, and for the same left endpoints the ones with earlier right endpoint occurs earlier in the order. Notice that feasibility of radius  $r$  means that there exist two hitting sets for the set of intervals  $\mathcal{I}(r)$ , the red centers and the blue centers such that they satisfy the separation constraint.

The interval endpoints  $a_i(r), b_i(r)$  can be computed by solving the equation,

$$(x - p_{i1})^2 + p_{i2}^2 + \dots + p_{id}^2 = r^2,$$

for  $x$ . Thus, they are given by  $a_i(r) = p_{i1} - \sqrt{r^2 - \sum_{j=2}^d p_{ij}^2}$ , and  $b_i(r) = p_{i1} + \sqrt{r^2 - \sum_{j=2}^d p_{ij}^2}$ . It is easy to see that for the range of  $r$  where the intersection is non-empty,  $a_i(r)$  is a strictly decreasing function of  $r$  and  $b_i(r)$  is a strictly increasing function of  $r$ .

**Model of computation.** We remark that our model of computation is the Real RAM model, where the usual arithmetic operations are assumed to take  $O(1)$  time.



■ **Figure 1** The functions  $a_i(r), b_i(r)$ .

**3** Approximation algorithms for the  $(n, p \wedge q, \alpha)$  problem in  $\mathbb{R}^d$

The  $(n, p \wedge q, \alpha)$  problem is NP-hard when  $p, q$  are part of the input, since the  $k$  center problem clearly reduces to the  $(n, p \wedge q, \alpha)$  problem when  $\alpha = 0$  and  $p = q = k$ . Here we show an approximation algorithm for the problem as well as one with a better approximation factor for the constrained problem. We assume that  $p \leq q$ , wlog.

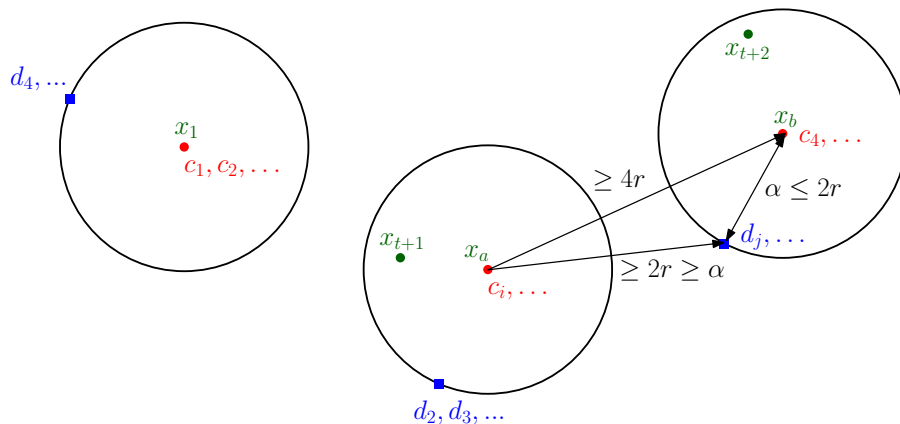
Here we show that there is a constant factor algorithm for the  $(n, p \wedge q, \alpha)$  problem in  $\mathbb{R}^d$ . We need a few preliminary results.

► **Lemma 1.** *Suppose that  $r \geq \alpha/2$  is a number such that there are points  $x_1, \dots, x_p$  satisfying  $P \subseteq \bigcup_{i=1}^p B(x_i, r)$ . Then, there are  $(p+q)$  points  $c_1, \dots, c_p, d_1, \dots, d_q$  all such that the following are met,*

- (I) Separation constraint:  $\text{dist}(c_i, d_j) \geq \alpha$  for all  $i, j$ , and,
- (II) Covering constraints:  $P \subseteq \bigcup_{i=1}^p B(c_i, 7r)$ , and,  $P \subseteq \bigcup_{j=1}^q B(d_j, 7r)$ .

**Proof.** First, from the points  $x_1, \dots, x_p$  we choose a maximal subset of them such that the distance between each pair of them is at least  $4r$ . This can be done by a simple scooping algorithm that starts with  $x_1$  as first point, then throws away all points  $x_i$  (for  $i > 1$ ) with  $\text{dist}(x_1, x_i) < 4r$ , then choose any one of the remaining points and proceed analogously.

Suppose after this (with some renaming) the points that remain are,  $x_1, \dots, x_t$ , where  $1 \leq t \leq p$ . Then, one can easily show that,  $P \subseteq \bigcup_{i=1}^t B(x_i, 5r)$ .



■ **Figure 2** Illustration for proof of Lemma 1.

Now, we choose the  $p$  red centers  $c_1, \dots, c_p$  at the points  $x_1, \dots, x_t$  such that each of them is chosen. Notice that this is possible since  $t \leq p$ . If  $t < p$ , some may be co-located at one  $x_i$ , though. Then, let  $d_1, \dots, d_q$  be any points on the surface of those balls (i.e., on the spheres). Since  $t \leq q$  we have enough points to hit all the balls. If necessary we can co-locate some points  $d_j$  to hit the target number  $q$ . See Figure 2.

The covering constraints are met since as remarked above, the balls of radius  $5r$  around all  $c_i$  (i.e., around all  $x_i$ ) covers  $P$ . Similarly, by the triangle inequality and because  $\alpha \leq 2r$ , the balls of radius  $7r$  around the  $d_j$  cover  $P$ .

To see the separation constraint, let  $c_i, d_j$  be any red and blue centers as defined above. Suppose  $c_i$  is located at the center  $x_a$  where  $1 \leq a \leq t$ , and  $d_j$  is located on the surface of the ball  $B(x_b, \alpha)$  where  $1 \leq b \leq t$ . If  $a = b$ , then since  $c_i$  is at the center and  $d_j$  on the surface of the ball  $B(x_a, \alpha)$  their distance is exactly  $\alpha$ . If, on the other hand,  $a \neq b$ , then by the triangle inequality we have that,  $\text{dist}(x_a, d_j) + \text{dist}(d_j, x_b) \geq \text{dist}(x_a, x_b) \geq 4r$ . Now,  $c_i = x_a$ , and  $\text{dist}(d_j, x_b) = \alpha \leq 2r$ , and so,  $\text{dist}(c_i, d_j) \geq 4r - 2r = 2r \geq \alpha$ , as desired.  $\blacktriangleleft$

► **Lemma 2.** *We have that,  $r_{p \wedge q, \alpha}(P) \geq \alpha/2$ .*

**Proof.** Consider any point  $p_l$ . This point is in some red ball  $B(c_i, r)$  and in some blue ball  $B(d_j, r)$ . Thus, by the triangle inequality,  $\text{dist}(c_i, d_j) \leq \text{dist}(c_i, p_l) + \text{dist}(p_l, d_j) \leq 2r$ . On the other hand,  $\alpha \leq \text{dist}(c_i, d_j)$ . Thus,  $\alpha \leq 2r$  and the claim follows.  $\blacktriangleleft$

Observe that since  $p \leq q$ ,  $r_p(P) \geq r_q(P)$ .

► **Lemma 3.** *We have that,  $r_{p \wedge q, \alpha}(P) \geq r_p(P)$ .*

**Proof.** Consider a feasible solution with the radius  $r^* = r_{p \wedge q, \alpha}(P)$ . The  $p$  red balls cover  $P$  with radius  $r^*$ . Thus,  $r^* \geq r_p(P)$ , since by definition,  $r_p(P)$  is the minimum  $p$ -center clustering radius.  $\blacktriangleleft$

Now, for the  $O(1)$  approximation to  $r_{p \wedge q, \alpha}(P)$  notice that we can easily compute by adapting Gonzalez's algorithm [7] a 2-approximation to the  $p$ -center clustering problem, i.e., to  $r_p(P)$ . (This is standard and well-known so we omit the details.) In other words we have now computed,  $p$  centers  $x_1, \dots, x_p$  all in  $P$ , and a radius  $r \leq 2r_p(P)$  such that balls  $B(x_i, r)$  cover  $P$ . We now consider the radius  $r' = \max(r, \alpha/2)$ , and clearly the balls  $B(x_i, r')$  also define such a covering. Now, using Lemma 1 we can find a feasible solution with radius at most  $7r' \leq 14r$ . Thus, we have shown the following theorem.

► **Theorem 4.** *Let  $r^*$  be the optimal radius for the  $\alpha$ -separated red blue clustering problem on an  $n$  point set  $P$  with parameters  $p, q \geq 1$ . Then, we can compute in polynomial-time, a feasible solution where the covering radius is at most  $14r^*$ .*

This approximation factor can be improved for the constrained problem, i.e., where all the centers are constrained to be on a fixed line  $\ell$ . The details can be found in the full version of this paper [6].

## 4 Polynomial-time algorithm for the constrained problem

Our basic approach will be to do a binary search for the optimal radius. We first present an algorithm to decide if a given value of the radius  $r$  is feasible. Then, we present an algorithm to determine a finite set of values such that the optimal radius must be within that set. Then, a binary search using the feasibility testing algorithm gives us the optimal radius.

## 4.1 Deciding feasibility for given radius $r$

Given a radius  $r > 0$  we give a polynomial-time method to decide if there is a solution to the constrained  $(n, p \wedge q, \alpha)$  problem with covering radius  $r$ . The algorithm is a dynamic programming algorithm. One of the challenges encountered is that the centers can be anywhere on the line, and thus a naive implementation of dynamic programming does not work since there are not finitely many sub-problems. As such, we first show how we can compute a finite set  $\mathcal{C}(r)$  of  $O(n^2)$  points such that if  $r$  is feasible, one can find a feasible solution with covering radius  $r$  with centers belonging to  $\mathcal{C}(r)$ .

### 4.1.1 Candidate points for the centers

Consider all the end-points of the intervals in  $\mathcal{I}(r)$ , i.e.,  $a_i(r), b_i(r)$ , and consider the sorted order of them. Assume that in sorted order the points are renamed to  $x_1, x_2, \dots, x_{2n}$  (possibly some of them are co-located). As remarked before, the feasibility problem is equivalent to finding two hitting sets for the intervals in  $\mathcal{I}(r)$  that satisfy the separation constraints. As in standard in such hitting set problems, we look at the *faces* of the arrangement of the intervals. Some of these faces might be open intervals, or half-open intervals, or even singleton points. Notice that all faces are disjoint by definition. It is easy to see that if  $F$  is such a face, then a point in the closure  $\bar{F}$  of  $F$  will hit at least the same intervals as points in  $F$  hit. To avoid confusion when we refer to faces vs. their closures, in the remaining discussion we will always say face  $F$  for the original face and closure face  $\bar{F}$  when referring to one of the closures (even though they may be the same set of points).

We explain now, why considering face closures is valid. Suppose a certain center belongs to a face  $F$  and suppose it is allowable to choose **any** point close enough to one of its boundary points, such that the separation constraints are met. Then, it is also valid to choose it at the boundary point and respecting the separation constraint since the separation constraint is that distance between the red and blue centers is  $\geq \alpha$  as opposed to a strict inequality  $> \alpha$ . Therefore, it is valid to replace faces with their closures.

To compute all the closures of the faces in the arrangement of the  $\mathcal{I}(r)$ , we sort the  $x_i$  and we retain all the consecutive intervals  $[x_i, x_{i+1}]$  that do not lie outside any of the intervals  $I_i(r)$ . This can be done by a simple line sweep algorithm. Notice that there are only  $O(n)$  such closure faces.

Next, we define a sequence for each such closure face. Consider such a closure face,  $[x_i, x_{i+1}]$  that is the **starting closure face** for this sequence. Consider the sequence,  $x_i, x_i + \alpha, x_i + 2\alpha, \dots$ . We only want to retain, for each closure face, (at most) the first three points that hit the closure face. So, given any starting closure face  $[x_i, x_{i+1}]$  there are only  $O(n)$  points in this sequence since it is bounded by the number of closure faces (in fact beyond the starting closure face  $[x_i, x_{i+1}]$ ) times three. Let the sequence of points that result due to starting closure face  $[x_j, x_{j+1}]$  be denoted by  $\text{SEQ}_j$ . We have the following lemma,

► **Lemma 5.** *For each starting closure face,  $[x_i, x_{i+1}]$  the associated sequence  $\text{SEQ}_i$  can be computed in  $O(n)$  time.*

**Proof.** We consider each closure face and compute the points of this sequence that possibly lie in this closure face. Consider such a closure face  $[x_j, x_{j+1}]$ . For a member of  $\text{SEQ}_i$  to lie in this closure face there is an integer  $k$  such that  $x_j \leq x_i + k\alpha \leq x_{j+1}$ . This is equivalent to,  $\frac{x_j - x_i}{\alpha} \leq k \leq \frac{x_{j+1} - x_i}{\alpha}$ . Thus, to find the first three points of  $\text{SEQ}_i$  hitting the closure face, we only need to find the three smallest integers in the interval,  $[\frac{x_j - x_i}{\alpha}, \frac{x_{j+1} - x_i}{\alpha}]$ , if there are such. This can be done in  $O(1)$  time per closure face. Since there are  $O(n)$  closure faces, the entire sequence can be constructed in  $O(n)$  time. ◀



Computing such sequence for each closure face as starting closure face leads to a total of  $O(n^2)$  points, and can be computed in  $O(n^2)$  time overall by the previous lemma. The set  $\mathcal{C}(r)$  is the set of the points in all these sequences. Let the sorted order of points in this set be denoted by  $c_1, c_2, \dots, c_m$  where  $m = O(n^2)$ . Thus  $\mathcal{C}(r) = \{c_1, \dots, c_m\}$ . For any such point  $c_k$  denote by  $s(k)$  the first index  $j$  such that  $c_j - c_k \geq \alpha$ . If there is no such point, let  $s(k) = m + 1$ . Notice that we can compute  $s(k)$  by a successor query in  $O(\log n)$  time if the set  $\mathcal{C}(r)$  is sorted. The following lemma says that we can assume that the centers (of both colors) are in  $\mathcal{C}(r)$ .

► **Lemma 6.** *Suppose that the constrained  $(n, p \wedge q, \alpha)$  problem has a feasible solution with covering radius  $r$ . Then there is a solution with all centers belonging to  $\mathcal{C}(r)$ .*

**Proof.** Consider a feasible solution with  $p$  red centers and  $q$  blue centers. First, we remark that we can assume, wlog, that in any face  $F$  there are at most two points, one red and one blue. This is true because having more than one red or more than one blue point in a face does not affect the covering constraints, as each point in a face hits (i.e., belongs to) the exact same set of intervals by definition. Thus, we may assume that there are at most  $T \leq (p + q)$  such centers, since we might need to throw away some of them when two centers of the same color belong to one face. Let the  $T$  centers be  $u_1, \dots, u_T$  where any of them can be red or blue. We show how to construct iteratively another feasible solution where all the centers are in  $\mathcal{C}(r)$ .

We will proceed face by face, and consider all the centers within the face. We know that there are at most two centers within a face. Moreover, if there are two they are of different colors. Our basic strategy is to move the first center left till we can, while remaining within the closure of the face, without violating any separation constraint. If there is only one point in a face, we are done. Otherwise, once the position of the first point is fixed, the second point can be similarly moved left until its position is determined within  $\mathcal{C}(r)$ . Let the sorted order of faces be  $F_1, F_2, \dots, F_N$  where  $N = O(n)$ .

We construct a new sequence  $v_1, \dots, v_T$  where  $v_i$  is assigned color of  $u_i$  and is obtained by shifting  $u_i$  to the left (so that it will lie in  $\mathcal{C}(r)$ , but never leaving closure of the face it belongs to). We prove the following claim by induction, which implies the claim that all the  $v_i$  belong to  $\mathcal{C}(r)$ .

▷ **Claim 7.** For each  $k \geq 1$ , all the points  $u_i$  belonging to face  $F_k$  are mapped to points  $v_i$  (belonging to  $\bar{F}_k$ ) such that, the at most two such  $v_i$ , lie on consecutive points of the same sequence  $\text{SEQ}_j$  for some  $j$ .

Consider the base case  $k = 1$ . If there are no points in  $F_1$ , the claim holds vacuously. If there is only one point in  $F_1$ , slide it left until it hits the boundary of  $F_1$ . This does not violate any constraints. The claim holds true trivially. Suppose there are two points in  $F_1$ . Now,  $\bar{F}_1 = [x_1, x_2]$  and after sliding the first point left till it coincides with  $x_1$ , and thus in  $\text{SEQ}_1$ , the second point clearly satisfies  $u_2 - x_1 \geq \alpha$  since even before the sliding the inequality was satisfied. Notice that the points are of different color. Thus we can place the second point  $v_2$  at  $x_1 + \alpha \in \text{SEQ}_1$  and they are consecutive points of  $\text{SEQ}_1$ .

Suppose that the claim is true up-to  $k$ . We now consider the case  $k + 1$ . Again, as before if there are no points in  $F_{k+1}$  the claim holds vacuously. If there is only one point, we slide it left to the first point in  $\mathcal{C}(r)$  which is allowable for it. The meaning of allowable is the following. Suppose this point is  $v_s$ . Then, if  $v_{s-1}$ , which is in a previous face, is of the same color as  $v_s$ , then  $v_s$  can be anywhere within its face. If it is of different color, then  $v_s$  has to be at least at  $v_{s-1} + \alpha$ . Since the starting point of the closure  $\bar{F}_{k+1}$  is in  $\mathcal{C}(r)$ , as is  $v_{s-1} + \alpha$

if it lies in  $\bar{F}_{k+1}$ , while sliding left we will hit a point in  $\mathcal{C}(r)$  eventually and we stop there. Now consider, the case where there are two points  $u_i, u_{i+1}$  in  $F_{k+1}$ . After  $v_i$  has been placed at its position in  $\mathcal{C}(r)$  as outlined for the case of single point in  $F_{k+1}$ , suppose it belongs to  $\text{SEQ}_j$  for some  $j$ . Clearly it is at most the second point, of  $\text{SEQ}_j$  in  $\bar{F}_{k+1}$  as the second point is already  $\alpha$  ahead of the first point of  $\text{SEQ}_j$  in  $\bar{F}_{k+1}$ . Now,  $u_{i+1} - v_i \geq \alpha$ . Thus, we can place the second point of  $F_{k+1}$  at the next point of  $\text{SEQ}_j$  in  $\bar{F}_{k+1}$ , which exists in  $\bar{F}_{k+1}$  since the next point is  $v_i + \alpha$  which is in  $\bar{F}_{k+1}$  by assumption. We observe that all claims hold true.  $\blacktriangleleft$

### 4.1.2 The dynamic programming algorithm

The dynamic programming algorithm computes two tables,  $\text{CanCover}^{\text{R}}[\mathcal{I}_1, \mathcal{I}_2, a, b, k]$ , and  $\text{CanCover}^{\text{B}}[\mathcal{I}_1, \mathcal{I}_2, a, b, k]$  of Boolean **True**, **False**. Here,  $\mathcal{I}_1, \mathcal{I}_2$  are prefixes of intervals of  $\mathcal{C}(r)$  (when they are ordered by their left and right end-points), and  $0 \leq a \leq p, 0 \leq b \leq q$  are integers, and  $1 \leq k \leq (m+1)$  is also an integer. The table entry  $\text{CanCover}^{\text{R}}[\mathcal{I}_1, \mathcal{I}_2, a, b, k]$  is **True**, if there is a hitting set consisting of (at most)  $a$  red points that hit the intervals in  $\mathcal{I}_1$ , (at most)  $b$  blue points that hit the intervals in  $\mathcal{I}_2$  and with the constraint that the first point to be possibly put is red and at  $c_k$  if  $k < m$ . Here  $k > m$  represents that there is no where to really put the first red point. Notice that the separation constraint between red/blue points must be met. Similarly the table entry  $\text{CanCover}^{\text{B}}[\mathcal{I}_1, \mathcal{I}_2, a, b, k]$  is true if the first point is blue and at  $c_k$  (for  $k \leq m$ ). Assuming that the above tables have been computed, we can answer whether the radius  $r$  is feasible by computing the following expression, where we denote  $\mathcal{I}(r)$  by  $\mathcal{I}$  for brevity,

$$\begin{aligned} & \text{CanCover}^{\text{R}}[\mathcal{I}, \mathcal{I}, p, q, 1] \vee \dots \vee \text{CanCover}^{\text{R}}[\mathcal{I}, \mathcal{I}, p, q, m] \\ & \quad \vee \\ & \text{CanCover}^{\text{B}}[\mathcal{I}, \mathcal{I}, p, q, 1] \vee \dots \vee \text{CanCover}^{\text{B}}[\mathcal{I}, \mathcal{I}, p, q, m]. \end{aligned}$$

In the above expression, we try to hit all the intervals in  $\mathcal{I}$  by both red and blue points and we try all possible starting locations and color for the first point. We know that the centers can be assumed to belong to  $\mathcal{C}(r) = \{c_1, \dots, c_m\}$ .

Now we present the recursive definition of the algorithm to fill the tables. We only present the definitions for  $\text{CanCover}^{\text{R}}[\cdot]$  but there is an entirely similar definition for  $\text{CanCover}^{\text{B}}[\cdot]$  with the roles of red and blue interchanged.

$$\text{CanCover}^{\text{R}}[\mathcal{I}_1, \mathcal{I}_2, a, b, k] = \begin{cases} \text{False} & \text{if } (\mathcal{I}_1 \neq \emptyset \wedge a = 0) \vee (\mathcal{I}_2 \neq \emptyset \wedge b = 0) \vee (\mathcal{I}_1 \cup \mathcal{I}_2 \neq \emptyset \wedge k > m), \\ \text{True} & \text{if } (\mathcal{I}_1 = \emptyset \wedge \mathcal{I}_2 = \emptyset), \\ \text{False} & \text{if there exists an interval in } \mathcal{I}_1 \cup \mathcal{I}_2 \text{ ending before } c_k, \\ B_a \vee B_b & \text{otherwise.} \end{cases}$$

The first case means that if there are not any red centers to put while some unhit intervals remain in  $\mathcal{I}_1$ , or not any blue centers to put but unhit intervals in  $\mathcal{I}_2$ , or if we have already passed over all centers ( $k > m$ ) but any unhit red or blue intervals remain, we return false. The next case means that if that all intervals have been hit already we should return true. The penultimate case means that if the first point  $c_k$  is so far ahead that at least one interval in  $\mathcal{I}_1 \cup \mathcal{I}_2$  ends before it, then there can be no solution. This is true because any later points, red or blue, will only be ahead of  $c_k$  and thus the ended interval cannot be hit. The last case

uses Boolean variables  $B_a, B_b$  that are defined as follows, and they also capture the main recursive cases. As required by the definition of the function, we must put the next center as red and at  $c_k$ . This would cause some intervals in  $\mathcal{I}_1$  to be hit by  $c_k$ . We remove those intervals from  $\mathcal{I}_1$ . Let  $\mathcal{I}'_1$  be the intervals in  $\mathcal{I}_1$  not hit by  $c_k$ . It is easy to see that if  $\mathcal{I}_1$  is a prefix of  $\mathcal{I}$ , then so is  $\mathcal{I}'_1$ . The definitions of  $B_a, B_b$  are as follows.

$$B_a \leftarrow (\mathcal{I}_1 = \emptyset) \vee \bigvee_{j=k+1}^m \text{CanCover}^R[\mathcal{I}'_1, \mathcal{I}_2, a-1, b, j]$$

This assignment ensures that if  $\mathcal{I}_1 = \emptyset$ , then we never really try to put any red point. If not, then we try all possibilities for the next position of the red point. Notice that putting another red point at  $c_k$  is not necessary so we start with the remaining positions and go up to  $m$ . The coverage requirements for blue points and their numbers remain unchanged. The red number decreases by 1. The Boolean  $B_b$  has the following definition,

$$B_b \leftarrow (\mathcal{I}_2 = \emptyset) \vee \bigvee_{j=s(k)}^m \text{CanCover}^B[\mathcal{I}'_1, \mathcal{I}_2, a-1, b, j].$$

This is because, if the next point (after the current red one) is to be blue, it can only be at index  $s(k)$  or later. Thus we look-up  $\text{CanCover}^B[\mathcal{I}'_1, \mathcal{I}_2, a-1, b, j]$  for all such possible  $j$ . The first check  $\mathcal{I}_2 = \emptyset$  means that if the blue intervals have already been hit, we do not need to put any blue point later. Both the tables  $\text{CanCover}^R[\cdot], \text{CanCover}^B[\cdot]$  are filled simultaneously by first filling in the entries fitting the base cases, and then traversing them in order of increasing  $a$ , increasing  $b$ , decreasing  $k$ , and increasing  $\mathcal{I}_1, \mathcal{I}_2$  (i.e., the smaller prefixes come earlier). It is easy to see that the traversal order meets the dependencies as written in the recursive definitions.

**Analysis.** First, observe that computing the candidate centers can be done in  $O(n^2)$  time as implied by Lemma 5 and the following discussion. Moreover, the successor points  $s(k)$  can all be computed in total  $O(n^2 \log n)$  time by first sorting  $\mathcal{C}(r)$  and then followed by successor queries. The time however is dominated by the main dynamic programming algorithm. Observe that there are  $O(n)$  prefixes, and  $m = O(n^2)$  possible center locations. Thus there are in total  $O(n^4 pq)$  entries to be filled. Except for the base cases, filling in an entry requires looking up  $O(n^2)$  previous entries, as well as some computation such as finding which intervals are not hit by the current point. Such queries can be handled easily for all the intervals say in  $\mathcal{I}_1$  wrt the point  $c_k$  in  $O(n)$  time. Thus for a particular table entry, we require  $O(n^2)$  time. Overall we will take  $O(n^6 pq)$  time. We get the following theorem,

► **Theorem 8.** *For the constrained  $(n, p \wedge q, \alpha)$  problem where the centers are constrained to lie on the  $x$ -axis, given a radius  $r$ , it can be decided if  $r$  is feasible in time  $T_{DP}(n, p, q) = O(n^6 pq)$ . Moreover, if  $r$  is feasible, a feasible solution with covering radius  $r$  can also be computed in the same time.*

To justify the comment about the feasible solution, note that by standard dynamic programming techniques, we can also remember while computing the table entries the solution, and it can be output at the end.

## 4.2 Candidate values for $r$

In this section, we will find a discrete candidate set for the optimal radii that facilitates a polynomial-time algorithm for solving the constrained  $(n, p \wedge q, \alpha)$  problem as presented in Section 4.3. For this purpose, we need to determine some properties of an optimal solution.

## 41:10 Separated Red Blue Center Clustering

First, we define a standard form solution and describe an easy approach to convert a feasible solution to the standard form. Then we present a lemma for proving a property of an optimal solution. Finally, we compute a finite candidate set for optimal radii.

Let  $U = \{u_1, u_2, \dots, u_{p+q}\}$  be a feasible solution with covering radius  $r$ . The closure face that contains  $u_i$  is denoted by  $[x_{i,1}(r), x_{i,2}(r)]$ , where  $x_{i,1}(r)$  and  $x_{i,2}(r)$  are the endpoints of some intervals (i.e.,  $a_j(r)$  or  $b_j(r)$ ). Since  $u_i$ s are on the  $x$ -axis, be a slight abuse of notation, we let  $u_i$  denotes the  $x$ -coordinate of point  $u_i$ . For a given feasible solution  $U = \{u_1, u_2, \dots, u_{p+q}\}$ , its **standard** form has two following properties:

1. If  $u_1$  and  $u_{p+q}$  are on the endpoints.
2. Any two consecutive same color centers are on the endpoints.

**Converting a given solution to standard form.** If  $u_1$  (resp.  $u_{p+q}$ ) is not on an endpoint, we move it to the left (resp. right) to hit  $x_{1,1}(r)$  (resp.  $x_{p+q,2}(r)$ ). For every pair of two consecutive same color centers  $u_i$  and  $u_{i+1}$ ,  $1 \leq i \leq p+q-1$ , if  $u_i$  is not on an endpoint, we move it to the right to hit  $x_{i,2}(r)$  and if  $u_{i+1}$  is not on an endpoint we move it to the left to hit  $x_{i+1,1}(r)$ .

Clearly, the standard form solution as constructed above satisfies the covering and separation constraints. Let  $S(k, j)$  denote a sequences of  $j+1$  consecutive centers in  $U$  starting from  $u_k$ , i.e.,  $(u_k, u_{k+1}, \dots, u_{k+j})$ . A sequence  $S(k, j)$  is called **alternate** if for all  $i, k \leq i \leq k+j-1$ ,  $u_i$  and  $u_{i+1}$  have different colors and centers  $u_k$  and  $u_{k+j}$  are on the endpoints and the other centers of the sequence are not on the endpoints (such a center is called **internal**).

Now note that if  $U$  is a standard solution, then the consecutive red-blue centers can be clustered in some alternate sequences. These alternate sequences can be provided by scanning the centers from left to right and clustering a couple of consecutive blue-red centers between two endpoints that include a center. To this end, we have the following simple approach:

**Clustering centers in alternate sequences.** Let  $u_{cur}$  be the first center that has not been visited yet. At the beginning,  $u_{cur} = u_1$ . Let  $u_i$  be the next closest different color center to  $u_{cur}$ . All centers from  $u_{cur}$  to  $u_{i-1}$  should be on the endpoints since they are the same color. We can construct the next sequence from  $k = i-1$ , i.e., we add  $u_{i-1}$  and  $u_i$  to a sequence. There are two events:

**Event 1:**  $u_i$  is on an endpoint, so the sequence is completed. If there are any unvisited centers, we continue scanning the centers by starting from  $u_i$ , i.e., we mark  $u_i$  as unvisited, set  $u_{cur} = u_i$ , and proceed as before until there is no unvisited center.

**Event 2:**  $u_i$  is not on an endpoint. Consider  $u_{i+1}$ .  $u_{i+1}$  and  $u_i$  should have different colors since  $U$  is standard. We add  $u_{i+1}$  to the sequence. If there are any unvisited centers, consider  $u_{i+2}$  and check Events 1 or 2 for  $i = i+2$ .

Note that some of the centers may belong to two alternate sequences (e.g., a center on an endpoint with different color adjacent centers) and some of them may not be in a sequence (e.g., a center with same-color adjacent centers).

Now we prove a property of the optimal solutions in standard form for being able to find a discrete candidate set for the optimal radii.

► **Lemma 9.** *Let  $U = \{u_1, u_2, \dots, u_{p+q}\}$  be a feasible solution for the constrained  $(n, p \wedge q, \alpha)$  problem with radius of covering  $r$ . If the distance between any two endpoints of the intervals in  $\mathcal{C}(r)$  is not  $t\alpha$ , where  $t \in \mathbb{Z}, 0 \leq t \leq p+q-1$ , then the constrained  $(n, p \wedge q, \alpha)$  problem has a feasible solution with radius less than  $r$ .*

**Proof.** We will show that there is a real number  $0 < \epsilon < r$  such that the constrained  $(n, p \wedge q, \alpha)$  problem has a feasible solution with radius of covering  $r - \epsilon$ . To this end, we obtain a set of centers,  $\bar{U}$ , from the given feasible solution  $U$  and show that the set of balls centered at the points in  $\bar{U}$  with radius  $r - \epsilon$  is a feasible solution for the problem. First, we need to modify  $U$  to find a feasible solution with the property that any two consecutive blue and red centers are at a distance strictly greater than  $\alpha$  (not exactly  $\alpha$ ). Then we use it for finding a solution,  $\bar{U}$ , with radius of covering  $r - \epsilon$  (that is explained later).

First of all, we convert  $U$  to standard form and compute all alternate sequences of the standard solution. Then we use them to find a feasible solution with the property that any two consecutive blue and red centers are at a distance of strictly greater than  $\alpha$ . Note that by the Lemma's assumption, each alternate sequence  $S(k, j)$  has at least a pair of two consecutive centers at a distance of strictly greater than  $\alpha$  (since each distance is at least  $\alpha$  and sum of them is not  $j\alpha$ ). But we need to have this strict inequality for all such pairs. So in each sequence  $S(k, j)$ , if there are two consecutive centers  $u_{k+i}$  and  $u_{k+i+1}$  at a distance of exactly  $\alpha$ , we should perturb the internal centers such that the distance between any two consecutive blue and red centers is strictly greater than  $\alpha$ .

For perturbing the internal centers of each alternate sequence  $S(k, j)$ , if  $j = 1$ , then  $u_{k+1} - u_k > \alpha$ , because  $u_k$  and  $u_{k+1}$  are on the endpoints so  $u_{k+1} - u_k \neq \alpha$ . If  $j > 1$ , we proceed by induction on the number of the pairs with the distance of  $\alpha$  which is denoted by  $n_{k,j}$ . For  $n_{k,j} = 1$ , let  $u_{k+i}, u_{k+i+1} \in S(k, j)$  such that  $u_{k+i+1} - u_{k+i} = \alpha$ . Since  $j > 1$ , at least one of  $u_{k+i}$  and  $u_{k+i+1}$  is internal, say  $u_{k+i}$ . Since  $n_{k,j} = 1$ ,  $u_{k+i} - u_{k+i-1} > \alpha$ . So we can shift  $u_{k+i}$  toward  $u_{k+i-1}$  infinitesimally such that  $u_{k+i} - u_{k+i-1} > \alpha$  and we still have  $u_{k+i+1} - u_{k+i} > \alpha$ . Assume for the induction hypothesis that for all integers  $m > 1$ , in a sequence  $S(k, j)$  with  $n_{k,j} < m$ , including a pair of consecutive red and blue centers at a distance greater than  $\alpha$ , we can perturb the internal centers such that all distances between two consecutive centers are strictly greater than  $\alpha$ . Now assume that  $n_{k,j} = m > 1$ . For some  $0 \leq i \leq j - 1$ , let  $u_{k+i}, u_{k+i+1} \in S(k, j)$  such that  $u_{k+i+1} - u_{k+i} = \alpha$ .  $S(k, j)$  has a pair of two consecutive centers at a distance greater than  $\alpha$ . This pair belongs to one of the sequences  $S(k, i)$  or  $S(k + i + 1, j)$ , say  $S(k, i)$ . It is clear that  $n_{k,i} < m$ , so by the induction hypothesis, we can perturb the internal centers of  $S(k, i)$  such that all distances between two consecutive centers are strictly greater than  $\alpha$ . Next, we can move  $u_{k+i}$  toward  $u_{k+i-1}$  infinitesimally such that  $u_{k+i} - u_{k+i-1} > \alpha$  and we now also have  $u_{k+i+1} - u_{k+i} > \alpha$ . Now we add  $u_{k+i}$  to  $S(k + i + 1, j)$  to obtain  $S(k + i, j)$  in which the distance between centers  $u_{k+i}$  and  $u_{k+i+1}$  is greater than  $\alpha$ . Since  $n_{k+i,j} < m$ , again by the induction hypothesis, we can perturb the internal centers of  $S(k + i, j)$  such that all distances between two consecutive centers are strictly greater than  $\alpha$ . It means that  $S(k, j)$  no longer contains a consecutive pair with distance  $\alpha$ .

Now we can compute  $\bar{U}$ . Let  $0 < \epsilon < r$  be a positive real number to be fixed later. If  $u_i$  is on an endpoint, say  $x_{i,1}(r)$ , let  $\bar{u}_i = x_{i,1}(r - \epsilon)$ , otherwise,  $\bar{u}_i = u_i$ . Notice that by our assumptions, there is no solution for  $t = 0$  so all endpoints are distinct. As such for an  $u_i$  on an endpoint, it is never on two endpoints simultaneously and its movement is unambiguously determined. We will show that there exists an  $\epsilon$  such that  $\bar{U} = \{\bar{u}_1, \bar{u}_2, \dots, \bar{u}_{p+q}\}$  is a feasible solution with radius of covering  $r - \epsilon$ , i.e.,  $\bar{U}$  should satisfy the covering and separation constraints. To this end, firstly, after decreasing  $r$  to  $r - \epsilon$ , the relative order of the endpoints of the intervals should not change, i.e., the displacement of an endpoint of a face  $F_i$  should be less than  $\|F_i\|/2$ , where  $\|F_i\|$  is the distance between the endpoints of face  $F_i$ . Secondly, for satisfying the covering constraint, the internal centers should remain in their faces, i.e.,  $x_{i,1}(r - \epsilon) < \bar{u}_i < x_{i,2}(r - \epsilon)$ . So the displacement of point  $x_{i,1}$  (resp.  $x_{i,2}$ ) should be less

## 41:12 Separated Red Blue Center Clustering

than  $u_i - x_{i,1}(r)$  (resp.  $x_{i,2}(r) - u_i$ ). Finally, for satisfying the separation constraint, in each sequence  $S(k, j)$ , we should have  $\bar{u}_{k+1} - \bar{u}_k \geq \alpha$  and  $\bar{u}_{k+j} - \bar{u}_{k+j-1} \geq \alpha$  (note that the distance between two internal centers does not change), i.e., the displacement of the endpoint that contains  $u_k$  (resp.  $u_{k+j}$ ) should be less than  $u_{k+1} - u_k - \alpha$  (resp.  $u_{k+j} - u_{k+j-1} - \alpha$ ). Therefore, by choosing real numbers  $\delta_1, \delta_2, \delta_3$  as follows, and  $0 < \delta < \min\{\delta_1, \delta_2, \delta_3\}$ , because of continuity of the movement of endpoints on line  $\ell$ , we can obtain a positive  $\epsilon$  such that the displacement of an endpoint becomes at most  $\delta$  when the radius decreases to  $r - \epsilon$ .

$$\begin{aligned} 0 < \delta_1 &< 1/2 \min_{1 \leq i \leq 2n-1} \{\|F_i\|\} \\ 0 < \delta_2 &< \min_{\forall S(k,j)} \left\{ \min_{k+1 \leq i \leq k+j-1} \{u_i - x_{i,1}(r), x_{i,2}(r) - u_i\} \right\} \\ 0 < \delta_3 &< \min_{\forall S(k,j)} \{u_{k+1} - u_k - \alpha, u_{k+j} - u_{k+j-1} - \alpha\} \end{aligned}$$

Consequently, there exists a non-zero  $\epsilon > 0$  such that the balls centered at points in  $\bar{U}$  with covering radius  $r - \epsilon$  is a feasible solution.  $\blacktriangleleft$

By Lemma 9, in the optimal solution, there is at least a pair of two endpoints at distance  $t\alpha$ , where  $t \in \mathbb{Z}, 0 \leq t \leq p + q - 1$ . The interval endpoints  $a_i(r)$  and  $b_i(r)$  are given by  $a_i(r) = p_{i1} - \sqrt{r^2 - \sum_{j=2}^d p_{ij}^2}$ , and  $b_i(r) = p_{i1} + \sqrt{r^2 - \sum_{j=2}^d p_{ij}^2}$ , so, a candidate set for the optimal radius can be computed by solving the following equations for all  $1 \leq i, k \leq n$  and  $t \in \mathbb{Z}, 0 \leq t \leq p + q - 1$ :

$$p_{i1} \pm \sqrt{r^2 - \sum_{j=2}^d p_{ij}^2} - p_{k1} \pm \sqrt{r^2 - \sum_{j=2}^d p_{kj}^2} = t\alpha,$$

since at least one of those equalities holds true. Due to our general position assumption, i.e., no two points in  $P$  have the same distance from  $\ell$ , these equations have a finite number of solutions. This is not too hard to show. See the full version of this paper [6] for the details. (We remark that the general position assumption can be removed. Without the assumption, Lemma 9 needs an amended statement and proof. Due to space constraints, we do not show this here. The amended statement and proof can be found in the full version of this paper [6].)

By solving these equations, we obtain  $O(n^2(p+q))$  candidates for the optimal radius and this proves the following lemma.

► **Lemma 10.** *There is a set of  $O(n^2(p+q))$  numbers, such that the optimal radius  $r_{p \wedge q, \alpha}(P)$  is one among them, and this set can be constructed in  $O(n^2(p+q))$  time.*

### 4.3 Main result

By first computing the candidates for  $r^*$  and then performing a binary search over them using the feasibility testing algorithm, we can compute the optimal radius. Thus we have the following theorem.

► **Theorem 11.** *The constrained  $(n, p \wedge q, \alpha)$  problem can be solved in  $O(n^2(p+q) + T_{DP}(n, p, q) \log n) = O(n^6 pq \log n)$  time.*

## 5 Conclusions

Improving the approximation factor of our main approximation algorithm (Theorem 4) and the running time of our polynomial-time algorithm for the constrained problem (Theorem 11) are obvious candidates for problems for future research work. Apart from this, it seems that a multi-color generalization of the  $k$ -center problem is worth studying for modeling similar practical applications. Here we want  $k$  different colored centers, and balls of each color covering all of  $P$  but with the separation constraints more general, i.e., between the centers of colors  $i, j$  the distance must be at least some given  $\alpha_{ij}$ . It seems that new techniques would be required for this general problem.

---

### References

- 1 P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.
- 2 P. Bose, S. Langerman, and S. Roy. Smallest enclosing circle centered on a query line segment. In *Proc. of Can. Conf. on Comp. Geom.*, 2008.
- 3 P. Bose and G. Toussaint. Computing the constrained euclidean geodesic and link center of a simple polygon with applications. In *Proc. Pacific Graph. Int.*, pages 102–112, 1996. doi:10.1109/CGI.1996.511792.
- 4 P. Brass, C. Knauer, H.-Suk Na, C.-Su Shin, and A. Vigneron. The aligned  $k$ -center problem. *Int. J. Comp. Geom. Appl.*, 21(2):157–178, 2011. doi:10.1142/S0218195911003597.
- 5 G. Das, S. Roy, S. Das, and S. Nandy. Variations of base-station placement problem on the boundary of a convex region. *Int. J. Found. Comput. Sci.*, 19:405–427, 2008. doi:10.1142/S0129054108005747.
- 6 M. Eskandari, B. B. Khare, and N. Kumar. Separated red blue center clustering, 2021. arXiv:2107.07914.
- 7 T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Th. Comp. Sc.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
- 8 F. Hurtado and G. Toussaint. Constrained facility location. *Studies of Location Analysis, Sp. Iss. on Comp. Geom.*, pages 15–17, 2000.
- 9 R. Z. Hwang, R. C. T. Lee, and R. C. Chang. The slab dividing approach to solve the euclidean  $p$ -center problem. *Algorithmica*, 9:1–22, 1993. doi:10.1007/BF01185335.
- 10 P. Kavand, A. Mohades, and M. Eskandari.  $(n, 1, 1, \alpha)$ -center problem. *Amirkabir Int. J. of Sc. & Res.*, 2014.
- 11 N. Megiddo. Linear time algorithms for linear programming in  $\mathbb{R}^3$ . *SIAM J. Comput.*, 12(4), 1983. doi:10.1137/0212052.
- 12 N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984. doi:10.1137/0213014.
- 13 S. Roy, D. Bardhan, and S. Das. Efficient algorithm for placing base stations by avoiding forbidden zone. In *Proc. of the Sec. Int. Conf. Dist. Comp. and Int. Tech.*, pages 105–116, 2005. doi:10.1007/11604655\_14.
- 14 C.-S. Shin, J.-H. Kim, S. K. Kim, and K.-Y. Chwa. Two-center problems for a convex polygon. In *Proc. of the 6th Ann. Euro. Symp. Alg.*, pages 199–210, 1998. doi:10.1007/3-540-68530-8\_17.
- 15 J. J. Sylvester. A question in the geometry of situation. *Quart. J. Math.*, 322(10):79, 1857.





# On the Extended TSP Problem

Julián Mestre  

Facebook Inc., Menlo Park, CA, USA  
The University of Sydney, Australia

Sergey Pupyrev  

Facebook Inc., Menlo Park, CA, USA

Seeun William Umboh  

The University of Sydney, Australia

---

## Abstract

We initiate the theoretical study of EXT-TSP, a problem that originates in the area of profile-guided binary optimization. Given a graph  $G = (V, E)$  with positive edge weights  $w : E \rightarrow \mathbb{R}^+$ , and a non-increasing discount function  $f(\cdot)$  such that  $f(1) = 1$  and  $f(i) = 0$  for  $i > k$ , for some parameter  $k$  that is part of the problem definition. The problem is to sequence the vertices  $V$  so as to maximize  $\sum_{(u,v) \in E} f(|d_u - d_v|) \cdot w(u, v)$ , where  $d_v \in \{1, \dots, |V|\}$  is the position of vertex  $v$  in the sequence.

We show that EXT-TSP is APX-hard to approximate in general and we give a  $(k + 1)$ -approximation algorithm for general graphs and a PTAS for some sparse graph classes such as planar or treewidth-bounded graphs.

Interestingly, the problem remains challenging even on very simple graph classes; indeed, there is no exact  $n^{o(k)}$  time algorithm for trees unless the ETH fails. We complement this negative result with an exact  $n^{O(k)}$  time algorithm for trees.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Approximation algorithms analysis

**Keywords and phrases** profile-guided optimization, approximation algorithms, bandwidth, TSP

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.42

**Related Version** *Full Version*: <https://arxiv.org/abs/2107.07815>

**Acknowledgements** We would like to thank Vahid Liaghat for fruitful discussions on the EXT-TSP problem.

## 1 Introduction

Profile-guided binary optimization (PGO) is an effective technique in modern compilers to improve performance by optimizing how binary code is laid out in memory. At a very high level, the idea is to collect information about typical executions of an application and then use this information to re-order how code blocks are laid out in the binary to minimize instruction cache misses, which in turn translates into running time performance gains. Newell and Pupyrev [20] recently introduced an optimization problem, which they call the Extended TSP (EXT-TSP) problem that aims at maximizing the number of block transitions that do not incur a cache miss.

The input to the EXT-TSP problem is a weighted directed graph  $G = (V, E)$ , which in the context of PGO corresponds to the control flow representation of the code we are trying to optimize: Every node  $u \in V$  corresponds to a basic block of code (for the purposes of this paper we can think of each of these blocks as a single instruction that takes a fixed amount of memory to encode); every edge  $(u, v) \in E$  represents the possibility of an execution jumping from  $u$  to  $v$ , and the weight  $w(u, v)$  captures how many times the profiler recorded said jump during the data collection phase. Our ultimate goal is to find a linear ordering of the nodes, each of which represents a possible code layout of the binary; we let this linear ordering be encoded by a one-to-one function  $d : V \rightarrow \{1, \dots, |V|\}$ . Finally, each edge  $(u, v)$



© Julián Mestre, Sergey Pupyrev, and Seeun William Umboh;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 42; pp. 42:1–42:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

contributes  $f(|d_u - d_v|) \cdot w(u, v)$  to the objective, where  $|d_u - d_v|$  is the distance between the edge endpoints in the linear ordering, and  $f(\cdot)$  is a non-increasing discount function such that  $f(1) = 1$  and  $f(i) = 0$  for  $i > k$ , where  $k = O(1)$  is part of the problem definition.

Newell and Pupyrev [20] designed and evaluated heuristics for EXT-TSP leading to significantly faster binaries. Their implementation is available in the open source project Binary Optimization and Layout Tool (BOLT) [17, 20, 22]. In their experiments, they found that setting  $k$  to be a small constant<sup>1</sup> and  $f(|d_u - d_v|) = \left(1 - \frac{|d_u - d_v|}{k}\right)$  for  $1 < |d_u - d_v| < k$ , yields the best results. The high level intuition is that the discount factor is a proxy for the probability that taking the jump causes a cache miss. Thus, the EXT-TSP objective aims at maximizing the number of jumps that do not cause a cache miss.

In this paper we initiate the theoretical study of EXT-TSP by providing a variety of hardness and algorithmic results for solving the problem both in the approximate and the exact sense in both general and restricted graph classes.

## 1.1 Our results

We show that EXT-TSP is APX-hard to approximate in general. We give a polynomial time  $(k + 1)$ -approximation algorithm and a  $n^{O(\frac{k}{\epsilon})}$  time  $(2 + \epsilon)$ -approximation for general graphs. We also give a  $n^{O(\frac{k}{\epsilon})}$  time  $(1 + \epsilon)$ -approximation for some sparse graphs classes such as planar or treewidth-bounded graphs.

Interestingly, the problem remains challenging even on very simple graph classes; indeed, there is no exact  $n^{O(k)}$  time algorithm for trees unless the ETH fail. Finally, we complement this negative result with an exact  $n^{O(k)}$  time algorithm for trees.

## 1.2 Related work

PGO techniques have been studied extensively in the compiler’s community. Code re-ordering is arguably the most impactful optimization among existing PGO techniques [22]. The classical approach for code layout is initiated by Pettis and Hansen [24], who formulated the problem of finding an ordering of basic blocks as a variant of the maximum directed TRAVELING SALESMAN PROBLEM on a control flow graph. They describe two greedy heuristics for positioning of basic blocks. Later, one of the heuristics (seemingly producing better results) has been adopted by the community, and it is now utilized by many modern compilers and binary optimizers, including LLVM and GCC. Very recently, Newell and Pupyrev [20] extended the classical model and suggested a new optimization problem, called EXTENDED-TSP. With an extensive evaluation of real-world and synthetic applications, they found the objective of EXT-TSP is closely related to the performance of a binary; thus, an improved solution of the problem yields faster binaries. We refer to [20] for a complete background on this literature.

The problem of laying out data in memory to minimize the cache misses has been studied in the Algorithms community [1, 12, 19, 30]. In this setting a number of requests arrives online and our job is to design an eviction policy [31]. Even though ultimately, we are also concerned with minimizing cache misses, there are two main differences: first, the profile data gives us information about future request that we can exploit to improve locality; second, this optimization is done at the compiler, which does not have control over the operating system’s cache eviction policy. The benchmark used for online algorithms is the competitive

---

<sup>1</sup> To be more specific,  $k$  is the number of blocks that can fit into 1024 bytes of memory.

ratio: the number of cache misses incurred by the online algorithm divided by the number of cache misses incurred by an optimal algorithm that knows the entire sequence of requests in advance. It is known that the best competitive ratio is  $\Theta(k)$  for deterministic algorithms and is  $\Theta(\log k)$  [1], where  $k$  is the size of the cache.

There are many classical optimization problems that seek for to sequence the vertex set of a graph to optimizing some objective function. The two most closely related to our problem are MAX TSP and MIN BANDWIDTH.

An instance of MAX TSP consists of a weighted undirected graph and our objective is to sequence the vertex set to maximize the weight of adjacent nodes. The problem is known to be APX-hard [23] and a number of approximation algorithms are known [4, 7, 13, 15, 16, 18, 21, 28], with the best being the  $5/4$ -approximation of Dudycz *et al.* [7] that runs in  $O(n^3)$ .

An instance of MIN BANDWIDTH consists of an undirected graph and our objective is to sequence the vertex set to minimize the maximum distance between the endpoints of any edge in the graph. The problem admits an  $n^{O(b)}$  time exact algorithm [26], where  $b$  is the bandwidth of the graph. On the negative side, there is no exact  $g(b)n^{o(b)}$  time algorithm [5] and unless the ETH fails, even in trees of pathwidth at most two. Several poly-logarithmic approximation algorithms exist for different graphs classes [9, 11, 14]; on the other hand, it is NP-hard to approximate the problem within any constant even for caterpillars [6].

A somewhat related problem is the MIN LINEAR ARRANGEMENT problem. An instance consists of an undirected graph and our objective is to sequence the vertex set to minimize the sum of the distances between the endpoints of each edge in the graph. Minimizing this objective function is equivalent to maximizing the EXT TSP objective function with the discount function  $f(i) = 1 - i/n$ . MIN LINEAR ARRANGEMENT admits polynomial-time exact algorithms on trees [29]; however, we are not aware of any results for higher treewidth. There are several poly-logarithmic approximation algorithms [3, 8, 10, 25] based on the spreading metrics technique of Even *et al.* [8]; however it is unclear how these techniques can be made to work for EXT TSP. Moreover, for our applications, we are interested in the regime where  $k \ll n$ , so this connection does not yield a result of practical relevance.

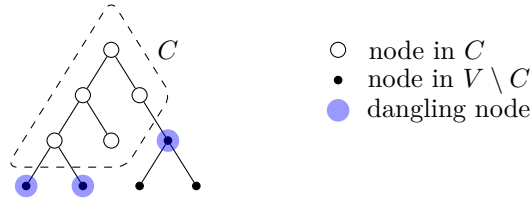
## 2 Problem definition and hardness

An instance of EXT-TSP problem consists of a directed graph  $G = (V, E)$  with positive edge weights  $w : E \rightarrow R^+$  and a non-increasing discount function  $f(\cdot)$  where  $f(1) = 1$  and  $f(i) = 0$  for  $i > k$ , where  $k$ , where  $k$  is a parameter that is part of the problem definition. The problem is to sequence the vertices  $V$  so that  $d_v \in \{1, \dots, |V|\}$  is position of vertex  $v$  with the objective to maximize

$$\sum_{(u,v) \in E} f(|d_u - d_v|) \cdot w(u, v)$$

The first thing to notice is that the fact that we could have defined the problem on an undirected graph since the contribution of an edge  $(u, v)$  to the objective only depends on its weight and the distance between its two endpoints, and is independent of whether it is a forward or a backward jump. Indeed, we can reduce the undirected case to the directed case and vice versa: Given an undirected graph, we can orient the edges arbitrarily; while given a directed graph we can combine pairs of anti-parallel edges into a single edge by adding up their weight.

In order to simplify our exposition, from now on we assume the input graph is undirected. Right away, this allows us to relate EXT-TSP to MAX TSP and MIN BANDWIDTH, which in turn yields the following hardness results. Finally, for the sake of succinctness, we extend the edge weight function to subsets of edges; namely,  $w(S) = \sum_{e \in S} w(e)$  for  $S \subseteq E$ .



■ **Figure 1** Dangling nodes of a root connected component  $C$ .

► **Theorem 1.** *The EXT-TSP problem exhibits the following hardness:*

1. *it is APX-hard, even when  $k = 1$ ,*
2. *does not admit an exact  $n^{o(k)}$  time algorithm unless the ETH fails, even in trees.*

**Proof.** For the first part, we use the relation between EXT-TSP and MAX TSP, which is known to be APX-hard [23]. Recall that the objective of the latter problem is to maximize  $\sum_{(u,v) \in E: |d_u - d_v| = 1} w(u,v)$  given an undirected graph. We can reduce an instance of MAX TSP to an undirected instance of EXT-TSP with  $k = 1$  where  $f(1) = 1$  and  $f(2) = 0$ . Therefore, EXT-TSP is APX-hard even when  $k = 1$ .

For the second part, we use the relation to MIN BANDWIDTH. Recall that the objective of the latter problem is to minimize  $\max_{(u,v) \in E} |d_u - d_v|$ , the optimal value of this objective is called the *bandwidth* of the graph. Given an instance  $G$  with bandwidth  $b$ , consider the EXT-TSP instance where  $f(i) = 1$  for  $0 \leq i \leq k$  and  $f(k+1) = 0$ ; if  $k = b$  then the objective of this instance must be  $w(E)$  as there exists a sequencing where the endpoints of every edge are within at most  $k$  of one another. It follows that, if we could have an  $n^{o(k)}$  time algorithm for EXT-TSP that implies an  $n^{o(b)}$  time algorithms, which does not exist even for very simple trees unless the ETH fails [5]. ◀

### 3 Exact Algorithms

In this section we complement the hardness from the previous section by developing an exact algorithm for trees whose running time is polynomial when  $k = \mathcal{O}(1)$ .

► **Theorem 2.** *There is an  $n^{O(k)}$  time algorithm for solving EXT-TSP optimally on trees.*

**Proof.** Let  $T$  be the input tree. Consider an optimal solution OPT, and let  $O$  be the set of *realized edges*, that is, the subset of edges whose endpoints are at distance at most  $k$  in OPT. Without loss of generality we assume that each connected component of  $O$  is laid out in a contiguous stretch in the optimal sequencing. Using this simple insight, we use dynamic programming (DP) to build a solution for the connected component  $C$  that has the root of the tree, and solve separately the subtree rooted at nodes that are not in  $C$  but that have a parent in  $C$ ; we call such nodes *dangling nodes* of  $C$  (see Figure 1). Without loss of generality, we assume that  $|C| \geq k$ . If  $C$  happens to be smaller, we can guess the optimal sequencing for  $C$  (there are only  $n^{k-1}$  choices), solve separately the subproblems rooted at dangling nodes of  $C$ , and keep the best solution.

Our algorithm is based on a subtle DP formulation. Each DP state represents succinctly a partial solution for a subtree of  $T$ , and it is defined by a tuple  $(z, \sigma, R)$ , where

- $z \in V$  is the root of the subtree of  $T$  we are trying to solve,
- $\sigma$  is a sequence of *exactly*  $k$  nodes in  $T_z$ , the subtree of  $T$  rooted at  $z$ ,
- $R$  is the set of edges incident on  $\sigma$  that have already been realized.

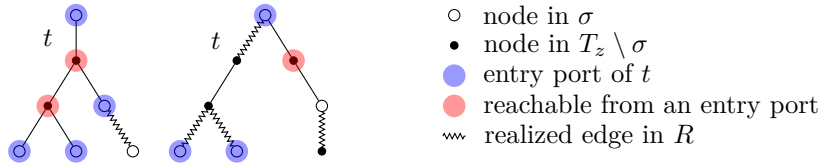


Figure 2 Two example showing the entry ports of a node  $t \in T_z \setminus \sigma$ . On the left, all entry ports of  $t$  are open, while on the right all entry ports of  $t$  are closed.

It is worth noting that although the structure of the DP states builds on that used in the algorithm of Saxe [26] for MIN BANDWIDTH, the fact that we do not necessarily realize all edges means we need new ideas and a more involved DP formulation to solve EXT-TSP.

Our high level goal is to build an edge weighted graph  $H$  over these tuples plus two dummy source and sink nodes  $s$  and  $t$  such that every optimal solution to the EXT-TSP problem on the subtree  $T_z$  induces an  $s$ - $t$  path whose weight equals the value of this solution; and conversely, every  $s$ - $t$  path induces an EXT-TSP solution of  $T_z$  whose value equals the weight of the path. Thus, once the graph is defined and the equivalence established, solving EXT-TSP amounts to a shortest path computation in  $H$ .

To provide some motivation and intuition on the definitions that will follow, consider an optimal solution of  $T_z$  realizing a subset of edges  $O$ , where  $C$  is the connected component of  $(T_z, O)$  that contains the root  $z$ , and let  $\tau$  be the optimal sequence for  $C$ . Note that  $\tau$  realizes  $O[C]$ , and by our earlier assumption  $|\tau| \geq k$ . For each  $j \in \{1, \dots, |C| - k + 1\}$  we let  $\sigma^j$  be the subsequence of  $\tau$  from  $j$  to  $j + k - 1$  and we let  $R^j$  be the subset of edges realized by the first  $j + k - 1$  positions of  $\tau$  that have at least one endpoint in  $\sigma^j$ . Then the path induced by  $\tau$  in  $H$  will be

$$s \rightarrow (z, \sigma^1, R^1) \rightarrow (z, \sigma^2, R^2) \rightarrow \dots \rightarrow (z, \sigma^{|C|-k+1}, R^{|C|-k+1}) \rightarrow t$$

The weight of the first edge  $s \rightarrow (z, \sigma^1, R^1)$  will be defined as the contribution of  $\sigma^1$  to the objective, that is the total discounted (according to  $\sigma^1$ ) weight of edges  $R^1$ . The weight of the last edge  $(z, \sigma^{|C|-k+1}, R^{|C|-k+1}) \rightarrow t$  will be defined as the value of the subproblems defined by dangling nodes of  $\sigma^{|C|-k+1}$  not spanned by  $R^{|C|-k+1}$ . Finally, the weight of an edge  $(z, \sigma^j, R^j) \rightarrow (z, \sigma^{j+1}, R^{j+1})$  will be defined as the value of the subproblem defined by dangling nodes of  $\sigma^j \setminus \sigma^{j+1}$  not spanned by  $R^{j+1}$  plus the discounted weight of  $R^{j+1} \setminus R^j$ . Since we do not double count any contributions, the weight of the path adds up to the value of the optimal solution for  $T_z$ .

Our goal is to impose some restrictions on the vertices and edges in  $H$  so that every  $s$ - $t$  path induces a solution of equal value in  $T_z$ . To that end we will define the notion of valid tuples and valid edges, but before we do that, we must introduce a few more concepts.

Given a tuple  $(z, \sigma, R)$  we say that a node  $u \in \sigma$  is an *entry port* for a node  $t \in T_z \setminus \sigma$  if the unique path  $P$  from  $t$  to  $u$  in  $T$  does not go through any other vertex in  $\sigma$ ; furthermore, we say that  $u$  is a *closed entry port* of  $t$  if the edge in  $P$  out of  $u$  is in  $R$ , otherwise, we say  $u$  is an *open entry port* of  $t$ . Finally, we say that  $t \in T_z \setminus \sigma$  is *reachable* if all the entry ports of  $t$  are open. See Figure 2 for an example illustrating these definitions.

A tuple  $(z, \sigma, R)$  is *valid* if for every  $t \in T_z \setminus \sigma$  the entry ports  $u \in \sigma$  for  $t$  are either all closed or all open. Indeed if  $(z, \sigma, R)$  was part of the path induced by some  $\tau$  then either  $t$  comes before  $\sigma$  in  $\tau$ , in which case  $t$  subtree spanned between the entry ports of  $t$  must have been already realized; or  $t$  comes after  $\sigma$  in  $\tau$ , in which case said subtree will be realized later on. Thus, we can focus only on valid tuples. We define a graph  $H$  over the valid tuples where we put a directed edge  $(z, \sigma, R) \rightarrow (z, \sigma', R')$  if:

## 42:6 On the Extended TSP Problem

- $\sigma'$  is obtained from  $\sigma$  by appending a reachable node (reachable with respect to the first tuple)  $v$  to  $\sigma$  and removing the first node  $u$  in  $\sigma$ ,
- $R'$  equals  $R$  minus edges in  $R$  that are incident on  $u$  but not on any other node in  $\sigma$ , plus edges from  $v$  to  $\sigma$ ,
- $(u, \text{parent}(u)) \in R \cup R'$ ,
- for each child  $c$  of  $u$  such that  $(c, u) \notin R \cup R'$ ,  $u$  is the unique (open) entry port of  $c$  (defined with respect to the first tuple) and  $v \notin T_c$ ; we call such  $c$ , a *dangling child* of  $u$ .

Furthermore, we define the weight of such an edge to be the discounted weight of newly realized edges (namely,  $R' \setminus R$ ) plus the total value of the optimal solutions for subtrees defined by dangling children of  $u$ . Note that the  $R' \setminus R$  must connect  $v$  to other nodes in  $\sigma$ , so we have all the information needed to discount their weight.

Finally, we connect  $s$  to each tuple  $(z, \sigma, R)$  where  $R$  is the set of edges with both endpoints in  $\sigma$  and the weight of the edge is the discounted (w.r.t.  $\sigma$ ) weight of  $R$ ; and we connect each tuple  $(z, \sigma, R)$  to  $t$  if the only reachable nodes adjacent to  $\sigma$  are dangling children and we set the weight of the edge to be the total value of the subproblems defined by those dangling children.

Given a path  $P$  in  $H$  we define  $\tau$  to be the induced solution by taking the  $\sigma$  of the first tuple in the path, and then extending the ordering by appending the new node of the  $\sigma$  in the next tuple and so on. Similarly, we can define the inverse operation: Given a sequencing  $\tau$  realizing a connected component of nodes that have the root of the tree, then we can define a sequence of tuples such that the sequence of tuples induces  $\tau$ .

▷ **Claim 3.** Let  $P$  be a path out of  $s$  in  $H$  inducing some ordering  $\tau$ . Then  $\tau$  realizes exactly the union of all the  $R$ -sets in  $P$ .

The claim is easy to prove by induction on the length of the sequence. If the sequence has only one tuple  $(z, \sigma, R)$ , then  $\tau = \sigma$  and  $R$  is the set of edges realized by  $\sigma$ , so the claim follows. Otherwise, if the last two tuples are  $(z, \sigma, R)$  and  $(z, \sigma', R')$  and  $v$  is the last node in  $\tau$  then  $R' \setminus R$  is the set of edges realized by  $\tau$  incident on  $v$  and we can use induction to account for the rest.

In order to prove the correctness of our dynamic programming formulation, we need to argue that every solution  $\tau$  to the original problem induces a path of equivalent cost, and vice-versa.

▷ **Claim 4.** Let  $\tau$  be the sequence of nodes in the connected component  $C$  of edges realized by the optimal solution OPT having  $z$ . The sequence of tuples induced by  $\tau$  forms a valid  $s$ - $t$  path whose weight equals

$$\sum_{(u,v) \in T[C]} f(|d_u - d_v|)w(u, v) + \sum_{\substack{u \notin C \\ \text{parent}(u) \in C}} \text{OPT}[T_u],$$

where  $d_u$  is the position of  $u$  in  $\tau$ .

If the sequence is a path, then by Claim 3,  $\tau$  realizes precisely the union of the  $R$ -sets in the sequence, and the weight of the path is precisely as stated in the claim. It only remains to show that the sequence is indeed a path. Consider two consecutive tuples  $(z, \sigma, R)$  and  $(z, \sigma', R')$  along the sequence. Our goal is to show that there is an edge connecting them. The first two conditions of a valid edge definition hold by definition of the induced sequence of tuples. For the third condition, note that  $(u, \text{parent}(u))$  must be realized by  $\tau$  and so  $\text{parent}(u)$  must occur within  $k$  positions of  $u$  so the edge must appear in  $R \cup R'$  and the



condition holds. For the fourth condition, if we let  $c$  be a child of  $u$  such that  $(c, u) \notin R \cup R'$ , we note that  $\tau$  cannot realize this edge after  $\sigma'$ , so it must be the case that  $v \notin T[c]$  (otherwise  $v$  would be disconnected from the root in  $C$ ) and that  $c$  is dangling child of  $u$  (otherwise  $c$  has a descendant in  $\sigma$  that would be disconnected from the root in  $C$ ).

▷ **Claim 5.** For a given  $s$ - $t$  path in  $H$ , let  $\tau$  be the ordering induced by the path. Then the set of edges realized by  $\tau$  forms a connected component  $C$  that contains the root and the weight of the path equals

$$\sum_{(u,v) \in T[C]} f(|d_u - d_v|)w(u, v) + \sum_{\substack{u \notin C \\ \text{parent}(u) \in C}} \text{OPT}[T_u],$$

where  $d_u$  is the position of  $u$  in  $\tau$ .

By Claim 3,  $\tau$  realizes precisely the union of the  $R$ -sets in the sequence. For every  $v \in \tau$  other than  $z$ , we argue that  $(v, \text{parent}(v))$  is realized by  $\tau$ . Indeed, let  $(z, \sigma, R)$  be the last tuple such that  $v \in \sigma$ . If  $(z, \sigma, R)$  is not the last tuple, by the third existence condition on the edge to the next tuple guarantees that  $(u, \text{parent}(u))$  is realized. If  $(z, \sigma, R)$  is the last tuple, by the existence condition on the edge to  $t$ , all reachable nodes adjacent to  $\sigma$  are dangling, in particular  $\text{parent}(u)$  is not reachable. Therefore, since  $(v, \text{parent}(v))$  is realized for all  $v$ , using induction we get that  $v$  must be connected all the way to the root with realized edges. Therefore the vertices in  $\tau$  form a connected subtree containing the root  $z$ , and the set of realized edges is precisely this subtree.

All this effort would be for naught, unless we could represent  $H$  succinctly. Recall that every node in  $H$  is a tuple  $(z, \sigma, R)$ ; clearly, there are only  $n$  choices for  $z$  and only  $n^k$  choices for  $\sigma$ ; furthermore, for an edge to be in  $R$ , since  $\sigma$  is a contiguous chunk of size  $k$ , they can only realize edges with connection to the previous  $k$  nodes, thus, we can represent  $R$  succinctly by listing those additional  $k$  nodes. Overall, there are  $n^{2k+1}$  edges in  $H$ ; we can list the outgoing neighboring tuples in  $O(n)$  time per tuple<sup>2</sup>. Therefore, we can run Dijkstra in  $O(n^{2k+2})$  time and identify the connected component of  $z$ . Since this has to be done for every node in  $T$ , we gain an extra factor of  $n$  for a running time of  $O(n^{2k+3})$ . ◀

## 4 Approximation Algorithms for special graph classes

In this section, we show that we can get very good approximations for special graph classes that go beyond trees.

► **Theorem 6.** *There is an  $n^{O(\frac{kt}{\epsilon})}$  time  $(1 + \epsilon)$ -approximation for EXT-TSP in graphs with a tree decomposition of tree-width  $t$ .*

**Proof.** Let  $T$  be the tree decomposition of our input graph  $G$  and let  $h = \lceil 1/\epsilon \rceil$ . To simplify the presentation of our algorithm we define an auxiliary problem, where the goal is to partition the vertex set into clusters of size at most  $hk$  and order each part separately, the EXT-TSP objective is computed for each part and summed up. If we let OPT be the value of the optimal solution for the original problem, we claim that OPT', the value of the optimal solution for the auxiliary problem is not much lower; more precisely,

$$\text{OPT}' \geq \frac{h-1}{h} \text{OPT}.$$

<sup>2</sup> We do not attempt to optimize this running time.

To see this, suppose that  $\text{OPT}$  lists the vertices in the order  $v_1, v_2, \dots, v_n$ . We pick a random threshold  $\alpha$  u.a.r. from  $\{0, 1, \dots, k-1\}$ , and cluster vertices together so that for each  $j$  we have a cluster  $\{v_{hkj+1+\alpha}, \dots, v_{hk(j+1)+\alpha}\}$ , yielding a solution to the auxiliary problem. Note that the probability of an edge that is realized by  $\text{OPT}$  must have endpoints that are at most  $k$  apart in the ordering, so there is only a  $1/h$  chance of that edge not being present in  $\text{OPT}'$ . Although this is a randomized construction, and it just shows that  $E[\text{OPT}'] \geq \frac{h-1}{h} \text{OPT}$ , it is easy to see that there must exist a value of  $\alpha$  that yields the desired bound<sup>3</sup>.

Given a tree decomposition for  $G$  with treewidth  $t$ , and a bag  $B$  in the decomposition we denote with  $T[B]$  the subset of vertices in the original graph spanned by the sub-decomposition rooted at  $B$ . For each  $u \in B$  we define a collection orderings of subsets  $\mathcal{S}_u$ , such that for an ordering  $\sigma$  of a subset  $S \subseteq V$  of vertices to be in  $\mathcal{S}_u$  we require that:

- $|S| \leq hk$ ,
- $u \in S$ , and
- the subgraph  $(S, \{(a, b) \in E[S] : |\sigma(a) - \sigma(b)| \leq k\})$  is connected.

We define a dynamic programming formulation for our auxiliary problem as follows. For each bag  $B$  in the decomposition and each  $|S|$ -tuple  $(\sigma_u : u \in B)$  where  $\sigma_u \in \mathcal{S}_u$ , we create a dynamic programming state  $A[B, (\sigma_u : u \in B)]$  that corresponds to the cost of the best solution for  $T[B]$  where each  $\sigma_u$  is the ordering of one of the clusters in the solution of the auxiliary problem. To keep the requirements feasible we ask that for any  $u, v \in B$  either  $\sigma_u = \sigma_v$ , or  $\sigma_u$  and  $\sigma_v$  do not share any vertices.

We work with a nice tree decomposition with join, forget, and introduce nodes. To define the recurrence for  $A$  we consider each case.

- Join node: Here we have two children with the same bag as the node. We simply pass the tuple constraining the solution space to each child. To compute its value we add the value of the two children and subtract the contribution of edges inside of  $B$  to avoid double counting. Notice that the distance between the endpoints of  $E[B]$  is specified by  $(\sigma_u : u \in B)$  so we can compute the appropriate discount of these edges.
- Introduce node: Here we have a single child with a bag having one fewer element; call it  $u$ . We remove  $\sigma_u$  from the tuple and  $u$  from  $B$ . To compute its value we add the contribution of edges between  $u$  and other nodes in  $\sigma_u$  to the value of the child. Again, we can use  $\sigma_u$  to discount the weight of these edges accordingly.
- Forget node: Here we have a single child with a bag with one additional element, call it  $u$ . To compute its value we need to guess the  $\sigma_u$  in the optimal solution. If  $u$  happens to already be in the sequence  $\sigma_v$  of some  $v \in B$  then  $\sigma_u = \sigma_v$ . Otherwise, we must guess  $\sigma_u$  by picking  $hk$  vertices from  $T[B] \setminus \cup_{v \in B} \sigma_v$  and checking that  $\sigma_u \in \mathcal{S}_u$ . Going over all possible valid states for the child bag and keeping the state with highest value yields the value of the state of the parent bag.

For the correctness, notice that there is no loss of information in the case of a introduce node. Let  $u$  be the node begin introduced. Either  $u$  is the only vertex in common between  $B$  and  $\sigma_u$ , in which case  $u$  is the only vertex in  $T[B]$  by virtue of  $\sigma_u$  being connected in  $G$ , and so it is safe to forget  $\sigma_u$  together with  $u$  in the child node. Or, there exists another  $v \in B - u$  such that  $v \in \sigma_u$ , which case  $\sigma_v = \sigma_u$  and so the information about the constraints we imposed in  $u$ 's part are preserved further down the decomposition.

<sup>3</sup> Note that the argument is non-constructive in the sense that given  $G$  it is not clear how to partition  $G$  into clusters of size  $hk$  so that  $\text{OPT}' \geq \frac{h-1}{h} \text{OPT}$ . The argument only guaranteed the existence of such a clustering.

For the correctness of the forget node case, note that the component that  $u$  belong to in the optimal solution is connected and that  $B$  acts like a separator from  $T[B]$  to the rest of the graph, so if  $u$  is not in the same component as any node in  $B$ , then it must be in a component with only nodes in  $T[B] \setminus \cup_{v \in B} \sigma_v$ .

There are  $n^{hkt+1}$  states in the decomposition and each one is considered once by a state associated with the parent bag in the decomposition, so the overall work is linear on the number of the states. We can enumerate the states on the fly by paying another  $O(n)$  term per state so the total running time is  $n^{hkt+2}$ .

Now, setting  $h = 1 + 1/\epsilon$ , the optimal solution found by DP is bound to be a  $1 + \epsilon$  approximation for the original problem in  $n^{O(\frac{k}{\epsilon})}$  as promised in the Theorem statement. ◀

We can use this result to obtain a  $(1 + \epsilon)$ -approximation for planar graphs.

► **Corollary 7.** *There is an  $n^{O(\frac{k}{\epsilon^2})}$  time  $(1 + \epsilon)$ -approximation for EXT-TSP in planar graphs.*

**Proof.** Using Baker’s technique [2] we can find an  $\ell$ -outerplanar subgraph  $G'$  of the input graph  $G$  such that value of the optimal solution to the EXT-TSP in  $G'$  is at least  $1 - 2/\ell$  the value of the optimal solution in  $G$ . Since the treewidth of  $G'$  is no more than  $3\ell$ , we can use the algorithm from Theorem 6 get a  $1 + \epsilon'$  approximation in  $G'$  in  $n^{O(\frac{k}{\epsilon'})}$  time. Setting  $\epsilon' = \epsilon/3$  and  $\ell = 6/\epsilon$ , we get the desired result for any  $\epsilon \leq 1$ . ◀

## 5 Approximation Algorithms for general graph

### 5.1 Greedy

Consider the following greedy algorithm: Start with an arbitrary vertex, and on each step append a vertex with the heaviest edge to the last-added vertex. That is, if  $u$  is the last-added vertex, then we append the vertex  $v$  maximizing  $w(u, v)$  where  $v \in V$  has not yet been added to the solution.

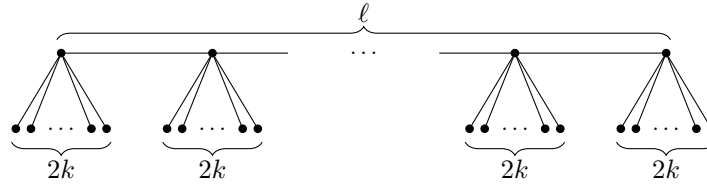
► **Lemma 8.** *Greedy is a  $2k$ -approximation and this is tight. It can be implemented to run in  $O(m \log n)$  time.*

**Proof.** Let  $O$  be the edges realized by the optimal solution and let  $u_1, u_2, \dots, u_n$  be the order computed by the greedy algorithm. Let  $d_u^*$  be the position of  $u$  in the optimal solution. Observe that the value of the greedy solution is at least  $\sum_{i=1}^{n-1} f(1)w(u_i, u_{i+1}) = \sum_{i=1}^{n-1} w(u_i, u_{i+1})$  as  $f(1) = 1$ . We partition  $O$  as follows, for each  $u_i$  we have a part  $O_i = \{(u_i, u_j) \in O : j > i\}$ . Using the fact that  $f$  is non-increasing and the definition of the greedy algorithm,  $f(|d_u^* - d_v^*|)w(u, v) \leq w(u, v) \leq w(u_i, u_{i+1})$  for all  $(u, v) \in O_i$ , and  $|O_i| \leq 2k$ . Thus, the value of the optimal solution is

$$\sum_{i=1}^{n-1} \sum_{(u,v) \in O_i} f(|d_u^* - d_v^*|)w(u, v) \leq 2k \sum_{i=1}^{n-1} w(u_i, u_{i+1}).$$

Thus, greedy is a  $2k$ -approximation.

To show that the analysis is tight, consider the following instance with  $n = (2k + 1)\ell$  consisting of  $\ell$   $2k$ -stars with the centers of the stars connected with a path of length of length  $\ell - 1$ . All edges have weight 1. The discount function  $f$  is such that  $f(i) = 1$  when  $i \leq k$  and  $f(i) = 0$  when  $i > k$ . The optimal solution sequences one star after the other and achieves a total cost of  $2k\ell$ . While the greedy solution may start at the center of the “left most star” and traverse the centers of all star and then add  $k$  pendant nodes, achieving a total cost of  $\ell - 1 + k$ . By making  $\ell$  large we get an approximation ratio that tends to  $2k$ .



■ **Figure 3** Tight instance for greedy. Optimal solution can realize  $2k\ell$  edges while Greedy may end up realizing only  $\ell - 1 + k$  edges.

For the implementation, we need to maintain a maximum priority queue with the nodes that are yet to be added to the greedy solution. The value associated with node  $u$  is the weight of the edge connecting  $u$  to the last node in the current partial greedy solution. When a new node is added to the greedy solution, this causes the priority of certain vertices to be updated (up for those incident on  $u$  or down for those incident on the second last-node of the partial solution, or either direction if incident on both nodes). The key observation is that each edge can cause the priority of a node to be changed twice (once when the first endpoint is added to the solution and again when that endpoint stops being the last node of the greedy solution). Therefore, the total number of priority updates is  $O(m)$ , which using a simple binary heap yields the desired time. ◀

### 5.2 Cycle cover based algorithm

We can do slightly better if we use a maximum weight cycle cover as the basis for our solution. A similar approach has been used to design approximation algorithms for max-TSP [13].

► **Theorem 9.** *There is a polynomial time  $\left(1 + \frac{1}{k+1}\right)$   $k$ -approximation for EXT-TSP in general graphs.*

**Proof.** Let  $A$  be a maximum weight set of edges such that the degree of every node is at most 2. This problem is also known as maximum weight simple 2-matching and can be reduced to regular maximum weight matching [27, Ch. 30]. Note that  $A$  is a collection of paths and cycles in  $G$ . If there exists a cycle  $C$  in  $A$ , we break  $C$  by removing the lightest edge. This gives us a collection of paths  $A'$ . Sequencing each path gives a solution to the EXT-TSP problem with value at least  $w(A')$ .

Now, given a solution to the EXT-TSP problem with value  $\text{OPT}$ , we claim that we can construct a solution to the degree bounded problem that has value at least  $\text{OPT}/k$ . To see this, note that the weight of the edges whose endpoints are at distance *exactly*  $i$  for  $i = 1, \dots, k$  is a candidate solution for  $A$ . It follows then that  $w(A) \geq \text{OPT}/k$ .

This is because the edges that are counted towards the objective in EXT-TSP have maximum degree  $2k$  and that solution can be scaled down by a factor of  $k$  to get a fractional solution to an exact LP formulation of the degree bounded problem. Thus, we get that  $w(A) \geq \text{OPT}/k$ .

Let  $\text{ALG}$  be the value of the solution found by our algorithm. Consider a cycle  $C$  in  $A$  with length  $\ell = |C|$ . Let  $e$  be the edge in  $C$  with minimum weight. Therefore,  $C$  contributes at least  $w(C) - w(e) + f(\ell - 1)w(e)$  to  $\text{ALG}$ . Since  $w(e) \leq w(C)/\ell$ , we can further simplify the previous expression to

$$w(C) \left(1 - \frac{1 - f(\ell - 1)}{\ell}\right).$$

Now if each cycle  $C$  in  $A$  had length at least  $\ell > k + 1$ , the weight of  $C \cap A'$  would be at least  $w(C) \left(1 - \frac{1}{k+2}\right)$ . Let ALG be the cost of the solution found by our algorithm. Then

$$\text{ALG} \geq w(A') \geq \frac{k+1}{k+2} w(A) \geq \frac{k+1}{(k+2)k} \text{OPT},$$

which matches the approximation factor of  $k + 1$  promised in the theorem statement. Unfortunately, cycles can be as small as  $\ell = 3$ , which depending on  $f$  could yield a worse approximation factor, so we need a different approach to our analysis.

Let  $\ell^*$  be the number in  $[3, 4, \dots, k + 2]$  maximizing  $\frac{1-f(\ell^*-1)}{\ell^*}$ . Using the same reasoning as above, we see that

$$\text{ALG} \geq w(A) \left(1 - \frac{1 - f(\ell^* - 1)}{\ell^*}\right).$$

The first thing to note is that if  $\ell^* = k + 2$  then the above analysis yield the desired approximation, so from now one assume  $\ell^* < k + 2$  and  $\frac{1-f(\ell^*-1)}{\ell^*} > \frac{1}{k+2}$ , or equivalently, that

$$1 - \frac{\ell^*}{k+2} > f(\ell^* - 1).$$

Consider the edges realized by the optimal solution and split them into  $X$  and  $Y$ . The first set,  $X$ , are the edges whose endpoints are at distance at most  $\ell^* - 2$  from each other; the second set,  $Y$ , are the edges whose endpoints are at distance between  $\ell^* - 1$  and  $k$ . Notice that

$$\text{OPT} \leq w(X) + f(\ell^* - 1)w(Y),$$

since all edges in  $Y$  are discounted at least  $f(\ell^* - 1)$ , and that

$$w(A) \geq \max \left\{ \frac{w(X)}{\ell^* - 2}, \frac{w(Y)}{k - \ell^* - 2} \right\},$$

since we can use the same scaling argument on  $X$  or  $Y$  but using a smaller scaling factor since the vertices in those edges sets have smaller degrees; namely,  $2\ell^* - 2$  and  $2(k - \ell^* - 1)$  respectively. Putting the above two inequalities together we get

$$\begin{aligned} \text{OPT} &\leq (\ell^* - 2)w(A) + f(\ell^* - 1)(k - \ell^* - 2)w(A) \\ &\leq \frac{(\ell^* - 2) + f(\ell^* - 1)(k - \ell^* - 2)}{\left(1 - \frac{1-f(\ell^*-1)}{\ell^*}\right)} \text{ALG}. \end{aligned}$$

Think of the above upper bound on the approximation ratio  $\text{OPT}/\text{ALG}$  as a function of  $f(\ell^* - 1)$ . We want to find the value  $0 \leq f(\ell^* - 1) \leq 1 - \ell^*/(k + 2)$  that yields the worst bound on the approximation ratio. The upper bound is the ratio of two linear functions of  $f(\ell^* - 1)$  and is thus maximized when either  $f(\ell^* - 1) = 0$  or  $f(\ell^* - 1) = 1 - \ell^*/(k + 2)$ . If  $f(\ell^* - 1) = 0$ , the ratio simplifies to  $\frac{\ell^*-2}{1-1/\ell^*}$ , which in turn is maximized at  $\ell^* = k + 2$  and yields a ratio of  $k + \frac{k}{k+1}$ , as desired. Finally, if  $f(\ell^* - 1) = 1 - \ell^*/(k + 2)$ , we again get the same approximation ratio. ◀

### 5.3 Local search algorithm

So far all the algorithms we have presented in this section have polynomial running times that are independent of  $k$ . If we are willing to have algorithms that run in  $n^{O(k)}$  we can get arbitrarily good approximations.

Our local search algorithm is parameterized by an integer value  $\ell \geq k$ . The algorithm maintains a solution  $\tau$  and performs local search moves where some subset of  $\ell$  nodes are taken out of  $\tau$  and sequenced optimally and attached to the end of the solution. At each step we perform the best such move and we stop once there is no move that improves the solution.

► **Lemma 10.** *A local optimal solution is a  $2 + \frac{2}{\ell/k-1}$  approximation for the EXT-TSP in general graphs.*

**Proof.** We will use the following notation throughout this proof: For a given solution  $\tau$  and a permutation  $\sigma$  of  $\ell$  elements, let  $\tau - \sigma$  the permutation of  $n - k$  elements that we get by removing the nodes in  $\sigma$  from  $\tau$ . Also, let  $\tau|\sigma$  be the permutation obtained by concatenating  $\sigma$  to  $\tau - \sigma$ . Finally, let  $w_\sigma(\tau)$  be the discounted weight of edges realized by  $\tau$  that are incident on vertices in  $\sigma$ , and  $w(\tau)$  be the discounted weight of all edges realized by  $\tau$ , i.e. the value of  $\tau$ .

Assume that  $\tau$  is locally optimal; namely, that no local move can improve its value:

$$w(\tau) \geq w(\tau|\sigma) \quad \forall \sigma : |\sigma| = \ell.$$

Notice that  $w(\tau) \leq w(\tau - \sigma) + w_\sigma(\tau)$  and that  $w(\tau|\sigma) \geq w(\tau - \sigma) + w(\sigma)$ . Therefore, a weaker necessary condition for being locally optimal is that

$$w_\sigma(\tau) \geq w(\sigma) \quad \forall \sigma : |\sigma| = \ell.$$

Let us build a collection for  $n + \ell$  sub-sequences of the optimal solution by sliding a window of size  $\ell$  over OPT. Call the resulting collection  $S$ . Adding up the above inequality for all  $\sigma \in S$  we get

$$\sum_{\sigma \in S} w_\sigma(\tau) \geq \sum_{\sigma \in S} w(\sigma).$$

Notice that every edge realized by  $\tau$  can appear in at most  $2\ell$  terms in the left-hand side of the above inequality (this is because every endpoint appears in at most  $\ell$  permutations), while every edge realized by OPT must appear in at least  $\ell - k$  terms in the right-hand side of the above inequality. These observation imply the following relation between  $\tau$  and OPT

$$2\ell w(\tau) \geq (\ell - k)w(\text{OPT}),$$

which in turn finish off the proof of the lemma. ◀

Of course, the issue with the above algorithm is that it is not clear how to compute a locally optimal solution. However, we can use the standard technique of only making a move if it improves the value of the objective by at least  $\delta/nw(\text{OPT})$ . This guarantees that we do not perform more than  $n/\delta$  and degrades the approximation ratio by no more than  $2\delta$ . This yields an algorithm that runs in  $O(n^{\ell+1}/\delta)$  time.

## 6 Conclusions and open problems

Our results can be generalized slightly. For example, one can get similar results for discount functions  $f$  that are non-symmetric (i.e.,  $f(u, v) \neq f(v, u)$ ), or when the block sizes are non-uniform. There are, however, some interesting questions that remain unanswered:

1. Is there a polynomial-time  $O(1)$ -approximation, independent of  $k$ ?
2. Is there an exact  $\mathcal{O}(f(k, t)n^{O(k)})$  time algorithm where  $t$  is the treewidth of the instance?

On the other hand, there are a few things that we can rule out. Note that we cannot expect  $(1 + \epsilon)$ -approximations even in  $n^{O(k)}$  time since that would contradict APX-hardness of MAX TSP, and we cannot expect to get exact algorithms for bounded treewidth instances in  $n^{o(k)}$  time either due to MIN BANDWIDTH hardness.

---

### References

- 1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.
- 2 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.
- 3 Moses Charikar, Mohammad Taghi Hajiaghayi, Howard J. Karloff, and Satish Rao.  $l_2^2$  spreading metrics for vertex ordering problems. *Algorithmica*, 56(4):577–604, 2010.
- 4 Zhi-Zhong Chen, Yuusuke Okamoto, and Lusheng Wang. Improved deterministic approximation algorithms for max TSP. *Inf. Process. Lett.*, 95(2):333–342, 2005.
- 5 Markus Sortland Dregi and Daniel Lokshtanov. Parameterized complexity of bandwidth on trees. In *International Colloquium on Automata, Languages, and Programming*, pages 405–416. Springer, 2014.
- 6 Chandan K. Dubey, Uriel Feige, and Walter Unger. Hardness results for approximating the bandwidth. *J. Comput. Syst. Sci.*, 77(1):62–90, 2011.
- 7 Szymon Dudycz, Jan Marcinkowski, Katarzyna E. Paluch, and Bartosz Rybicki. A  $4/5$ -approximation algorithm for the maximum traveling salesman problem. In *Proc of 19th International Conference on Integer Programming and Combinatorial Optimization*, volume 10328, pages 173–185, 2017.
- 8 Guy Even, Joseph Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *J. ACM*, 47(4):585–616, 2000. doi:10.1145/347476.347478.
- 9 Uriel Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci.*, 60(3):510–539, 2000.
- 10 Uriel Feige and James R. Lee. An improved approximation ratio for the minimum linear arrangement problem. *Inf. Process. Lett.*, 101(1):26–29, 2007. doi:10.1016/j.ipl.2006.07.009.
- 11 Uriel Feige and Kunal Talwar. Approximating the bandwidth of caterpillars. *Algorithmica*, 55(1):190–204, 2009.
- 12 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.
- 13 Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for finding a maximum weight hamiltonian circuit. *Oper. Res.*, 27(4):799–809, 1979.
- 14 Anupam Gupta. Improved bandwidth approximation for trees and chordal graphs. *J. Algorithms*, 40(1):24–36, 2001.
- 15 Refael Hassin and Shlomi Rubinstein. An approximation algorithm for the maximum traveling salesman problem. *Inf. Process. Lett.*, 67(3):125–130, 1998.



- 16 Refael Hassin and Shlomi Rubinfeld. Better approximations for max TSP. *Inf. Process. Lett.*, 75(4):181–186, 2000.
- 17 Facebook Inc. Binary optimization and layout tool, 2020. URL: <https://github.com/facebookincubator/BOLT>.
- 18 S. Rao Kosaraju, James K. Park, and Clifford Stein. Long tours and short superstrings (preliminary version). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 166–177. IEEE Computer Society, 1994.
- 19 Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991. doi:10.1007/BF01759073.
- 20 Andy Newell and Sergey Pupyrev. Improved basic block reordering. *IEEE Transactions in Computers*, 69(12):1784–1794, 2020.
- 21 Katarzyna E. Paluch, Marcin Mucha, and Aleksander Madry. A  $7/9$  - approximation algorithm for the maximum traveling salesman problem. In *Proc of 12th International Workshop on Approximation, Randomization, and Combinatorial Optimization*, volume 5687, pages 298–311, 2009.
- 22 Maksim Panchenko, Rafael Auler, Bill Nell, and Guilherme Ottoni. Bolt: A practical binary optimizer for data centers and beyond. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, pages 2–14, 2019.
- 23 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993.
- 24 Karl Pettis and Robert C Hansen. Profile guided code positioning. In *Proc. of the 11th ACM Conference on Programming Language Design and Implementation*, pages 16–27, 1990.
- 25 Satish Rao and Andréa W. Richa. New approximation techniques for some linear ordering problems. *SIAM J. Comput.*, 34(2):388–404, 2004. doi:10.1137/S0097539702413197.
- 26 James B Saxe. Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM Journal on Algebraic Discrete Methods*, 1(4):363–369, 1980.
- 27 Alexander Schrijver. *Combinatorial Optimization*. Springer-Verlag, 2003.
- 28 A. I. Serdyukov. An algorithm with an estimate for the traveling salesman problem of maximum (in russian). *Upravlyaemye Sistemy*, 25:80–86, 1984.
- 29 Yossi Shiloach. A minimum linear arrangement algorithm for undirected trees. *SIAM J. Comput.*, 8(1):15–32, 1979. doi:10.1137/0208002.
- 30 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.
- 31 Neal E. Young. Online paging and caching. In *Encyclopedia of Algorithms*, pages 1457–1461. Springer, 2016.

# Probabilistic Analysis of Euclidean Capacitated Vehicle Routing

Claire Mathieu  

CNRS, IRIF, Université de Paris, France

Hang Zhou  

École Polytechnique, Institut Polytechnique de Paris, France

---

## Abstract

We give a probabilistic analysis of the unit-demand Euclidean capacitated vehicle routing problem in the random setting, where the input distribution consists of  $n$  unit-demand customers modeled as independent, identically distributed uniform random points in the two-dimensional plane. The objective is to visit every customer using a set of routes of minimum total length, such that each route visits at most  $k$  customers, where  $k$  is the capacity of a vehicle. All of the following results are in the random setting and hold asymptotically almost surely.

The best known polynomial-time approximation for this problem is the iterated tour partitioning (ITP) algorithm, introduced in 1985 by Haimovich and Rinnooy Kan [15]. They showed that the ITP algorithm is near-optimal when  $k$  is either  $o(\sqrt{n})$  or  $\omega(\sqrt{n})$ , and they asked whether the ITP algorithm was “also effective in the intermediate range”. In this work, we show that when  $k = \sqrt{n}$ , the ITP algorithm is at best a  $(1 + c_0)$ -approximation for some positive constant  $c_0$ .

On the other hand, the approximation ratio of the ITP algorithm was known to be at most  $0.995 + \alpha$  due to Bompadre, Dror, and Orlin [10], where  $\alpha$  is the approximation ratio of an algorithm for the traveling salesman problem. In this work, we improve the upper bound on the approximation ratio of the ITP algorithm to  $0.915 + \alpha$ . Our analysis is based on a new lower bound on the optimal cost for the metric capacitated vehicle routing problem, which may be of independent interest.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Approximation algorithms analysis

**Keywords and phrases** capacitated vehicle routing, iterated tour partitioning, probabilistic analysis, approximation algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.43

**Related Version** *Full Version:* <http://arxiv.org/abs/2109.06958>

**Funding** This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

## 1 Introduction

**Unit-Demand Euclidean CVRP.** In the *capacitated vehicle routing problem (CVRP)*, we are given a set of  $n$  customers and a depot. There is an unlimited number of identical vehicles, each of an integer capacity  $k$ . The route of a vehicle starts at the depot and returns there after visiting at most  $k$  customers. The objective is to visit every customer, using a set of routes of minimum total length. Vehicle routing is a basic type of problems in operations research, and several books (see [3, 11, 14, 25] among others) have been written on those problems. We study the *unit-demand Euclidean* version of the problem, in which each customer has unit demand, all locations (the customers and the depot) lie in the two-dimensional plane, and distances are given by the Euclidean metric. The unit-demand Euclidean CVRP is a generalization of the Euclidean traveling salesman problem and is known to be NP-hard for all  $k \geq 3$  (see [5]). Unless explicitly mentioned, all CVRP instances in this paper are assumed to be unit-demand Euclidean.



© Claire Mathieu and Hang Zhou;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 43; pp. 43:1–43:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**ITP Algorithm.** The best known polynomial-time approximation for the CVRP is a very simple algorithm, called *iterated tour partitioning (ITP)*. This algorithm first computes a traveling salesman tour (ignoring the capacity constraint) using some other algorithm, then partitions the tour into segments such that the number of customers in each segment is at most  $k$ , and finally connects the endpoints of each segment to the depot so as to make a tour. The ITP algorithm was introduced and refined by Haimovich and Rinnooy Kan [15] and Altinkemer and Gavish [2] in the 1980s. Its performance is parameterized by the choice of traveling salesman tour: the approximation ratio of the ITP algorithm is  $1 + \alpha$ , where  $\alpha$  is the approximation ratio of the algorithm used to compute the traveling salesman tour in the first step. Since the Euclidean traveling salesman problem admits a *polynomial-time approximation scheme (PTAS)* by Arora [4] and Mitchell [20],  $\alpha$  can be set to any constant strictly greater than 1.

**Random Setting.** Given the difficult challenges posed by the CVRP, researchers turned to an analysis beyond worst case, by making some probabilistic assumptions on the distribution of the input instance. In 1985, Haimovich and Rinnooy Kan [15] gave the first probabilistic analysis on the ITP algorithm for the CVRP, where the customers are *independent, identically distributed (i.i.d.)* random points. An event  $\mathcal{E}$  occurs *asymptotically almost surely (a.a.s.)* if  $\lim_{n \rightarrow \infty} \mathbb{P}[\mathcal{E}] = 1$ . They showed that, the ITP algorithm is a.a.s. an  $(\alpha + o(1))$ -approximation for the CVRP when  $k$  is either  $o(\sqrt{n})$  or  $\omega(\sqrt{n})$ .<sup>1</sup> The performance of the ITP algorithm in the intermediate range of  $k = \Theta(\sqrt{n})$  was unknown. They asked in [15] whether the ITP algorithm was “also effective in the intermediate range”.

In our work, we study this question raised by Haimovich and Rinnooy Kan [15]. We give a probabilistic analysis of the ITP algorithm when the points are i.i.d. random, with a focus on the range of  $k = \Theta(\sqrt{n})$ . Our first main result is a lower bound: even in the random setting, the ITP algorithm is at best a  $(1 + c_0)$ -approximation a.a.s., for some constant  $c_0 > 0$  (Theorem 1), see Section 3.

► **Theorem 1.** *Consider the iterated tour partitioning algorithm for the unit-demand Euclidean capacitated vehicle routing problem. Let  $V$  be a set of  $n$  i.i.d. uniform random points in  $[0, 1]^2$ . Let  $k = \sqrt{n}$ . For some fixed depot  $O \in \mathbb{R}^2$ , there exists a constant  $c_0 > 0$ , such that, for any constant  $\alpha > 1$ , there exists an  $\alpha$ -approximate traveling salesman tour on  $V \cup \{O\}$ , such that the approximation ratio of the algorithm is at least  $1 + c_0$  asymptotically almost surely.*

► **Remark.** The  $\alpha$ -approximate traveling salesman tour in Theorem 1 is constructed using Karp’s partitioning algorithm [16].

On the other hand, the approximation ratio of the ITP algorithm is at most  $1 + \alpha$  due to Altinkemer and Gavish [2]. In 2007, this ratio was improved by Bompadre, Dror, and Orlin [10] to  $0.995 + \alpha$ , a.a.s., when the points are i.i.d. uniform random in the unit square.<sup>2</sup> Here, using a different approach (Theorem 4), we further improve the upper bound on the approximation ratio in this random setting to  $0.915 + \alpha$ , a.a.s. (Theorem 2), see Section 4. We generalize our results to multiple depots in the full version of the paper.

<sup>1</sup> As observed in [15], a solution of the ITP algorithm consists of two types of costs: the *radial cost* and the *local cost*. When  $k$  is  $o(\sqrt{n})$  or  $\omega(\sqrt{n})$ , one of the two types dominates, the reason for which the solution is an  $(\alpha + o(1))$ -approximation (or even a  $(1 + o(1))$ -approximation for the case of  $k = o(\sqrt{n})$ ).

<sup>2</sup> The analysis in [10] focused on the case of  $\alpha = 1$ , though that analysis can be easily generalized to any  $\alpha \geq 1$ . Bompadre, Dror, and Orlin [10] noted in their work that a ratio of  $0.985 + \alpha$  is achievable without giving the proof.

► **Theorem 2.** *Consider the iterated tour partitioning algorithm for the unit-demand Euclidean capacitated vehicle routing problem. Let  $V$  be a set of  $n$  i.i.d. uniform random points in  $[0, 1]^2$ . Let  $k$  be any integer in  $[1, n]$ . Let the depot  $O$  be any point in  $\mathbb{R}^2$ . For any constant  $\alpha \geq 1$  and any  $\alpha$ -approximate traveling salesman tour on  $V \cup \{O\}$ , the approximation ratio of the algorithm is at most  $0.915 + \alpha$  asymptotically almost surely.*

## 1.1 Other Related Work

**PTAS and Quasi-PTAS Results for the CVRP.** Despite the difficulty of the CVRP, there has been progress on several special cases. A series of papers designed PTAS algorithms for small  $k$ : work by Haimovich and Rinnooy Kan [15], when  $k$  is constant; by Asano et al. [5] extending techniques in [15], for  $k = O(\log n / \log \log n)$ ; and by Adamaszek, Czumaj, and Lingas [1], when  $k \leq 2^{\log^{f(\epsilon)}(n)}$ . For higher dimensional Euclidean metrics, Khachay and Dubinin [17] gave a PTAS for fixed dimension  $\ell$  and  $k = O(\log^{\frac{1}{\ell}}(n))$ . For unbounded  $k$ , Das and Mathieu [13] designed a quasi-polynomial time approximation scheme.

**Probabilistic Analyses.** The instance distribution when the customers are i.i.d. random points is perhaps the most natural probabilistic setting. In that setting, Rhee [23] and Daganzo [12] analyzed the value of an optimal solution to the CVRP for the case when  $k$  is fixed. Baltz et al [6] studied the multiple depot vehicle routing problem when both the customers and the depots are i.i.d. random points and assuming unlimited tour capacity.

**Analyses of the ITP Algorithm.** Because of the popularity of the ITP algorithm, its approximation ratio has already been much studied and bounds were utilized in a design of best-to-date approximation algorithms for the CVRP, see, e.g., [9]. In the metric version of the CVRP, the approximation ratio of the ITP algorithm is at most  $1 + (1 - \frac{1}{k})\alpha$  due to Altinkemer and Gavish [2]. Bompadre, Dror, and Orlin [9] reduced this bound by a factor of  $\Omega(\frac{1}{k^3})$ . On the other hand, Li and Simchi-Levi [18] showed that the ITP algorithm is at best a  $(2 - \frac{1}{k})$ -approximation algorithm on general metrics even if  $\alpha = 1$ . Despite of a huge amount of research, the ITP algorithm by Haimovich and Rinnooy Kan [15] and Altinkemer and Gavish [2] remains the polynomial-time algorithm with the best approximation guarantee for the Euclidean CVRP.

**Other Applications of the ITP Algorithm.** Very recently, Blauth, Traub, and Vygen [8] exploited properties of tight instances in the analysis of the ITP algorithm, and used those properties in their design of the best-to-date approximation algorithm for metric CVRP with a ratio of  $1 + \alpha - \epsilon$ , where  $\epsilon$  is roughly  $\frac{1}{3000}$ .

Because of its simplicity, the ITP algorithm is versatile and has been adapted to other vehicle routing problems. For example, Mosheiov [21] studied the vehicle routing with pick-up and delivery services. They showed that the ITP algorithm is efficient through worst-case analysis and numerical tests. Li, Simchi-Levi, and Desrochers [19] considered the vehicle routing problem with constraints on the total distance traveled by each vehicle. They showed that the ITP algorithm has a good worst-case performance when the number of vehicles is relatively small.

## 1.2 Overview of Techniques

To show that the ITP algorithm is at best a  $(1 + c_0)$ -approximation for the CVRP in the random setting (Theorem 1), we construct a significantly better solution.

In the random setting, one may view an ITP solution as partitioning the unit square into small regions and dedicating one tour to each small region. The cost of the solution is then roughly the sum of two terms: the *radial cost*, incurred by traveling between the depot and the small region; and the *local cost*, incurred by traveling from customer to customer within the small region.

To improve that solution, the idea is that instead of traveling straight between the depot and the small region, a smarter tour might as well make some small detours to visit some additional nearby customers en route to the small region. We call that a *mixed tour*. This modification of the solution has a positive effect because those nearby customers are covered at little additional cost, thus saving the local cost of covering those customers; but it also has a negative effect because visiting those nearby customers uses up some of the tour's capacity, and to account for that the definition of the small regions must be adjusted, and their area shrunk. Controlling the two competing effects so that on balance the net result is an improvement requires a delicate definition of regions. We start by decomposing the plane into regions of three types. Then we construct a solution in which a single tour may visit regions of different types, see Figure 1. The mixed structure of the tours enables us to show that the constructed solution has significantly smaller cost. See Section 3 for more details.

Our proof of the improved upper bound on the approximation ratio of the ITP algorithm in the random setting (Theorem 2) relies on a new lower bound on the optimal cost (Theorem 11). To achieve the new lower bound, we consider the gap between the average distance to the depot and the maximum distance to the depot among all points in a single tour of an optimal solution. Intuitively, if this gap is large, then the gap itself contributes to the lower bound on the optimal cost; and if this gap is small, then there are many points whose distances to the depot is close to the maximum distance, and the total local cost of those points contributes to the lower bound. Our analysis for the lower bound is completely different from [10] and enables us to obtain a better approximation ratio of  $0.915 + \alpha$ .

Our new lower bound on the optimal cost also enables us to generalize our results to the setting of multiple depots. This lower bound holds in the metric CVRP in general, and may be of independent interest.

► **Remark.** The restriction to i.i.d. uniform random points in  $[0, 1]^2$  is made to simplify the presentation. With extra work, our analysis can be extended to higher dimensional Euclidean spaces, to general density functions, and to general bounded supports (though the approximation guarantees in those settings may differ from that in Theorem 2).

## 2 Notations and Preliminaries

Let  $\delta(\cdot, \cdot)$  denote the Euclidean distance between two points or between a point and a set of points. For any path  $P$  of points  $x_1, x_2, \dots, x_m$  in  $\mathbb{R}^2$  where  $m \in \mathbb{N}$ , define  $\text{cost}(P) = \sum_{i=1}^{m-1} \delta(x_i, x_{i+1})$ .

**Capacitated Vehicle Routing Problem (CVRP).** Given a set  $V$  of  $n$  points in  $\mathbb{R}^2$ , a depot  $O$  in  $\mathbb{R}^2$ , and an integer capacity  $k \in [1, n]$ , the goal is to find a collection of tours covering  $V$  of minimum total cost, such that each tour visits  $O$  and at most  $k$  points in  $V$ . Let  $\text{OPT}$  denote the value of an optimal solution to the CVRP.

For any point  $x \in V$ , let  $\ell(x) = \delta(O, x)$ . Let  $\text{rad}$  denote the *radial cost*, defined by  $\text{rad} = \frac{2}{k} \cdot \sum_{x \in V} \ell(x)$ .

► **Lemma 3** ([15]). *Let  $T^*$  be an optimal traveling salesman tour on  $V \cup \{O\}$ . Then  $\text{OPT} \geq \max(\text{rad}, \text{cost}(T^*))$ .*

**Iterated Tour Partitioning (ITP).** We review the *iterated tour partitioning (ITP)* algorithm defined by Altinkemer and Gavish [2]. The ITP algorithm consists of a preprocessing phase and a main phase. In the preprocessing phase, it runs an approximation algorithm for the traveling salesman problem on  $V \cup \{O\}$ . Let  $\alpha$  denote the approximation ratio of this algorithm. Let  $T = (O, x_1, x_2, \dots, x_n, O)$  denote the resulting traveling salesman tour. In the main phase, the ITP algorithm selects the best of the  $k$  solutions constructed as follows. For each  $i \in [1, k]$ , let  $n_i = \lceil (n - i)/k \rceil + 1$  and define a solution  $S_i$  to the CVRP to be the union of the  $n_i$  tours  $(O, x_1, \dots, x_i, O)$ ,  $(O, x_{i+1}, \dots, x_{i+k}, O)$ ,  $(O, x_{i+k+1}, \dots, x_{i+2k}, O)$ ,  $\dots$ ,  $(O, x_{i+(n_i-2)k+1}, \dots, x_n, O)$ . In other words, the solution  $S_i$  partitions the traveling salesman tour  $T$  into segments with  $k$  points each, except possibly the first and the last segments. The output of the ITP algorithm is a solution among  $S_1, \dots, S_k$  that achieves the minimum cost. It is easy to see that the main phase of the ITP algorithm can be carried out in  $O(nk)$  time.<sup>3</sup>

Let  $\text{ITP}(T)$  denote the cost of the output solution. The following classic bound on  $\text{ITP}(T)$  was due to Altinkemer and Gavish [2] and, together with Lemma 3, immediately implies that the ITP algorithm is a  $(1 + \alpha)$ -approximation, where  $\alpha$  is the approximation ratio of the traveling salesman tour  $T$ .

► **Lemma 4** ([2]). *Let  $T$  be any traveling salesman tour on  $V \cup \{O\}$ . Then*

$$\text{OPT} \leq \text{ITP}(T) \leq \text{rad} + \left(1 - \frac{1}{k}\right) \cdot \text{cost}(T).$$

**Probabilistic Analysis of the Traveling Salesman Problem.** Beardwood, Halton, and Hammersley [7] analyzed the value of an optimal solution to the traveling salesman problem in the random setting.

► **Lemma 5** ([7, 24]). *Let  $V$  be a set of  $n$  i.i.d. uniform random points with bounded support in  $\mathbb{R}^2$ . Let  $M$  denote the measure of the support. Let  $T^*$  denote an optimal traveling salesman tour on  $V$ . Then there exists a universal constant  $\beta$  such that, for any  $\epsilon > 0$ , we have*

$$\lim_{n \rightarrow \infty} \frac{\text{cost}(T^*)}{\sqrt{M \cdot n}} = \beta, \quad \text{with probability 1.}$$

*In addition,  $\beta_0 < \beta < \beta_1$ , where  $\beta_0 = 0.62866$  and  $\beta_1 = 0.92117$ .*

► **Remark.** Up to scaling, Lemma 5 holds for any support that is a rectangle with constant aspect ratio.

### 3 Lower Bound on the Approximation Ratio

In this section, we prove Theorem 1 by providing a lower bound on the approximation ratio  $\text{ITP}(T)/\text{OPT}$  of the ITP algorithm, where  $T$  is a traveling salesman tour. Let the depot  $O = (\frac{1}{2}, -1000)$ . Lemmas 6 and 7 are the key ingredients in the proof of Theorem 1.

► **Lemma 6.** *Let  $\beta$  be defined as in Lemma 5. Then there exists a constant  $c_1 \in (0, \beta)$  such that for any  $\epsilon_1 > 0$ ,  $\text{OPT} < (1 + \epsilon_1)(\text{rad} + \beta\sqrt{n}) - c_1\sqrt{n}$ , a.a.s.*

► **Lemma 7.** *Let  $\beta$  be defined as in Lemma 5. Then for any  $\alpha > 1$ , there exists an  $\alpha$ -approximate traveling salesman tour  $T$  on  $V \cup \{O\}$ , such that for any  $\epsilon_1 > 0$ ,  $\text{ITP}(T) > (1 - \epsilon_1)(\text{rad} + \beta\sqrt{n})$ , a.a.s.*

<sup>3</sup> The running time of the main phase can even be improved to  $O(n)$ .

Lemma 6 contains main novelties in this section. The proof of Lemma 7 is in the full version of the paper. First, we show how Lemmas 6 and 7 imply Theorem 1.

**Proof of Theorem 1 using Lemmas 6 and 7.** Let  $\epsilon_1 > 0$  be a constant to be set later. From Lemma 6, there exists an absolute constant  $c_1 \in (0, \beta)$ , such that  $\text{OPT} < (1 + \epsilon_1)(\text{rad} + \beta\sqrt{n}) - c_1\sqrt{n}$ , a.a.s. From Lemma 7, for any  $\alpha > 1$ , there exists an  $\alpha$ -approximate traveling salesman tour  $T$  on  $V \cup \{O\}$ , such that  $\text{ITP}(T) > (1 - \epsilon_1)(\text{rad} + \beta\sqrt{n})$ , a.a.s. Hence

$$\frac{\text{ITP}(T)}{\text{OPT}} > \frac{(1 - \epsilon_1)(\text{rad} + \beta\sqrt{n})}{(1 + \epsilon_1)(\text{rad} + \beta\sqrt{n}) - c_1\sqrt{n}}, \quad \text{a.a.s.}$$

To analyze  $\text{rad}$ , let  $L$  be the expectation of  $\ell(x)$  for  $x \in [0, 1]^2$  with uniform distribution. Since the depot  $O$  is at a constant distance from  $[0, 1]^2$ ,  $L$  is a constant. By the law of large numbers,

$$\left| \frac{1}{n} \sum_{x \in V} \ell(x) - L \right| < \epsilon_1, \quad \text{a.a.s.}$$

Recall that  $\text{rad} = \frac{2}{\sqrt{n}} \sum_{x \in V} \ell(x)$ . Hence  $(L - \epsilon_1) \cdot 2\sqrt{n} < \text{rad} < (L + \epsilon_1) \cdot 2\sqrt{n}$ , a.a.s. Denoting the function  $f$  as

$$f(\epsilon_1) = \frac{(1 - \epsilon_1)(2L - 2\epsilon_1 + \beta)}{(1 + \epsilon_1)(2L + 2\epsilon_1 + \beta) - c_1},$$

we have

$$\frac{\text{ITP}(T)}{\text{OPT}} > f(\epsilon_1), \quad \text{a.a.s.}$$

Since  $L$ ,  $\beta$  and  $c_1$  are positive constants and  $c_1 < \beta$  (Lemma 6), we have

$$\lim_{\epsilon_1 \rightarrow 0} f(\epsilon_1) = 1 + \frac{c_1}{2L + \beta - c_1}.$$

Let  $c_0 = \frac{1}{2} \cdot \frac{c_1}{2L + \beta - c_1}$ , which is a positive constant. Choosing  $\epsilon_1$  small enough such that  $f(\epsilon_1) > 1 + c_0$ , we have

$$\frac{\text{ITP}(T)}{\text{OPT}} > f(\epsilon_1) > 1 + c_0, \quad \text{a.a.s.}$$

The claim follows. ◀

The rest of the section is dedicated to prove Lemma 6.

Without loss of generality, we assume that  $\epsilon_1 \leq 1$ , since otherwise it suffices to prove the claim for the case of  $\epsilon_1 = 1$ . We construct a solution to the CVRP whose cost is less than  $(1 + \epsilon_1)(\text{rad} + \beta\sqrt{n}) - c_1\sqrt{n}$ , a.a.s., where the constant  $c_1 > 0$  will be chosen at the end of the proof.

### 3.1 Decomposition of the Plane

In order to construct a solution to the CVRP, we describe a decomposition of  $[0, 1]^2$  into rectangles of three types. Let  $\epsilon_2 = \frac{\epsilon_1}{10}$ . We partition<sup>4</sup>  $[0, 1]^2$  into a lower part  $[0, 1] \times [0, \frac{3+\epsilon_2}{4}]$  which is a rectangle of type III, and a collection of *boxes* of the form  $[(i-1)D, iD] \times [\frac{3+\epsilon_2}{4}, 1]$ , with  $D = n^{-1/4}$  and  $1 \leq i \leq 1/D$ . For simplicity, assume that  $1/D$  is an integer. See Figure 1a.

---

<sup>4</sup> The decomposition is a partition except for the boundaries, that have measure 0.



Next, we decompose each box, see Figure 1b. Let  $m = \frac{5}{40-\beta} \cdot n^{1/4}$ . For simplicity, assume that  $m$  is an integer. The upper half of a box is partitioned into  $m$  *type I rectangles* of the form  $[(i-1)D, iD] \times [1 - (j-1)H, 1 - jH]$ , where  $H = \frac{1-\epsilon_2}{8 \cdot m}$  and  $1 \leq j \leq m$ . The lower left part of a box is partitioned into  $2m$  slices such that each slice is a *type II rectangles* of the form  $[(i-1)D + (j-1)W, (i-1)D + jW] \times [\frac{3+\epsilon_2}{4}, \frac{7+\epsilon_2}{8}]$ , with  $W = \frac{\beta}{10} \cdot n^{-1/2}$  and  $1 \leq j \leq 2m$ . The rest of the box is a single *type III rectangle*.

For any rectangle  $R$  in the resulting decomposition, let  $n_R$  denote the number of points of  $V$  that are in the rectangle  $R$ , and let  $M_R$  denote the measure of the rectangle  $R$ . The following fact relates  $n_R$  with  $M_R$ .

► **Fact 8.** *A.a.s. the following event  $\mathcal{E}$  occurs:  $(1 - \epsilon_2) \cdot M_R \cdot n < n_R < (1 + \epsilon_2) \cdot M_R \cdot n$  for all rectangles  $R$  in the resulting decomposition.*

**Proof.** Let  $R$  be any rectangle in the resulting decomposition. Observe that  $M_R = \Omega(1/\sqrt{n})$ . The expectation of  $n_R$  is  $M_R \cdot n = \Omega(\sqrt{n})$ . By Chernoff bound,

$$\mathbb{P}[n_R \leq (1 - \epsilon_2)M_R \cdot n] \leq e^{-\Omega(\sqrt{n})} \quad \text{and} \quad \mathbb{P}[n_R \geq (1 + \epsilon_2)M_R \cdot n] \leq e^{-\Omega(\sqrt{n})}.$$

Since there are  $\Theta(\sqrt{n})$  rectangles in the decomposition, the event  $\mathcal{E}$  occurs with probability at least  $1 - \Theta(\sqrt{n}) \cdot e^{-\Omega(\sqrt{n})} = 1 - o(1)$ . ◀

From now on, we condition on the occurrence of  $\mathcal{E}$  in Fact 8.

### 3.2 Construction of a Solution

To construct a relative cheap solution, the main observation is that it is profitable for a tour to visit points in rectangles of both types I and II. In each box, there are  $m$  rectangles of type I and  $2m$  rectangles of type II. We form  $m$  groups with those, such that each group contains one rectangle of type I and two rectangles of type II. For each group, we cover the points in the group by a particular tour on these points in addition to the depot  $O$ , in a way to be described shortly. For all points in the rectangles of type III, we construct an optimal solution to the CVRP on those points with depot  $O$  and with capacity  $\sqrt{n}$ .

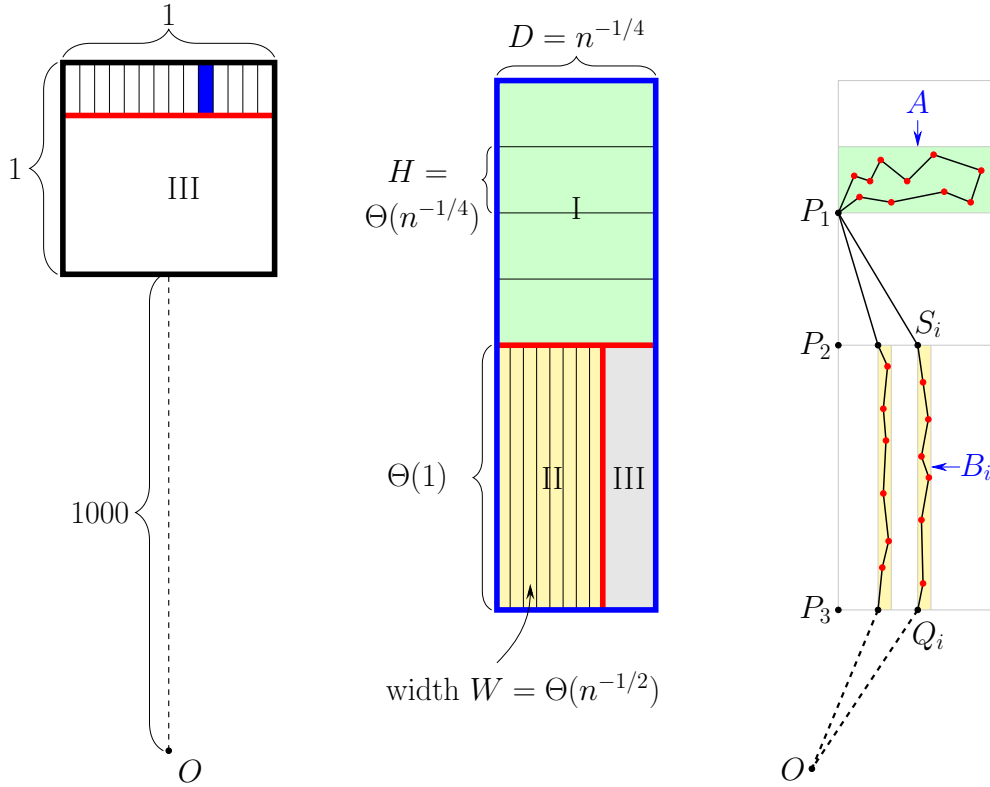
A *mixed tour* is a tour covering points in rectangles of types I and II. Consider a box  $\mathcal{B}$  and a group in  $\mathcal{B}$  consisting of a rectangle  $A$  of type I and two rectangles  $B_1$  and  $B_2$  of type II. We construct a specific mixed tour  $T_{\text{mix}}$  defined as follows.

Let  $P_1$  denote the bottom left corner of the rectangle  $A$ . Let  $T_0$  denote an optimal traveling salesman tour on the points in  $A \cup \{P_1\}$ . For each  $i \in \{1, 2\}$ , we define an  $O$ -to- $P_1$  path  $T_i$  visiting the points in  $B_i$  as follows. Let  $S_i$  and  $Q_i$  denote the top left and bottom left corners of the rectangle  $B_i$ . Let  $T_i$  denote the concatenation of the segment  $OQ_i$ , a  $Q_i$ -to- $S_i$  path visiting the points in  $B_i$  in non-decreasing order on the  $y$ -coordinate (breaking ties arbitrarily), and the segment  $S_iP_1$ . Finally, the tour  $T_{\text{mix}}$  is defined as the concatenation of  $T_0$ ,  $T_1$ , and  $T_2$ . See Figure 1c. This completes our construction.

Since the measure of  $A$  is  $D \cdot H$  and the measure of  $B_i$  (for each  $i \in \{1, 2\}$ ) is  $\frac{1-\epsilon_2}{8} \cdot W$ , Event  $\mathcal{E}$  implies that the total number of points in  $A \cup B_1 \cup B_2$  is at most

$$(1 + \epsilon_2) \left( D \cdot H + 2 \cdot \frac{1 - \epsilon_2}{8} \cdot W \right) \cdot n = (1 + \epsilon_2)(1 - \epsilon_2)n^{-1/2} \cdot n < \sqrt{n},$$

so the constructed solution is feasible.



(a)  $[0, 1]^2$  decomposition. (b) Box decomposition. (c) A mixed tour.

■ **Figure 1** Decomposition and tour construction. Figure 1a illustrates the decomposition of  $[0, 1]^2$ . The highlighted area in Figure 1a represents a box. Figure 1b illustrates the decomposition of a box into rectangles of types I, II, and III. Figure 1c describes a tour covering points in one rectangle  $A$  of type I and in two rectangles  $B_i$ , for  $i \in \{1, 2\}$ , of type II.

### 3.3 Cost of a Mixed Tour

Consider any mixed tour  $T_{\text{mix}}$ . We follow the same notations as in Section 3.2. From the construction,

$$\text{cost}(T_{\text{mix}}) = \text{cost}(T_0) + \text{cost}(T_1) + \text{cost}(T_2). \tag{1}$$

Let  $T_A^*$  denote an optimal traveling salesman tour on the points in  $A$ . The cost of  $T_0$  is at most  $\text{cost}(T_A^*)$  plus the cost of the detour to include the point  $P_1$ . The cost of the detour is less than  $2(D + H)$ , so

$$\text{cost}(T_0) < \text{cost}(T_A^*) + 2(D + H). \tag{2}$$

Let  $n_1$  and  $n_2$  denote the number of points of  $V$  that are in  $B_1$  and  $B_2$ , respectively. The costs of  $T_1$  and  $T_2$  are bounded by the following fact, whose proof is in the full version of the paper.

► **Fact 9.** For  $i \in \{1, 2\}$ , we have

$$\text{cost}(T_i) < \delta(O, P_1) + \frac{1}{4000} + n_i W + (W + 2D). \tag{3}$$

From Equations (1)–(3), and using the definition of  $W$ , we have

$$\text{cost}(T_{\text{mix}}) < \text{cost}(T_A^*) + 2\delta(O, P_1) + \frac{1}{2000} + \frac{\beta}{10} \cdot \frac{n_1 + n_2}{\sqrt{n}} + (2W + 6D + 2H). \quad (4)$$

It remains to bound  $\delta(O, P_1)$ . Observe that by the definition of  $\ell(\cdot)$  and the triangle inequality, and since the height of a box  $\mathcal{B}$  is less than  $\frac{1}{4}$ ,

$$\delta(O, P_1) < \begin{cases} \ell(x) + D + H & \text{for any } x \in A, \\ \ell(x) + \frac{1}{4} + D + H & \text{for any } x \in B_1 \cup B_2. \end{cases}$$

Let  $V_{\text{mix}}$  denote the set of the points of  $V$  in  $A \cup B_1 \cup B_2$ . By averaging we have

$$\delta(O, P_1) < \frac{1}{|V_{\text{mix}}|} \left( \sum_{x \in V_{\text{mix}}} \ell(x) \right) + \frac{1}{|V_{\text{mix}}|} \frac{n_1 + n_2}{4} + (D + H).$$

Since the measure of  $A \cup B_1 \cup B_2$  is  $(1 - \epsilon_2)n^{1/2}$ , Event  $\mathcal{E}$  implies that  $|V_{\text{mix}}| > (1 - \epsilon_2)^2 \cdot \sqrt{n}$ , which is at least  $\frac{\sqrt{n}}{1 + \epsilon_1}$  since  $\epsilon_2 = \frac{\epsilon_1}{10}$ . Hence

$$\delta(O, P_1) < \frac{1 + \epsilon_1}{\sqrt{n}} \left[ \left( \sum_{x \in V_{\text{mix}}} \ell(x) \right) + \frac{n_1 + n_2}{4} \right] + (D + H). \quad (5)$$

Since  $n_1 + n_2 = \Theta(\sqrt{n})$ , we have  $(2W + 6D + 2H) + 2(D + H) < \frac{\epsilon_1}{\sqrt{n}} \cdot \frac{\beta}{10} \cdot (n_1 + n_2)$  when  $n$  is large enough. From Equations (4) and (5), we conclude that

$$\text{cost}(T_{\text{mix}}) < \text{cost}(T_A^*) + \frac{1 + \epsilon_1}{\sqrt{n}} \left[ 2 \left( \sum_{x \in V_{\text{mix}}} \ell(x) \right) + \left( \frac{\beta}{10} + \frac{1}{2} \right) (n_1 + n_2) \right] + \frac{1}{2000}. \quad (6)$$

### 3.4 Cost of the Solution

#### 3.4.1 Solution in the Rectangles of Types I and II

Let  $V_I$  and  $V_{II}$  denote the subsets of the points of  $V$  that are in the rectangles of type I and type II, respectively. Let  $K$  denote the number of mixed tours. We have  $K = n^{1/4} \cdot m = \frac{5}{40 - \beta} \cdot \sqrt{n}$ . Applying Equation (6) on each mixed tour and summing, we have

$$\text{cost}(\mathcal{S}_{\text{mix}}) \leq Y + \frac{1 + \epsilon_1}{\sqrt{n}} \left[ 2 \left( \sum_{x \in V_I \cup V_{II}} \ell(x) \right) + \left( \frac{\beta}{10} + \frac{1}{2} \right) |V_{II}| \right] + \frac{\sqrt{n}}{400 \cdot (40 - \beta)}, \quad (7)$$

where  $Y$  denotes the overall cost of  $T_A^*$  over all rectangles  $A$  of type I.

To analyze  $Y$ , we consider any rectangle  $A$  of type I. The measure of  $A$  is  $M_A = D \cdot H = (1 - \epsilon_2) \left(1 - \frac{\beta}{40}\right) \frac{1}{\sqrt{n}}$ . The event  $\mathcal{E}$  implies that

$$(1 - \epsilon_2)^2 \cdot \left(1 - \frac{\beta}{40}\right) \cdot \sqrt{n} < n_A < \left(1 - \frac{\beta}{40}\right) \cdot \sqrt{n}. \quad (8)$$

We investigate the expectation of  $\text{cost}(T_A^*)$ . By a construction given in [7], there exists a constant  $C$  such that the cost of an optimal traveling salesman tour through any  $n_A$  points in  $A$  is at most  $C\sqrt{M_A \cdot n_A}$ . Together with Lemma 5, it follows that

$$\mathbb{E}[\text{cost}(T_A^*)] < (1 + \epsilon_2) \cdot \beta \sqrt{M_A \cdot n_A} < (1 + \epsilon_2) \cdot \beta \left(1 - \frac{\beta}{40}\right),$$

### 43:10 Probabilistic Analysis of Euclidean Capacitated Vehicle Routing

for  $n$  large enough (and thus  $n_A$  large enough). Since there are  $K = \Theta(\sqrt{n})$  rectangles  $A$  of type I, by the law of large numbers,

$$Y = \sum_A \text{cost}(T_A^*) < (1 + \epsilon_2) \cdot K \cdot (1 + \epsilon_2) \cdot \beta \left(1 - \frac{\beta}{40}\right), \quad \text{a.a.s.}$$

On the other hand, summing Equation (8) over all  $A$ , we have

$$|V_I| = \sum_A n_A > K \cdot (1 - \epsilon_2)^2 \cdot \left(1 - \frac{\beta}{40}\right) \cdot \sqrt{n}.$$

Therefore,

$$Y < (1 + \epsilon_2)^2 \cdot \frac{1}{(1 - \epsilon_2)^2} \cdot \frac{\beta \cdot |V_I|}{\sqrt{n}} < (1 + \epsilon_1) \cdot \frac{\beta \cdot |V_I|}{\sqrt{n}}, \quad \text{a.a.s.},$$

where the last inequality follows since  $\epsilon_2 = \frac{\epsilon_1}{10}$ . From Equation (7), we conclude that, a.a.s.,

$$\text{cost}(\mathcal{S}_{\text{mix}}) \leq \frac{1 + \epsilon_1}{\sqrt{n}} \left[ 2 \left( \sum_{x \in V_I \cup V_{II}} \ell(x) \right) + \beta \cdot |V_I| + \left( \frac{\beta}{10} + \frac{1}{2} \right) |V_{II}| \right] + \frac{\sqrt{n}}{400 \cdot (40 - \beta)}. \quad (9)$$

#### 3.4.2 Solution in the Rectangles of Type III

Let  $\hat{V}$  denote the subset of the points of  $V$  that are in the rectangles of types III. Let  $\hat{T}$  denote an optimal traveling salesman tour on  $\hat{V} \cup \{O\}$ . Let  $\hat{\mathcal{S}}$  denote an optimal solution to the CVRP on  $\hat{V}$  with depot  $O$  and with capacity  $k = \sqrt{n}$ . By Lemma 4,

$$\text{cost}(\hat{\mathcal{S}}) \leq \frac{2}{\sqrt{n}} \left( \sum_{x \in \hat{V}} \ell(x) \right) + \text{cost}(\hat{T}).$$

The cost of  $\hat{T}$  is at most the cost  $C_{\text{TSP}}$  of an optimal traveling salesman tour on  $\hat{V}$  plus the detour to visit  $O$ . Since the distance between the depot and any point in  $[0, 1]^2$  is  $O(1)$ , the cost of the detour is  $O(1)$ , so  $\text{cost}(\hat{T}) \leq C_{\text{TSP}} + O(1)$ .

Next, we analyze  $C_{\text{TSP}}$ . Let  $L$  denote the width of a type III rectangle inside a box. Then  $L = D - W \cdot m = \Theta(n^{-1/4})$ . We observe that the length of a side of any type III rectangle is either  $L$  or  $\omega(L)$ . So we partition every type III rectangle into squares of side length  $L$ . For each square, consider an optimal traveling salesman tour on the points inside that square. Let  $Z$  denote the overall cost of the optimal traveling salesman tours inside all squares from all rectangles of type III. Then  $C_{\text{TSP}}$  is at most  $Z$  plus the total lengths of the boundaries of all squares. Using the same argument from Section 3.4.1, we have

$$Z < \left(1 + \frac{\epsilon_1}{3}\right) \cdot \frac{\beta \cdot |\hat{V}|}{\sqrt{n}}, \quad \text{a.a.s.}$$

Since the boundary length of each square is negligible compared with the TSP cost inside that square, we have

$$C_{\text{TSP}} = (1 + o(1)) \cdot Z < \left(1 + \frac{\epsilon_1}{2}\right) \cdot \frac{\beta \cdot |\hat{V}|}{\sqrt{n}}, \quad \text{a.a.s.}$$

Noting that  $|\hat{V}| = \Theta(n)$ , we have

$$\text{cost}(\hat{T}) \leq C_{\text{TSP}} + O(1) < (1 + \epsilon_1) \cdot \frac{\beta \cdot |\hat{V}|}{\sqrt{n}}, \quad \text{a.a.s.}$$

Therefore,

$$\text{cost}(\hat{\mathcal{S}}) \leq \frac{2}{\sqrt{n}} \left( \sum_{x \in \hat{V}} \ell(x) \right) + \frac{1 + \epsilon_1}{\sqrt{n}} \cdot \beta \cdot |\hat{V}|, \quad \text{a.a.s.} \quad (10)$$

### 3.4.3 The Global Solution

Let  $\mathcal{S} = \mathcal{S}_{\text{mix}} \cup \hat{\mathcal{S}}$  denote the global solution. From Equations (9) and (10), and using  $\frac{2}{\sqrt{n}} \cdot \sum_{x \in V} \ell(x) = \text{rad}$  and  $V_I \cup V_{II} \cup \hat{V} = V$ , we have

$$\text{cost}(\mathcal{S}) \leq (1 + \epsilon_1) \left[ \text{rad} + \beta \cdot \frac{|V|}{\sqrt{n}} - \left( \beta - \frac{\beta}{10} - \frac{1}{2} \right) \frac{|V_{II}|}{\sqrt{n}} \right] + \frac{\sqrt{n}}{400 \cdot (40 - \beta)}, \quad \text{a.a.s.} \quad (11)$$

Observe that the rectangles of type II have an overall measure of  $(1 - \epsilon_2) \cdot \frac{\beta}{8 \cdot (40 - \beta)}$ . Event  $\mathcal{E}$  implies that  $|V_{II}| > (1 - \epsilon_2)^2 \cdot \frac{\beta \cdot n}{8 \cdot (40 - \beta)} > \frac{1}{1 + \epsilon_1} \cdot \frac{\beta \cdot n}{8 \cdot (40 - \beta)}$  since  $\epsilon_2 = \frac{\epsilon_1}{10}$ . By Lemma 5,  $\beta > \beta_0 = 0.62866$ . Thus  $\frac{9\beta}{10} - \frac{1}{2} > 0$ . From Equation (11), we have, a.a.s.,

$$\begin{aligned} (1 + \epsilon_1)(\text{rad} + \beta\sqrt{n}) - \text{cost}(\mathcal{S}) &\geq (1 + \epsilon_1) \cdot \left( \frac{9\beta}{10} - \frac{1}{2} \right) \cdot \frac{|V_{II}|}{\sqrt{n}} - \frac{\sqrt{n}}{400 \cdot (40 - \beta)} \\ &> \left( \frac{9\beta}{10} - \frac{1}{2} \right) \cdot \frac{\beta\sqrt{n}}{8 \cdot (40 - \beta)} - \frac{\sqrt{n}}{400 \cdot (40 - \beta)} \\ &= \frac{\sqrt{n}}{8 \cdot (40 - \beta)} \cdot \left( \left( \frac{9\beta}{10} - \frac{1}{2} \right) \cdot \beta - \frac{1}{50} \right) \\ &> \frac{\sqrt{n}}{8 \cdot (40 - \beta_0)} \cdot \left( \left( \frac{9\beta_0}{10} - \frac{1}{2} \right) \cdot \beta_0 - \frac{1}{50} \right). \end{aligned}$$

Let  $c_1$  denote the leading constant in the above bound, i.e.,

$$c_1 = \frac{1}{8 \cdot (40 - \beta_0)} \cdot \left( \left( \frac{9\beta_0}{10} - \frac{1}{2} \right) \cdot \beta_0 - \frac{1}{50} \right).$$

The value of  $c_1$  is roughly 0.000068. We conclude that

$$\text{cost}(\mathcal{S}) < (1 + \epsilon_1)(\text{rad} + \beta\sqrt{n}) - c_1\sqrt{n}, \quad \text{a.a.s.}$$

We complete the proof of Lemma 6.

## 4 Upper Bound on the Approximation Ratio

In this section, we prove Theorem 2 by providing an upper bound on the approximation ratio  $\text{ITP}(T)/\text{OPT}$  of the ITP algorithm, where  $T$  is a traveling salesman tour.

Let  $\lambda$  and  $\epsilon$  be positive constants such that  $\lambda + \epsilon < 1$ . We analyze the performance of the ITP algorithm with respect to  $\lambda$  and  $\epsilon$ . The values of  $\lambda$  and  $\epsilon$  will be set in the end of the proof.

### 4.1 Structural analysis

► **Lemma 10.** *Let  $T = (O, y_1, y_2, \dots, y_m, O)$  be any tour starting and ending at  $O$ . Let  $L = \frac{1}{m} \left( \sum_{j=1}^m \ell(y_j) \right)$ . Let  $\Delta = \max_{1 \leq j \leq m} \{\ell(y_j)\} - L$ . Then there exists a set  $W \subseteq \{y_1, \dots, y_m\}$  which is of cardinality greater than  $(\lambda + \epsilon) \cdot m - 1$  such that*

$$\text{cost}(T) \geq 2 \left( L - \frac{\lambda + \epsilon}{1 - \lambda - \epsilon} \cdot \Delta \right) + \sum_{x \in W} \delta(x, W \setminus \{x\}).$$

## 43:12 Probabilistic Analysis of Euclidean Capacitated Vehicle Routing

**Proof.** Let  $W$  denote the set of points  $x$  such that  $\ell(x) \geq L - \frac{\lambda+\epsilon}{1-\lambda-\epsilon} \cdot \Delta$ , other than the last one in the order of traversal by  $T$  starting from  $O$ . Tour  $T$  must first travel through a path to a first point of  $W$ , paying at least  $L - \frac{\lambda+\epsilon}{1-\lambda-\epsilon} \cdot \Delta$ , then proceed from each point  $x$  of  $W$  through a path to another point of  $W$ , paying at least  $\delta(x, W \setminus \{x\})$ , and finally, go to one more point such that  $\ell(x) \geq L - \frac{\lambda+\epsilon}{1-\lambda-\epsilon} \cdot \Delta$ , and travel from there through a path back to the depot, paying at least  $L - \frac{\lambda+\epsilon}{1-\lambda-\epsilon} \cdot \Delta$ . Hence the cost of  $T$  is at least as stated in Lemma 10.

Next, we bound the size of  $W$ . When  $\Delta = 0$ , we have  $\ell(y_j) = L$  for all  $j \in [1, m]$ . Hence  $|W| = m - 1$ , which is greater than  $(\lambda + \epsilon) \cdot m - 1$ , since  $\lambda + \epsilon < 1$ . The claim follows.

It remains to analyze the case when  $\Delta > 0$ . Every point of  $T$  is at distance at most  $L + \Delta$  from the depot. Letting  $m'$  denote the number of points whose distance from the depot is at least  $L - \frac{\lambda+\epsilon}{1-\lambda-\epsilon} \cdot \Delta$ , we have

$$mL = \sum_{j=1}^m \ell(y_j) < m'(L + \Delta) + (m - m') \left( L - \frac{\lambda + \epsilon}{1 - \lambda - \epsilon} \cdot \Delta \right).$$

Since  $1 - \lambda - \epsilon > 0$ , this implies  $m' - (\lambda + \epsilon) \cdot m > 0$ , hence  $|W| = m' - 1 > (\lambda + \epsilon) \cdot m - 1$ . ◀

The following result is a strengthening of the lower bound  $\text{OPT} \geq \text{rad}$  from Lemma 3, and will lead to our improved analysis of the ITP algorithm.

► **Theorem 11.** *Let  $\lambda$  and  $\epsilon$  be positive constants such that  $\lambda + \epsilon < 1$ . Let  $V$  be a set of  $n$  points in any distance metric. Let  $k = \omega(1)$ . There exists a set  $U \subseteq V$  which is of cardinality greater than  $(\lambda + \frac{\epsilon}{2}) \cdot n$  for  $n$  large enough, and such that*

$$\text{OPT} \geq \text{rad} + (1 - \lambda - \epsilon) \left( \sum_{x \in U} \delta(x, U \setminus \{x\}) \right).$$

**Proof.** Let  $T_1, \dots, T_q$  be the tours in an optimal solution to the CVRP. Let  $m_i$  be the number of points in  $V$  that are visited by the tour  $T_i$ . Up to combining tours that visit few points, we may assume that  $m_i > \frac{k}{2}$  for all but at most one tour, so  $q \leq \frac{2n}{k} + 1 = o(n)$ .

For each tour  $T_i$ , define the corresponding  $L_i$ ,  $\Delta_i$ , and  $W_i$  with respect to the tour  $T_i$  using the notations of Lemma 10. By summation, letting  $U = \bigcup_i W_i$ , Lemma 10 then implies (using  $q = o(n)$ ,  $n$  large enough, and  $\delta(x, W_i \setminus \{x\}) \geq \delta(x, U \setminus \{x\})$ )

$$|U| = \sum_{i \leq q} |W_i| > \left( (\lambda + \epsilon) \sum_i m_i \right) - q = (\lambda + \epsilon) \cdot n - q > \left( \lambda + \frac{\epsilon}{2} \right) \cdot n$$

and

$$\sum_i \text{cost}(T_i) \geq \left( \sum_i 2 \left( L_i - \frac{\lambda + \epsilon}{1 - \lambda - \epsilon} \cdot \Delta_i \right) \right) + \left( \sum_{x \in U} \delta(x, U \setminus \{x\}) \right). \quad (12)$$

On the other hand, we trivially have

$$\sum_i \text{cost}(T_i) \geq \sum_i 2(L_i + \Delta_i). \quad (13)$$

A linear combination of Equation (12) with coefficient  $(1 - \lambda - \epsilon)$  and of Equation (13) with coefficient  $(\lambda + \epsilon)$  leads to:

$$\text{OPT} = \sum_i \text{cost}(T_i) \geq \left( \sum_i 2L_i \right) + (1 - \lambda - \epsilon) \left( \sum_{x \in U} \delta(x, U \setminus \{x\}) \right).$$

Observe that

$$\sum_i 2L_i = \sum_i \sum_{x \in T_i} \frac{2\ell(x)}{m_i} \geq \sum_i \sum_{x \in T_i} \frac{2\ell(x)}{k} = \sum_{x \in V} \frac{2\ell(x)}{k} = \text{rad.}$$

The Lemma follows. ◀

## 4.2 Probabilistic Analysis

The following result suggests that the closest point distance follows the law of large numbers. It is a corollary of Theorem 2.4 in [22].<sup>5</sup>

► **Lemma 12** ([22]). *Let  $\mathcal{P}$  be a homogeneous Poisson point process of intensity 1 on  $\mathbb{R}^2$  and  $\delta_{\mathcal{P}}$  denote the distance from the origin of  $\mathbb{R}^2$  to a closest point in  $\mathcal{P}$  by the Euclidean norm. Let  $V$  be a set of  $n$  i.i.d. uniform random points. Then, given any bounded function  $\phi : [0, \infty) \rightarrow [0, \infty)$ , as  $n \rightarrow \infty$  we have:*

$$\frac{1}{n} \sum_{x \in V} \phi(\sqrt{n} \cdot \delta(x, V \setminus \{x\})) \rightarrow \mathbb{E}[\phi(\delta_{\mathcal{P}})].$$

Lemma 12 provides the rigorous setting enabling us to derive a new lower bound on the sum of the closest point distances over a subset of a set of i.i.d. uniform random points, which we now state.

► **Lemma 13.** *Let  $V$  be a set of  $n$  i.i.d. uniform random points. Let  $U$  be any subset of  $V$  such that  $|U| > (\lambda + \frac{\epsilon}{2}) \cdot n$ . Then, asymptotically almost surely,*

$$\sum_{x \in U} \delta(x, V \setminus \{x\}) > (\xi_{\lambda} - \epsilon) \cdot \sqrt{n},$$

where  $\xi_{\lambda}$  is a constant defined by

$$\xi_{\lambda} := \frac{1}{2} \operatorname{erf} \left( \sqrt{\ln \frac{1}{1-\lambda}} \right) - (1-\lambda) \cdot \sqrt{\frac{1}{\pi} \cdot \ln \frac{1}{1-\lambda}}$$

in which  $\operatorname{erf}(\cdot)$  is the Gauss error function  $\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$ .

**Proof.** Recall the definition of  $\delta_{\mathcal{P}}$  from Lemma 12. By definition of the Poisson point process, the probability  $g(r)$  of the event  $\delta_{\mathcal{P}} \leq r$  equals  $1 - e^{-\pi r^2}$ .

Let  $Z \subseteq V$  be the set of points  $x \in V$  such that  $\delta(x, V \setminus \{x\}) \leq \frac{r_0}{\sqrt{n}}$ , with  $r_0 = \sqrt{\frac{1}{\pi} \cdot \ln \frac{1}{1-\lambda}}$ . We apply Lemma 12 with  $\phi$  equals  $\phi_1$ , the indicator function of whether  $r \leq r_0$ , to obtain that, as  $n \rightarrow \infty$ ,

$$\frac{|Z|}{n} \rightarrow \mathbb{E}[\phi_1(\delta_{\mathcal{P}})] = g(r_0) = \lambda.$$

Thus  $|Z| \leq (\lambda + \frac{\epsilon}{2}) \cdot n < |U|$ , a.a.s. Since  $Z$  consists of the points  $x \in V$  with the smallest values of  $\delta(x, V \setminus \{x\})$ , we have

$$\sum_{x \in U} \delta(x, V \setminus \{x\}) \geq \sum_{x \in Z} \delta(x, V \setminus \{x\}), \quad \text{a.a.s.} \tag{14}$$

<sup>5</sup> To apply Theorem 2.4 in [22], we consider a *directed* graph  $G$  with vertex set  $V$ , such that from every vertex  $x \in V$ , there is a unique outgoing edge, let it be  $(x, y)$ , where  $y$  is the closest point to  $x$  among the points in  $V \setminus \{x\}$ , breaking ties arbitrarily. Theorem 2.4 in [22] is interpreted with reference to Remark (h) in [22].



### 43:14 Probabilistic Analysis of Euclidean Capacitated Vehicle Routing

To analyze  $\sum_{x \in Z} \delta(x, V \setminus \{x\})$ , we define a bounded function  $\phi_2$  as follows.

$$\phi_2(r) = \begin{cases} r, & r \leq r_0 \\ 0, & \text{otherwise.} \end{cases}$$

Applying Lemma 12 with  $\phi = \phi_2$ , we have, as  $n \rightarrow \infty$ ,

$$\frac{1}{n} \sum_{x \in Z} \sqrt{n} \cdot \delta(x, V \setminus \{x\}) \rightarrow \mathbb{E}[\phi_2(\delta_{\mathcal{P}})],$$

thus

$$\sum_{x \in Z} \delta(x, V \setminus \{x\}) > (\mathbb{E}[\phi_2(\delta_{\mathcal{P}})] - \epsilon) \cdot \sqrt{n}, \quad \text{a.a.s.} \quad (15)$$

Observe that  $\mathbb{E}[\phi_2(\delta_{\mathcal{P}})] = \int_0^\infty \phi_2(r) \cdot g'(r) dr = \int_0^{r_0} r \cdot g'(r) dr$ , where  $g'(r) = 2\pi r \cdot e^{-\pi r^2}$ . Integrating by parts, recalling the definition of the Gauss error function, and plugging in the value of  $r_0$ , we have

$$\mathbb{E}[\phi_2(\delta_{\mathcal{P}})] = \left( \int_0^{r_0} e^{-\pi r^2} dr \right) - \left[ r \cdot e^{-\pi r^2} \right]_0^{r_0} = \left[ \frac{\text{erf}(\sqrt{\pi} \cdot r)}{2} - r \cdot e^{-\pi r^2} \right]_0^{r_0} = \xi_\lambda.$$

The claim follows. ◀

### 4.3 Proof of Theorem 2

Let  $T$  denote an  $\alpha$ -approximate traveling salesman tour on  $V \cup \{O\}$ , where  $\alpha \geq 1$  is a constant. When  $k = O(1)$ , for any  $\epsilon > 0$ ,  $\text{ITP}(T) < (1 + \epsilon)\text{OPT}$  a.a.s. according to [15], which implies the claim. In the following, we assume that  $k = \omega(1)$ .

According to Lemma 4, we have

$$\text{ITP}(T) < \text{cost}(T) + \text{rad.}$$

First, we bound  $\text{cost}(T)$ . Letting  $T^*$  denote an optimal traveling salesman tour on  $V \cup \{O\}$ , we have  $\text{cost}(T) \leq \alpha \cdot \text{cost}(T^*)$ . By Lemma 5, the value of an optimal traveling salesman tour on  $V$  is less than  $(\beta + \frac{\epsilon}{2}) \cdot \sqrt{n}$ , a.a.s. Since the distance between the depot and any point in  $[0, 1]^2$  is  $O(1)$ , we have  $\text{cost}(T^*) < (\beta + \frac{\epsilon}{2}) \cdot \sqrt{n} + O(1)$ , which is less than  $(\beta + \epsilon) \cdot \sqrt{n}$  when  $n$  is large enough. Thus  $\text{cost}(T) < \alpha(\beta + \epsilon) \cdot \sqrt{n}$ , a.a.s.

Next, we analyze  $\text{rad}$ . By Theorem 11, for some  $U \subseteq V$  of size greater than  $(\lambda + \frac{\epsilon}{2})n$  we have

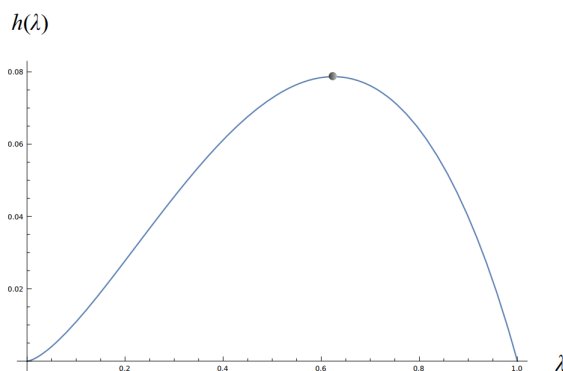
$$\text{rad} \leq \text{OPT} - (1 - \lambda - \epsilon) \left( \sum_{x \in U} \delta(x, U \setminus \{x\}) \right).$$

By Lemma 13 and the fact that  $\delta(x, U \setminus \{x\}) \geq \delta(x, V \setminus \{x\})$ , we have a.a.s.

$$\sum_{x \in U} \delta(x, U \setminus \{x\}) > (\xi_\lambda - \epsilon) \cdot \sqrt{n}.$$

Noting that  $1 - \lambda - \epsilon > 0$ , we have

$$\text{rad} \leq \text{OPT} - (1 - \lambda - \epsilon) \cdot (\xi_\lambda - \epsilon) \cdot \sqrt{n}.$$



■ **Figure 2** Plot of the function  $h(\lambda) = (1 - \lambda) \cdot \xi_\lambda$  for  $\lambda \in [0, 1)$ . The maximum value of  $h(\lambda)$  is greater than 0.078674, which is achieved when  $\lambda$  is roughly 0.62468.

Combining the above bounds gives a.a.s.

$$\text{ITP}(T) < \text{OPT} + (\alpha(\beta + \epsilon) - (1 - \lambda - \epsilon) \cdot (\xi_\lambda - \epsilon)) \cdot \sqrt{n}. \quad (16)$$

Note that the coefficient of  $\sqrt{n}$  in Equation (16) must be positive, because  $\text{ITP}(T) \geq \text{OPT}$  (Lemma 4). Using Lemmas 3 and 5, and assuming  $\epsilon < \beta$ , we have a.a.s.  $\sqrt{n} < \frac{\text{cost}(T^*)}{\beta - \epsilon} \leq \frac{\text{OPT}}{\beta - \epsilon}$ , and substituting into Equation (16) gives a.a.s.

$$\text{ITP}(T) < \left( 1 + \frac{\alpha(\beta + \epsilon) - (1 - \lambda - \epsilon) \cdot (\xi_\lambda - \epsilon)}{\beta - \epsilon} \right) \cdot \text{OPT}.$$

Since  $\beta$  is a positive constant (Lemma 5), choosing  $\lambda$  to maximize  $(1 - \lambda) \cdot \xi_\lambda$  and  $\epsilon$  small enough yields

$$\frac{\text{ITP}(T)}{\text{OPT}} < 1 + \alpha - \frac{\max_\lambda \{(1 - \lambda) \cdot \xi_\lambda\}}{\beta} + 0.00001.$$

A numerical calculation (Figure 2) gives  $\max_\lambda \{(1 - \lambda) \cdot \xi_\lambda\} > 0.078674$ , and Lemma 5 tells us that  $\beta < \beta_1 = 0.92117$ . Substituting those values concludes the proof.


---

## References

- 1 A. Adamaszek, A. Czumaj, and A. Lingas. PTAS for  $k$ -tour cover problem on the plane for moderately large values of  $k$ . *International Journal of Foundations of Computer Science*, 21(06):893–904, 2010.
- 2 K. Altinkemer and B. Gavish. Heuristics for delivery problems with constant error guarantees. *Transportation Science*, 24(4):294–297, 1990.
- 3 S. P. Anbuudayasankar, K. Ganesh, and S. Mohapatra. *Models for practical routing problems in logistics*. Springer, 2016.
- 4 S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.
- 5 T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama. Covering points in the plane by  $k$ -tours: towards a polynomial time approximation scheme for general  $k$ . In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 275–283, 1997.
- 6 A. Baltz, D. Dubhashi, A. Srivastav, L. Tansini, and S. Werth. Probabilistic analysis for a multiple depot vehicle routing problem. *Random Structures & Algorithms*, 30(1-2):206–225, 2007.

- 7 J. Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 55, pages 299–327. Cambridge University Press, 1959.
- 8 J. Blauth, V. Traub, and J. Vygen. Improving the approximation ratio for capacitated vehicle routing. In *Integer Programming and Combinatorial Optimization (IPCO)*, volume 12707. Springer, 2021.
- 9 A. Bompadre, M. Dror, and J. B. Orlin. Improved bounds for vehicle routing solutions. *Discrete Optimization*, 3(4):299–316, 2006.
- 10 A. Bompadre, M. Dror, and J. B. Orlin. Probabilistic analysis of unit-demand vehicle routing problems. *Journal of applied probability*, 44(1):259–278, 2007.
- 11 T. G. Crainic and G. Laporte. *Fleet management and logistics*. Springer Science & Business Media, 2012.
- 12 C. F. Daganzo. The distance traveled to visit  $n$  points with a maximum of  $c$  stops per vehicle: An analytic model and an application. *Transportation science*, 18(4):331–350, 1984.
- 13 A. Das and C. Mathieu. A quasi-polynomial time approximation scheme for Euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015.
- 14 B. Golden, S. Raghavan, and E. Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008.
- 15 M. Haimovich and A. H. G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.
- 16 Richard M. Karp. Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. *Mathematics of operations research*, 2(3):209–224, 1977.
- 17 M. Khachay and R. Dubinin. PTAS for the Euclidean capacitated vehicle routing problem in  $\mathbb{R}^d$ . In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.
- 18 C. L. Li and D. Simchi-Levi. Worst-case analysis of heuristics for multidepot capacitated vehicle routing problems. *ORSA Journal on Computing*, 2(1):64–73, 1990.
- 19 C. L. Li, D. Simchi-Levi, and M. Desrochers. On the distance constrained vehicle routing problem. *Operations research*, 40(4):790–799, 1992.
- 20 J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP,  $k$ -MST, and related problems. *SIAM Journal on computing*, 28(4):1298–1309, 1999.
- 21 G. Mosheiov. Vehicle routing with pick-up and delivery: tour-partitioning heuristics. *Computers & Industrial Engineering*, 34(3):669–684, 1998.
- 22 M. D. Penrose and J. E. Yukich. Weak laws of large numbers in geometric probability. *The Annals of Applied Probability*, 13(1):277–303, 2003.
- 23 W. T. Rhee. Probabilistic analysis of a capacitated vehicle routing problem II. *The Annals of Applied Probability*, 4(3):741–764, 1994.
- 24 S. Steinerberger. New bounds for the traveling salesman constant. *Advances in Applied Probability*, 47(1):27–36, 2015.
- 25 P. Toth and D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

# Exact and Approximation Algorithms for Many-To-Many Point Matching in the Plane

Sayan Bandyapadhyay ✉ 

Department of Informatics, University of Bergen, Norway

Anil Maheshwari ✉ 

School of Computer Science, Carleton University, Ottawa, Canada

Michiel Smid ✉ 

School of Computer Science, Carleton University, Ottawa, Canada

---

## Abstract

Given two sets  $S$  and  $T$  of points in the plane, of total size  $n$ , a many-to-many matching between  $S$  and  $T$  is a set of pairs  $(p, q)$  such that  $p \in S$ ,  $q \in T$  and for each  $r \in S \cup T$ ,  $r$  appears in at least one such pair. The cost of a pair  $(p, q)$  is the (Euclidean) distance between  $p$  and  $q$ . In the minimum-cost many-to-many matching problem, the goal is to compute a many-to-many matching such that the sum of the costs of the pairs is minimized. This problem is a restricted version of minimum-weight edge cover in a bipartite graph, and hence can be solved in  $O(n^3)$  time. In a more restricted setting where all the points are on a line, the problem can be solved in  $O(n \log n)$  time [3]. However, no progress has been made in the general planar case in improving the cubic time bound. In this paper, we obtain an  $O(n^2 \cdot \text{poly}(\log n))$  time exact algorithm and an  $O(n^{3/2} \cdot \text{poly}(\log n))$  time  $(1 + \epsilon)$ -approximation in the planar case.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Approximation algorithms analysis

**Keywords and phrases** Many-to-many matching, bipartite, planar, geometric, approximation

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.44

**Related Version** *Full Version:* <http://arxiv.org/abs/2109.07524>

**Funding** This work is supported by the European Research Council (ERC) via grant LOPPRE, reference 819416 and by NSERC, Canada.

**Acknowledgements** We thank Saeed Mehrabi for introducing the many-to-many matching problem to us.

## 1 Introduction

Let  $G = (V = S \cup T, E)$  be a simple bipartite graph where each edge has a non-negative real weight, and no vertex is isolated. The *many-to-many* matching problem on  $G$  is to find a subset of edges  $E' \subseteq E$  of minimum total weight such that for each vertex  $v \in V$  there is an edge in  $E'$  incident on  $v$ . This is often referred to as the *minimum-weight edge cover* problem. A standard method to compute a minimum-weight edge cover of  $G$  in polynomial-time is to reduce the problem to the minimum-weight perfect matching problem on an equivalent graph, see [15, 3, 4, 5]. Since the reduction takes  $O(|V| + |E|)$  time, the running time is the same as the fastest known algorithm for computing a minimum-weight perfect matching. A faster  $\frac{3}{2}$ -approximation algorithm is proposed in [5].

Motivated by the computational problems in musical rhythm theory, Colannino et al. [3] studied the many-to-many matching problem in a geometric setting. Suppose we are given two sets  $S$  and  $T$  of points in the plane, of total size  $n$ . A *many-to-many* matching between  $S$  and  $T$  is a set of pairs  $(p, q)$  such that  $p \in S$ ,  $q \in T$  and for each  $r \in S \cup T$ ,  $r$  appears in at least one such pair. The *cost of a pair*  $(p, q)$  is the (Euclidean) distance  $d(p, q)$  between  $p$



© Sayan Bandyapadhyay, Anil Maheshwari, and Michiel Smid;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 44; pp. 44:1–44:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and  $q$ . The *cost of a set of pairs* is the sum of the costs of the pairs in the set. In the geometric *minimum-cost many-to-many matching* problem, the goal is to compute a many-to-many matching such that the cost of the corresponding set of pairs is minimized. For points on a line, an  $O(n \log n)$  time dynamic-programming algorithm is proposed in [3]. In the same paper, the importance of the many-to-many matching problem for points in the plane is stated in the context of melody matching. Furthermore, the authors state that this version in the plane is an important open problem, especially since geometry has helped in designing efficient algorithms for the computation of the minimum-weight perfect matching for points in the plane (see, e.g., [16, 17]). Several variants of the 1-dimensional many-to-many matching problem are considered in [12, 13, 14].

## 1.1 Our Results and Techniques

In this work, we design several exact and approximation algorithms for minimum-cost many-to-many matching in the plane, thus affirmatively addressing the open problem posed by [3]. First, we obtain an  $O(n^2 \cdot \text{poly}(\log n))$  time<sup>1</sup> exact algorithm for this problem using a connection to minimum-weight perfect matching. We note that our time-bound matches the time bound for solving minimum-weight bipartite matching for points in the plane. Next, for any  $\epsilon > 0$ , we obtain a  $(1 + \epsilon)$ -approximation algorithm for this problem with improved  $O((1/\epsilon^c) \cdot n^{3/2} \cdot \text{poly}(\log n))$  running time for some small constant  $c$ . We also obtain a simple 2-approximation in  $O(n \log n)$  time.

Next, we give an overview of our techniques. A major reason behind the scarcity of results for geometric many-to-many-matching is the lack of techniques to directly approach this problem. The  $O(n^3)$  algorithm known for general graphs reduces the problem to (minimum-weight) bipartite perfect matching and uses a graph matching algorithm to solve the problem on the new instance. However, this standard reduction [9] from many-to-many matching to regular matching changes the weights of the edges in a convoluted manner. Hence, even if one starts with the planar Euclidean distances, the new interpoint distances in the constructed instance cannot be embedded in the plane or even in any metric space. As the algorithms for planar bipartite (perfect) matching heavily exploit the properties of the plane, they cannot be employed for solving the new instance of bipartite perfect matching. Thus, even though there is a wealth of literature for planar bipartite matching, no progress has been made in understanding the structure of many-to-many matching in the plane.

In our approach, we use a rather unconventional connection to bipartite perfect matching. First, we use a different reduction to convert our instance of many-to-many matching in the plane to an equivalent instance of bipartite perfect matching. The new instance cannot be embedded in the plane, however it does not modify the original distances between the points. Rather the new bipartite graph constructed is the union of (i) the original geometric bipartite graph, (ii) a new bipartite clique all of whose edges have the same weight and (iii) a linear number of additional edges. Thus, even though we could not use a planar matching algorithm directly, we could successfully use ideas from the literature of planar matching exploiting the structure of the constructed graph. A similar reduction was used in [4] in the context of computation of link distance of graphs. However, we cannot use this reduction directly, as we cannot afford to explicitly store the constructed graph which contains  $\Omega(n^2)$  edges. Recall that we are aiming for an  $O((1/\epsilon^c) \cdot n^{3/2} \cdot \text{poly}(\log n))$  time bound. Nevertheless, we exploit the structure of this graph to implicitly store it in  $O(n)$  space.

---

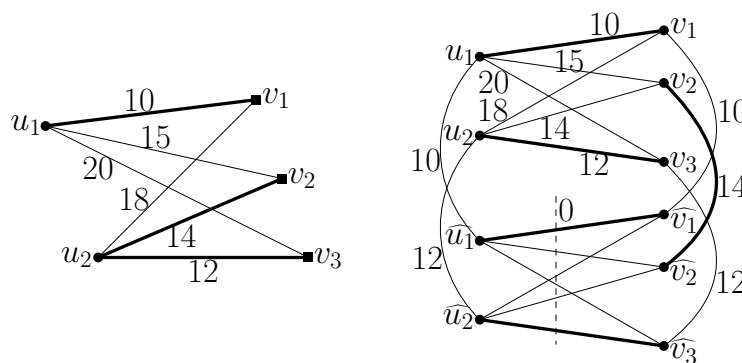
<sup>1</sup> We use the notation  $\text{poly}()$  to denote a polynomial function.

In Section 3, we show that the Hungarian algorithm can be implemented on our constructed graph in  $O(n^2 \cdot \text{poly}(\log n))$  time using ideas from planar matching. To obtain a subquadratic time bound, we implement the algorithm due to Gabow and Tarjan [6]. A straightforward implementation of this algorithm might need  $\Omega(n^2)$  time. We note that this algorithm has been used in several works on planar matching for obtaining efficient algorithms [11, 17, 2]. Our algorithm is closest to the one in [17] among these algorithms from a moral point of view. The difference is that their algorithm is for planar points. However, we deal with a graph which is partly embeddable. Nevertheless, in Section 4, we show that using additional ideas and data structures, the Gabow-Tarjan algorithm can be implemented in  $O(n^{3/2} \cdot \text{poly}(\log n))$  time, albeit with a  $(1 + \epsilon)$ -factor loss on the quality of the solution.

## 2 Preliminaries

In the *bipartite perfect matching* problem, we are given an edge-weighted bipartite graph  $G = (R, B, E)$  containing a perfect matching, and the goal is to find a perfect matching having the minimum cost or sum of the edge-weights. For our convenience, sometimes we would assume that the edges of  $G$  are not given explicitly. For example,  $R$  and  $B$  might be two sets of points in the plane, and  $G$  is the complete bipartite graph induced by the bipartition  $(R, B)$ . In this case, the points in  $R \cup B$  can be used to implicitly represent the graph  $G$ . In our case, we will use similar implicit representation of input graphs for designing subquadratic algorithm. Now, we have the following lemma, which reduces our problem to an equivalent instance of bipartite perfect matching.

► **Lemma 1.** *Given an instance  $I'$  of minimum-cost many-to-many matching, one can compute in  $O(n \log n)$  time an instance of bipartite perfect matching  $I$  such that (i) if there is a many-to-many matching for  $I'$  of cost  $C$ , there is a perfect matching for  $I$  of cost at most  $C$ , and (ii) if there is a perfect matching for  $I$  of cost  $C$ , there is a many-to-many matching for  $I'$  of cost  $C$ .*



■ **Figure 1** The figure on the left shows an instance  $I'$  of minimum-cost many-to-many matching along with the interpoint distances, where  $S = \{u_1, u_2\}$  and  $T = \{v_1, v_2, v_3\}$ . The right figure depicts the graph constructed from  $I'$  along with the edge weights, where  $R_0 = \{u_1, u_2\}$ ,  $R_1 = \{\hat{v}_1, \hat{v}_2, \hat{v}_3\}$ ,  $B_0 = \{v_1, v_2, v_3\}$  and  $B_1 = \{\hat{u}_1, \hat{u}_2\}$ . Solution pairs (or edges) are shown in bold.

**Proof.** Given the instance  $I'$  consisting of the two sets of points  $S$  and  $T$ , we construct a bipartite graph  $G = (R = R_0 \cup R_1, B = B_0 \cup B_1, E)$ , where  $R_0 = S$ ,  $B_0 = T$ ,  $R_1$  contains copies of the points in  $T$ ,  $B_1$  contains copies of the points in  $S$ .  $E$  contains

all the edges of  $E_0 = R_0 \times B_0$  and  $E_1 = R_1 \times B_1$ , and also the ones in  $E_2 = \{(u, \hat{u}) \mid u \in R_0, \hat{u} \text{ is the copy of } u \text{ in } B_1\}$  and  $E_3 = \{(\hat{v}, v) \mid v \in B_0, \hat{v} \text{ is the copy of } v \text{ in } R_1\}$ . The weight of each edge  $(u, v) \in E_0$  is the distance between the points  $u$  and  $v$ . The weight of each edge in  $E_1$  is 0. The weight of any edge  $(u, \hat{u}) \in E_2$  is the distance between  $u \in S$  and its closest neighbor in  $T$ . The weight of any edge  $(\hat{v}, v) \in E_3$  is the distance between  $v \in T$  and its closest neighbor in  $S$ . See Figure 1 for an example. Note that  $G$  can be represented implicitly in  $O(n)$  space, where  $|S \cup T| = n$ . Let  $I$  be the constructed instance of bipartite perfect matching that consists of  $G$ . As the closest neighbors of  $n$  points in the plane can be found in total  $O(n \log n)$  time using Voronoi diagram, construction of  $G$  takes  $O(n \log n)$  time.

Now, suppose  $I'$  has a many-to-many matching  $M'$ . Consider the pairs in  $M'$  as the edges of a graph  $G'$  with vertices being the points in  $S \cup T$ . We will use the term pairs and edges in  $G'$  interchangeably. First, note that wlog we can assume that  $G'$  does not contain any path of length 3. Otherwise, we can remove the middle edge of such a path from  $M'$ , and  $M'$  still remains a many-to-many matching. Thus, each component in  $G'$  is a star. We compute a perfect matching in  $G$  from  $M'$  as follows. For each star in  $G'$  having only one edge  $(u, v)$  with  $u \in S, v \in T$ , add  $(u, v) \in E_0$  to  $M$ . Also, add the edge  $(\hat{u}, \hat{u}) \in E_1$  to  $M$ . Now, consider any star  $H = \{(u, v_1), (u, v_2), \dots, (u, v_t)\}$  in  $G'$ ; wlog assume that  $u \in S$ . Add  $(u, v_1) \in E_0$  to  $M$ . Also, add the edge  $(\hat{v}_1, \hat{u}) \in E_1$  to  $M$ . For each  $2 \leq i \leq t$ , add the edge  $(\hat{v}_i, v_i) \in E_3$  to  $M$ . It is not hard to verify that all the vertices of  $G$  are matched in  $M$ . Also, as the weight of a star edge  $(u, v_i)$  above is at least the weight of  $(\hat{v}_i, v_i)$  in  $G$  by definition, the cost of  $M$  is at most the cost of  $M'$ .

Next, suppose  $I$  has a perfect matching  $M$ ; we construct a many-to-many matching  $M'$  for  $I'$ . Consider any  $u \in S = R_0$ . If  $(u, \hat{u}) \in M$ , add the pair  $(u, u')$  to  $M'$ , where  $u'$  is the closest neighbor of  $u$  in  $T$ . Otherwise,  $u$  is matched (in  $M$ ) to some  $v_1 \in B_0$ . In this case, simply add the pair  $(u, v_1)$  to  $M'$ . Similarly, consider any  $v \in T = B_0$ . If  $(\hat{v}, v) \in M$ , add the pair  $(v', v)$  to  $M'$ , where  $v'$  is the closest neighbor of  $v$  in  $S$ . Otherwise,  $v$  is matched (in  $M$ ) to some  $u_1 \in R_0$ . In this case, simply add the pair  $(u_1, v)$  to  $M'$ . It is not hard to verify that  $M'$  is a many-to-many matching, and the cost of  $M'$  is same as the cost of  $M$ . ◀

Now, consider any matching in a graph. An *alternating* path is a path whose edges alternate between matched and unmatched edges. Similarly, one can define *alternating* cycles and trees. A vertex is called *free* if it is not matched. An *augmenting* path is an alternating path which starts and ends at free vertices. Given an augmenting path  $P$  w.r.t. a matching  $M$ , we can augment  $M$  by one edge if we remove the edges of  $P \cap M$  from  $M$  and add the edges in  $P \setminus M$  to  $M$ . The new matching is denoted by  $M \oplus P$ . Throughout the paper,  $m$  and  $n$  denote the number of edges and vertices, respectively, unless otherwise specified. We denote the weight or cost of an edge  $(u, v)$  by  $c(u, v)$ . In our discussions, a path can be treated as an ordered set of vertices or edges depending on the context.

### 3 An Exact Algorithm

Consider the instance  $I$  obtained by the reduction in Lemma 1. In this section, we prove the following theorem.

► **Theorem 2.** *Bipartite perfect matching can be solved exactly on  $I$  in time  $O(n^2 \cdot \text{poly}(\log n))$ , and hence there is an  $O(n^2 \cdot \text{poly}(\log n))$  time exact algorithm for minimum-cost many-to-many matching.*



In the rest of this section, we prove Theorem 2. To prove this theorem we use the Hungarian algorithm [10] for computing a (minimum-cost) bipartite perfect matching.

In the Hungarian algorithm, there is a dual variable  $y(v)$  corresponding to each vertex  $v$ . Every feasible matching  $M$  must satisfy the following two conditions.

$$y(u) + y(v) \leq c(u, v) \text{ for every edge } (u, v) \quad (1)$$

$$y(u) + y(v) = c(u, v) \text{ for every edge } (u, v) \in M \quad (2)$$

Given any matching  $M$  and an augmenting path  $P$ , the net-cost or augmentation cost is defined as,

$$\phi(P) = \sum_{(u,v) \in P \setminus M} c(u, v) - \sum_{(u,v) \in P \cap M} c(u, v)$$

$\phi(P)$  is basically the cost increment for augmenting  $M$  along  $P$ . The net-cost of any alternating cycle can be defined in the same way. The Hungarian algorithm starts with an empty matching  $M$ . In every iteration, it computes an augmenting path of the minimum net-cost and augments the current matching. The algorithm halts once a perfect matching is found. If there is a perfect matching in the graph, this algorithm returns one after at most  $n$  iterations. Moreover, an augmenting path can be found in  $O(m)$  time leading to  $O(mn)$  running time in total.

It is possible to show that any perfect matching is a minimum-cost matching if and only if there is no negative net-cost alternating cycle with respect to it. Moreover, a feasible matching with dual values  $\{y(v)\}$  satisfies this property. Thus, it is sufficient to find a perfect matching that is feasible.

For finding an augmenting path, a Hungarian search procedure is employed. Hungarian search uses Dijkstra's shortest path algorithm to find a minimum net-cost path in the graph where the value  $y(u) + y(v)$  is subtracted from the weight of each edge  $(u, v)$ . This along with the first feasibility condition ensure that each edge has a non-negative weight, and hence there is no negative cycle in the graph. So, one can correctly employ Dijkstra's algorithm to find such a shortest path. Finally, one can show that the minimum net-cost augmenting path with original weights corresponds to a shortest path with the modified weights, and vice versa. After augmenting the current matching with the newly found path, the dual values are adjusted appropriately to ensure feasibility of the new matching.

Now, we describe in detail how the Hungarian search procedure is implemented in each iteration. An edge  $(u, v)$  is called *admissible* if  $y(u) + y(v) = c(u, v)$ . It can be shown that it suffices to find an augmenting path consisting of only admissible edges. Let  $M$  be the current matching and  $F$  be the free vertices of  $R$ . To obtain the desired augmenting path, a forest  $\mathcal{F}$  is grown whose roots are in  $F$ . Each tree in  $\mathcal{F}$  is an augmenting tree rooted at a vertex in  $F$ . Once  $\mathcal{F}$  contains an augmenting path the search is completed. At any moment, let  $R'$  be the vertices of  $R$  in  $\mathcal{F}$  and  $B'$  be the vertices of  $B$  not in  $\mathcal{F}$ . Initially,  $R' = F$  and  $B' = B$ . Also, let

$$\delta = \min_{\{u \in R', v \in B'\}} c(u, v) - y(u) - y(v).$$

Note that  $\delta = 0$  means there is an admissible edge. In each step, if  $\delta = 0$ , an admissible edge  $(u, v)$  is selected where  $u \in R'$  and  $v \in B'$ . If  $v$  is free,  $(u, v)$  is added to  $\mathcal{F}$  and the desired augmenting path is found. Otherwise, let  $v$  be matched to  $u'$  (which is not in  $\mathcal{F}$  by an invariant). In this case, the edges  $(u, v)$  and  $(v, u')$  are added to  $\mathcal{F}$ ;  $u'$  is added to  $R'$  and  $v$  is removed from  $B'$ .

If at some moment,  $\delta$  becomes more than 0 (no admissible edge), we perform dual adjustments. In particular, for each  $u \in R'$ ,  $y(u)$  is updated to  $y(u) + \delta$  and for each  $v \in \mathcal{F} \cap B$ ,  $y(v)$  is updated to  $y(v) - \delta$ . This ensures that at least one edge becomes admissible, e.g., the edge corresponding to which the  $\delta$  value is achieved. Thus, eventually the search halts with an augmenting path in  $\mathcal{F}$ .

It can be shown that if  $\delta$  can be computed efficiently, then the desired augmenting path can also be found efficiently [6, 1, 16]. For that purpose, another variable  $\Delta$  is maintained. Also, for each vertex  $v$ , a weight  $\sigma_v$  is stored. In the beginning of each step,  $\Delta = 0$ ,  $\sigma_v = y(v)$ . When the edges  $(u, v)$  and  $(v, u')$  are added to  $\mathcal{F}$ ,  $\sigma_{u'}$  is updated to  $y(u') - \Delta$  and  $\sigma_v$  is updated to  $y(v) + \Delta$ . Once  $\delta$  becomes more than 0,  $\Delta$  is updated to  $\Delta + \delta$ .

Note that the weight of a vertex is updated only once when it is added to  $\mathcal{F}$ . We have the following observation.

► **Observation 3** ([6, 1, 16]). *For each  $u \in R'$ , the current dual value of  $u$ ,  $y(u)$ , is equal to  $\sigma_u + \Delta$ . For each  $v \in \mathcal{F} \cap B$ , the current dual value of  $v$ ,  $y(v)$ , is equal to  $\sigma_v - \Delta$ .*

It follows from the above observation that  $\delta$  can be equivalently expressed as follows.

$$\delta = \min_{u \in R', v \in B'} \{c(u, v) - \sigma_u - \sigma_v\} - \Delta.$$

Hence, Hungarian search boils down to the following task ignoring the trivial details. We need to maintain two sets  $R' \subseteq R$  and  $B' \subseteq B$ . Initially,  $B' = B$ . In each step, a vertex  $r$  is added to  $R'$  and a vertex  $b$  is removed from  $B'$ . Additionally, each vertex  $v$  has a weight  $\sigma_v$ . In every step, the goal is to maintain the bichromatic closest pair, which is the pair  $(r, b) \in R' \times B'$  with the minimum  $c(r, b) - \sigma_r - \sigma_b$  value. In the following, we construct a data structure that can be used to perform the above task (a Hungarian search) in  $O(n \cdot \text{poly}(\log n))$  time. As we need at most  $n$  such searches, Theorem 2 follows.

Recall that in our instance  $I$ , we are given the graph  $G = (R = R_0 \cup R_1, B = B_0 \cup B_1, E)$ , where  $R_1$  contains copies of the vertices in  $B_0$ ,  $B_1$  contains copies of the vertices in  $R_0$ .  $E = E_0 \cup E_1 \cup E_2 \cup E_3$ , where  $E_0 = R_0 \times B_0$ ,  $E_1 = R_1 \times B_1$ ,  $E_2 = \{(u, \hat{u}) \mid u \in R_0, \hat{u} \in B_1\}$ , and  $E_3 = \{(\hat{v}, v) \mid v \in B_0, \hat{v} \in R_1\}$ . Let  $n = |R| = |B|$ .

Our data structure  $\mathcal{D}$  is a collection of three data structures. First, we construct the dynamic bichromatic closest pair data structure from [8] for the two point sets  $R_0$  and  $B_0$  with the distance function  $c(r, b) - \sigma_r - \sigma_b$  for each pair  $(r, b)$ . We refer to this data structure as  $\mathcal{D}_1$ . Initially, it contains only the points in  $R' \cup B'$ . Next, we construct two max-heaps  $H_1^r$  and  $H_1^b$  for the vertices in  $R_1 \cap R'$  and  $B_1 \cap B'$ , respectively. Initially,  $H_1^r$  contains vertices in  $R'$  and  $H_1^b$  contains all the vertices in  $B_1$ . The key value of each vertex  $v$  is its weight  $\sigma_v$ . Using  $H_1^r$  and  $H_1^b$ , the pair  $(u, v) \in (R_1 \cap R') \times (B_1 \cap B')$  with the maximum  $\sigma_u + \sigma_v$  value can be found in  $O(1)$  time. We also construct another min-heap  $H_{23}$  to store the edges in  $E_2 \cup E_3$  with key value  $c(r, b) - \sigma_r - \sigma_b$  for each pair  $(r, b)$ . Initially, it is empty. In every iteration, it contains only those edges  $(u, v)$  such that  $u \in R'$  and  $v \in B'$ . We also maintain a global closest pair  $(r, b)$  over the three data structures  $\mathcal{D}_1$ ,  $H_1^r \cup H_1^b$  and  $H_{23}$  with the minimum key value  $c(r, b) - \sigma_r - \sigma_b$ .

Using  $\mathcal{D}$ , we implement each step as follows. If  $v \in B_0$ , remove  $v$  from  $\mathcal{D}_1$ . Also remove  $(\hat{v}, v)$  from  $H_{23}$  if it is in  $H_{23}$ . If  $v \in B_1$ , remove  $v$  from  $H_1^b$ . Also remove the edge  $(u', v)$  with  $u' \in R_0$  from  $H_{23}$  if its in  $H_{23}$ . If  $u \in R_0$ , add  $u$  to  $\mathcal{D}_1$ . Also add the edge  $(u, \hat{u})$  to  $H_{23}$  if  $\hat{u} \in B'$ . If  $u \in R_1$ , add  $u$  to  $H_1^r$ . Also add the edge  $(u, v')$  with  $v' \in B_0$  to  $H_{23}$  if  $v' \in B'$ .

It is not hard to verify that at each step the correct global closest pair is stored in  $\mathcal{D}$ . The correctness for  $\mathcal{D}_1$  and  $H_{23}$  follow trivially. Also, as the weight of the edges in  $E_1$  are same, it is sufficient to find a pair  $(u, v) \in E_1$  for which  $\sigma_u + \sigma_v$  value is maximized. As  $E_1$  contains all the edges in  $R_1 \times B_1$ , equivalently it suffices to find a  $u \in R_1$  with the maximum  $\sigma_u$  value and a  $v \in B_1$  with the maximum  $\sigma_v$  value.

We note that the total number of steps in a Hungarian search is at most  $n$ . Also,  $O(n)$  operations (insertions, deletions and searching) on  $\mathcal{D}_1$  can be performed in amortized  $O(n \cdot \text{poly}(\log n))$  time [8]. The construction time and space of  $\mathcal{D}_1$  are also  $O(n \cdot \text{poly}(\log n))$ . Moreover, the  $O(n)$  operations on all max-heaps can be performed in total  $O(n \log n)$  time. It follows that Hungarian search can be implemented in  $O(n \cdot \text{poly}(\log n))$  time leading to the desired running time of our algorithm.

#### 4 An Improved $(1 + \epsilon)$ -approximation

Consider the instance  $I$  obtained by the reduction in Lemma 1. In this section, we prove the following theorem.

► **Theorem 4.** *A  $(1 + \epsilon)$ -approximate bipartite perfect matching for  $I$  can be computed in time  $O((1/\epsilon^c) \cdot n^{3/2} \cdot \text{poly}(\log n))$  for some constant  $c$ , and hence there is an  $O((1/\epsilon^c) \cdot n^{3/2} \cdot \text{poly}(\log n))$  time  $(1 + \epsilon)$ -approximation algorithm for minimum-cost many-to-many matching.*

In the rest of this section, we prove Theorem 4. In particular, we show that the bipartite perfect matching algorithm by Gabow and Tarjan [6] can be implemented on the instance  $I$  in the mentioned time.

The Gabow-Tarjan algorithm is based on a popular scheme called the bit-scaling paradigm. The algorithm is motivated by two classic matching algorithms: Hopcroft-Karp [7] for maximum cardinality bipartite matching with  $O(m\sqrt{n})$  running time and Hungarian algorithm [10] for minimum-cost bipartite matching with  $O(mn)$  running time. The Hopcroft-Karp algorithm chooses an augmenting path of the shortest length. The Hungarian algorithm, as mentioned before, chooses an augmenting path whose augmentation cost is the minimum. When the weights on the edges are small an augmenting path of the shortest length approximates the latter path. The Gabow-Tarjan algorithm scales the weights in a manner so that all the effective weights are small. This helps to combine the ideas of the two algorithms, which leads towards an  $O(m\sqrt{n} \log(nN))$  time algorithm for (minimum-cost) bipartite perfect matching, where  $N$  is the largest edge weight.

Next, we describe the Gabow-Tarjan algorithm. This algorithm is based on the ideas of the Hungarian algorithm. However, here instead of a feasible matching we compute a *1-feasible* matching. A matching  $M$  is called 1-feasible if it satisfies the following two conditions.

$$y(u) + y(v) \leq c(u, v) + 1 \text{ for every edge } (u, v) \tag{3}$$

$$y(u) + y(v) = c(u, v) \text{ for every edge } (u, v) \in M \tag{4}$$

Note that the only difference is that now the sum of dual variables  $y(u) + y(v)$  can be 1 plus the cost of the edge. This additive error of 1 on every unmatched edge ensures that longer augmenting paths have larger cost. As an effect, the algorithm picks short augmenting paths as in the Hopcroft-Karp algorithm.

A *1-optimal* matching is a perfect 1-feasible matching. Note that a 1-optimal matching costs more than the original optimal matching. However, as the error is at most +1 for every edge, one can show the following.

► **Lemma 5 ([6]).** *Let  $M$  be a 1-optimal matching and  $M'$  be any perfect matching. Then  $c(M') \geq c(M) - n$ .*

It follows from the above lemma that, to annihilate the error introduced, one can scale the weight of each edge by a factor of  $(n + 1)$ , and then with the scaled weights the cost of a 1-optimal matching is same as the cost of any optimal matching. Let  $\bar{c}(u, v)$  be the scaled cost of  $(u, v)$ , i.e.,  $\bar{c}(u, v) = (n + 1) \cdot c(u, v)$ . Let  $k = \lfloor \log((n + 1)N) \rfloor + 1$  be the maximum number of bits needed to represent any new weight.

The Gabow-Tarjan algorithm runs in  $k$  different scales. In each scale  $i$  ( $1 \leq i \leq k$ ), the most significant  $i$  bits of  $\bar{c}(u, v)$  are used for defining the current cost of each edge  $(u, v)$ . The dual values are also modified to maintain 1-feasibility of an already computed perfect matching in the following way:  $y(v) \leftarrow 2y(v) - 1$  for every vertex  $v$ . Then with the current edge costs and dual values, the algorithm computes a 1-optimal matching.

By the above claim, that any 1-optimal matching is also optimal with scaling factor  $n + 1$ , the 1-optimal matching computed by the algorithm at  $k$ -th scale must be optimal.

To find a 1-optimal matching on a particular scale a procedure called *match* is employed which we describe below. Before that we need a definition. Consider any 1-feasible matching  $M$ . An edge  $(u, v)$  is called *eligible* if it is in  $M$  or  $y(u) + y(v) = c(u, v) + 1$ . It can be shown that for the purpose of computing a 1-optimal matching it suffices to consider the augmenting paths which consist of eligible edges only.

**The *match* procedure**

Initialize all the dual variables  $y(v)$  to 0 and  $M$  to  $\emptyset$ . Repeat the following two steps until a perfect matching is obtained in step 1.

1. Find a maximal set  $\mathcal{A}$  of augmenting paths of eligible edges. For each path  $P \in \mathcal{A}$ , augment the current matching  $M$  along  $P$  to obtain a new matching which is also denoted by  $M$ . For each vertex  $v \in P \cap B$ , decrease  $y(v)$  by 1. (This is to ensure that the new matching  $M$  also is 1-feasible.) If  $M$  is perfect, terminate.
2. Employ a Hungarian search to adjust the values of the dual variables (by keeping  $M$  1-feasible), and find an augmenting path of eligible edges.

Note that the number of free vertices in every iteration of *match* is at least 1 less than that in the previous iteration, as step 2 always ends with finding an augmenting path. By also showing that the dual value of a variable is increased by at least 1 in every call of Hungarian search, they proved that  $O(\sqrt{n})$  iterations are sufficient to obtain a 1-optimal matching. It can be shown that each iteration of *match* can be executed in general bipartite graphs in  $O(m)$  time leading to the complexity of  $O(m\sqrt{n})$  in each scale. As we have  $O(\log(nN))$  scales, in total the running time is  $O(m\sqrt{n} \log(nN))$ .

Next, we use the Gabow-Tarjan algorithm to compute a perfect matching for our instance of bipartite perfect matching. In particular, we show that by exploiting the structure of our instance, it is possible to implement every iteration of *match* in  $O(n \cdot \text{poly}(\log n))$  time. First, we show that one can consider a modified instance with bounded aspect ratio of the weights, for the purpose of computing a  $(1 + \epsilon)$ -approximation. For this, we need the following theorem. (The proofs of the results marked with  $(*)$  are deferred to the full version.)

► **Theorem 6**  $(*)$ . *There is an  $O(n \log n)$  time 2-approximation algorithm for minimum-cost many-to-many matching.*

The algorithm in Theorem 6 is fairly simple. For each point in a set ( $S$  or  $T$ ), it adds the pair corresponding to its nearest neighbor in the other set.

Let  $\text{OPT}$  be the optimal cost of perfect matching on the instance  $I$ . Recall that the instance  $I$  consists of the graph  $G = (R = R_0 \cup R_1, B = B_0 \cup B_1, E)$ . Given the implicit representation of  $G$ , we compute a 2-approximate solution for minimum-cost many-to-many

matching on the two sets of points  $R_0$  and  $B_0$  using the algorithm in Theorem 6. Let  $C$  be the cost of this solution. By Lemma 1,  $\text{OPT} \leq C \leq 2\text{OPT}$ . We construct a new instance  $I_1$  which consists of the implicit representation of the same graph  $G$ . Additionally, we assume that any edge with weight more than  $C$  in  $I_1$  will not be part of the solution, and each edge has cost at least  $\epsilon C/(2n)$ . Note that it is not possible to explicitly set the weight of each and every edge if we are allowed to spend  $o(n^2)$  time. Thus, for the time being, we make the above assumptions implicitly. Later, we will make them explicit in our algorithm. Note that the construction time of  $I_1$  is dominated by the time of the 2-approximation algorithm, which is  $O(n \log n)$ . We obtain the following lemma.

► **Lemma 7** (\*). *Given  $I$ , one can compute in  $O(n \log n)$  time another instance  $I_1$  of bipartite perfect matching, such that (i) the weight of every edge is in  $[\epsilon C/(2n), C]$  where  $\text{OPT} \leq C \leq 2\text{OPT}$ , (ii)  $I_1$  has a perfect matching of weight at most  $(1 + \epsilon)\text{OPT}$ , and (iii) any perfect matching in  $I_1$  of weight  $C'$  is also a perfect matching in  $I$  of weight at most  $C'$ .*

Henceforth, we solve the problem on  $I_1$ . Note that the minimum edge weight in  $I_1$  is  $\epsilon C/(2n)$  and the maximum is  $C$ . By scaling the weights by  $2n/(\epsilon C)$ , we can assume wlog that the edge weights in  $I_1$  are in  $[1, n^2]$ . Moreover, we can assume that each edge weight  $w$  is rounded up to the nearest integer at least  $w$ . We can afford to remove the fractions, as each fraction costs less than 1, which is at most  $\epsilon \text{OPT}/n$  w.r.t. the original weights. For our convenience, we also divide the weights into  $O(\log_{1+\epsilon} n)$  classes as follows. For each weight  $a$  (an integer) with  $(1 + \epsilon)^i \leq a < (1 + \epsilon)^{i+1}$ ,  $a$  is rounded to the largest integer in the range  $[(1 + \epsilon)^i, (1 + \epsilon)^{i+1})$ , where  $0 \leq i \leq \lceil 2 \log_{1+\epsilon} n \rceil$ . We denote this largest integer corresponding to the  $i$ -th weight class by  $w_i$ . We note that the above weight scalings are performed implicitly. It is not hard to verify that these still preserve a  $(1 + O(\epsilon))$ -approximate solution, which is sufficient for our purpose. Henceforth, we treat  $\epsilon$  as a constant and hide function of  $\epsilon$  in time complexity as a constant in  $O()$  notation. It will not be hard to verify that the dependency on  $\epsilon$  that we hide is  $(1/\epsilon)^c$  for some true small constant  $c$ .

To implement the Gabow-Tarjan algorithm, we show how the *match* procedure can be implemented efficiently. To implement step 1 of *match*, we store the information about the input graph  $G$  in a data structure that we refer to as MATCH. We allow the following operation on MATCH. In the following, we denote the current matching by  $M$ .

**FIND\_MAXIMAL\_APS.** Find a maximal set of vertex disjoint augmenting paths of eligible edges with respect to  $M$ .

Given the MATCH data structure, we implement the step 1 of *match* as follows. The dual values are stored in an array indexed by the vertices. Note that the dual values remain fixed in step 1 while the augmenting paths are found. Afterwards, the dual values are updated. We first make a call to FIND\_MAXIMAL\_APS to obtain a maximal set  $\mathcal{A}$  of paths. For each  $P \in \mathcal{A}$ , we augment  $M$  along  $P$  to obtain a new matching  $M$ . Also, for each  $v \in P \cap B$ , we decrease  $y(v)$  by 1.

In Section 5, we show how to construct and maintain MATCH so that the above subroutine can be performed in time  $O(n \cdot \text{poly}(\log n))$ . The building time of MATCH is  $O(n \cdot \text{poly}(\log n))$ , and it takes  $O(n \cdot \text{poly}(\log n))$  space. Thus, by noting that  $\mathcal{A}$  contains disjoint paths, step 1 can be implemented in  $O(n \cdot \text{poly}(\log n))$  time and space.

We also need another data structure which will help us implement step 2 of *match*, which we refer to as the Hungarian search data structure. As described before, Hungarian search boils down to maintaining a bichromatic closest pair  $(r, b)$  of two sets with the minimum  $c(r, b) - \sigma_r - \sigma_b$  value. However, here we have to be more careful, as for an unmatched

eligible edge  $(u, v)$ ,  $y(u) + y(v) = c(u, v) + 1$ . In contrast, in the Hungarian algorithm, we had  $y(u) + y(v) = c(u, v)$  in that case. Hence, we have to consider the  $c(u, v) + 1 - y(u) - y(v)$  value as the distance of such an unmatched pair  $(u, v)$ . This apparently makes our life harder, as now we have to deal with two types of distance functions: one for matched pairs and one for unmatched pairs. However, we use the following observation to consider only one type of distances, which follows from our description of Hungarian search in Section 3.

► **Observation 8.** *Consider any matched edge  $(u, v)$  with  $u \in R$  and  $v \in B$ . In the Hungarian search, if  $u \in R'$ , i.e.,  $u$  is already in the forest, then  $v$  must also be in the forest, i.e.,  $v \notin B'$ .*

Recall that for computing  $\delta$ , we look into the pairs  $(u, v)$  where  $u \in R'$  and  $v \in B'$ . By the above observation, it suffices to probe only unmatched edges. Hence, we can again work with only one distance function  $c(r, b) + 1 - \sigma_r - \sigma_b$  for the purpose of computing  $\delta$ . As the  $+1$  term is common in all distances, we also drop that, and work with our old distance function.

In Section 5, we show how to construct and maintain Hungarian search data structure so that the task of maintaining closest pair can be performed in total  $O(n \cdot \text{poly}(\log n))$  time. Moreover, the data structure uses  $O(n \cdot \text{poly}(\log n))$  construction time and space.

► **Lemma 9.** *Using the MATCH and Hungarian search data structures, one can implement the Gabow-Tarjan algorithm on the instance  $I_1$  in time  $O(n^{3/2} \cdot \text{poly}(\log n))$ , i.e., a minimum cost bipartite perfect matching in  $I_1$  can be computed in time  $O(n^{3/2} \cdot \text{poly}(\log n))$ .*

## 5 Data Structures

### 5.1 The MATCH Data Structure

We would like to construct a data structure where given a matching at a fixed scale, a maximal set of augmenting paths can be computed efficiently. In particular, we are given the graph  $G = (R = R_0 \cup R_1, B = B_0 \cup B_1, E)$ , where  $R_1$  contains copies of the vertices in  $B_0$ ,  $B_1$  contains copies of the vertices in  $R_0$ .  $E = E_0 \cup E_1 \cup E_2 \cup E_3$ , where  $E_0 = R_0 \times B_0$ ,  $E_1 = R_1 \times B_1$ ,  $E_2 = \{(u, \hat{u}) \mid u \in R_0, \hat{u} \in B_1\}$ , and  $E_3 = \{(\hat{v}, v) \mid v \in B_0, \hat{v} \in R_1\}$ . Let  $n = |R| = |B|$ . Also, note that each edge weight is an integer  $w_i$  for  $0 \leq i \leq \lceil (c' \log n) / \epsilon \rceil$ , where  $c'$  is a constant. Let  $E_i^0$  be the set of edges in  $R_0 \times B_0$  with weights  $w_i$ . We define a bi-clique cover for each  $E_i^0$  as a collection  $\mathcal{C}_i = \{(P_{i1}, Q_{i1}), (P_{i2}, Q_{i2}), \dots, (P_{it(i)}, Q_{it(i)})\}$  where  $P_{ij} \subseteq R_0$ ,  $Q_{ij} \subseteq B_0$ , all the edges in  $P_{ij} \times Q_{ij}$  are in  $E_i^0$  and  $\bigcup_{j=1}^{t(i)} P_{ij} \times Q_{ij} = E_i^0$ . The size of  $E_i^0$  is  $\sum_{j=1}^{t(i)} |P_{ij}| + |Q_{ij}|$ . Given the points in  $R_0$  and  $B_0$ , using standard range searching data structures, one can compute such a bi-clique cover of size  $O((n/\epsilon) \log^2 n)$  in  $O((n/\epsilon) \log^2 n)$  time. We note that bi-clique covers are also used for the algorithm in [17]. Let  $\mathcal{C} = \mathcal{C}_i$ . Thus  $\mathcal{C}$  can be computed in  $O((n/\epsilon^2) \log^3 n)$  time and space.

In MATCH we store the bi-clique covers in  $\mathcal{C}$  corresponding to the edges in  $E_0$ . Also, we store the edges in  $E_2$  along with their weights in an array  $A_2$  indexed by the vertices of  $R_0$  and the edges in  $E_3$  and their weights in an array  $A_3$  indexed by the vertices of  $B_0$ . For finding an augmenting path efficiently, we need to store additional information. Before describing that, we describe in more detail how the augmenting paths are found.

The maximal set of augmenting paths are found by a careful implementation of depth first search. In this implementation, vertices can be labeled as marked. Initially all vertices are unmarked. We select any free unmarked vertex of  $R$  and initialize a path  $P$  at that vertex.  $P$  is extended from the last vertex  $u$  (in  $R$  as an invariant) as follows. We probe an eligible edge  $(u, v)$ . If  $v$  is already marked, the next eligible edge is considered. If no such



edge exists, the last two edges (one unmatched and one matched) are deleted from  $P$ . If  $P$  becomes empty, a new path is initialized. For the remaining cases, the following subroutine is called.

**AUGMENTING\_PATH**( $v$ ). If  $v$  is unmarked and free, we have found an augmenting path;  $v$  is marked,  $P$  is added to  $\mathcal{A}$  and a new path is initialized. In this case, return DONE. If  $v$  is unmarked, but matched with another vertex  $w$ ,  $(u, v)$  and  $(v, w)$  are added to  $P$ ;  $v, w$  are marked and the extension continues from  $w$  (in  $R$ ).

► **Observation 10** (\*). *In the above procedure, we always maintain the invariant that when we extend a path it suffices to probe only unmatched edges.*

Note that in the above procedure we cannot afford to probe all the unmatched edges. So, we have to implement the above step carefully. First, note that an edge  $(u, v)$  ( $u \in R$ ) is never scanned twice. When  $(u, v)$  is probed the first time, if  $v$  is unmarked, it becomes marked in all the cases. Also, once  $v$  is marked,  $(u, v)$  is never used to extend  $P$ . Thus, we can eliminate  $(u, v)$  from further probing.

From the above discussion, as  $E_2$  and  $E_3$  contain  $O(n)$  edges in total they can be probed in  $O(n)$  time. However,  $E_0$  and  $E_1$  contain  $\Omega(n^2)$  edges and thus for probing them we need specialized data structures. Next, we describe those.

Let  $c(u, v)$  be the weight of  $(u, v)$  at the current scale.  $y(u)$  is the dual value of the vertex  $u$  which remains fixed throughout the augmenting paths finding process. Again consider the bi-clique covers in  $\mathcal{C}$ . For each  $0 \leq i \leq \lceil (c' \log n)/\epsilon \rceil$ ,  $w_i$  denotes the weight of the edges in  $i$ -th class. Also, let  $\ell$  be the weight class to which the edges in  $E_1$  belong, i.e., all of their weights are  $w_\ell$ . For each such  $i$  and  $1 \leq j \leq t(i)$ , we store in  $match$  the vertices of  $Q_{ij}$  in a Red-Black tree  $T_{ij}$  with  $w_i + 1 - y(v)$  as the key of each such vertex  $v$ . Moreover, for each  $u \in R_0$ , we keep an ordered set of indexes  $I(u) = \{(i, j) \mid u \in P_{ij}\}$ . Similarly, define the index set  $I(v)$  for  $v \in B_0$ . We also store the vertices of  $B_1$  in a Red-Black tree  $T_1$  with  $w_\ell + 1 - y(v)$  as the key of  $v \in B_1$ .

► **Observation 11**. *MATCH uses  $O(n \cdot \text{poly}(\log n))$  construction time and space.*

The above space bound follows from the fact that the space complexity of MATCH is dominated by the space needed for the Red-Black trees, which is  $O(|B_1|) + \sum_{(i,j)} O(|Q_{ij}|) = O((n/\epsilon^2) \log^3 n)$ , as a Red-Black tree uses linear space. The time bound follows trivially.

**FIND\_MAXIMAL\_APS**. Let  $F$  be the set of free vertices and  $\Pi$  be the set of vertices that are already marked. Initially  $\Pi = \emptyset$ . Let  $\mathcal{A}$  be the set of augmenting paths found so far, which is initialized to  $\emptyset$ . For each vertex  $u \in R_1$ , set its  $E_1$ -failed flag to 0. While there is a vertex  $r_1 \in (R \cap F) \setminus \Pi$ , do the following.

- Initialize a path  $P$  at  $r_1$ .
- While  $P$  is not empty, do the following.
  - Let  $P = \{r_1, b_1, \dots, r_{\tau-1}, b_{\tau-1}, r_\tau\}$  be the current augmenting path that we need to extend.
  - (Case 1.  $r_\tau \in R_0$ ) Access the array  $A_2$  to find whether the copy of  $r_\tau$  in  $B_1$ , i.e.,  $\hat{r}_\tau$ , is marked and  $(r_\tau, \hat{r}_\tau)$  is eligible.
    - \* If  $\hat{r}_\tau$  is unmarked and  $(r_\tau, \hat{r}_\tau)$  is eligible, call the subroutine AUGMENTING\_PATH( $\hat{r}_\tau$ ). If this subroutine returns DONE, terminate this while loop. Otherwise, jump to the next iteration.



- \* Otherwise, search the Red-Black tree  $T_{ij}$  where  $(i, j)$  is the first index in  $I(r_\tau)$ , to find a vertex  $b_\tau$  with key value  $y(r_\tau)$ . If such a vertex  $b_\tau$  is found, call the subroutine  $\text{AUGMENTING\_PATH}(b_\tau)$ . Remove  $b_\tau$  from all the Red-Black trees with indexes in  $I(b_\tau)$ , as it is marked in the subroutine. If this subroutine returns DONE, terminate this while loop. Otherwise, jump to the next iteration. If no such vertex  $b_\tau$  is found in  $T_{ij}$ , remove the index  $(i, j)$  from  $I(r_\tau)$  and repeat the above step (performed for  $(i, j)$ ) for the next index in  $I(r_\tau)$ . If  $I(r_\tau)$  becomes empty, remove  $r_\tau$  and  $b_{\tau-1}$  from  $P$ , and continue to the next iteration.
- (Case 2.  $r_\tau \in R_1$ ) Access the array  $A_3$  to find whether the original copy of  $r_\tau$  in  $B_0$ , say  $v$ , is marked and  $(r_\tau, v)$  is eligible. If  $v$  is unmarked and  $(r_\tau, v)$  is eligible, call the subroutine  $\text{AUGMENTING\_PATH}(v)$ . If this subroutine returns DONE, terminate this while loop. Otherwise, jump to the next iteration. If  $E_1$ -failed is not set, i.e., it is 0 for  $r_\tau$ , search the Red-Black tree  $T_1$  to find a vertex  $b_\tau$  with key value  $y(r_\tau)$ . If such a vertex  $b_\tau$  is found, call the subroutine  $\text{AUGMENTING\_PATH}(b_\tau)$ . Remove  $b_\tau$  from  $T_1$ , as it is marked in the subroutine. If this subroutine returns DONE, terminate this while loop. Otherwise, jump to the next iteration. If no such vertex  $b_\tau$  is found in  $T_1$ , set  $E_1$ -failed flag for  $r_\tau$  to 1, remove  $r_\tau$  and  $b_{\tau-1}$  from  $P$ , and continue to the next iteration.

The above procedure is self-explanatory. Next, we prove its correctness and bound the implementation time.

► **Lemma 12** (\*). *FIND\_MAXIMAL\_APS correctly computes a maximal set of disjoint augmenting paths.*

► **Lemma 13** (\*). *FIND\_MAXIMAL\_APS can be implemented in  $O(n \cdot \text{poly}(\log n))$  time.*

## 5.2 Hungarian Search Data Structure

Recall that in Hungarian search, we need to maintain two sets  $R' \subseteq R$  and  $B' \subseteq B$ . Initially,  $R' = \emptyset$  and  $B' = B$ . In each iteration, a vertex is added to  $R'$  and removed from  $B'$ . Additionally, each vertex  $v$  has a weight  $\sigma_v$ . In every iteration, the goal is to maintain the bichromatic closest pair, which is the pair  $(u, v) \in R' \times B'$  with the minimum  $c(u, v) - \sigma_u - \sigma_v$  value.

Consider the bi-clique cover  $\mathcal{C}$  and in particular a pair  $(P_{ij}, Q_{ij})$  in  $\mathcal{C}$ . Then any edge in  $P_{ij} \times Q_{ij}$  has the same weight  $w_i$ . Thus, the pair  $(u, v)$  with the maximum  $\sigma_u + \sigma_v$  value has the minimum  $c(u, v) - \sigma_u - \sigma_v$  value in  $P_{ij} \times Q_{ij}$ . In other words, it is sufficient to keep track of a vertex  $u \in P_{ij}$  with the maximum  $\sigma_u$  value and a vertex  $v \in Q_{ij}$  with the maximum  $\sigma_v$  value.

Now, we describe the Hungarian Search data structure. For each index  $(i, j)$  in  $\mathcal{C}$ , we construct two max-heaps  $H_{ij}^r$  and  $H_{ij}^b$ , for  $P_{ij} \cap R'$  and  $Q_{ij} \cap B'$ , respectively. Initially,  $H_{ij}^r$  is empty and  $H_{ij}^b$  contains all the vertices in  $Q_{ij}$ . The key value of each vertex  $v$  is its weight  $\sigma_v$ . Note that using  $H_{ij}^r$  and  $H_{ij}^b$ , the pair  $(u, v) \in (P_{ij} \cap R') \times (Q_{ij} \cap B')$  with the maximum  $\sigma_u + \sigma_v$  value can be found in  $O(1)$  time. Similarly, we construct two heaps  $H_1^r$  and  $H_1^b$  for the vertices in  $R_1 \cap R'$  and  $B_1 \cap B'$ , respectively. Initially,  $H_1^r$  is empty and  $H_1^b$  contains all the vertices in  $B_1$ . The key value of each vertex  $v$  is its weight  $\sigma_v$ . Again using  $H_1^r$  and  $H_1^b$ , the pair  $(u, v) \in (R_1 \cap R') \times (B_1 \cap B')$  with the maximum  $\sigma_u + \sigma_v$  value can be found in  $O(1)$  time. We also construct another max-heap  $H_{23}$  to store the edges in  $E_2 \cup E_3$ . Initially, it is empty. In every iteration, it contains only those edges  $(u, v)$  such that  $u \in R'$  and

$v \in B'$ . Finally, to keep track of the global maximum pair, we create another max-heap  $H$ .  $H$  stores a maximum pair of  $(P_{ij} \cap R') \times (Q_{ij} \cap B')$  for each index  $(i, j)$  and maximum pairs of  $(R_1 \cap R') \times (B_1 \cap B')$  and  $H_{23}$ .

Note that the space complexity of the data structure is  $\sum_{(i,j)} O(|P_{ij}| + |Q_{ij}|) + O(n) = O((n/\epsilon^2) \log^3 n)$ . This follows, as a max-heap uses linear space.

► **Observation 14.** *The Hungarian Search data structure uses  $O(n \cdot \text{poly}(\log n))$  construction time and space.*

Next, we describe a procedure which will help us implement Hungarian search. This procedure takes a pair  $(u, v)$  as input, where  $u \in R \setminus R'$  and  $v \in B'$ . We need to add  $u$  to  $R'$  and remove  $v$  from  $B'$  while maintaining the correct maximum pair.

**UPDATE\_CLOSEST\_PAIR( $u, v$ ).** If  $v \in B_0$ , remove  $v$  from all  $H_{ij}^b$  that contains  $v$ . Also remove  $(\hat{v}, v)$  from  $H_{23}$  if its in  $H_{23}$ . If  $v \in B_1$ , remove  $v$  from  $H_1^b$ . Also remove the edge  $(u', v)$  with  $u' \in R_0$  from  $H_{23}$  if its in  $H_{23}$ . If  $u \in R_0$ , add  $u$  to all  $H_{ij}^r$  such that  $P_{ij}$  contains  $u$ . Also add the edge  $(u, \hat{u})$  to  $H_{23}$  if  $\hat{u} \in B'$ . If  $u \in R_1$ , add  $u$  to  $H_1^r$ . Also add the edge  $(u, v')$  with  $v' \in B_0$  to  $H_{23}$  if  $v' \in B'$ . Update the maximum pairs in  $H$  accordingly by selecting the updated maximum pairs from the other max-heaps.

► **Lemma 15** (\*). *Hungarian search can be performed in  $O(n \cdot \text{poly}(\log n))$  time.*

---

## References

- 1 Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):912–953, 1999.
- 2 Mudabir Kabir Asathulla, Sanjeev Khanna, Nathaniel Lahn, and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite perfect matching in planar graphs. *ACM Trans. Algorithms*, 16(1):2:1–2:30, 2020.
- 3 Justin Colannino, Mirela Damian, Ferran Hurtado, Stefan Langerman, Henk Meijer, Suneeta Ramaswami, Diane L. Souvaine, and Godfried Toussaint. Efficient many-to-many point matching in one dimension. *Graphs Comb.*, 23(Supplement-1):169–178, 2007. doi:10.1007/s00373-007-0714-3.
- 4 Thomas Eiter and Heikki Mannila. Distance measures for point sets and their computation. *Acta informatica*, 34(2):109–133, 1997.
- 5 S. M. Ferdous, Alex Pothén, and Arif Khan. New approximation algorithms for minimum weighted edge cover. In Fredrik Manne, Peter Sanders, and Sivan Toledo, editors, *Proceedings of the Eighth SIAM Workshop on Combinatorial Scientific Computing, CSC 2018, Bergen, Norway, June 6-8, 2018*, pages 97–108. SIAM, 2018. doi:10.1137/1.9781611975215.10.
- 6 Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.
- 7 John E Hopcroft and Richard M Karp. An  $n^{5/2}$  algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.
- 8 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64(3):838–904, 2020.
- 9 Judith Keijsper and Rudi Pendavingh. An efficient algorithm for minimum-weight bibranching. *Journal of Combinatorial Theory, Series B*, 73(2):130–145, 1998.
- 10 Harold W Kuhn. Variants of the Hungarian method for assignment problems. *Naval Research Logistics Quarterly*, 3(4):253–258, 1956.
- 11 Sharath Raghvendra and Pankaj K. Agarwal. A near-linear time  $\epsilon$ -approximation algorithm for geometric bipartite matching. *J. ACM*, 67(3):18:1–18:19, 2020.


#### 44:14 Algorithms for Many-To-Many Point Matching in the Plane

- 12 Fatemeh Rajabi-Alni and Alireza Bagheri. An  $O(n^2)$  algorithm for the limited-capacity many-to-many point matching in one dimension. *Algorithmica*, 76(2):381–400, 2016. doi: 10.1007/s00453-015-0044-4.
- 13 Fatemeh Rajabi-Alni and Alireza Bagheri. A fast and efficient algorithm for many-to-many matching of points with demands in one dimension. *CoRR*, abs/1904.05184, 2019. arXiv:1904.05184.
- 14 Fatemeh Rajabi-Alni and Alireza Bagheri. A faster algorithm for the limited-capacity many-to-many point matching in one dimension. *CoRR*, abs/1904.03015, 2019. arXiv:1904.03015.
- 15 A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.
- 16 Pravin M Vaidya. Geometry helps in matching. *SIAM Journal on Computing*, 18(6):1201–1225, 1989.
- 17 Kasturi R Varadarajan and Pankaj K Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *SODA*, volume 99, pages 805–814, 1999.

# Augmenting Graphs to Minimize the Radius

Joachim Gudmundsson 

The University of Sydney, Australia

Yuan Sha 

The University of Sydney, Australia

Fan Yao

The University of Sydney, Australia

---

## Abstract

We study the problem of augmenting a metric graph by adding  $k$  edges while minimizing the radius of the augmented graph. We give a simple 3-approximation algorithm and show that there is no polynomial-time  $(5/3 - \epsilon)$ -approximation algorithm, for any  $\epsilon > 0$ , unless  $P = NP$ .

We also give two exact algorithms for the special case when the input graph is a tree, one of which is generalized to handle metric graphs with bounded treewidth.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Design and analysis of algorithms

**Keywords and phrases** graph augmentation, radius, approximation algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.45

## 1 Introduction

We study the problem of minimizing the radius of a metric graph by inserting  $k$  new edges.

Let  $G = (V, E)$  be a non-negative weighted graph with  $n$  vertices, and  $V^2$  be the set of all possible edges on  $V$ . A non-edge of  $G$ , also referred to as shortcut, is an edge in  $\bar{E} = V^2 \setminus E$ . The graph distance  $d_G(u, v)$  between two vertices  $u, v \in V$  is the smallest weight of any path in  $G$  joining  $u$  and  $v$ . The eccentricity of a vertex  $v \in V$  is the maximum graph distance between  $v$  and any vertex in  $V$ . The radius of  $G$  is the minimum eccentricity of any vertex in  $G$  and the center of  $G$  is a vertex with minimum eccentricity.

The radius of a graph is closely related to the diameter. The diameter of a graph is the maximum graph distance between any pair of vertices. The problem of minimizing the diameter of a graph by adding  $k$  new edges has been extensively studied [2, 6, 7, 8, 9, 14] and has applications in areas such as communication networks, information networks, flight scheduling and protein interaction.

In the general case each added edge may also have a cost, and the edge augmentation problem for minimizing the radius or diameter can then be seen as a bicriteria optimization problem. The two criteria are: (1) the total cost of the added edges, and (2) the radius or diameter of the augmented graph. A bicriteria optimization problem is then either (1) given a budget on the total cost of the added edges, minimize the radius or diameter, or (2) given a target on the radius or diameter of the augmented graph, minimize the total cost of the added edges. For radius, the first bicriteria optimization problem is formally defined as follows.

- **PROBLEM:** Bounded Cost Minimum Radius Edge Addition (BCMR)
- **INPUT:** An undirected graph  $G = (V, E)$  with weight function  $\ell : V^2 \rightarrow \mathbb{R}^+ \cup \{0\}$ , a cost function  $\varsigma : \bar{E} \rightarrow \mathbb{R}^+ \cup \{0\}$  and a positive number  $B$ .
- **GOAL:** Add a set  $F \subseteq \bar{E}$  with  $\sum_{e \in F} \varsigma(e) \leq B$  such that the radius of  $\hat{G} = (V, E \cup F)$  is minimized.



© Joachim Gudmundsson, Yuan Sha, and Fan Yao;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 45; pp. 45:1–45:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider the BCMR problem restricted to the special case when the costs are unit and the weights satisfy the triangle inequality. That is, given a metric graph  $G = (V, E, \ell)$ , add  $k = \lfloor B \rfloor$  shortcuts to  $G$  to minimize the radius of the augmented graph. We will refer to our restricted variant of the BCMR problem as the metric BCMR problem.

### Related results

To the best of the authors' knowledge there is only one paper on the BCMR problem. Johnson and Wang [13] showed a linear-time algorithm for the special case when  $k = 1$  and  $G$  is a path embedded in a metric space. They considered the continuous version where the center of the graph can be in the interior of an edge. However, the closely related problem of minimizing the diameter of the augmented graph has a long and rich history.

The problem of minimizing the diameter of the augmented graph such that the total cost of the shortcuts is within a budget (referred to as BCMD) and the problem of minimizing the cost of added shortcuts such that the diameter of the augmented graph is within a given value (referred to as BDMC) were shown to be NP-hard in [16] and  $W[2]$ -hard in [9, 11]. Li et al. [14] gave a constant factor approximation algorithm for the BCMD problem restricted to unit weights and unit costs. Later Bilò et al. [2] improved the analysis of Li et al.'s algorithm. Bilò et al.'s analysis also implies that Li et al.'s algorithm gives a 4-approximation for BCMD on metric graphs. Demaine and Zadimoghaddam [7] considered a variant of BCMD where the costs are unit and all non-edges have length  $\delta$ , where  $\delta$  is a small constant compared to the diameter of the graph. We will refer to this model as the DZ model. Demaine and Zadimoghaddam gave a constant factor approximation algorithm for the BCMD problem in the DZ model, which was later improved by Bilò et al. [2]. For the general BCMD problem where the weights and costs are arbitrary integers, Frati et al. [9] gave a 4-approximate Fixed-Parameter Tractable (FPT) algorithm (under the cost parameter). More restricted variants of the BCMD problem have also been considered in the literature, where the input graph is either a path or a tree and one shortcut is added. Section 1.1 in [13] gives a nice overview of these results.

Compared to the BCMD problem, the BDMC problem is traditionally harder to approximate. Dodis and Khanna [8] studied the BDMC problem in depth and gave extensive inapproximability results for different weight and cost functions. They also gave almost matching upper bounds for these variants of the BDMC problem.

### Our results

In this paper we study the metric BCMR problem. Our main results are:

1. A simple  $O(kn(m+n \log n))$  time 3-approximation algorithm and a  $(5/3-\epsilon)$  approximation hardness bound, even for metric BCMR on geometric graph<sup>1</sup>.
2. An exact  $O(n^3 \log n)$  time algorithm for metric BCMR when the input graph is a tree and an exact  $O((k^2b^2 + kb^32^b) \cdot n^{2b+2})$  time algorithm when the input graph has treewidth  $b - 1$ .

Our second result shows that the metric BCMR problem on trees is in  $P$ , while whether the BCMD problem on trees is in  $P$  is still an open problem, even for unit weights and unit costs.

---

<sup>1</sup> A geometric graph is a graph where all vertices are embedded in the Euclidean space.

Similar to Lemma 3 in [7], we can prove that any  $\alpha$ -approximation for the BCMR problem is a  $2\alpha$ -approximation for the corresponding BCMD problem. Thus our algorithm for BCMR on metric graphs with bounded treewidth is a 2-approximation for BCMD on metric graphs with bounded treewidth, which is slightly better than Li et al.'s 4-approximation (however, their result holds for metric graphs).

## Paper organization

The rest of the paper is organized as follows. The 3-approximation algorithm for the metric BCMR problem is presented in Section 2 and the approximation hardness results are given in Section 3. Section 4 describes our algorithms for the metric BCMR problem on trees. Finally, we present an algorithm on graphs with bounded treewidth in Section 5.

## 2 3-approximation algorithm

Let  $G = (V, E, \ell)$  and  $k$  be an instance of the metric BCMR problem. Let  $F^*$  be an optimal solution for the instance, and let  $c^*$  be a center of  $G^* = (V, E \cup F^*, \ell)$ .

If the center  $c^*$  is known, then the metric BCMR problem is just adding  $k$  shortcuts to minimize the eccentricity of  $c^*$ . However, since  $c^*$  is not known, we will just try every vertex in  $V$  as a candidate. We define the Graph Augmentation Eccentricity Minimization problem, GAEM for short, as follows: given a metric graph  $G = (V, E, \ell)$ , an integer  $k$  and a vertex  $s$  in  $V$ , add  $k$  shortcuts in  $\bar{E}$  to  $G$  such that the eccentricity of  $s$  is minimized. Obviously solving GAEM for every vertex in  $V$  gives us an optimal solution to the metric BCMR problem. Thus in the rest of this paper we will focus our attention on solving the GAEM problem.

The following lemma gives an important property for the GAEM problem, which we will utilize throughout the paper. The proof can be found in Appendix A.

► **Lemma 1.** *Given a metric graph  $G = (V, E, \ell)$ , an integer  $k$  and a vertex  $s$  in  $V$  for the GAEM problem, there is an optimal solution where every shortcut is incident to  $s$ .*

Lemma 1 allows us to only consider solutions in which all the shortcuts are incident to  $s$ . It also applies when we consider approximate solutions. Our 3-approximation algorithm uses the well-known farthest-first traversal technique popularized by Gonzalez [12]. Next we state the approximation algorithm.

■ **Algorithm 1** FarthestAdditionGAEM( $G, k, s$ ).

---

**Require:** A metric graph  $G = (V, E, \ell)$ , an integer  $k$  and a vertex  $s$  in  $V$ .

**Ensure:** An approximate optimal solution for the GAEM instance.

```

1:  $\hat{G} \leftarrow G$ 
2: for  $i \leftarrow 1$  to  $k$  do
3:   find the vertex  $u$  farthest from  $s$  in  $\hat{G}$ .
4:   add  $(s, u)$  to  $\hat{G}$ .
5: end for
6: return  $\hat{G}$ 

```

---

The approximation bound of Algorithm 1 is given in Lemma 2. The proof is found in Appendix B.

► **Lemma 2.** *Algorithm 1 is a 3-approximation algorithm for the GAEM problem.*

## 45:4 Augmenting Graphs to Minimize the Radius

Algorithm 1 is very simple and can be implemented using standard graph algorithms. Let  $n = |V|$  and  $m = |E|$ . In each iteration of the for loop, we use Dijkstra's algorithm to compute the single-source shortest paths from  $s$  to all other vertices in  $V$ . Using Fibonacci heaps [10], Dijkstra's algorithm runs in time  $O(m' + n \log n)$ , where  $m' < m + k$ . Without loss of generality, we may assume that  $k \leq n - 1 \leq m$ . The total running time is thus  $O(km + kn \log n)$ .

Running Algorithm 1 for  $n$  times gives our main result in this section.

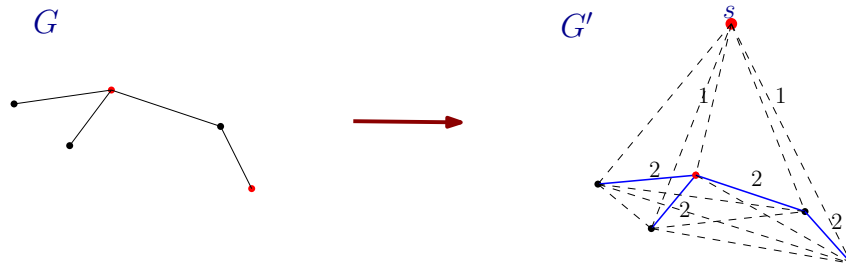
► **Theorem 3.** *There is an  $O(kn(m + n \log n))$  time, 3-approximation algorithm for the metric BCMR problem.*

When the input graph is unweighted or in the DZ model, arguments similar to the proof of Lemma 2 give an approximation factor of 2. For unweighted graphs, the 3-approximation algorithm runs in  $O(knm)$  time by using BFS instead of Dijkstra's algorithm in Step 3 of Algorithm 1. Note that for general metric graphs we cannot use BFS in Step 3.

### 3 Approximation hardness

We show the approximation hardness of the metric BCMR problem by showing the approximation hardness of the GAEM problem. This follows from the fact that any polynomial time  $\alpha$ -approximation algorithm for GAEM will trivially lead to a polynomial time  $\alpha$ -approximation algorithm for the metric BCMR problem. The hardness proof is done using a reduction from the Dominating Set problem.

Given a graph  $G = (V, E)$  and an integer  $k$ , the Dominating Set decision problem asks whether there is a subset  $S \subset V$  of size  $k$  such that every vertex not in  $S$  is adjacent to a vertex in  $S$ . The GAEM decision problem is: given a metric graph  $G = (V, E, \ell)$ , a vertex  $s \in V$ , an integer  $k$  and a target value  $r$ , decide if there is a set  $F$  of  $k$  shortcuts such that their addition to  $G$  reduces the eccentricity of  $s$  to at most  $r$ .



■ **Figure 1** Reduction from the Dominating Set decision problem. Edges in  $E'$  are colored blue and have weight 2. All shortcuts are dashed and have weight 1.

► **Theorem 4.** *For any  $\epsilon > 0$ , finding a  $(\frac{5}{3} - \epsilon)$  approximate solution for the GAEM problem is NP-hard.*

**Proof.** Let  $G = (V, E)$  be a graph. Let  $I = (G, k)$  be an instance of the Dominating Set decision problem. We transform  $I$  into an instance  $I' = (G', s, k, 3)$  of the GAEM decision problem by adding an extra vertex  $s$ , that is,  $G' = (V' = V \cup \{s\}, E' = E, \ell)$ . We set the weight function  $\ell$  as: the weight of any shortcut in  $[V']^2 \setminus E'$  is 1 and the weight of any edge in  $E'$  is 2. See Figure 1. Note that  $\ell$  satisfies triangle inequality.



If there is a dominating set  $S \subset V$  of size  $k$  in  $G$ , we can choose the same  $k$  vertices in  $G'$  and add  $k$  shortcuts from  $s$  to every vertex in this subset. It follows that every vertex in  $V$  is connected to  $s$  by 1 shortcut and at most 1 edge in  $E$ . The eccentricity of  $s$  is thus at most 3.

If we can add  $k$  shortcuts to  $G'$  such that the eccentricity of  $s$  in the resulting graph is at most 3, we can add such  $k$  shortcuts which are all incident to  $s$ , by Lemma 1. For such  $k$  shortcuts, all other vertices of the shortcuts except  $s$  form a subset  $S' \subset V$  of size  $k$ . Since the eccentricity of  $s$  is at most 3, every vertex in  $V \setminus S'$  must be adjacent to a vertex in  $S'$  by an edge in  $E$ . Thus  $S'$  is dominating set of  $G$ .

Any vertex that is connected to  $s$  through 1 shortcut and 1 edge in  $E$  has distance 3 to  $s$ . Any vertex that is connected to  $s$  through 1 shortcut and 2 edges in  $E$  has distance 5 to  $s$ . If there is a  $(\frac{5}{3} - \epsilon)$  approximation algorithm for the GAEM problem then we can use it to solve  $I'$  and thus any dominating set instance. Thus finding a  $(\frac{5}{3} - \epsilon)$  approximate solution for the GAEM problem is NP-hard. ◀

The same construction can be used for the DZ model while only modifying the length function. Assuming the length of all non-edges to be very small, we get:

► **Lemma 5.** *For any  $\epsilon > 0$ , finding a  $(2 - \epsilon)$  approximate solution for the GAEM problem in the DZ model is NP-hard.*

Note that this is a tight lower bound since Algorithm 1 is a 2-approximation algorithm for the GAEM problem in the DZ model.

Somewhat surprising is that the GAEM problem on geometric graphs (the weights are Euclidean distances) is as hard to approximate as the GAEM problem. This is stated in Lemma 6 and the proof is found in Appendix C.

► **Lemma 6.** *For any  $\epsilon > 0$ , finding a  $(\frac{5}{3} - \epsilon)$  approximate solution for the GAEM problem on geometric graphs is NP-hard.*

## 4 The GAEM problem on metric trees

In this section we consider the GAEM problem on trees. We give two algorithms: (1) a near quadratic-time algorithm, and (2) a slower algorithm with running time  $O(\min\{k^2n^3, n^4\})$ . The first algorithm is more efficient and elegant, however, we are not able to generalize it to graphs of bounded treewidth. Therefore we will describe the second algorithm below (Section 4.1) while the description of the first algorithm can be found in Appendix D. We state the result of the first algorithm here:

► **Theorem 7.** *The GAEM problem on trees can be solved in  $O(n^2 \log n)$  time.*

Note that while Theorem 7 proves that the metric BCMR problem on trees can be solved in polynomial time, the complexity status for the BCMR problem on trees remains open, even for unit weights and unit costs.

Throughout this section we will assume that the input tree has degree at most 3. Otherwise, we transform it into a tree of degree 3 by splitting vertices with degree greater than 3. When a vertex of degree 3 is picked as the root, we split the vertex and get a binary tree. In the following, we assume that the input is a binary tree rooted at  $s$ .

### 4.1 The second algorithm for trees

By Lemma 1 all the shortcuts we add are incident to  $s$ . For a binary tree  $T = (V, E, \ell)$  rooted at  $s$ , we aim to add  $k$  shortcuts all incident to  $s$  so that the eccentricity of  $s$  in the augmented graph is minimized.

Our algorithm is a dynamic programming approach with three parameters. Before we define the subproblems we need to define some notations.

For a vertex  $v$  in  $V$ , let  $lc(v)$  and  $rc(v)$  denote the left and right child of  $v$ , respectively. Let  $T_v$  denote the subtree of  $T$  rooted at  $v$ . Let  $D^\uparrow(v) = \bigcup_{u \in T \setminus T_v} \{d_T(v, u) + |us|\}$  and let  $D^\downarrow(v) = \bigcup_{u \in T_v} \{d_T(v, u) + |us|\}$ . Each distance in  $D^\uparrow(v)$  is the weight of a tree path from  $v$  to a vertex  $u$  in  $T \setminus T_v$ , plus the shortcut  $(u, s)$ . The distances in  $D^\downarrow(v)$  are defined in the same way but through a vertex  $u$  in  $T_v$ . This is illustrated in Fig. 2(a), where the weight of the blue path is in  $D^\uparrow(v)$  and the weight of the red path is in  $D^\downarrow(v)$ . Note that the number of distances in  $D^\downarrow(v)$  is  $|T_v|$  and the number of distances in  $D^\uparrow(v)$  is  $|T \setminus T_v|$ . In our recursion we will need the subset  $D^\downarrow(v, d)$  of  $D^\downarrow(v)$  containing all distances in  $D^\downarrow(v)$  that are at least  $d$ .

For a subtree  $T_v$  and three parameters  $d^\uparrow$ ,  $d^\downarrow$  and  $\bar{k}$ , the subproblem is to find a set of  $\bar{k}$  shortcuts from  $s$  to vertices in  $T_v$  so that the maximum distance from vertices in  $T_v$  to  $s$  is minimized. The parameter  $d^\uparrow$  is the (assumed) weight of the shortest path from  $v$  to  $s$  via the parent of  $v$  (thus through a shortcut added to a vertex out of  $T_v$ ), and the parameter  $d^\downarrow$  is the (assumed) weight of the shortest path from  $v$  to  $s$  *not* going through the parent of  $v$  (thus through one of the  $\bar{k}$  shortcuts). Note that  $d^\uparrow$  is in  $D^\uparrow(v)$  and its path goes through shortcut  $(v^\uparrow, s)$  for some vertex  $v^\uparrow$  in  $T \setminus T_v$ . Also  $d^\downarrow$  is in  $D^\downarrow(v)$  and its path goes through shortcut  $(v^\downarrow, s)$  for some vertex  $v^\downarrow$  in  $T_v$ .

For given  $d^\uparrow$ ,  $d^\downarrow$  and  $\bar{k}$ , let  $R_v[d^\uparrow][d^\downarrow][\bar{k}]$  denote the minimum maximum distance from a vertex in  $T_v$  to  $s$ , when  $\bar{k}$  shortcuts are allowed to be added from  $s$  to vertices in  $T_v$ .

We have two base cases. If  $\bar{k} = 0$  then, by definition,  $d^\downarrow = \infty$  and

$$R_v[d^\uparrow][d^\downarrow][0] = d^\uparrow + \max_{u \in T_v} \{d_T(u, v)\},$$

and if  $v$  is a leaf and  $\bar{k} = 1$  then

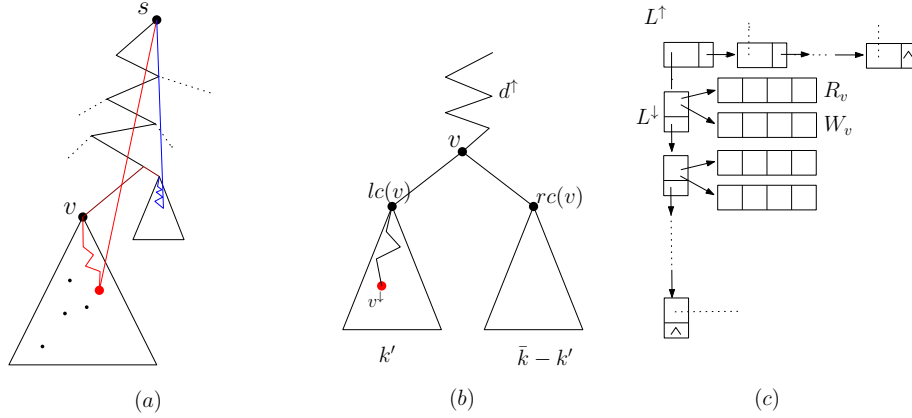
$$R_v[d^\uparrow][d^\downarrow][1] = |vs|.$$

Next we state the recursion. Depending on the location of  $v^\downarrow$ , we have three different cases for  $\bar{k} > 0$ : (a)  $v^\downarrow$  lies in  $T_{lc(v)}$ , (b)  $v^\downarrow$  lies in  $T_{rc(v)}$ , or (c)  $v = v^\downarrow$  which implies that there is a shortcut connecting  $s$  to  $v$ . In all three cases, the allowed  $\bar{k}$  shortcuts are split between  $T_{lc(v)}$  and  $T_{rc(v)}$ .

**Case (a):  $v^\downarrow$  lies in  $T_{lc(v)}$ .**

$$R_v[d^\uparrow][d^\downarrow][\bar{k}] = \min_{1 \leq k' \leq \bar{k}} \{ \max\{R_{lc(v)}[d^\uparrow + |v, lc(v)|][d^\downarrow - |v, lc(v)|][k']\}, \\ \min_{d \in D^\downarrow(rc(v), d^\downarrow - |v, rc(v)|)} \{R_{rc(v)}[\min\{d^\uparrow, d^\downarrow\} + |v, rc(v)|][d][\bar{k} - k']\}, \min\{d^\uparrow, d^\downarrow\} \},$$

where  $\hat{k} = \min\{\bar{k}, |T_{lc(v)}|\}$ . Note that  $k'$  shortcuts are added to vertices in  $T_{lc(v)}$  and  $\bar{k} - k'$  shortcuts are added to vertices in  $T_{rc(v)}$ . Since  $v^\downarrow$  lies in  $T_{lc(v)}$ , the second parameter to  $R_{lc(v)}$  equals  $d^\downarrow - |v, lc(v)|$  and the second parameter to  $R_{rc(v)}$  is at least  $d^\downarrow - |v, rc(v)|$ . See Figure 2(b). Finally,  $\min\{d^\uparrow, d^\downarrow\}$  is the distance from  $v$  to  $s$ .



■ **Figure 2** (a) Shortest paths from  $v$  to  $s$ . (b)  $v^\downarrow$  lies in  $T_{lc(v)}$ . (c) Illustrating the list structures defined in Section 4.1.1.

**Case (b):  $v^\downarrow$  lies in  $T_{rc(v)}$ .**

$$R_v[d^\uparrow][d^\downarrow][\bar{k}] = \min_{1 \leq k' \leq \bar{k}} \{ \max\{R_{rc(v)}[d^\uparrow + |v, rc(v)|][d^\downarrow - |v, rc(v)|][k'],$$

$$\min_{d \in D^\downarrow(lc(v), d^\downarrow - |v, lc(v)|)} \{R_{lc(v)}[\min\{d^\uparrow, d^\downarrow\} + |v, lc(v)|][d][\bar{k} - k']\}, \min\{d^\uparrow, d^\downarrow\} \},$$

where  $\hat{k} = \min\{\bar{k}, |T_{rc(v)}|\}$ . The formula is symmetric to the formula for Case (a).

**Case (c):  $v = v^\downarrow$ .**

$$R_v[d^\uparrow][d^\downarrow][\bar{k}] = \min_{0 \leq k' \leq \bar{k}} \{ \max\{ \min_{d_l \in D^\downarrow(lc(v), |s, lc(v)|)} \{R_{lc(v)}[|s, v| + |v, lc(v)|][d_l][k']\},$$

$$\min_{d_r \in D^\downarrow(rc(v), |s, rc(v)|)} \{R_{rc(v)}[|s, v| + |v, rc(v)|][d_r][\bar{k} - k' - 1]\}, |vs| \} \},$$

where  $\hat{k} = \min\{\bar{k} - 1, |T_{lc(v)}|\}$ . The distance from  $v$  to  $s$  equals  $|vs|$ .

### 4.1.1 Data structures for DP

For efficient computation we define:

$$W_v[d^\uparrow][d^\downarrow][\bar{k}] = \min_{d \in D^\downarrow(v, d^\downarrow)} \{R_v[d^\uparrow][d][\bar{k}]\}.$$

Then the  $\min_{d \in D^\downarrow(\dots)}$  terms in the formulae of  $R_v$  in the last subsection are replaced by  $W_v$  terms. By definition,  $W_v$  can be computed from  $R_v$  in an ordered manner.

For efficiency, we will use the following list structure for  $R_v$  and  $W_v$  in the dynamic programming steps. The first level is a list,  $L^\uparrow$ , ordered on the values of  $d^\uparrow$ . Each node in  $L^\uparrow$  contains a second level list, denoted  $L^\downarrow$ , ordered on the values of  $d^\downarrow$ . And each node in  $L^\downarrow$  contains two arrays of size at most  $k$ , one for  $R_v$  and one for  $W_v$ . See Fig. 2(c).

The dynamic programming is done in a bottom-up manner. At a leaf node  $v$ ,  $d^\uparrow$  has  $n - 1$  values and  $d_v^\downarrow$  has one value. We perform a tree traversal starting from  $v$  to get all the values of  $d^\uparrow$ , and order these values in  $L^\uparrow$ . For each  $d^\uparrow$  value, build an  $L^\downarrow$  list containing one node. At an internal node  $v$ , the values of its  $L^\uparrow$  are obtained from the  $L^\downarrow$  of its right child and the  $L^\uparrow$  of its left child. Its  $L^\downarrow$  is formed by merging the  $L^\downarrow$  lists of its left and right child. The array elements of  $R_v$  are computed by using the formulae in the last subsection. Array elements of  $W_v$  are computed accumulatively from the array elements of  $R_v$ .

To get the shortcuts of an optimal solution, we need to keep some additional information. At a node  $v$ , the  $L^\uparrow$  list also stores the vertex  $v^\uparrow$  for each  $d^\uparrow$  value. The  $L^\downarrow$  list also stores the  $v^\downarrow$  for each  $d^\downarrow$  value. At an internal node  $v$ , when we use the formulae of  $R_v$  to compute an array element, we also keep track of the array elements of  $lc(v)$ 's structure and  $rc(v)$ 's structure that together give the solution for the current array element of  $R_v$ . When using the formula of  $W_v$ , we keep track of the array element of  $R_v$  that gives the value for the current array element of  $W_v$ . We can then backtrack with the above information to get the shortcuts of the optimal solution. The extra information we kept is constant per array element.

### 4.1.2 Running time

To analyze the running time, we first give two bounds. Their proofs are available in Appendix E.  $T$  is a rooted binary tree with  $n$  vertices.

► **Lemma 8.**  $\sum_{v \in T} |T_{lc(v)}| \cdot |T_{rc(v)}| \leq n^2$ .

► **Corollary 9.**  $\sum_{v \in T} |T_v| \cdot \min\{|T_{lc(v)}|, |T_{rc(v)}|\} \leq 2n^2 + n \log n$ .

At a leaf node  $v$ , the tree traversal for computing values of  $d^\uparrow$  takes  $O(n)$  time. Sorting these values takes  $O(n \log n)$  time. Plus computing  $R_v$  and  $W_v$ , we spend  $O(n \log n)$  time in total. At an internal node  $v$ ,  $O(n)$  time is spent on obtaining the sorted  $d^\uparrow$  values for  $L^\uparrow$  and the sorted  $d^\downarrow$  values for  $L^\downarrow$ .  $0 \leq k' \leq \min\{\bar{k}, |T_{vs}|\}$ , where  $|T_{vs}| = \min\{|T_{lc(v)}|, |T_{rc(v)}|\}$ .  $\bar{k} \leq \min\{k, |T_v|\}$ . Thus computing the array elements of  $R_v$  for given  $d^\uparrow$  and  $d^\downarrow$  takes  $O(\min\{k^2, |T_v| \cdot |T_{vs}|\})$  time. Both  $d^\uparrow$  and  $d^\downarrow$  have at most  $n$  values. Using  $O(k^2)$  as the upper bound gives a total running time of  $O(k^2 n^3)$ . Using  $O(|T_v| \cdot |T_{vs}|)$  as the upper bound gives a total running time  $\sum_{v \in T} n^2 \cdot |T_v| \cdot |T_{vs}| = O(n^4)$ , by Corollary 9. So the total running time is  $O(\min\{k^2 n^3, n^4\})$ . To use less space, we can use *depth first search*. The space requirement is thus  $O(k^2 n^2)$ , and we get:

► **Theorem 10.** *When the input graph is a tree embedded in a metric space, GAEM can be solved in  $O(\min\{k^2 n^3, n^4\})$  time, using  $O(k^2 n^2)$  storage.*

## 5 Metric BCMR on graphs with bounded treewidth

Treewidth, introduced by Robertson and Seymour [15], measures how similar a graph is to a tree. Many NP-hard graph problems can be solved efficiently when the input graph is restricted to graphs with bounded treewidth [1, 3]. The notion of treewidth is based on the notion of tree decomposition of a graph.

► **Definition 11** ([15]). *A tree decomposition of a graph  $G = (V, E)$ , denoted by  $TD(G)$ , is a pair  $(X, T)$  in which  $T = (I, F)$  is a tree and  $X = \{X_i | i \in I\}$  is a family of subsets of  $V(G)$  such that:*

1.  $\bigcup_{i \in I} X_i = V$ ;
2. for each edge  $e = \{u, v\} \in E$  there exists an  $i \in I$  such that both  $u$  and  $v$  belong to  $X_i$ ;  
and
3. for all  $v \in V$ , the set of nodes  $\{i \in I | v \in X_i\}$  forms a connected subtree of  $T$ .

$X_i$ , a subset of  $V(G)$ , is called the *bag* of node  $i$ . The maximum size of a bag in  $TD(G)$  minus one is called the *width* of the tree decomposition. The *treewidth* of a graph, denoted as  $tw(G)$ , is the minimum width over all possible tree decompositions of the graph. The

problem of deciding whether the treewidth of a given graph is at most  $k$  is NP-complete. However, there are efficient constructive algorithms when  $k$  is small ( $k \leq 4$ ) or the input graph is outerplanar. There are also FPT approximation algorithms that guarantee a constant approximation factor [4].

A *nice* tree decomposition  $(X = \{X_i | i \in I\}, T = (I, F))$  is a tree decomposition such that  $|I| = O(\text{tw}(G) \cdot |V|)$ ,  $T$  is a rooted binary tree with three types of internal nodes:

1. a join node that has two children  $lc(i), rc(i)$ ,  $X_{lc(i)} = X_{rc(i)} = X_i$ .
2. a forget node that has one child  $lc(i)$ ,  $X_i \subset X_{lc(i)}$  and  $|X_i| = |X_{lc(i)}| - 1$ .
3. an introduce node that has one child  $lc(i)$ ,  $X_{lc(i)} \subset X_i$  and  $|X_{lc(i)}| = |X_i| - 1$ .

Note that a tree decomposition can be transformed into a nice tree decomposition in linear time.

In this section, we assume that a nice tree decomposition  $(X, T)$  for  $G$  with width  $\text{tw}(G)$  is given. We also assume that the bags of the root and the leaves are empty. Let  $T_i$  denote the subtree of  $T$  rooted at node  $i$ . Let  $X_{T_i} = \bigcup_{i \in T_i} X_i$  and  $b = \text{tw}(G) + 1$ .

We solve GAEM for graphs with bounded treewidth by generalizing the dynamic programming algorithm described in Section 4.1 for trees. Consider a tree  $\hat{T} = (\hat{V}, \hat{E})$ . A node  $\hat{v}$  in  $\hat{V}$  is the link point between the two subtrees  $\hat{T}_{\hat{v}}$  and  $\hat{T} \setminus (\hat{T}_{\hat{v}} \setminus \{\hat{v}\})$ .

A tree decomposition defines a sequence of separators of the graph. The separator is the link between the two separated parts of the graph. For any  $V' \subseteq V$ , let  $G(V')$  denote the subgraph of the input graph  $G$  induced by vertices in  $V'$ . For a node  $i$  of  $T$  in  $(X, T)$ ,  $X_i$  is a separator between  $G(X_{T_i})$  and  $G(V \setminus (X_{T_i} \setminus X_i))$ . Each vertex in  $X_i$  is a link point. For each vertex  $v_{i,j}$  in  $X_i$ , let  $D^\uparrow(v_{i,j}) = \bigcup_{u \in V \setminus (X_{T_i} \setminus X_i)} (d_G(v_{i,j}, u) + |us|)$  and let  $D^\downarrow(v_{i,j}) = \bigcup_{u \in X_{T_i}} (d_G(v_{i,j}, u) + |us|)$ . Each distance in  $D^\uparrow(v_{i,j})$  is the weight of a path from  $v_{i,j}$  to a vertex  $u$  in  $V \setminus (X_{T_i} \setminus X_i)$ , plus the shortcut  $(u, s)$ . We say the distance is *realized* by  $u$ . The distances in  $D^\downarrow(v_{i,j})$  are defined in the same way but through a vertex  $u$  in  $X_{T_i}$ . As in Section 4.1 we will need the subset  $D^\downarrow(v_{i,j}, d)$  of  $D^\downarrow(v_{i,j})$ , containing all distances in  $D^\downarrow(v_{i,j})$  that are at least  $d$ .

At node  $i$  of  $(X, T)$ , let  $t_i^\uparrow = (d_{i,1}^\uparrow, \dots, d_{i,|X_i|}^\uparrow)$  denote a tuple where  $d_{i,j}^\uparrow \in D^\uparrow(v_{i,j})$  is the weight of the shortest path from  $v_{i,j}$  to  $s$  going through a shortcut added to a vertex in  $V \setminus (X_{T_i} \setminus X_i)$ ,  $1 \leq j \leq |X_i|$ . Similarly we can define  $t_i^\downarrow = (d_{i,1}^\downarrow, \dots, d_{i,|X_i|}^\downarrow)$ . Let  $v_{i,j}^\uparrow, 1 \leq j \leq |X_i|$  denote the vertex in  $V \setminus (X_{T_i} \setminus X_i)$  that realizes  $d_{i,j}^\uparrow$ , and  $v_{i,j}^\downarrow$  denote the vertex in  $X_{T_i}$  that realizes  $d_{i,j}^\downarrow$ . We say that  $d_{i,j}^\uparrow$  and  $d_{i,j}^\downarrow$  are *components* of  $t_i^\uparrow$  and  $t_i^\downarrow$ , respectively. For a given node  $i$  and three parameters  $t_i^\uparrow, t_i^\downarrow$  and  $\bar{k}$  the subproblem is to find a set of  $\bar{k}$  shortcuts from  $s$  to vertices in  $X_{T_i}$  so that the maximum distance from vertices in  $X_{T_i}$  to  $s$  is minimized, assuming the distance tuples  $t_i^\uparrow$  and  $t_i^\downarrow$ .

Before we define the recursive steps we will need a few more notations. Let  $R_i[t_i^\uparrow][t_i^\downarrow][\bar{k}]$  denote the minimum maximum distance from a vertex in  $X_{T_i}$  to  $s$ , where  $\bar{k}$  shortcuts are allowed to be added between  $s$  and vertices in  $X_{T_i}$ , assuming the distance tuples  $t_i^\uparrow$  and  $t_i^\downarrow$ .

In Section 4.1, we defined  $W_{\hat{v}}$  for a link point  $\hat{v}$  between  $\hat{T}_{\hat{v}}$  and  $\hat{T} \setminus (\hat{T}_{\hat{v}} \setminus \{\hat{v}\})$ . For graphs of bounded treewidth we have  $|X_i|$  link points at node  $i$  of  $(X, T)$ . Assume  $i$  is a join node with left child  $lc(i)$  and right child  $rc(i)$ . A component  $d_{i,j}^\downarrow$  of  $t_i^\downarrow$  is realized by a vertex  $v_{i,j}^\downarrow$ . If  $v_{i,j}^\downarrow$  is in the left subtree  $T_{lc(i)}$  of  $T_i$  then  $d_{rc(i),j}^\downarrow$  must be at least  $d_{i,j}^\downarrow$ . The reverse is symmetrically true. So we let  $t_i^\downarrow = (t_{i1}^\downarrow, t_{i2}^\downarrow) = \{d_{i,1}^\downarrow, \dots, d_{i,|X_i|}^\downarrow\}$  be the  $t^\downarrow$  parameter to a  $W_i$ . If  $d_{i,j}^\downarrow \in t_{i1}^\downarrow$  then the component equals  $d_{i,j}^\downarrow$ . If  $d_{i,j}^\downarrow \in t_{i2}^\downarrow$  then the component is at least  $d_{i,j}^\downarrow$ .

## 45:10 Augmenting Graphs to Minimize the Radius

Thus

$$W_i[t_i^\uparrow][t_i^\downarrow] = (t_{i1}^\downarrow; t_{i2}^\downarrow)[\bar{k}] = \min\{R_i[t_i^\uparrow][d_{i,1}, \dots, d_{i,|X_i|}][\bar{k}]\},$$

where  $d_{i,j} = d_{i,j}^\downarrow$  if  $d_{i,j}^\downarrow \in t_{i1}^\downarrow$  and  $d_{i,j} \in D^\downarrow(v_{i,j}, d_{i,j}^\downarrow)$  if  $d_{i,j}^\downarrow \in t_{i2}^\downarrow$ . There are  $2^{|X_i|} - 1$  nonempty subset of  $X_i$ , so we need to define  $2^{|X_i|} - 1$   $W_i$ s with different components in  $t_{i1}^\downarrow$  and  $t_{i2}^\downarrow$ .

**Additional complexity for graphs with bounded treewidth.** The definitions and notations above corresponds closely to similar concepts in Section 4.1. However, there are a couple of complications that we have to take care of specifically for graphs with bounded treewidth. Firstly, we have to handle the three types of nodes (join, introduce and forget nodes) differently. This will be discussed in detail below. Secondly, two aspects of the computation at a node of  $(X, T)$  become more complex.

1. *Feasible* values of  $t_i^\uparrow$  and  $t_i^\downarrow$ . Not every combination of the values of each  $d_{i,j}^\downarrow$  forms a feasible value of  $t_i^\uparrow$ . We say a  $t_i^\uparrow$  is *feasible* only if it can be realized by some set of shortcuts (The formal definition appears in Appendix F).
2. There are  $(2^{|X_i|} - 1)$   $W_i$ s to be computed from  $R_i$ .

We discuss how to handle these two problems in Appendix F.

Next we explain the recursion steps at join, introduce and forget nodes.

**Join nodes.** A join node  $i$  has two children  $lc(i)$  and  $rc(i)$ . We need to compute  $R_i$  from  $R_{lc(i)}$ ,  $W_{lc(i)}$ ,  $R_{rc(i)}$  and  $W_{rc(i)}$ . More specifically, we need to determine the tuples  $t_{lc(i)}^\uparrow$ ,  $t_{lc(i)}^\downarrow$ ,  $t_{rc(i)}^\uparrow$  and  $t_{rc(i)}^\downarrow$  from  $t_i^\uparrow$  and  $t_i^\downarrow$ .

A partition of  $V$  for the separations between  $i$  and its children is shown in Figure 3(a). We use three copies of  $X_i$  for ease of illustration. For a component  $d_{i,j}^\downarrow$  of  $t_i^\downarrow$ ,  $v_{i,j}^\downarrow$  can be a vertex in  $X_i$ , or in  $(X_{T_{lc(i)}} \setminus X_i)$ , or in  $(X_{T_{rc(i)}} \setminus X_i)$ . If  $v_{i,j}^\downarrow$  is in  $X_i$ , both  $d_{lc(i),j}^\downarrow$  and  $d_{rc(i),j}^\downarrow$  equal  $d_{i,j}^\downarrow$ . Any path from  $v_{i,j}$  to  $s$  that goes through a shortcut incident to a vertex in  $X_{T_{rc(i)}} \setminus X_i$  has weight at least  $d_{lc(i),j}^\downarrow$  so we can set  $d_{lc(i),j}^\uparrow = d_{i,j}^\uparrow$ , and similarly  $d_{rc(i),j}^\uparrow = d_{i,j}^\uparrow$ .

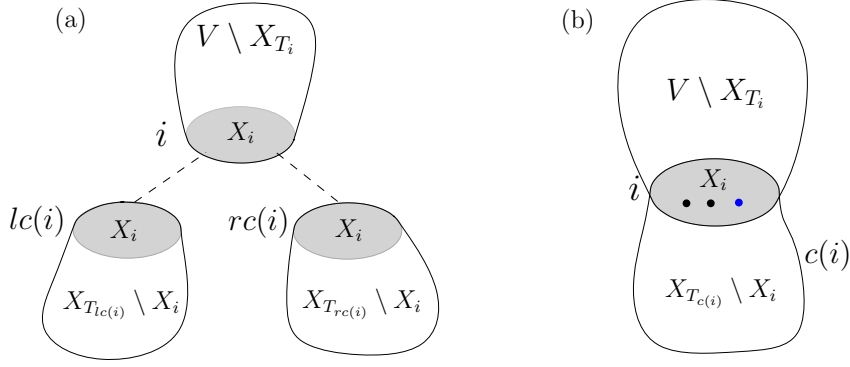
If  $v_{i,j}^\downarrow$  is in  $X_{T_{lc(i)}} \setminus X_i$ , then  $d_{lc(i),j}^\downarrow = d_{i,j}^\downarrow$  and  $d_{rc(i),j}^\downarrow$  is at least  $d_{i,j}^\downarrow$ . Any path from  $v_{i,j}$  to  $s$  that goes through a shortcut incident to a vertex in  $X_{T_{rc(i)}} \setminus X_i$  has length at least  $d_{lc(i),j}^\downarrow$  so we can set  $d_{lc(i),j}^\uparrow = d_{i,j}^\uparrow$ . For  $rc(i)$ , it is possible that  $d_{i,j}^\downarrow < d_{i,j}^\uparrow$ , so we set  $d_{rc(i),j}^\uparrow = \min\{d_{i,j}^\uparrow, d_{i,j}^\downarrow\}$ . The case when  $v_{i,j}^\downarrow$  is in  $X_{T_{rc(i)}} \setminus X_i$  is handled symmetrically.

The recursive relation for  $R_i$  is then:

$$R_i[t_i^\uparrow][t_i^\downarrow][\bar{k}] = \min_{k_1, k_2} \{ \max\{ W_{lc(i)}[t_{lc(i)}^\uparrow][t_{lc(i)}^\downarrow; t_{lc(i)2}^\downarrow][k_1], \\ W_{rc(i)}[t_{rc(i)}^\uparrow][t_{rc(i)1}^\downarrow; t_{rc(i)2}^\downarrow][k_2] \} \},$$

where  $k_1$  plus  $k_2$  equals  $\bar{k}$  plus the number of shortcuts added to vertices in  $X_i$ . The  $t^\uparrow$  and  $t^\downarrow$  tuples for  $W_{lc(i)}$  and  $W_{rc(i)}$  are derived as discussed above. Note that when  $t_{lc(i)2}^\downarrow = \emptyset$ ,  $W_{lc(i)} = R_{lc(i)}$ .

Next we analyze the time spent at a join node. As shown in Appendix F, computing all feasible values of  $t_i^\uparrow$  and  $t_i^\downarrow$  for  $R_i$  requires  $O(bn^b)$  time. For a given  $t_i^\uparrow$  value, computing  $R_i$  for all feasible values of  $t_i^\downarrow$  and  $\bar{k}$  takes  $O(k^2b \cdot |X_{T_i}|^{|X_i|}) = O(k^2bn^b)$  time. As discussed in Appendix F, computing all values for a  $W_i$  takes  $O(kb^2n^{2b})$  time. Since there are  $(2^b - 1)$  different  $W_i$ s the total time spent at a join node is  $O((k^2b + kb^22^b) \cdot n^{2b})$ .



■ **Figure 3** (a) Partition of  $V$  at a join node  $i$  for the separations between  $i$  and its children. (b) Separation at an introduce node  $i$ ,  $v_{i,|X_i|}$  is colored blue.

**Introduce node.** Consider an introduce node  $i$  and let  $c(i)$  be its child. Without loss of generality, let  $v_{i,|X_i|}$  be the vertex introduced at  $i$ . From the property of a tree decomposition, any path from  $v_{i,|X_i|}$  to a vertex in  $\{X_{T_i} \setminus X_i\}$  must go through a vertex in  $X_i \setminus \{v_{i,|X_i|}\}$ , see Fig. 3(b). We compute  $R_i$  from  $R_{c(i)}$  or  $W_{c(i)}$ . For given  $t_i^\uparrow$ ,  $t_i^\downarrow$  and  $\bar{k}$ , we need to determine the corresponding  $t_{c(i)}^\uparrow$  and  $t_{c(i)}^\downarrow$ . We consider two cases, depending on whether  $v_{i,|X_i|}$  is incident to one of the  $\bar{k}$  shortcuts, or not.

1.  $d_{i,|X_i|}^\downarrow \neq |s, v_{i,|X_i|}|$ , that is,  $v_{i,|X_i|}$  is not incident to any of the  $\bar{k}$  shortcuts. For any  $d_{i,j}^\uparrow$ ,  $j \neq |X_i|$ , the corresponding  $d_{c(i),j}^\uparrow$  is realized by the same vertex as  $d_{i,j}^\uparrow$ . Thus  $d_{c(i),j}^\uparrow = d_{i,j}^\uparrow$ , where  $j \neq |X_i|$ . Similarly,  $d_{c(i),j}^\downarrow = d_{i,j}^\downarrow$ , where  $j \neq |X_i|$ . For given  $t_i^\uparrow$ ,  $t_i^\downarrow$  and  $\bar{k}$  at introduce node  $i$ , the maximum distance from all vertices in  $X_{T_{c(i)}}$  to  $s$  is  $R_{c(i)}[t_{c(i)}^\uparrow][t_{c(i)}^\downarrow][\bar{k}]$ . The distance from  $v_{i,|X_i|}$  to  $s$  is the minimum of  $d_{i,|X_i|}^\uparrow$  and  $d_{i,|X_i|}^\downarrow$ . As a result we have:

$$R_i[t_i^\uparrow][t_i^\downarrow][\bar{k}] = \max\{R_{c(i)}[t_i^\uparrow \setminus d_{i,|X_i|}^\uparrow][t_i^\downarrow \setminus d_{i,|X_i|}^\downarrow][\bar{k}], \min\{d_{i,|X_i|}^\uparrow, d_{i,|X_i|}^\downarrow\}\}.$$

2.  $d_{i,|X_i|}^\downarrow = |s, v_{i,|X_i|}|$ , that is,  $v_{i,|X_i|}$  is incident to one of the  $\bar{k}$  shortcuts. By definition,  $d_{c(i),j}^\uparrow$ ,  $j \neq |X_i|$ , equals the minimum of  $d_{i,j}^\uparrow$  and  $d_G(v_{i,j}, v_{i,|X_i|}) + |s, v_{i,|X_i|}|$ . For any such  $d_{i,j}^\downarrow$ ,  $j \neq |X_i|$ , that  $v_{i,j}^\downarrow$  is not  $v_{i,|X_i|}$ , the corresponding  $d_{c(i),j}^\downarrow$  equals  $d_{i,j}^\downarrow$ . For any such  $d_{i,j}^\downarrow$ ,  $j \neq |X_i|$ , that  $v_{i,j}^\downarrow$  is  $v_{i,|X_i|}$ , the corresponding  $d_{c(i),j}^\downarrow$  is greater than  $d_{i,j}^\downarrow$ . For given  $t_i^\uparrow$ ,  $t_i^\downarrow$  and  $\bar{k}$  at an introduce node  $i$ , the maximum distance from all vertices in  $X_{T_{c(i)}}$  to  $s$  is  $W_{c(i)}[t_{c(i)}^\uparrow][t_{c(i)}^\downarrow][\bar{k} - 1]$ . The distance from  $v_{i,|X_i|}$  to  $s$  is  $|s, v_{i,|X_i|}|$ . Thus,

$$R_i[t_i^\uparrow][t_i^\downarrow][\bar{k}] = \max\{W_{c(i)}[(d_{c(i),1}^\uparrow, \dots, d_{c(i),|X_i|-1}^\uparrow)][(t_{c(i)1}^\downarrow; t_{c(i)2}^\downarrow)][\bar{k} - 1], |s, v_{i,|X_i|}|\},$$

where  $d_{c(i),j}^\uparrow = \min\{d_{i,j}^\uparrow, d_G(v_{i,j}, v_{i,|X_i|}) + |s, v_{i,|X_i|}|\}$ , for  $j < |X_i|$ . If  $d_{i,j}^\downarrow$  is not realized by  $v_{i,|X_i|}$ , for  $j < |X_i|$ , then  $d_{c(i),j}^\downarrow = d_{i,j}^\downarrow$  and is a component of  $t_{c(i)1}^\downarrow$ . Otherwise,  $d_{c(i),j}^\downarrow > d_{i,j}^\downarrow$  and is a component of  $t_{c(i)2}^\downarrow$ .

The time required to compute the  $R_i$  values is  $O(kbn^{2b})$ . The  $W_i$ s are computed from  $R_i$  in the same way as for a join node, hence, in totally  $O(kb^22^bn^{2b})$  time. As a result, the time spent at an introduce node is  $O(kb^22^b \cdot n^{2b})$ .



**Forget node.** Assume  $i$  is a forget node and  $c(i)$  is its child. Without loss of generality, assume  $v_{c(i),|X_{c(i)}|}$  is the vertex forgotten at  $i$ . As usual, we compute  $R_i$  from values of  $R_{lc(i)}$  or  $W_{c(i)}$ .  $v_{i,j} = v_{c(i),j}$ ,  $j < |X_{c(i)}|$ . By definition,  $d_{c(i),j}^\uparrow = d_{i,j}^\uparrow$ , for all  $j < |X_{c(i)}|$ . In the input graph  $G$ , any path from  $v_{c(i),|X_{c(i)}|}$  to a vertex in  $V \setminus X_{T_i}$  must go through a vertex in  $X_i$ . Thus the corresponding  $d_{c(i),|X_{c(i)}|}^\uparrow$  equals  $\min\{d_{i,j}^\uparrow + d_G(v_{i,j}, v_{c(i),|X_{c(i)}|}) \mid j < |X_{c(i)}|\}$ . By definition,  $d_{c(i),j}^\downarrow = d_{i,j}^\downarrow$  for all  $j < |X_{c(i)}|$ . However,  $d_{c(i),|X_{c(i)}|}^\downarrow$  can take a number of different values. The only constraint is that it forms a feasible  $t_{c(i)}^\downarrow$  together with all other  $d_{c(i),j}^\downarrow$  values,  $j < |X_{c(i)}|$ . Thus  $R_i$  is computed from  $W_{c(i)}$  where  $d_{c(i),|X_{c(i)}|}^\downarrow$  is the only component in  $t_{c(i)2}^\downarrow$  and takes the minimum value that forms a feasible  $t_{c(i)}^\downarrow$  with all other  $d_{c(i),j}^\downarrow$ ,  $j < |X_{c(i)}|$ .

$$R_i[t_i^\uparrow][t_i^\downarrow][\bar{k}] = W_{c(i)}[t_{c(i)}^\uparrow][t_{c(i)1}^\downarrow = t_i^\downarrow; t_{c(i)2}^\downarrow][\bar{k}],$$

where  $d_{c(i),|X_{c(i)}|}^\downarrow \in t_{c(i)2}^\downarrow$ . The  $W_i$ s are computed from  $R_i$  as before. The time spent at a forget node is  $O(kb^22^b \cdot n^{2b})$ .

Since there are  $O(bn)$  nodes in a nice tree decomposition we conclude this section with the following theorem.

► **Theorem 12.** *When the input graph  $G$  is a metric graph with bounded treewidth, GAEM problem can be solved in  $O((k^2b^2 + kb^32^b) \cdot n^{2b+1})$  time, where  $b = tw(G) + 1$ . The metric BCMR problem on graphs with bounded treewidth can be solved in  $O((k^2b^2 + kb^32^b) \cdot n^{2b+2})$  time.*

---

## References

- 1 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial  $k$ -trees. *Discret. Appl. Math.*, 23(1):11–24, 1989.
- 2 Davide Bilò, Luciano Gualà, and Guido Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theor. Comput. Sci.*, 417:12–22, 2012.
- 3 Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 19–36, 1997.
- 4 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A  $c^k n$  5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.
- 5 Gerard J. Chang and George L. Nemhauser. The  $k$ -domination and  $k$ -stability problems for sun-free chordal graphs. *SIAM J. Algebraic Discrete Methods*, 5(3):332–345, 1984.
- 6 Victor Chepoi and Yann Vaxès. Augmenting trees to meet biconnectivity and diameter constraints. *Algorithmica*, 33(2):243–262, 2002.
- 7 Erik D. Demaine and Morteza Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 420–431, 2010.
- 8 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 750–759, 1999.
- 9 Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72(4):995–1010, 2015.

- 10 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 338–346. IEEE Computer Society, 1984.
- 11 Yong Gao, Donovan R. Hare, and James Nastos. The parametric complexity of graph diameter augmentation. *Discret. Appl. Math.*, 161(10-11):1626–1631, 2013.
- 12 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.
- 13 C. Johnson and H. Wang. A linear-time algorithm for radius-optimally augmenting paths in a metric space. In *Proceedings of the 15th International Symposium on Algorithms and Data Structures*, pages 466–480, 2019.
- 14 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Oper. Res. Lett.*, 11(5):303–308, 1992.
- 15 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
- 16 Anneke A. Schoone, Hans L. Bodlaender, and Jan van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987.

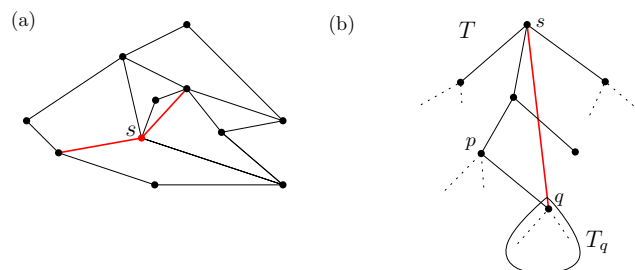
**A Proof of Lemma 1**

► **Lemma 1** Given a metric graph  $G = (V, E, \ell)$ , an integer  $k$  and a vertex  $s$  in  $V$  for the GAEM problem, there is an optimal solution where every shortcut is incident to  $s$ .

**Proof.** Let  $F^*$  be an optimal solution and  $G^* = (V, E \cup F^*, \ell)$ . Let  $ecc^*(s)$  be the eccentricity of  $s$  in  $G^*$  and let  $T$  be the shortest path tree rooted at  $s$ . If every edge in  $F^*$  is incident to  $s$  then the lemma immediately holds. Otherwise there exists at least one edge  $(p, q)$  in  $F^*$  that is not incident to  $s$ . We will show that  $(p, q)$  can be replaced by either  $(s, p)$  or  $(s, q)$  such that the eccentricity of  $s$  in the resulting graph does not increase.

If  $(p, q)$  is an edge in  $T$ , then there is a subset  $U$  of  $V$  that go through  $(p, q)$  on their shortest paths to  $s$  in  $T$ . Without loss of generality assume  $d_T(p, s) < d_T(q, s)$ , as shown in Fig. 4. The set  $U$  is the vertices in the subtree  $T_q$  of  $T$  rooted at  $q$ . By triangle inequality  $|sq| \leq d_T(s, q)$ . Replace  $(p, q)$  by  $(s, q)$  to get  $T'$ . Then every vertex in  $T'_q$  now has a path to  $s$  no longer than its shortest path in  $T$ , and every vertex not in  $T_q$  can still use their shortest path in  $T$ . Consequently,  $ecc(s)$  in  $T'$  is at most  $ecc^*(s)$ .

If  $(p, q)$  is not an edge in  $T$  then replace  $(p, q)$  by either  $(s, p)$  or  $(s, q)$ . Since every vertex can still use its shortest path in  $T$  the eccentricity of  $s$  does not increase. ◀



■ **Figure 4** (a) An example where all the edges of the graph have unit length. The two shortcuts in red show an optimal solution for the GAEM instance with  $k = 2$ . (b) Illustrating the proof for Lemma 1.

**B Proof of Lemma 2**

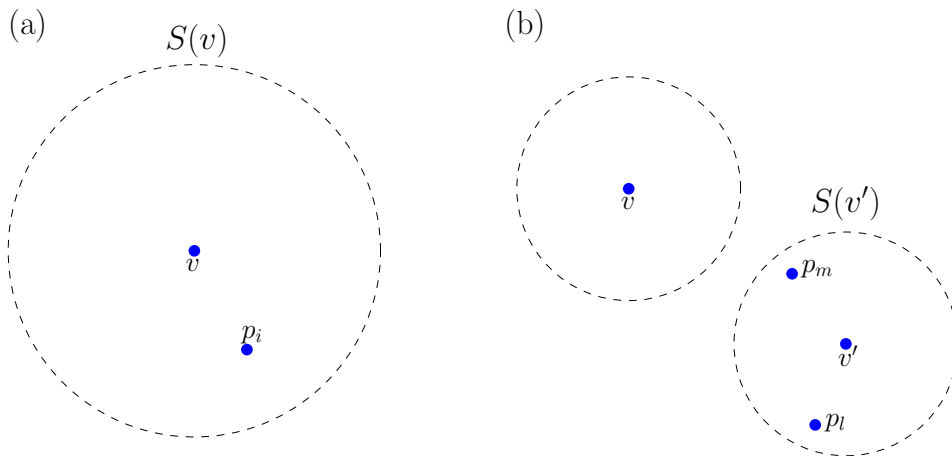
► **Lemma 2** *Algorithm 1 is a 3-approximation algorithm for the GAEM problem.*

**Proof.** Let  $A = \{q_1, q_2, \dots, q_t\}$  be the set of vertices that are adjacent to  $s$  in  $G$ . If Algorithm 1 adds less than  $k$  shortcuts, the augmentation is optimal. Otherwise, let  $U = \{p_1, p_2, \dots, p_k\}$  be the set of vertices such that  $(s, p_i) (i = 1, \dots, k)$  is a shortcut added by Algorithm 1. Remember from Lemma 1 there is always an optimal solution  $F^*$  where all the shortcuts are incident to  $s$ . Assume  $B = \{c_1, c_2, \dots, c_k\}$  is the set of vertices such that  $(s, c_i)$  is a shortcut in  $F^*$ . Let  $\text{ecc}^*(s)$  denote the eccentricity of  $s$  in  $G^* = (V, E \cup F^*, \ell)$ .  $A \cup B$  is the set of vertices that are adjacent to  $s$  in  $G^*$ . Any vertex  $w$  other than  $s$  must go through exactly one vertex  $v$  in  $A \cup B$  (may be itself) on its shortest path to  $s$  in  $G_s^*$ . Let  $S(v), v \in A \cup B$  denote the set of vertices that go through  $v$  on their shortest paths to  $s$  in  $G^*$ .  $\{S(v) : v \in A \cup B\}$  forms a partition of all the vertices other than  $s$ . We need to prove that the graph distance from  $w$  to  $s$  in  $\hat{G}$  is at most  $3\text{ecc}^*(s)$ .

If  $v \in A$ ,  $d_G(w, s) \leq \text{ecc}^*(s)$  so the distance from  $w$  to  $s$  in  $\hat{G}$  is at most  $\text{ecc}^*(s)$ . If any  $p_i \in U$  is in a  $S(q_j)$  where  $q_j \in A$ ,  $d_G(p_i, s) \leq \text{ecc}^*(s)$ . At the time  $p_i$  is picked by Algorithm 1,  $p_i$  is the farthest vertex to  $s$ , so Algorithm 1 returns an optimal solution. What is left is when  $v \in B$  and no  $p_i \in U$  is in any  $S(q_j)$  where  $q_j \in A$ . Any  $p_i$  must be in some  $S(c_i)$  where  $c_i \in B$ .

If there is a  $p_i \in U$  in  $S(v)$ , as in Figure 5(a), there is a path from  $w$  to  $s$  in  $\hat{G}$  that goes through  $v$  and  $p_i$  and has weight  $d_G(w, v) + d_G(v, p_i) + |sp_i|$ . Both  $d_G(w, v)$  and  $d_G(v, p_i)$  are at most  $\text{ecc}^*(s)$ . By the triangle inequality,  $|sp_i| \leq \text{ecc}^*(s)$ . Thus the distance from  $w$  to  $s$  in  $\hat{G}$  is at most  $3\text{ecc}^*(s)$ .

Otherwise, there is no  $p_i \in U$  in  $S(v)$ .  $|U| = |B|$ . By the pigeonhole principle, there must exist a vertex  $v' \in B$  such that two vertices  $p_l, p_m \in U$  are in  $S(v')$ , as shown in Figure 5(b). Without loss of generality, assume  $p_l$  is picked before  $p_m$  in  $\hat{G}$ . When  $p_m$  is about to be added a shortcut, it is the farthest vertex to  $s$ . The distance from  $w$  to  $s$  in  $\hat{G}$  is thus at most  $d_G(p_m, v') + d_G(v', p_l) + |sp_l| \leq 3\text{ecc}^*(s)$ , which is an upper bound of  $p_m$ 's distance to  $s$  before it is added a shortcut. ◀



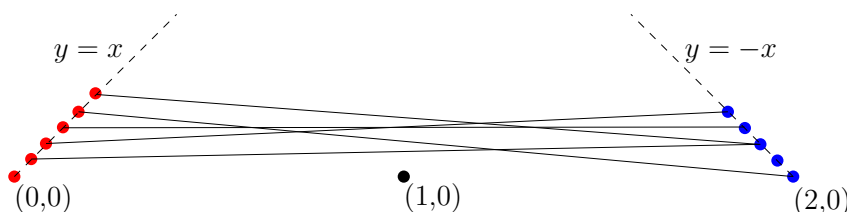
■ **Figure 5** (a) There is a vertex  $p_i \in U$  in  $S(v)$ . (b) There is no vertex  $p_i \in U$  in  $S(v)$ .

**C Proof of Lemma 6**

► **Lemma 6** For any  $\epsilon > 0$ , finding a  $(\frac{5}{3} - \epsilon)$  approximate solution for the GAEM problem on geometric graphs is NP-hard.

**Proof.** In [5], Chang and Nemhauser proved that the Dominating Set decision problem for bipartite graphs is NP-hard. We use a reduction from this problem to show the approximation hardness of GAEM on geometric graphs.

Let  $G = (V = V_1 \cup V_2, E)$  be a bipartite graph where  $V_1$  and  $V_2$  are disjoint and every edge in  $E$  connects one vertex in  $V_1$  to one vertex in  $V_2$ . Let  $I = (G, k)$ ,  $k < |V_1 \cup V_2|$ , be an instance of the Dominating Set decision problem for bipartite graphs. Without loss of generality, assume  $|V_1| \geq |V_2|$  and let  $m = |V_1|$ . We can embed  $G$  in the plane. Place vertices in  $V_1$  at positions  $(0, 0), (\epsilon/2(m-1), \epsilon/2(m-1)), \dots, (\epsilon/2, \epsilon/2)$  and place vertices in  $V_2$  at positions  $(2, 0), (2 - \epsilon/2(m-1), 2 - \epsilon/2(m-1)), \dots, (2 - \epsilon(n-1)/2(m-1), 2 - \epsilon(n-1)/2(m-1))$ . Then add vertex  $s$  at position  $(1, 0)$ . See Figure 6. We have a graph  $G' = (V' = V \cup \{s\}, E' = E, \ell)$  where  $\ell$  is the Euclidean distance function.  $I$  is transformed into an instance  $I' = (G', s, k, 3)$  of GAEM on geometric graphs. For any point  $p$  in  $V_1$  and any point  $q$  in  $V_2$ , we have  $1 - \epsilon/2 \leq |sp| \leq 1$  and  $2 - \epsilon \leq |pq| \leq 2$ .  $G$  has a dominating set of size  $k$  if and only if we can add  $k$  shortcuts to  $G'$  so that the eccentricity of  $s$  in the resulting graph is at most 3. The shortest path from a vertex to  $s$  that goes through 1 shortcut and 2 edges in  $E$  has length at least  $5 - 5\epsilon/2$ . Since  $3 \cdot (\frac{5}{3} - \epsilon) = 5 - 3\epsilon < 5 - 5\epsilon/2$ , a  $(\frac{5}{3} - \epsilon)$  approximation algorithm for GAEM on geometric graphs solves  $I'$  and thus any instance of the Dominating Set decision problem. ◀



■ **Figure 6** A bipartite graph embedded in the plane.

**D An  $O(n^2 \log n)$  time algorithm for GAEM on trees**

We first devise an  $O(\min\{k^2n, n^2\})$  time algorithm that solves the decision problem of GAEM on trees. Then we search for an optimal solution by using the decision algorithm as a subroutine.

**D.1 The decision algorithm**

Given a value  $D$ , the decision algorithm decides whether we can add  $k$  shortcuts all incident to  $s$  to the input tree such that the eccentricity of  $s$  in the augmented graph is at most  $D$ . We introduce some notations that will be used in the discussion.

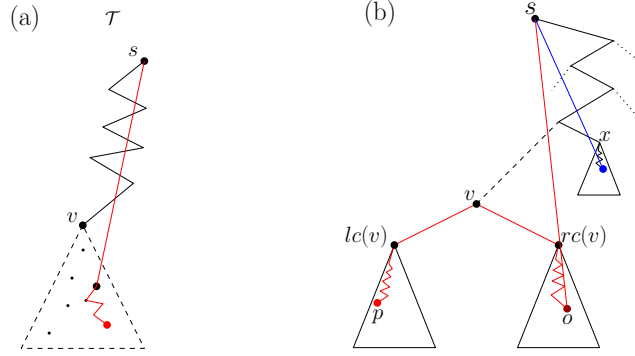
Let  $d_T(u, v)$  denote the distance between any two vertices  $u$  and  $v$  in  $T$ . Let  $\zeta$  be a set of  $\bar{k}$  ( $\bar{k} \leq k$ ) shortcuts added to vertices in  $T_v$ , where  $T_v$  is the subtree of  $T$  rooted at  $v$ . If from a vertex in  $T_v$  we can go along a path inside  $T_v$  and then through a shortcut in  $\zeta$  to reach  $s$  within distance  $D$ , we say this vertex is covered by  $\zeta$ , for example the red vertex

## 45:16 Augmenting Graphs to Minimize the Radius

in Figure 7(a); otherwise we say it is uncovered by  $\zeta$ . Let  $U_v(\zeta)$  denote the set of vertices uncovered by  $\zeta$  in  $T_v$ . When  $U_v(\zeta)$  is empty, we say  $T_v$  is covered by  $\zeta$ ; otherwise we say  $T_v$  is uncovered by  $\zeta$ . To reach  $s$  within  $D$  from an uncovered vertex in  $U_v(\zeta)$ , we have to go along a path out of  $T_v$  and then through a shortcut<sup>2</sup> added to a vertex in  $T \setminus T_v$ , if at all possible. See the red vertex  $p$  in Figure 7(b) for an example. Let  $g_v(\zeta) = \max_{u \in U_v(\zeta)} d_T(u, v)$  denote the maximum distance from a vertex in  $U_v(\zeta)$  to  $v$  in  $T$ . Let  $d_v(\zeta)$  denote the length of the shortest path from  $v$  to  $s$  that goes through a shortcut in  $\zeta$ .

The decision algorithm is used to solve an optimization problem. The optimization problem aims to find shortcuts that are optimal in incurring a “Yes” solution to the decision problem. Generally, given any integer  $\bar{k} \leq k$  and any vertex  $v$ , the optimization problem is to add  $\bar{k}$  shortcuts to vertices in  $T_v$  so that they are optimal in incurring a “Yes” solution to the decision problem on  $T$ . Before we formally define the meaning of being optimal in incurring a “Yes” solution, we first give a supporting lemma.

As discussed above,  $\zeta$  is a set of  $\bar{k}$  shortcuts added to vertices in  $T_v$ .



■ **Figure 7** (a)  $\bar{k}$  shortcuts are added to  $T_v$ . (b) Some vertices in  $T_v$  are uncovered.

► **Lemma 13.** *If  $U_v(\zeta)$  is nonempty and  $\zeta$  is part of a “Yes” solution to the decision problem of GAEM on  $T$ , then in the “Yes” solution*

- 1) *all vertices in  $U_v(\zeta)$  go through the same shortcut added to a vertex in  $T \setminus T_v$  on their shortest paths to  $s$ .*
- 2) *vertices in  $T \setminus T_v$  does not go through any shortcut in  $\zeta$  on their shortest paths to  $s$ .*

**Proof.** In the “Yes” solution, all vertices in  $U_v(\zeta)$  must first go up to  $v$ , then follow the  $v - s$  shortest path which cannot go through a shortcut in  $\zeta$ . So property 1) is true.  $U_v(\zeta)$  is nonempty, thus

$$g_v(\zeta) + d_v(\zeta) > D. \quad (1)$$

Let  $\gamma$  denote the shortcut that the shortest paths from vertices in  $U_v(\zeta)$  to  $s$  go through. If  $\gamma$  is incident to a vertex in  $T_v$ 's sibling  $T_w$ , for example the darkred vertex  $o$  in Figure 7(b), we have

$$g_v(\zeta) + |uv| + |uw| + d_w(\zeta') \leq D, \quad (2)$$

where  $\zeta'$  is the set of shortcuts added to vertices in  $T_w$ . Equation 1 and 2 imply that  $d_v(\zeta) > |uw| + d_w(\zeta')$ , which means for every vertex in  $T \setminus T_v$ , the shortest path through  $\gamma$  to  $s$  is shorter than any shortest path through a shortcut in  $\zeta$ . If  $\gamma$  is incident to a vertex in

<sup>2</sup> For ease of discussion, we consider the edges that are incident to  $s$  in  $T$  as pre-added shortcuts.

$T \setminus T_v \setminus T_w$ , like the blue vertex in Figure 7(b), we can similarly show that for every vertex in  $T \setminus T_v$ , a shortest path through  $\gamma$  to  $s$  is shorter than any shortest path through a shortcut in  $\zeta$ . Property 2) is proved. ◀

We now discuss the meaning of a  $\zeta$  being optimal in incurring a “Yes” solution to the decision problem on  $T$ . Consider two sets  $\zeta_1$  and  $\zeta_2$ :

1. if both  $U_v(\zeta_1)$  and  $U_v(\zeta_2)$  are nonempty. Without loss of generality, assume  $g_v(\zeta_1) < g_v(\zeta_2)$ . Assume  $\zeta_2$  is part of a “Yes” solution to the decision problem. Replace  $\zeta_2$  by  $\zeta_1$ . Lemma 13 implies that the resulting solution is still a “Yes” solution. The reverse is not true.  $\zeta_1$  place lower requirement on shortcuts added to vertices outside  $T_v$ . Thus  $\zeta_1$  is better in making a “Yes” solution to the decision problem than  $\zeta_2$ .
2. one of  $U_v(\zeta_1)$  and  $U_v(\zeta_2)$  is empty while the other is nonempty. Without loss of generality, assume  $U_v(\zeta_1)$  is empty. Assume  $\zeta_2$  is part of a “Yes” solution to the decision problem. Replace  $\zeta_2$  by  $\zeta_1$ . From Lemma 13(b), the resulting solution is still a “Yes” solution.  $\zeta_1$  places no requirement on the shortcuts added to vertices in  $T \setminus T_v$  but  $\zeta_2$  does. Thus  $\zeta_1$  is better in making a “Yes” solution to the decision problem than  $\zeta_2$ .
3. both  $U_v(\zeta_1)$  and  $U_v(\zeta_2)$  are empty. Assume  $d_v(\zeta_1) < d_v(\zeta_2)$ . If  $\zeta_2$  is part of a “Yes” solution to the decision problem, replacing  $\zeta_2$  by  $\zeta_1$  will still give a “Yes” solution. But for vertices in  $T \setminus T_v$ ,  $\zeta_1$  provides better shortcut than  $\zeta_2$ . Thus  $\zeta_1$  is better in making a “Yes” solution to the decision problem.

We can formally define the above *better in making a ‘Yes’ solution* relation on  $\zeta$ s. Use  $\prec$  to denote this relation. Then

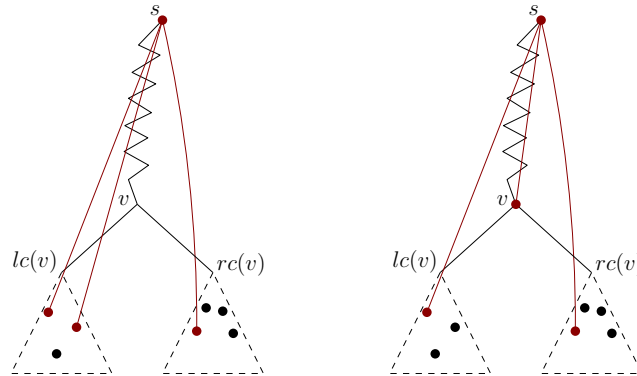
$$\begin{aligned}
 &U_v \text{ empty, smaller } d_v \prec U_v \text{ empty, greater } d_v \\
 &\prec U_v \text{ nonempty, smaller } g_v \prec U_v \text{ nonempty, greater } g_v.
 \end{aligned}$$

A  $\zeta$  that has no other  $\zeta$ s preceding it in the  $\prec$  relation is an optimal  $\zeta$ .

The optimization problem is well-defined. By the optimal substructure of the optimization problem, we can use dynamic programming to solve the problem. Let  $opt(v, \bar{k})$  denote an optimal solution of the problem on  $T_v$  with  $\bar{k}$ . When a solution uncovers  $T_v$ , we keep track of  $g_v$ . When a solution covers  $T_v$ , we keep track of  $d_v$ . The recursive formula of  $opt(v, \bar{k})$  is:

$$opt(v, \bar{k}) = \min_{\prec} \{combine(f, opt(lc(v), k'), opt(rc(v), \bar{k} - k' - f)) | 0 \leq k' \leq \bar{k} - f, f = 0/1\},$$

where  $\min_{\prec}$  is the “minimum” in relation “ $\prec$ ” and  $f$  flags whether a shortcut is added to  $v$ . When combining the optimal sub-solutions of the left child  $lc(v)$  and the right child  $rc(v)$ ,



■ **Figure 8** (a)  $(s, v)$  is not added as a shortcut. (b)  $(s, v)$  is added as a shortcut.

there are two cases: 1)  $v$  is not added a shortcut, 2)  $v$  is added a shortcut. For case 1),  $k'$  shortcuts are added to vertices in  $T_{lc(v)}$  and  $\bar{k}-k'$  shortcuts are added to vertices in  $T_{rc(v)}$ . For case 2),  $k'$  shortcuts are added to vertices in  $T_{lc(v)}$  and  $\bar{k}-1-k'$  shortcuts are added to vertices in  $T_{rc(v)}$ . See Figure 8 for an illustration. The *combine* procedures for case 1) and case 2) are similar and we only discuss case 1) below.  $combine(0, opt(lc(v), k'), opt(rc(v), \bar{k}-k'))$  works as follows:

**Case 1:**  $opt(lc(v), k')$  covers  $T_{lc(v)}$  and  $opt(rc(v), \bar{k}-k')$  covers  $T_{rc(v)}$ . Check if  $\min\{|v, lc(v)| + d_{lc(v)}, |v, rc(v)| + d_{rc(v)}\} \leq D$ . If so,  $v$  and all other vertices in  $T_v$  are covered and  $d_v = \min\{|v, lc(v)| + d_{lc(v)}, |v, rc(v)| + d_{rc(v)}\}$ ; else  $T_v$  is uncovered and  $g_v = 0$ .

**Case 2:**  $opt(lc(v), k')$  covers  $T_{lc(v)}$  and  $opt(rc(v), \bar{k}-k')$  uncovers  $T_{rc(v)}$ . We need to check whether vertices in  $T_{rc(v)}$  uncovered by  $opt(rc(v), \bar{k}-k')$  is covered by  $opt(lc(v), k')$ . Check if  $g_{rc(v)} + |v, rc(v)| + |v, lc(v)| + d_{lc(v)} \leq D$ . If so,  $T_v$  is covered and  $d_v = |v, lc(v)| + d_{lc(v)}$ ; else  $T_v$  is uncovered and  $g_v = g_{rc(v)} + |v, rc(v)|$ .

**Case 3:**  $opt(lc(v), k')$  uncovers  $T_{lc(v)}$  and  $opt(rc(v), \bar{k}-k')$  covers  $T_{rc(v)}$ . Symmetric to case 2.

**Case 4:**  $opt(lc(v), k')$  uncovers  $T_{lc(v)}$  and  $opt(rc(v), \bar{k}-k')$  uncovers  $T_{rc(v)}$ .  $T_v$  is uncovered and  $d_v = \max\{g_{lc(v)} + |v, lc(v)|, g_{rc(v)} + |v, rc(v)|\}$ .

It is not hard to verify the correctness of the procedure.  $combine(1, opt(lc(v), k'), opt(rc(v), \bar{k}-k'))$  works similarly and is left to the interested reader. The combining procedure at the root is slightly different at the root, we omit the details here.

All that is left is analyzing the running time of the dynamic programming. Once  $opt(lc(v), k')$  and  $opt(rc(v), \bar{k}-k')$  are known, the *combine* procedure takes constant time. Thus we only need to count the number of times  $combine(0, \dots)$  and  $combine(1, \dots)$  are called for computing every  $opt(v, \bar{k})$ .  $\bar{k} \leq \min\{k, |T_v|\}$  and  $k' \leq \min\{\bar{k}, |T_{lc(v)}|\}$ . Using  $\bar{k} \leq k$  and  $k' \leq \bar{k}$  as upper bounds, at a vertex  $v$ ,  $combine(0, \dots)$  is called  $O(\sum_{\bar{k}=1}^k \bar{k}) = O(k^2)$

times. Similarly,  $combine(0, \dots)$  is called  $O(k^2)$  times. So we spend  $O(k^2)$  time at a vertex  $v$ . This gives a total running time of  $O(k^2n)$ . We also use  $\bar{k} \leq |T_v|$  and  $k' \leq |T_{lc(v)}|$  as upper bounds. We assume that  $T_{lc(v)} = T_{vs}$ , otherwise we can just swap  $lc(v)$  and  $rc(v)$ . Thus for all vertices in  $T$ ,  $combine(0, \dots)$  is called  $O(\sum_{v \in T} |T_v| \cdot |T_{vs}|) = O(n^2)$  times, by Lemma 9.

Similarly, for all vertices,  $combine(1, \dots)$  is called  $O(n^2)$  times. This gives a total running time of  $O(n^2)$ . Thus the total running time is  $O(\min\{k^2n, n^2\})$ .

► **Theorem 14.** *The decision problem of GAEM on trees can be solved in  $O(\min\{k^2n, n^2\})$  time.*

## D.2 The search algorithm

For any solution of  $k$  shortcuts all incident to  $s$ , there is a farthest vertex  $v$  to  $s$  in the augmented graph. The shortest path from  $s$  to  $v$  goes from  $s$  to a vertex  $u$  (which may be  $v$ ) through a shortcut or an edge in  $T$ , then follows the shortest path from  $u$  to  $v$  in  $T$ . Both  $u$  and  $v$  are vertices in  $T$ . So there are at most  $n^2$  possible values for the eccentricity of  $s$  in any augmented graph.

► **Lemma 15.** *There are at most  $n^2$  possible values for the eccentricity of  $s$  in any augmented graph where all shortcuts are incident to  $s$ .*

Since our input is a tree, we can compute all possible values in  $O(n^2)$  time. Then we sort the values and do a binary search over the sorted values, using the decision algorithm as a subroutine. The minimum value for which the decision algorithm returns a “Yes” solution is the minimum eccentricity incurred by an optimal solution. We can find an optimal solution in  $O(\min\{k^2n, n^2\} \cdot \log n + n^2 \log n) = O(n^2 \log n)$  time.



► **Theorem 16.** *GAEM on trees can be solved in  $O(n^2 \log n)$  time.*

## E

 Proofs of Lemma 8 and Corollary 9

► **Lemma 8**  $\sum_{v \in T} |T_{lc(v)}| \cdot |T_{rc(v)}| \leq n^2$ .

**Proof.** We use induction on the number of tree nodes to prove the bound. When  $n = 1$ , the tree contains only one node and the bound holds trivially.

Assume  $\sum_{v \in T} |T_{lc(v)}| \cdot |T_{rc(v)}| \leq n^2$  holds for all  $n < k$ . Then for trees of size  $n = k$ ,  $\sum_{v \in T} |T_{lc(v)}| \cdot |T_{rc(v)}| = \sum_{v \in T_{lc(s)}} |T_{lc(v)}| \cdot |T_{rc(v)}| + \sum_{v \in T_{rc(s)}} |T_{lc(v)}| \cdot |T_{rc(v)}| + |T_{lc(s)}| \cdot |T_{rc(s)}|$ , where  $s$  is the root of  $T$ . By the induction hypothesis,  $\sum_{v \in T_{lc(s)}} |T_{lc(v)}| \cdot |T_{rc(v)}| \leq |T_{lc(s)}|^2$ ,  $\sum_{v \in T_{rc(s)}} |T_{lc(v)}| \cdot |T_{rc(v)}| \leq |T_{rc(s)}|^2$ . Thus  $\sum_{v \in T} |T_{lc(v)}| \cdot |T_{rc(v)}| \leq |T_{lc(s)}|^2 + |T_{rc(s)}|^2 + |T_{lc(s)}| \cdot |T_{rc(s)}| \leq (|T_{lc(s)}| + |T_{rc(s)}|)^2 < n^2$ . ◀

► **Corollary 9**  $\sum_{v \in T} |T_v| \cdot \min\{|T_{lc(v)}|, |T_{rc(v)}|\} \leq 2n^2 + n \log n$ .

**Proof.** Let  $T_{vs}$  be the subtree rooted at a child of  $v$  such that  $|T_{vs}| = \min\{|T_{lc(v)}|, |T_{rc(v)}|\}$ . We know that  $\sum_{v \in T} |T_v| \cdot |T_{vs}| = \sum_{v \in T} (|T_{lc(v)}| + |T_{rc(v)}| + 1) \cdot |T_{vs}|$ , and by Lemma 8,  $\sum_{v \in T} (|T_{lc(v)}| + |T_{rc(v)}|) \cdot |T_{vs}| \leq 2n^2$ . For  $\sum_{v \in T} |T_{vs}|$ , we can see that a node in  $T$  is counted at most  $\log n$  times since for any  $T_v$ , only nodes in the subtree rooted at a child of  $v$  with fewer nodes are counted. Thus  $\sum_{v \in T} |T_{vs}| \leq n \log n$  and the corollary follows. ◀

## F

 How to compute feasible values and  $W_i$ s

We first formally define what is a feasible  $t_i^\uparrow$  value. The definition of a feasible  $t_i^\downarrow$  value is similar. Let  $t_i^\uparrow = (d_{i,1}^\uparrow, \dots, d_{i,|X_i|}^\uparrow)$ . For any two components  $d_{i,k}^\uparrow$  and  $d_{i,l}^\uparrow$ , the length of the shortest path from  $v_{i,k}$  to  $s$  that goes through shortcut  $(v_{i,l}, s)$  has to be at least  $d_{i,k}^\uparrow$ , otherwise  $d_{i,k}^\uparrow$  would have a smaller value. Conversely, the length of the shortest path from  $v_{i,l}$  to  $s$  that goes through shortcut  $(v_{i,k}, s)$  has to be at least  $d_{i,l}^\uparrow$ . The definition follows.

► **Definition 17.** *Let  $t_i^\uparrow = (d_{i,1}^\uparrow, \dots, d_{i,|X_i|}^\uparrow)$ .  $t_i^\uparrow$  is feasible if and only if for any  $1 \leq k < l \leq |X_i|$ ,  $d_G(v_{i,k}, v_{i,l}^\uparrow) + |v_{i,l}^\uparrow, s| \geq d_{i,k}^\uparrow$  and  $d_G(v_{i,l}, v_{i,k}^\uparrow) + |v_{i,k}^\uparrow, s| \geq d_{i,l}^\uparrow$ .*

To save space, we store feasible values of  $t_i^\uparrow$  (and  $t_i^\downarrow$ ) in a multilist. As a preprocessing step, we compute distances between any pair of vertices in  $G$  by using an all-pairs shortest paths algorithm. The distance between any pair of vertices in  $G$  can then be obtained in constant time. Let  $\mathcal{L}_i^\uparrow$  denote the multilevel list for  $t_i^\uparrow$ . Levels of  $\mathcal{L}_i^\uparrow$  are contain ordered values of  $d_{i,1}^\uparrow, \dots, d_{i,|X_i|}^\uparrow$ , respectively. We store the sorted values of  $\{d_G(v_{i,j}, u) + |us| \mid u \in V \setminus (X_{T_i} \setminus X_i)\}$  for each  $v_{i,j} \in X_i$  in a list  $l_j^\uparrow$ , separately. We build  $\mathcal{L}_i^\uparrow$  from  $\{l_j^\uparrow\}$ .

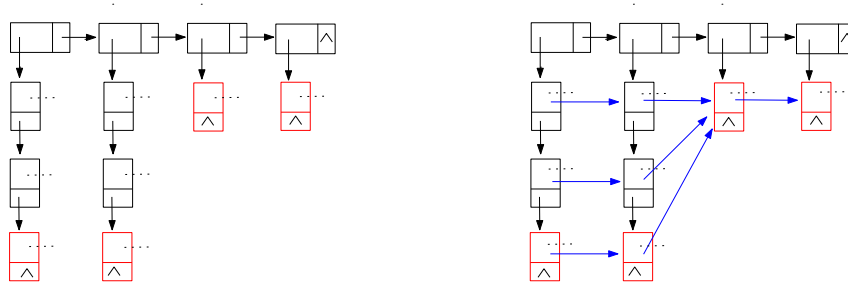
Besides  $d_{i,j}^\uparrow$  value, each node of the first level of  $\mathcal{L}_i^\uparrow$  stores a  $(|X_i| - 1)$ -level list. Each node of a second level of  $\mathcal{L}_i^\uparrow$  stores a  $(|X_i| - 2)$ -level list, and so on. Generally, a node in a  $k$ th ( $1 \leq k \leq |X_i|$ ) level list stores a  $(|X_i| - k)$ -level list. A  $k$ th level list is built recursively as follows. Assume  $d_{i,1}^\uparrow, \dots, d_{i,k-1}^\uparrow$  are the distance values contained in nodes of previous levels. We build this  $k$ th level list by  $l_k^\uparrow, \dots, l_{|X_i|}^\uparrow$ . For each  $d_{i,k}^\uparrow$  value in  $l_k^\uparrow$ , we check whether

the condition in Definition 17 is satisfied between  $d_{i,k}^\uparrow$  and each of  $d_{i,1}^\uparrow, \dots, d_{i,k-1}^\uparrow$ . If so, we create a node with the current  $d_{i,k}^\uparrow$  value, and build its  $(|X_i| - k)$ -level list recursively by  $l_{k+1}^\uparrow, \dots, l_{|X_i|}^\uparrow$ . In this way, we can build  $\mathcal{L}_i^\uparrow$  in  $O(bn^b)$  time. An example  $\mathcal{L}_i^\uparrow$  is shown in Figure 9(a). The  $d_{i,2}^\uparrow$  value stored at a red node in a vertical (second level) list is realized by the same vertex as the  $d_{i,1}^\uparrow$  value stored at its first level list node. Feasible values of  $t_i^\downarrow$  are constructed in the same way. To access values of  $R_i$  fast, we can encode feasible  $t_i^\uparrow$  and  $t_i^\downarrow$  values as indices into a multidimensional array. We can then access  $R_i$  values in  $O(b)$  time.

► **Lemma 18.** *For any node  $i$  of  $(X, T)$ , the feasible values of  $t_i^\uparrow$  ( $t_i^\downarrow$ ) can be computed in  $O(bn^b)$  time.  $R_i$  values can be accessed in  $O(b)$  time.*

We now discuss how to compute a  $W_i$ . The feasible  $t_i^\downarrow$  values for a  $W_i$  are just the feasible  $t_i^\downarrow$  values for  $R_i$ . However, to deal with the components in  $t_{i2}^\downarrow$ , we build the multilist for  $t_i^\downarrow$  by using components in  $t_{i1}^\downarrow$  for the first  $|t_{i1}^\downarrow|$  levels and using components in  $t_{i2}^\downarrow$  for the remaining  $|t_{i2}^\downarrow|$  levels. After building the multilist for  $t_i^\downarrow$  in this order, we compute values of  $W_i$  in  $|t_{i2}^\downarrow|$  rounds. Assume  $t_{i2}^\downarrow = (d_{i,j_1}^\downarrow, \dots, d_{i,j_s}^\downarrow)$ . The first round computes values such that  $d_{i,j_1}^\downarrow, \dots, d_{i,j_{s-1}}^\downarrow$  are fixed while  $d_{i,j_s}^\downarrow$  is greater than or equal to the specified value. The second round computes values such that  $d_{i,j_1}^\downarrow, \dots, d_{i,j_{s-2}}^\downarrow$  are fixed while  $d_{i,j_{s-1}}^\downarrow, d_{i,j_s}^\downarrow$  are greater than or equal to the specified values. And so on. By adding links between nodes in the multilist, each round build its result on the previous round. An example is shown in Figure 9(b). In the example,  $t_{i2}^\downarrow = t_i^\downarrow$  and  $|X_i| = 2$ . The links added for the 2nd round are drawn in blue.

► **Lemma 19.** *For a given  $t_i^\uparrow$ ,  $W_i$  values for all feasible  $t_i^\downarrow$  and  $\bar{k}$  can be computed in  $O(kb^2n^b)$  time. All values of a  $W_i$  can be computed in  $O(kb^2n^{2b})$  time.*



■ **Figure 9** (a)  $\mathcal{L}_i^\downarrow$  where  $|X_i| = 2$ . (b) Blue links between nodes in  $\mathcal{L}_i^\downarrow$ . The value stored in the pointed to node is greater than or equal to the value stored in the pointed from node.

# Linear-Time Approximation Scheme for $k$ -Means Clustering of Axis-Parallel Affine Subspaces

Kyungjin Cho ✉

POSTECH, Pohang, South Korea

Eunjin Oh ✉

POSTECH, Poahng, South Korea

---

## Abstract

In this paper, we present a linear-time approximation scheme for  $k$ -means clustering of *incomplete* data points in  $d$ -dimensional Euclidean space. An *incomplete* data point with  $\Delta > 0$  unspecified entries is represented as an axis-parallel affine subspace of dimension  $\Delta$ . The distance between two incomplete data points is defined as the Euclidean distance between two closest points in the axis-parallel affine subspaces corresponding to the data points. We present an algorithm for  $k$ -means clustering of axis-parallel affine subspaces of dimension  $\Delta$  that yields an  $(1 + \epsilon)$ -approximate solution in  $O(nd)$  time. The constants hidden behind  $O(\cdot)$  depend only on  $\Delta, \epsilon$  and  $k$ . This improves the  $O(n^2d)$ -time algorithm by Eiben et al. [SODA'21] by a factor of  $n$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases**  $k$ -means clustering, affine subspaces

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.46

**Related Version** *Full Version:* <https://arxiv.org/abs/2106.14176>

**Funding** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2020R1C1C1012742).

## 1 Introduction

Clustering is a fundamental research topic in computer science, which arises in various applications [13], including pattern recognition and classification, data mining, image analysis, and machine learning. In clustering, the objective is to group a set of data points into clusters so that the points from the same cluster are similar to each other. Usually, input points lie in a high-dimensional space, and the similarity between two points is defined as their distance. Two of the popular clusterings are  $k$ -median and  $k$ -means clusterings. In the  $k$ -means clustering problem, we wish to partition a given point set into  $k$  clusters to minimize the sum of squared distances of each point to its cluster center. Similarly, in the  $k$ -median clustering problem, we wish to partition a given point set into  $k$  clusters to minimize the sum of distances of each point to its cluster center.

In this paper, we consider clustering for *incomplete data points*. The analysis of incomplete data is a long-standing challenge in practical statistics. There are lots of scenarios where entries of points of a given data set are incomplete [2]. For instance, a few questions are left blank on a questionnaire; weather records for a region omit the figures for one weather station for a short period because of a malfunction; stock exchange data is absent for one stock on one day because of a trading suspension. Various heuristic, greedy, convex optimization, statistical, or even ad hoc methods were proposed throughout the years in different practical domains to handle missing data [2].

Gao et al. [10] introduced a geometric approach to deal with incomplete data points for clustering problems. An *incomplete* point has one or more unspecified entries, which can be represented as an axis-parallel affine subspace. The distance between two incomplete data



© Kyungjin Cho and Eunjin Oh;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 46; pp. 46:1–46:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

points is defined as the Euclidean distance between two closest points in the axis-parallel affine subspaces corresponding to the data points. Since the distance between an axis-parallel affine subspace and a point is well-defined, the classical clustering problems such as  $k$ -means,  $k$ -median, and  $k$ -center can be defined on a set of axis-parallel affine subspaces.

The  $k$ -center problem in this setting was studied by [10, 11, 15]. Gao et al. [10, 11] focused on the  $k$ -center clustering for  $k \leq 3$ , and presented an approximation algorithm for the  $k$ -center clustering of axis-parallel affine subspaces. Later, Lee and Schulman [15] improved the running time of the algorithm by Gao et al., and then presented an  $O(nd)$ -time approximation algorithm for the  $k$ -center clustering problem for a larger  $k$ . The constant hidden behind  $O(\cdot)$  depends on  $\Delta, \epsilon$  and  $k$ . Moreover, they showed that the running time of an approximation algorithm with any approximation ratio cannot be polynomial in even one of  $k$  and  $\Delta$  unless  $P = NP$ , and thus the running time of their algorithm is almost tight.

Very recently, Eiben et al. [7] presented an approximation algorithm for the  $k$ -means clustering of  $n$  axis-parallel affine subspaces of dimension  $\Delta$ . Their algorithm yields an  $(1 + \epsilon)$ -approximate solution in  $O(n^2 d)$  time with probability  $O(1)$ . The constant hidden behind  $O(\cdot)$  depends on  $\Delta, \epsilon$  and  $k$ . Since the best-known algorithm for the  $k$ -center clustering in this setting runs in time linear in both  $n$  and  $d$  (but exponential in both  $k$  and  $\Delta$ ), it is a natural question if a similar time bound can be achieved for the  $k$ -means clustering. In this paper, we resolve this natural question by presenting an  $(1 + \epsilon)$ -approximation algorithm for the  $k$ -means clustering problem running in time linear in  $n$  and  $d$ .

**Related work.** The  $k$ -median and  $k$ -means clustering problems for *points* in  $d$ -dimensional Euclidean space have been studied extensively. Since these problems are NP-hard even for  $k = 2$  or  $d = 2$  [3, 16, 18], the study of  $k$ -means and  $k$ -median clusterings have been devoted to obtain  $(1 + \epsilon)$ -approximation algorithms for these problems [1, 5, 8, 12, 14]. These algorithms run in time polynomial time in the input size if one of  $k$  and  $d$  is constant. Indeed, it is NP-hard to approximate Euclidean  $k$ -means clustering within a factor better than a certain constant larger than one [4]. That is, the  $k$ -means clustering problem does not admit a PTAS for arbitrary  $k$  and  $d$  unless  $P=NP$ .

Also, the clustering problems for *lines* (which are not necessarily axis-parallel) also have been studied [17, 19]. More specifically, Ommer and Malik [19] presented a heuristic for  $k$ -median clustering of lines in three-dimensional space. Later, Marom and Feldman [17] presented an algorithm for computing a coresets of size  $dk^{O(k)} \log n/\epsilon^2$ , which gives a polynomial-time  $(1 + \epsilon)$ -approximation algorithm for the  $k$ -means clustering of lines in  $d$ -dimensional Euclidean space.

**Our results.** We present an algorithm for  $k$ -means clustering of axis-parallel affine subspaces of dimension  $\Delta$  that yields an  $(1 + \epsilon)$ -approximate solution in  $2^{O(\frac{\Delta^4 k}{\epsilon} (\log \frac{\Delta}{\epsilon} + k))} dn$  time with a constant probability. This improves the previously best-known algorithm by Eiben et al [7], which takes  $2^{O(\frac{\Delta^7 k^3}{\epsilon} (\log \frac{k\Delta}{\epsilon}))} dn^2$  time. Since it is a generalization of the  $k$ -means clustering problem for points ( $\Delta = 0$ ), it does not admit a PTAS for arbitrary  $k$  and  $d$  unless  $P=NP$ . Similarly to Lee and Schulman [15], we show in the full version of this paper that an approximation algorithm with any approximation ratio cannot run in polynomial time in even one of  $k$  and  $\Delta$  unless  $P=NP$ . The running time of our algorithm is almost tight in the sense that it is linear in  $nd$  and exponential in  $k$  and  $\Delta$ .

► **Theorem 1** ([4, 15]). *No algorithm for computing an  $(1+\alpha)$ -approximate  $k$ -means clustering runs in time polynomial of  $n, d$  and  $\Delta$  (or polynomial of  $n, d$  and  $k$ ) unless  $P=NP$ .*

This lower bound does not rule out the possibility that this problem can be solved in  $O(nd + f(k, \Delta))$  time for an exponential function  $f$  of  $k$  and  $\Delta$ . However, it seems hard to achieve this goal using the framework of Kumar et al. [14] and Ackermann et al. [1] as their algorithms (for the standard clustering problem) also run in  $O(nd \cdot f(k))$  time for an exponential function  $f$  of  $k$ .

► **Remark 2.** Although it seems hard to achieve a significantly better running time using the framework of Ackermann et al., there is a merit of using this framework: we can handle outliers without additional effort as shown in [9]. Details will be discussed in Conclusion.

## 2 Preliminaries

We consider points in  $\mathbb{R}^d$  with missing entries in some coordinates. Let us denote the missing entry value by  $\otimes$ , and let  $\mathbb{H}^d$  denote the set of elements of  $\mathbb{R}^d$  where we allow some coordinates to take the value  $\otimes$ . Furthermore, we call a point in  $\mathbb{H}^d$  a  $\Delta$ -missing point if at most  $\Delta$  of its coordinates have value  $\otimes$ . We use  $[k]$  to denote the set  $\{1, \dots, k\}$  for any integer  $k \geq 1$ . For any point  $u \in \mathbb{H}^d$  and an index  $i \in [d]$ , we use  $(u)_i$  to denote the entry of the  $i$ -th coordinate of  $u$ . If it is clear from the context, we simply use  $u_i$  to denote  $(u)_i$ . Throughout this paper, we use  $i$  or  $j$  to denote an index of the coordinates of a point, and  $t$  to denote an index of a sequence (of points or sets). We use  $(u_t)_{t \in [k]}$  to denote a  $k$ -tuple consisting of  $u_1, u_2, \dots, u_k$ .

**Distance between two  $\Delta$ -missing points.** The *domain* of a point  $u$  in  $\mathbb{H}^d$ , denoted by  $\text{DOM}(u)$ , is defined as the set of coordinate-indices  $i \in [d]$  with  $(u)_i \neq \otimes$ . For a set  $I$  of coordinate-indices in  $[d]$ , we say that  $u$  is *fully defined* on  $I$  if  $\text{DOM}(u) \subseteq I$ . Similarly, we say that  $u$  is *partially defined* on  $I$  if  $\text{DOM}(u) \cap I \neq \emptyset$ . For a set  $P$  of points of  $\mathbb{H}^d$  and a set  $I$  of coordinate-indices in  $[d]$ , we use  $\text{FD}(P, I)$  to denote the set of points of  $P$  fully defined on  $I$ . Similarly, we use  $\text{PD}(P, I)$  to denote the set of points of  $P$  partially defined on  $I$ . The *null* point is a point  $p \in \mathbb{H}^d$  such that  $(p)_i = \otimes$  for all indices  $i \in [d]$ . With a slight abuse of notation, we denote the null point by  $\otimes$  if it is clear from the context. Also, we sometimes use  $I_t$  to denote  $\text{DOM}(u_t)$  if it is clear from the context.

Notice that a  $\Delta$ -missing point in  $\mathbb{H}^d$  can be considered as a  $\Delta$ -dimensional affine subspace in  $\mathbb{R}^d$ . The distance between two  $\Delta$ -missing points in  $\mathbb{H}^d$  is defined as the Euclidean distance between their corresponding  $\Delta$ -dimensional affine subspaces in  $\mathbb{R}^d$ . More generally, we define the *distance* between two points  $x$  and  $y$  in  $\mathbb{H}^d$  on a set  $I \subseteq [d]$  as

$$d_I(x, y) = \sqrt{\sum_{i \in I} |x_i - y_i|^2},$$

where  $|a - b| = 0$  for  $a = \otimes$  or  $b = \otimes$  by convention.

**The  $k$ -Means clustering of  $\Delta$ -missing points.** In this paper, we consider the  *$k$ -means clustering* of  $\Delta$ -missing points of  $\mathbb{H}^d$ . As in the standard setting (for  $\Delta = 0$ ), we wish to partition a given point set  $P$  into  $k$  clusters to minimize the sum of squared distances of each point to its cluster center. For any partition  $(P_t)_{t \in [k]}$  of  $P$  into  $k$  clusters such that each cluster  $P_t$  is associated with a cluster center  $c_t \in \mathbb{R}^d$ , the *cost* of the partition is defined as the sum of squared distances of each point in  $P$  to its cluster center.

To be more precise, we define the *clustering cost* as follows. For a set  $P \subset \mathbb{H}^d$  and a  $\Delta$ -missing point  $y$ , we use  $\text{COST}(P, y)$  to denote the sum of squared distances of each point in  $P$  to  $y$ . We also define the cost on a coordinate set  $I \subseteq [d]$ , denoted by  $\text{COST}_I(P, y)$ , as

the sum of squared distances on  $I$  between the points in  $P$  and their cluster centers. That is,  $\sum_{x \in P} d_I(x, y)^2$ . For convention,  $\text{COST}_i(P, y) = \text{COST}_{\{i\}}(P, y)$  for  $i \in [d]$ . The clustering cost  $\text{COST}((P_t)_{t \in [k]}, (c_t)_{t \in [k]})$  of clustering  $((P_t)_{t \in [k]}, (c_t)_{t \in [k]})$  is defined as  $\sum_{t \in [k]} \text{COST}(P_t, c_t)$ .

Now we introduce two properties of an optimal clustering  $((P_t^*)_{t \in [k]}, (c_t^*)_{t \in [k]})$  that minimizes the clustering cost, which will be frequently used throughout this paper. For each cluster  $P_t^*$ ,  $\text{COST}(P_t^*, c_t)$  is minimized when  $c_t$  is the *centroid* of  $P_t^*$  [7]. That is,  $c_t^*$  is the centroid of  $P_t^*$ . For a set  $P$  of points in  $\mathbb{H}^d$ , the *centroid* of  $P$ , denoted by  $c(P)$ , is defined as

$$(c(P))_i = \begin{cases} \otimes & \text{if } \text{PD}(P, i) = \emptyset, \\ \frac{\sum_{u \in \text{PD}(P, i)} u_i}{|\text{PD}(P, i)|} & \text{otherwise.} \end{cases}$$

Also, the clustering cost is minimized when  $(P_t^*)_{t \in [k]}$  forms the Voronoi partition of  $P$  induced by  $(c_t^*)_{t \in [k]}$  [7]. That is,  $(P_t^*)_{t \in [k]}$  is the partition of  $P$  into  $k$  clusters in such a way that  $c_t^*$  is the closest cluster point from any point  $p$  in  $P_t^*$ .

**Sampling.** Our algorithm uses random sampling to compute an approximate  $k$ -means clustering. Lemma 3 is a restatement of [1, Lemma 2.1], and Lemma 4 is used in [7] implicitly. Since Lemma 4 is not explicitly stated in [7], we give a sketch of the proof in the full version of this paper [6].

► **Lemma 3** ([1, Lemma 2.1]). *Assume that we are given a set  $P$  of points in  $\mathbb{H}^d$ , an index  $i \in [d]$ , and an approximation factor  $\alpha > 0$ . Let  $Q$  be a subset of  $P$  with  $|\text{PD}(Q, i)| \geq c|P|$  for some constant  $c$ , which is not given explicitly. Then we can compute a point  $x$  of  $\mathbb{R}$  in  $O(|P|d m_{\alpha, \delta})$  time satisfying with probability  $\frac{1-\delta}{5} 2^{\Omega(-m_{\alpha, \delta} \log(\frac{1}{c} m_{\alpha, \delta}))}$  that*

$$\text{COST}_i(Q, x) \leq (1 + \alpha) \text{COST}_i(Q, c(Q)),$$

where  $m_{\alpha, \delta} \in O(1/(\alpha\delta))$ .

► **Lemma 4** ([7]). *Assume that we are given a set  $P$  of  $\Delta$ -missing points in  $\mathbb{H}^d$  and an approximation factor  $\alpha > 0$ . Let  $Q$  be a subset of  $P$  with  $|Q| \geq c|P|$  for some constant  $c$  with  $0 < c < 1$ , which is not given explicitly. Then we can compute a  $\Delta$ -missing point  $u \in \mathbb{H}^d$  in  $O(|P|d\lambda)$  time satisfying with probability  $\frac{c^{8(\Delta+1)\lambda+1}}{4(4\Delta)^{8\Delta\lambda}}$  that*

$$\text{COST}_I(Q, u) < (1 + \alpha) \text{COST}_I(Q, c(Q)),$$

where  $I$  denotes the domain of  $u$ , and  $\lambda = \max\{(\frac{3}{\alpha})^{1/(2\Delta)}, (128\Delta^3)^{1/(2\Delta)}\}$ .

### 3 Overview of the Algorithm

We first briefly describe a  $(1 + \epsilon)$ -approximation algorithm for  $k$ -means clustering for points in  $d$ -dimensional Euclidean space given by Kumar et al. [14]. Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$ , and  $((P_t^*)_{t \in [k]}, (c_t^*)_{t \in [k]})$  be an optimal  $k$ -means clustering for  $P$ .

**Sketches of [1] and [14].** The algorithm of Kumar et al. [14] consists of several phases of two types: sampling phases and pruning phases. Their idealized strategy is as follows. At the beginning of a phase, it decides the type of the phase by computing the index  $t$  that maximizes  $|P_t^*|$ . If the cluster center of  $P_t^*$  has not been obtained, the algorithm enters the sampling phase. This algorithm picks a random sample of a constant size from  $P$ , and hopefully this sample would contain enough random samples from  $P_t^*$ . Then one can compute a good approximation  $c_t$  to  $c_t^*$  using Lemma 3.

If it is not the case, the algorithm enters a pruning phase, and it assigns each point to its closest cluster if their distance is at most  $L$ , where  $L$  denotes the smallest distance between two cluster centers we have obtained so far. They repeat this until all cluster centers are obtained, and finally obtain a good approximation to  $(P_t^*)_{t \in [k]}$ .

However, obviously, it is hard to implement this idealized strategy. To handle this, they try all possibilities (for both pruning and sampling phases and for all indices  $t \in [k]$  to be updated for sampling phases), and return the best solution found this way. Kumar et al. [14] showed that their algorithm runs in  $O(2^{(k/\epsilon)^{O(1)}} dn)$  time, and returns an  $(1 + \epsilon)$ -approximate  $k$ -means clustering with probability  $1/2$ . Later, Ackermann et al. [1] gave a tighter bound on the running time of this algorithm.

**Sketch of Eiben et al. [7].** To handle  $\Delta$ -missing points, Eiben et al. generalized the algorithm in [14]. Their idealized strategy can be summarized as follows. It maintains  $k$  centers  $(u_t)_{t \in [k]}$ , which are initially set to the null points. In each sampling phase, it obtains one (or at least  $\lfloor d \rfloor - \Delta$ ) coordinate of one of the centers.

At the beginning of a phase, it decides the type of the phase by computing the index  $t$  that maximizes  $|\text{PD}(P_t^*, [d] - I_t)|$ . A sampling phase happens if  $|\text{PD}(P_t^*, [d] - I_t)| > c|R|$ , where  $R$  denotes the number of points which are not yet assigned to any cluster. In this case, a random sample of constant size from  $R$  would contain enough random samples from  $|\text{PD}(P_t^*, j)|$  with  $j \in [d] - I_t$ . Thus, using the random sample, one can obtain a good approximation to  $(c_t^*)_j$ .

Otherwise, a pruning phase happens. In a pruning phase, the algorithm assigns points which are not yet assigned to any cluster to clusters. Here, a main difficulty is that even though the distance between a point  $p$  in  $R$  and its closest center  $u_t$  is at most  $L$ , where  $L$  denotes the distance between two cluster centers,  $p$  is not necessarily in  $P_t^*$ . They resolved this in a clever way by ignoring  $\Delta$  coordinates for comparing the distances from two cluster centers.

**Comparison of our contribution and Eiben et al. [7].** Our contribution is two-fold: the dependency on  $n$  decreases to  $O(n)$  from  $O(n^2)$ , and the dependency of  $\Delta$  and  $k$  decreases significantly. First, the improvement on the dependency of  $n$  comes from introducing a faster and simpler procedure for a pruning phase. In the previous algorithm, it cannot be guaranteed that a constant fraction of points of  $R$  is removed from  $R$ . This yields the quadratic dependency of  $n$  in their running time. We overcome the difficulty they (and we) face in a pruning phase in a different way. For each subset  $T$  of  $[k]$ , we consider the set  $S_T$  of points  $x \in R$  such that  $\text{DOM}(x) \subset I_t$  for every  $t \in T$  and  $\text{DOM}(x) \not\subset I_{t'}$  for every  $t' \notin T$ . Then  $S_T$ 's for all subsets  $T \subset [k]$  form a partition of  $R$ . In a pruning phase, we choose the set  $S_T$  that maximizes  $|S_T|$ . We show that the size of  $S_T$  is at least a constant fraction of  $|R|$  (unless we enter the sampling phase). Moreover, in this case, if the distance between a point  $p$  in  $S_T$  and its closest center  $u_t$  is at most  $L$ , where  $L$  denotes the distance between two cluster centers, it holds that  $p \in P_t^*$ .

Second, we improved the dependency of  $\Delta$  and  $k$  using the framework of Ackermann et al. [1]. To adapt this framework for our problem, we are required to handle several technical difficulties. This is mainly because the center of each cluster changes during the execution of the algorithm in our case unlike the standard setting considered in [1].

Due to the lack of space, some proofs are omitted. All missing proofs can be found in the full version of this paper.



---

**Algorithm 1 Idealized  $k$ -Means.**


---

```

input : A set  $P$  of  $\Delta$ -missing points in the plane
output : A  $(1 + \epsilon)$ -approximate  $k$ -means clustering for  $P$ 
1  $R \leftarrow P$  and  $P_t \leftarrow \emptyset$  for all cluster-indices  $t \in [k]$ 
2 Initialize  $\mathcal{U} = \langle u_1, \dots, u_k \rangle$  so that  $(u_t)_i = \otimes$  for all cluster-indices  $t \in [k]$  and  $i \in [d]$ 
3 while  $R \neq \emptyset$  do
4   Let  $t$  be the cluster-index that maximizes  $|\text{PD}(R \cap P_t^*, [d] - I_t)|$ 
5   if  $|\text{PD}(R \cap P_t^*, [d] - I_t)| \geq c|R|$  then
6     /* sampling phase */
7     if  $I_t = \emptyset$  then
8       /* We resrepresent this sampling phase as  $(t, \text{DOM}(u_t))$  */
9        $u_t \leftarrow$  The  $\Delta$ -missing point obtained from Lemma 4
10    else
11      /* We resrepresent this sampling phase as  $(t, \{j\})$  */
12      Let  $j$  be the coordinate-index in  $[d] - I_t$  that maximizes  $\text{PD}(R \cap P_t^*, j)$ 
13       $(u_t)_j \leftarrow$  The value obtained from Lemma 3
14      Assign the points in  $\text{FD}(R, \cap_{t \in [k]} I_t)$  to their closest cluster centers in  $\mathcal{U}$ 
15       $R \leftarrow R - \text{FD}(R, \cap_{t \in [k]} I_t)$ 
16    else
17      /* pruning phase */
18      /*  $\mathcal{U}_T$  is the set of all  $u_t$  for  $t \in T$  */
19       $T \leftarrow$  The set of cluster-indices that maximizes  $|S_T|$ 
20       $B \leftarrow$  The first half of  $S_T$  sorted in ascending order of distance from  $\mathcal{U}_T$ 
21      Assign the points in  $B$  to their closest cluster centers in  $\mathcal{U}_T$ 
22       $R \leftarrow R - B$ 
23 return  $\mathcal{U}$ 

```

---

**4 For  $k$ -Means Clustering**

In this section, we describe and analyze for a  $k$ -means clustering algorithm. Let  $\mathcal{U} = \langle u_1, u_2, \dots, u_k \rangle$  be a sequence of points in  $\mathbb{H}^d$ .

**4.1 Algorithm Using the Counting Oracle**

We first sketch an algorithm for  $k$ -clustering assuming that we can access the *counting oracle*. Let  $\langle P_1^*, \dots, P_k^* \rangle$  be an optimal  $k$ -clustering for  $P$  induced by the centroids  $\langle c_1^*, \dots, c_k^* \rangle$ . The counting oracle takes a subset of  $P$  and a cluster-index  $t \in [k]$ , and it returns the number of points in the subset which are contained in  $P_t^*$ . Then in Section 4.3, we show how to compute an approximate clustering without the counting oracle.

The algorithm consists of several phases of two types: a *sampling phase* or a *pruning phase*. We initialize  $\mathcal{U}$  to a  $k$ -tuple of null points. In a sampling phase, we obtain values of  $(u_t)_j$  for indices  $t \in [k]$  and  $j \in [d]$  which were set to  $\otimes$ . Also, we assign points of  $P$  to one of the  $k$  clusters in sampling and pruning phases. The pseudocode of the algorithm is described in Algorithm 1. At any moment during the execution of the algorithm, we maintain a set  $R$  of remaining points and a  $k$ -tuple  $\mathcal{U} = \langle u_1, \dots, u_k \rangle$  of (partial) centers in  $\mathbb{H}^d$ . We let  $I_t$  be  $\text{DOM}(u_t)$ . Initially,  $R$  is set to  $P$ , and  $\mathcal{U}$  is set to the  $k$ -tuple of null points. The algorithm terminates if  $R = \emptyset$ , and finally  $\mathcal{U}$  becomes a set of points in  $\mathbb{R}^d$ .

We consider the partition  $\mathcal{F}$  of  $R$  defined as follows. For a subset  $T$  of  $[k]$ , let  $S_T$  denote the set of points  $x \in R$  such that  $\text{DOM}(x) \subset I_t$  for every  $t \in T$  and  $\text{DOM}(x) \not\subset I_{t'}$  for every  $t' \notin T$ . Let  $\mathcal{F} = \{S_T \mid T \text{ is a proper subset of } [k]\}$ . The following lemma shows that  $\mathcal{F}$  is a partition of  $R$ .

► **Lemma 5.** *For any point  $x \in R$ , there exists a unique set in  $\mathcal{F}$  containing  $x$ .*

At the beginning of each phase, we decide the type of the current phase. Let  $t$  be the cluster-index of  $[k]$  that maximizes  $|\text{PD}(R \cap P_t^*, [d] - I_t)|$ , where  $R$  is the set of points of  $P$  which are not assigned to any cluster. The one of the following cases always happen:  $|\text{PD}(R \cap P_t^*, [d] - I_t)| \geq c|R|$ , or there exists a set  $T$  of cluster-indices in  $[k]$  such that  $|S_T \cap (\cup_{t \in T} P_t^*)| \geq c|R|$  for any constant  $c < 1/(2^k + k)$ , which will be specified later.<sup>1</sup>

► **Lemma 6.** *One of the following always holds for any constant  $c < 1/(2^k + k)$ .*

- $|\text{PD}(R \cap P_t^*, [d] - I_t)| \geq c|R|$  for a cluster-index  $t \in [k]$ , or
- $|S_T \cap (\cup_{t \in T} P_t^*)| \geq c|R|$  for a proper subset  $T$  of  $[k]$ .

**Sampling phase.** If the first case happens, we enter a sampling phase. Let  $\alpha$  be a constant, which will be specified later. We use it as an approximation factor used in Lemmas 3 and 4 for sampling. If  $I_t$  is empty, we replace  $u_t$  with a  $\Delta$ -missing point in  $\mathbb{H}^d$  obtained from Lemma 4. If  $I_t$  is not empty, then it is guaranteed that  $|I_t|$  is at least  $d - \Delta$ . We compute the coordinate-index  $j$  in  $[d] - I_t$  that maximizes  $|\text{PD}(R \cap P_t^*, j)|$  using the counting oracle. Clearly,  $(u_t)_j = \otimes$  and  $|\text{PD}(R \cap P_t^*, j)|$  is at least  $c|R|/\Delta$ . Then we replace  $(u_t)_j$  with a value obtained from Lemma 3. At the end of the phase, we check if  $\text{FD}(R, \cap_{t \in [k]} I_t)$  is not empty. If it is not empty, we assign those points to their closest cluster centers.

**Pruning phase.** Otherwise, we enter a pruning phase. Instead of obtaining a new coordinate value of  $u_t$ , we assign points of  $R$  to cluster centers in a pruning phase. To do this, we find a proper subset  $T$  of  $[k]$  which maximizes  $|S_T|$ . Then among the points of  $S_T$ , we choose the  $|S_T|/2$  points closest to their closest centers in  $\mathcal{U}_T$ , where  $\mathcal{U}_T$  is the set of all  $u_t$  for  $t \in T$ . Then we assign each of them to its closest center in  $\mathcal{U}_T$ . In this way, points in  $\cup_{t' \notin T} P_{t'}^*$  might be assigned (incorrectly) to  $u_t$  for a cluster-index  $t \in T$ . We call such a point a *stray point*.

## 4.2 Analysis of the Approximation Factor

In this section, we analyze the approximation factor of the algorithm. We let  $\mathcal{S}$  be the sequence of sampling phases happened during the execution of the algorithm. At the end of the algorithm,  $\mathcal{U}$  becomes a  $(1 + \epsilon)$ -approximate clustering as we will see later. In the following, we use  $\mathcal{C} = \langle c_1, \dots, c_k \rangle$  to denote the output of the algorithm. For a sequence  $T$  of cluster-indices, we use  $\mathcal{C}_T$  to denote a  $|T|$ -tuple consisting of  $c_t$  for  $t \in T$ . Let  $\text{OPT}_k(P)$  denote the clustering cost of an optimal  $k$ -means clustering of  $P$ .

► **Lemma 7.** *The size of  $\mathcal{S}$  is at most  $k(\Delta + 1)$ .*

**Preliminaries.** Recall that  $\mathcal{S}$  denotes the sequence of sampling phases. Here, we represent a sampling phase as the pair  $(t, I)$ , where  $t$  is the cluster-index considered in the sampling phase, and  $I$  is the set of indices  $i$  such that  $(u_t)_i$  is obtained during the sampling phase.

<sup>1</sup> We set  $\alpha = \epsilon/3$ , and  $c = \frac{\alpha}{8 \cdot 2^k k^2 (\Delta + 1)}$ .

Note that the size of  $I$  is either one or at least  $d - \Delta$  for any sampling phase. Also, for  $s = (t, I) \in \mathcal{S}$ , we let  $t^s = t$  and  $I^s = I$ . For each sampling phase  $s = (t, I)$ , let  $R^s$  be the set of points of  $P_t^*$  which are not assigned at the *beginning* of the phase. Furthermore, let  $I_t^s$  denote  $\text{DOM}(u_t)$  we have at the end of the sampling phase  $s$  for a cluster-index  $t' \in [k]$ . For two sampling phases  $s$  and  $s'$  in  $\mathcal{S}$ , we use  $s \preceq s'$  if  $s$  comes before  $s'$  in  $\mathcal{S}$  or equals to  $s'$ . We denote  $P_T^* = \cup_{t \in T} P_t^*$  and  $P_{\bar{T}}^* = \cup_{t' \notin T} P_{t'}^*$ .

**Sketch.** A point of  $P$  is assigned to one of the clusters during a sampling phase or a pruning phase. That is, at the end of the execution of the algorithm,  $P$  is partitioned into  $k$  clusters  $P_1, \dots, P_k$ . Note that it is not necessarily a Voronoi partition of  $P$  with respect to  $\mathcal{C}$ . Our goal in this section is that the clustering cost  $\text{COST}((P_t)_{t \in [k]}, (c_t)_{t \in [k]})$  is at most  $(1 + \epsilon)\text{OPT}_k(P)$ . We first show that the clustering cost induced by non-stray points is  $(1 + \alpha)\text{OPT}_k(P)$ , and then show that the clustering cost induced by stray points is  $\alpha\text{OPT}_k(P)$ .

For this, we use the two following technical lemmas. Lemma 8 is a consequence of Lemmas 3 and 4, and Claim 9 follows from construction. Proofs can be found in the Appendix of the full version [6].

► **Lemma 8.**  $\sum_{s \in \mathcal{S}} \text{COST}_{I^s}(R^s, c_{t^s}) \leq (1 + \alpha)\text{OPT}_k(P)$  with a probability at least  $p^k q^{k\Delta}$ , where  $q$  and  $p$  are the probabilities in Lemmas 3 and 4.

▷ **Claim 9.** For a sampling phase  $s'$  in  $\mathcal{S}$  and a proper subset  $T$  of  $[k]$ , let  $X$  be a point subset of  $S_T$  which are not assigned at the end of a sampling phase  $s'$ . Then

$$\text{COST}(X \cap P_T^*, \mathcal{C}_T) \leq \sum \text{COST}_{I^s}(\text{PD}(X \cap P_{t^s}^*, I^s), c_{t^s}),$$

where the summation is taken over all sampling phases  $s$  in  $\mathcal{S}$  with  $s \preceq s'$  and  $t^s \in T$ .

#### 4.2.1 Clustering Cost Induced by Non-Stray Points

We first show that the clustering cost induced by non-stray points is at most  $(1 + \alpha)\text{OPT}_k(P)$ . There are two types of non-stray points: the points assigned during the sampling phases, and the points assigned during the pruning phases which are not stray. The first term in the following lemma is the clustering cost induced by points of the first type, and the second term is the clustering cost induced by points of the second type. For a proper subset  $T$  of  $[k]$ , let  $A_T^s$  be the set of points in  $S_T$  assigned to  $\mathcal{U}$  during the consecutive pruning phases lying between two adjacent sampling phases  $s$  and  $s'$  with  $s \preceq s'$ .

► **Lemma 10.** Let  $S$  be the set of points assigned during the sampling phases.

$$\text{COST}(S, \mathcal{C}) + \sum_{s \in \mathcal{S}} \sum_{T \subsetneq [k]} \text{COST}(A_T^s \cap P_T^*, \mathcal{C}_T) \leq (1 + \alpha)\text{OPT}_k(P),$$

with a probability at least  $p^k q^{k\Delta}$ , where  $q$  and  $p$  are the probabilities in Lemmas 3 and 4.

**Proof.** A point  $p \in S$  is assigned to its closest center in  $\mathcal{C}$ . This is because we assign a point during a sampling phase only if it is fully defined on  $\text{DOM}(u_t)$  for all cluster-indices  $t \in [k]$ . Thus the following holds.

$$\text{COST}(S, \mathcal{C}) \leq \sum_{t \in [k]} \text{COST}(S \cap P_t^*, c_t) = \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(\text{PD}(S \cap R^s, I), c_{t^s}).$$

For the second term of this claim, we have the following:

$$\begin{aligned}
\sum_{s' \in \mathcal{S}} \sum_{T \subsetneq [k]} \text{COST}(A_T^{s'} \cap P_T^*, \mathcal{C}_T) &\leq \sum_{s' \in \mathcal{S}} \sum_{T \subsetneq [k]} \sum_{\substack{s \preceq s' \\ t^s \in T}} \text{COST}_{I^s}(\text{PD}(A_T^{s'} \cap P_{t^s}^*, I^s), c_{t^s}) \\
&= \sum_{s \in \mathcal{S}} \sum_{s \preceq s'} \sum_{t^s \in T \subsetneq [k]} \text{COST}_{I^s}(\text{PD}(A_T^{s'} \cap P_{t^s}^*, I^s), c_{t^s}) \\
&\leq \sum_{s \in \mathcal{S}} \sum_{s \preceq s'} \sum_{t^s \in T \subsetneq [k]} \text{COST}_{I^s}(\text{PD}(A_T^{s'} \cap R^s, I^s), c_{t^s}) \\
&\leq \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(\text{PD}(A \cap R^s, I^s), c_{t^s}),
\end{aligned}$$

where  $A$  denotes the set of points of  $P$  assigned during all pruning phases. The first inequality holds by Claim 9. The second and the last inequalities hold since they change only the ordering of summation. The third inequality holds since  $A_T^{s'} \cap P_{t^s}^* \subset R^s$  if  $s \preceq s'$ .

By combining previous properties together with Claim 9, we have:

$$\begin{aligned}
\text{COST}(\mathcal{S}, \mathcal{C}) + \sum_{s \in \mathcal{S}} \sum_{T \subsetneq [k]} \text{COST}(A_T^s \cap P_T^*, \mathcal{C}_T) &\leq \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(\text{PD}(S \cap R^s, I^s), c_{t^s}) \\
&\quad + \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(\text{PD}(A \cap R^s, I^s), c_{t^s}) \\
&\leq \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(R^s, c_{t^s}) \\
&\leq (1 + \alpha) \text{OPT}_k(P). \quad \blacktriangleleft
\end{aligned}$$

## 4.2.2 Clustering Cost Induced by Stray Points

Now we show that the clustering cost induced by stray points is  $\alpha \text{OPT}_k(P)$ . Recall that the stray points are assigned to clusters during pruning phases. We first analyze the clustering cost by considering consecutive pruning phases lying between two adjacent sampling phases of  $\mathcal{S}$ . Then we show that the overall clustering cost induced by stray points.

### 4.2.2.1 During consecutive pruning phases

Consider a sequence  $\mathcal{P}$  of the consecutive pruning phases lying between two adjacent sampling phases  $s$  and  $s'$  in  $\mathcal{S}$  with  $s \preceq s'$ . Let  $N$  denote the number of pruning phases in  $\mathcal{P}$ . For a proper subset  $T$  of  $[k]$ , recall that  $A_T^s$  denotes the set of points in  $S_T$  assigned to  $\mathcal{U}$  during the phases in  $\mathcal{P}$ . During this period,  $u_t$  remains the same for each cluster-index  $t \in [k]$ . By definition,  $A_T^s \cap P_T^*$  is the set of stray points assigned during the pruning phases in  $\mathcal{P}$ . For each pruning phase of  $\mathcal{P}$ , we choose a subset  $T$  of  $[k]$  and assign a half of  $S_T$  to  $\mathcal{U}$ . Here, note that distinct pruning phases of  $\mathcal{P}$  might choose distinct subsets  $T$ . Therefore, there might be more than one subsets  $T$  of  $[k]$  with  $A_T^s \neq \emptyset$ .

We first analyze the clustering cost induced by stray points assigned during the pruning phases in  $\mathcal{P}$ . Recall that  $R^s$  denotes the set of points of  $P$  which are not yet assigned to any cluster at the end of the sampling phase  $s$ . Let  $S_T^s$  denotes the set of points of  $x \in R^s$  such that  $\text{DOM}(x) \subset I_t$  for every  $t \in T$  and  $\text{DOM}(x) \not\subset I_{t'}$  for every  $t' \notin T$ .

► **Lemma 11.**  $\sum_{T \subsetneq [k]} \text{COST}(A_T^s \cap P_T^*, \mathcal{C}_T) \leq 8 \cdot 2^k ck \cdot \sum_{T \subsetneq [k]} \text{COST}(S_T^s \cap P_T^*, \mathcal{C}_T)$ .

**Proof (Sketch).** Since  $S_T^s$ 's form a partition of  $R^s$ , it suffices to fix a proper subset  $T$  of  $[k]$  and show that  $\text{COST}(A_T^s \cap P_T^*, \mathcal{C}_T) \leq 8 \cdot 2^k ck \cdot \text{COST}(S_T^s \cap P_T^*, \mathcal{C}_T)$ . We partition  $A_T^s$  into  $N$  subsets so that  $A_T^{(x)}$  is the set of points of  $S_T$  assigned to  $\mathcal{U}$  at the  $x$ th pruning phase of  $\mathcal{P}$ . By construction, in the  $x$ th pruning phase, there exists a unique index-set  $T$  with  $A_T^{(x)} \neq \emptyset$ . Let  $\mathcal{X}_T$  be the increasing sequence of indices  $x$  of  $[N]$  with  $A_T^{(x)} \neq \emptyset$ .

For any consecutive indices  $x$  and  $x'$  in  $\mathcal{X}_T$  with  $x' < x$  and any proper subset  $T$  of  $[k]$ , we show that the clustering cost induced by stray points assigned during the  $x'$ th pruning phase of  $\mathcal{P}$  is at most  $8 \cdot 2^k ck$  times the clustering cost induced by non-stray points assigned during the  $x$ th pruning phase of  $\mathcal{P}$ . Also, we analyze the clustering cost induced by stray points assigned during the last phase of  $\mathcal{P}$  similarly. The clustering cost induced by all non-stray points assigned during the pruning phases of  $\mathcal{P}$  is at most  $\sum_{T \subseteq [k]} \text{COST}(S_T^s \cap P_T^*, \mathcal{C}_T)$ , and thus the lemma holds. Details can be found in the full version of this paper.  $\blacktriangleleft$

#### 4.2.2.2 During the entire pruning phases

Recall that  $\mathcal{S}$  denotes the sequence of sampling phases, and each sampling phase is represented as  $(t, I)$ , where  $t$  is the cluster-index considered in the sampling phase, and  $I$  is the set of indices  $i$  such that  $(u_t)_i$  is obtained during the sampling phase.

The following lemma gives an upper bound of the total cost induced by the stray points.

► **Lemma 12.**  $\sum_{s \in \mathcal{S}} \sum_{T \subseteq [k]} \text{COST}(A_T^s \cap P_T^*, \mathcal{C}_T) \leq 8 \cdot 2^k ck^2 (\Delta + 1)(1 + \alpha) \text{OPT}_k(P)$ ,

with a probability at least  $p^k q^{k\Delta}$ , where  $q$  and  $p$  are the probabilities in Lemmas 3 and 4.

**Proof.** The lemma holds by the following inequalities. The first and second inequalities hold by Claims 11 and 9, respectively. The third one holds since it changes only the ordering of summation. The fourth one holds since for a fixed sampling phase  $s'$  in  $\mathcal{S}$ ,  $S_T^{s'}$  are disjoint for all proper subsets  $T$  of  $[k]$ . Also notice that, for two sampling phases  $s, s'$  in  $\mathcal{S}$  and a proper subset  $T$  of  $[k]$ ,  $R^s$  contains  $S_T^{s'} \cap P_{t^s}^*$  if  $s \preceq s'$ . The fifth one holds since the size of  $\mathcal{S}$  is at most  $k(\Delta + 1)$ . The last two hold by the definition of  $\text{COST}(\cdot)$ .

$$\begin{aligned}
\sum_{s \in \mathcal{S}} \sum_{T \subseteq [k]} \text{COST}(A_T^s \cap P_T^*, \mathcal{C}_T) &\leq 8 \cdot 2^k ck \sum_{s \in \mathcal{S}} \sum_{T \subseteq [k]} \text{COST}(S_T^s \cap P_T^*, \mathcal{C}_T) \\
&\leq 8 \cdot 2^k ck \sum_{s' \in \mathcal{S}} \sum_{T \subseteq [k]} \sum_{\substack{s \preceq s' \\ t^s \in T}} \text{COST}_{I^s}(\text{PD}(S_T^{s'} \cap P_{t^s}^*, I^s), c_{t^s}) \\
&= 8 \cdot 2^k ck \sum_{s \in \mathcal{S}} \sum_{s \preceq s'} \sum_{t^s \in T \subseteq [k]} \text{COST}_{I^s}(\text{PD}(S_T^{s'} \cap P_{t^s}^*, I^s), c_{t^s}) \\
&\leq 8 \cdot 2^k ck \sum_{s \in \mathcal{S}} \sum_{s \preceq s'} \text{COST}_{I^s}(\text{PD}(R^s, I^s), c_{t^s}) \\
&\leq 8 \cdot 2^k ck^2 (\Delta + 1) \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(\text{PD}(R^s, I^s), c_{t^s}) \\
&\leq 8 \cdot 2^k ck^2 (\Delta + 1) \sum_{s \in \mathcal{S}} \text{COST}_{I^s}(R^s, c_{t^s}), \\
&\leq 8 \cdot 2^k ck^2 (\Delta + 1)(1 + \alpha) \text{OPT}_k(P). \quad \blacktriangleleft
\end{aligned}$$

We can obtain the following lemma by combining Lemma 10 and Lemma 12.

► **Lemma 13.** For a constant  $\alpha > 0$ , the algorithm returns an  $(1 + 8 \cdot 2^k ck^2 (\Delta + 1))(1 + \alpha)$ -approximate  $k$ -means clustering for  $P$  with probability at least  $p^k q^{k\Delta}$ , where  $q$  and  $p$  are the probabilities in Lemmas 3 and 4.

---

**Algorithm 2** *k*-Means.

---

```

1  $R \leftarrow R - \text{FD}(R, \cap_{t \in [k]} I_t)$ 
2  $\mathcal{E} \leftarrow \emptyset$ 
3  $\mathcal{U}' = \langle u'_1, \dots, u'_k \rangle \leftarrow \mathcal{U} = \langle u_1, \dots, u_k \rangle$ 
4 if  $R = \emptyset$  then return  $u_{[k]}$ 
5 for  $t \in [k]$  do
6   if  $I_t = \emptyset$  then
7      $u'_t \leftarrow$  the  $\Delta$ -missing point obtained from Lemma 4
8     Add the clustering returned by k-Means  $(\mathcal{U}', R)$  to  $\mathcal{E}$ 
9   else
10    foreach  $j \in [d] - I_t$  do
11       $u'_t \leftarrow u_t$ 
12       $(u'_t)_j \leftarrow$  The value obtained from Lemma 3
13      Add the clustering returned by k-Means  $(\mathcal{U}', R)$  to  $\mathcal{E}$ 
14  $T \leftarrow$  the non-empty proper subset of  $[k]$  that maximizes  $|R \cap S_T|$ 
15 if  $|R \cap S_T| \geq |R|/(2^k - 1)$  then
16    $\mathcal{U}_T$  is the set of all  $u_t$  for  $t \in T$ 
17    $B \leftarrow$  The first half of  $|R \cap S_T|$  sorted in ascending order of distance from  $\mathcal{U}_T$ 
18   Add the clustering returned by k-Means  $(\mathcal{U}, R - B)$  to  $\mathcal{E}$ 
19 return the clustering  $\mathcal{C}$  in  $\mathcal{E}$  which minimizes  $\text{COST}(R, \mathcal{C})$ 

```

---

### 4.3 Algorithm without Counting Oracle

The algorithm we have described uses the counting oracle in two places: determining the type of the phase and selecting a pair of the cluster-index and coordinate-index to be updated in a sampling phase. In this section, we explain how to avoid using the counting oracle. To do this, we simply try all possible cases: run both phases and update each possible cluster for all indices during a sampling phase. The main algorithm, *k*-Means  $(\mathcal{U}, R)$ , is described in Algorithm 2. Its input consists of cluster centers  $\mathcal{U}$  of a partial clustering of  $P$  and a set  $R$  of points of  $P$  which are not yet assigned. Finally, *k*-Means  $(\otimes_k, P)$  returns an  $(1 + 8 \cdot 2^k ck^2(\Delta + 1))(1 + \alpha)$ -approximate *k*-means clustering of  $P$ , where  $\otimes_k$  denotes the *k*-tuple of  $\mathbb{H}^d$  of null points.

The clustering cost returned by *k*-Means  $(\otimes_k, P)$  is at most the cost returned by the algorithm which uses the counting oracle in Section 4.1. In the following, we analyze the running time of *k*-Means  $(\otimes_k, P)$ . Let  $T(n, \delta)$  be the running of *k*-Means  $(\mathcal{U}, R)$  when  $n = |R|$  and  $\delta = \sum_{t \in [k]} \min\{d - |I_t|, \Delta + 1\}$ . Here,  $\delta$  is an upper bound on the number of updates required to make  $I_t = [d]$  for every cluster-index  $t$  in  $[k]$ . Then we have the following recurrence relation for  $T(n, \delta)$ .

▷ **Claim 14.**  $T(n, \delta) \leq \delta \cdot T(n, \delta - 1) + T\left(\left(1 - \frac{1}{2^{k+1}-2}\right)n, \delta\right) + O\left(\frac{k\delta\Delta^3 dn}{\alpha}\right)$

*Proof.* In a sampling phase, *k*-Means calls itself at most  $\delta$  times recursively with different parameters. Each recursive call takes  $T(n, \delta - 1)$  time. Also, the time for updating cluster centers takes  $O(\delta\Delta^3 dn/\alpha)$  in total by Lemma 3 and 4. For a pruning phase, we compute  $|R \cap S_T|$  for each  $T \subsetneq [k]$  in total  $O(dn)$  time, and then choose the first half of  $S_T$  in increasing order of the distances from  $u_T$  in total  $O(kdn)$  time. The recursive call invoked in the pruning phase takes  $T\left(\left(1 - \frac{1}{2^{k+1}-2}\right)n, \delta\right)$  time. We have  $\delta + 1$  clusterings returned by recursive calls in total, and we can choose  $c_{[k]}$  in  $O(\delta kdn)$  time. Thus, the claim holds. ◁

By solving the recurrence relation, we can obtain an upper bound of  $T(n, \delta)$ .

▷ **Claim 15.**  $T(n, \delta) \leq (2\delta(2^k - 1))^{2\delta+1} (1 + \frac{1}{2^{k+1}-3})^{\delta^2} \Delta^3 kdn/\alpha$

We obtain the following theorem by setting  $\alpha = \epsilon/3$ , and  $c = \frac{\alpha}{8 \cdot 2^k k^2 (\Delta+1)}$ .

► **Theorem 16.** *Given a  $\Delta$ -missing  $n$ -point set  $P$  in  $\mathbb{H}^d$ , a  $(1 + \epsilon)$ -approximate solution to the  $k$ -means clustering problem can be found in  $2^{O(\max\{\Delta^4 k(\log \Delta + k), \frac{\Delta^k}{\epsilon}(\log \frac{1}{\epsilon} + k)\})} dn$  time with a constant probability  $1/2$ .*

## 5 Concluding Remarks

In this paper, we gave a linear-time approximation algorithm for  $k$ -means clustering on axis-parallel affine subspaces. Our algorithm runs in time linear in  $nd$ , which is the size of the input. The bound is almost tight in the sense that no  $(1 + \epsilon)$ -approximation algorithm for this problem runs in time polynomial in even one of  $k$  and  $\Delta$ .

Our algorithm is based on the framework of Kumar et al. [14] and Ackerman et al. [1]. One merit of using this framework is that we can handle outliers without additional effort as shown in [9]. In this case, the goal is to minimize the clustering cost allowing to remove a small portion of the input data. The problem of computing a  $k$ -means clustering of missing data has not been explicitly considered in the presence of outliers. However, the observation of [9] allows us to extend our algorithm to handle outliers.

A main idea of [9] for clustering points in  $\mathbb{R}^d$  is as follows. Let  $m$  be the number of outliers (the number of points which are allowed to be removed). Consider an optimal  $k$ -means clustering  $(P_1^*, \dots, P_k^*)$  of the input point set  $P$  after removing  $m$  outliers. Then the largest cluster, say  $P_1^*$ , has size at least  $(|P| - m)/k$ . Then by picking a random sample of a constant size (but the sample size depends on  $m$ ), one can compute a good approximation to  $c(P_1^*)$  using Lemma 3. Using this observation, Feng et al. [9] showed that the algorithm by Kumar et al. [14] (using a parameter slightly larger than the standard one for Lemma 3) computes the cluster centers of an approximation  $k$ -means clustering in the presence of  $m$  outliers. Then the  $m$  points farthest from the cluster centers are  $m$  outliers.

The observation by Feng et al. also holds for  $\Delta$ -missing points. In a sampling phase, the set  $R$  of remaining points contains at most  $m$  outliers. This means that the largest set  $S_T$  contains at least  $|R|/(2^k + k + m)$  points of  $P_T^*$ . Therefore, we can apply Lemma 6 using a constant  $c < 1/(2^k + k + m)$ . Thus we can handle  $m$  outliers in  $O(nd)$  time, where the constant hidden behind  $O(\cdot)$  depends on  $m$ .

As mentioned in Introduction, the lower bound in Theorem 1 does not rule out the possibility that this problem can be solved in  $O(nd + f(k, \Delta))$  time for an exponential function  $f$  of  $k$  and  $\Delta$ . Moreover, it seems hard to achieve this goal using the framework of Kumar et al. [14] and Ackermann et al. [1] as their algorithms also run in  $O(nd \cdot f(k))$  time for an exponential function of  $k$ . It is an interesting open question whether one can improve the running time to  $O(nd + f(k, \Delta))$ . Also, obtaining a coresets for this problem is also an interesting open question.

---

## References

- 1 Marcel R. Ackermann, Johannes Blömer, and Christian Sohler. Clustering for metric and nonmetric distance measures. *ACM Transactions on Algorithms*, 6(4), September 2010.
- 2 P.D. Allison. *Missing Data*. Number no. 136 in Missing Data. SAGE Publications, 2001. URL: <https://books.google.co.kr/books?id=ZtYArHXjpB8C>.



- 3 Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Papat. NP-hardness of Euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248, 2009.
- 4 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of Euclidean  $k$ -means. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG 2015)*, 2015.
- 5 Ke Chen. On coresets for  $k$ -median and  $k$ -means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.
- 6 Kyungjin Cho and Eunjin Oh. Linear-time approximation scheme for  $k$ -means clustering of affine subspaces. *CoRR*, abs/2106.14176, 2021. [arXiv:2106.14176](https://arxiv.org/abs/2106.14176).
- 7 Eduard Eiben, Fedor V Fomin, Petr A Golovach, Willian Lochet, Fahad Panolan, and Kirill Simonov. EPTAS for  $k$ -means clustering of affine subspaces. In *Proceedings of the Thirty-Second ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2649–2659, 2021.
- 8 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43th Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 569–578, 2011.
- 9 Qilong Feng, Zhen Zhang, Ziyun Huang, Jinhui Xu, and Jianxin Wang. Improved algorithms for clustering with outliers. In *Proceedings of the 30th International Symposium on Algorithms and Computation (ISAAC 2019)*, pages 61:1–61:12, 2019.
- 10 Jie Gao, Michael Langberg, and Leonard J Schulman. Analysis of incomplete data and an intrinsic-dimension helly theorem. *Discrete & Computational Geometry*, 40(4):537–560, 2008.
- 11 Jie Gao, Michael Langberg, and Leonard J. Schulman. Clustering lines in high-dimensional space: Classification of incomplete data. *ACM Trans. Algorithms*, 7(1), 2010.
- 12 Sariel Har-Peled and Akash Kushal. Smaller coresets for  $k$ -median and  $k$ -means clustering. *Discrete & Computational Geometry*, 37(1):3–19, January 2007.
- 13 Anil Kumar Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- 14 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *Journal of the ACM*, 57(2):1–32, 2010.
- 15 Euiwoong Lee and Leonard J Schulman. Clustering affine subspaces: hardness and algorithms. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms (SODA 2013)*, pages 810–827, 2013.
- 16 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar  $k$ -means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012.
- 17 Yair Marom and Dan Feldman.  $k$ -means clustering of lines for big data. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
- 18 Nimrod Megiddo. On the complexity of some geometric problems in unbounded dimension. *Journal of Symbolic Computation*, 10(3):327–334, 1990.
- 19 Björn Ommer and Jitendra Malik. Multi-scale object detection by clustering lines. In *Proceedings of the IEEE 12th International Conference on Computer Vision (ICCV 2009)*, pages 484–491, 2009.



# Feedback Vertex Set on Geometric Intersection Graphs

Shinwoo An ✉

POSTECH, Pohang, South Korea

Eunjin Oh ✉

POSTECH, Pohang, South Korea

---

## Abstract

In this paper, we present an algorithm for computing a feedback vertex set of a unit disk graph of size  $k$ , if it exists, which runs in time  $2^{O(\sqrt{k})}(n+m)$ , where  $n$  and  $m$  denote the numbers of vertices and edges, respectively. This improves the  $2^{O(\sqrt{k} \log k)}n^{O(1)}$ -time algorithm for this problem on unit disk graphs by Fomin et al. [ICALP 2017]. Moreover, our algorithm is optimal assuming the exponential-time hypothesis. Also, our algorithm can be extended to handle geometric intersection graphs of similarly sized fat objects without increasing the running time.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** Feedback vertex set, intersection graphs, parameterized algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.47

**Funding** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2020R1C1C1012742).

## 1 Introduction

The FEEDBACK VERTEX SET problem is a classical and fundamental graph problem, which is one of Karp's 21 NP-complete problems. Given an undirected graph  $G = (V, E)$  and an integer  $k$ , the goal is to find a set  $S$  of vertices of size  $k$  such that every cycle of  $G$  contains at least one vertex of  $S$ . In other words, this problem asks to find a set  $S$  of vertices of size  $k$  whose removal from  $G$  results in a forest. This problem has been studied extensively from the viewpoint of exact exponential-time algorithms [13], parameterized algorithms [6, 18], and approximation algorithms [1].

In this paper, we study the FEEDBACK VERTEX SET problem from the viewpoint of parameterized algorithms. When the parameter is the size  $k$  of a feedback vertex set, the best known parameterized algorithm takes  $3.62^k n^{O(1)}$  time [18]. On the other hand, it is known that no algorithm for FEEDBACK VERTEX SET runs in  $2^{o(k)} n^{O(1)}$  time assuming the exponential-time hypothesis (ETH) [8]. For special classes of graphs such as planar graphs and  $H$ -minor-free graphs for any fixed  $H$ , there are  $2^{O(\sqrt{k})} n^{O(1)}$ -time algorithms for the FEEDBACK VERTEX SET problem [10]. Moreover, for planar graphs, FEEDBACK VERTEX SET admits a linear kernel [4].

We present a subexponential-time algorithm for FEEDBACK VERTEX SET on *geometric intersection graphs*, which can be considered as a natural generalization of planar graphs. Consider a set  $F$  of geometric objects (for example, disks and polygons) in the plane. The *intersection graph*  $G[F]$  for  $F$  is defined as the undirected graph whose vertices correspond to the objects in  $F$  such that two vertices are connected by an edge if and only if the two objects corresponding to them intersect. In the case that  $F$  is a set of disks, its intersection graph is called a *disk graph*, which has been studied extensively for various algorithmic problems [5, 7, 17]. It can be used as a model for broadcast networks: The disks of  $F$



© Shinwoo An and Eunjin Oh;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 47; pp. 47:1–47:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

represent transmitter-receiver stations with the same transmission power. A planar graph can be represented as a disk graph, and thus the class of disk graphs is a generalization of the class of planar graphs.

**Previous Work.** Prior to our work, the best known algorithm for FEEDBACK VERTEX SET parameterized by the size  $k$  of a feedback vertex set on unit disk graphs takes  $2^{O(\sqrt{k} \log k)} n^{O(1)}$  time [14, 16]. Since the best known lower bound on the computation time is  $2^{o(\sqrt{n})}$  assuming the exponential-time hypothesis (ETH) [9], it is a natural question if FEEDBACK VERTEX SET on unit disk graphs can be solved optimally. De Berg et al. [9] presented an (non-parameterized) ETH-tight algorithm for this problem, which runs in  $2^{O(\sqrt{n})}$  time. However, it was not known if there is an ETH-tight parameterized algorithm for FEEDBACK VERTEX SET on unit disk graphs.

Recently, several NP-complete problems have been studied for unit disk graphs (and geometric intersection graphs) from the viewpoint of parameterized algorithms, for example, the STEINER TREE, FEEDBACK VERTEX SET, VERTEX COVER,  $k$ -PATH and CYCLE PACKING problems [2, 14, 19]. In the case of VERTEX COVER, the work by de Berg et al. [9] implies an ETH-tight parameterized algorithm. Also, in the case of  $k$ -PATH problem, Fomin et al. [15] presented an ETH-tight parameterized algorithm which runs in  $2^{O(\sqrt{k})} O(n+m)$  time. However, for the other problems, there is a gap between the running time of the best known algorithms and the best known lower bounds.

**Our Result.** In this paper, we present an ETH-tight parameterized algorithm for the FEEDBACK VERTEX SET problem on unit disk graphs, which runs in  $2^{O(\sqrt{k})} (n+m)$  time, where  $n$  and  $m$  denote the numbers of vertices and edges, respectively. This improves the  $2^{O(\sqrt{k} \log k)} n^{O(1)}$ -time algorithm for this problem on unit disk graphs by Fomin et al. [14]. Moreover, unlike the algorithm in [14], our algorithm works on the graph itself and do not require the geometric representation. Also, our algorithm indeed handles geometric intersection graphs of similarly sized fat objects in the plane without increasing the running time, which will be defined in Section 2.1.

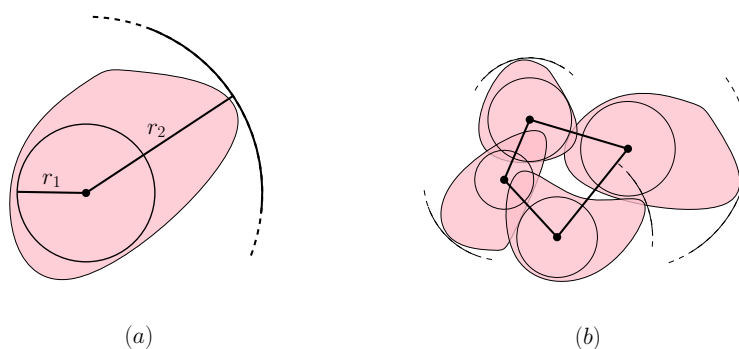
## 2 Preliminaries

For a graph  $G = (V, E)$ , let  $V(G)$  and  $E(G)$  denote the sets of vertices and edges, respectively. For a subset  $S$  of  $V$ , let  $G[S]$  be the subgraph of  $G$  induced by  $S$ . Also, let  $G - S$  be the subgraph of  $G$  induced by  $V - S$ .

### 2.1 Geometric Intersection Graphs

For a constant  $0 < \alpha < 1$ , an object  $o \subseteq \mathbb{R}^2$  is said to be  $\alpha$ -fat if there are two disks  $B_1$  and  $B_2$  with  $B_1 \subseteq o \subseteq B_2$  such that the radius ratio of  $B_1$  to  $B_2$  is at least  $\alpha$ .<sup>1</sup> See Figure 1(a). For example, a disk is a 1-fat object, and a square is a  $1/\sqrt{2}$ -fat object. We call a set  $F$  of  $\alpha$ -fat objects a *similarly sized set* if the ratio of the largest diameter to the smallest diameter of the objects in  $F$  is bounded by a fixed constant  $\gamma$ . For an  $\alpha$ -fat object  $o$  of  $F$ , there are two *concentric* disks  $B_3$  and  $B_4$  with  $B_3 \subseteq o \subseteq B_4$  such that the radius ratio of  $B_3$  and  $B_4$  is at least  $\alpha/2$ . We consider the center of  $B_3$  and  $B_4$  as the *center* of  $o$ . For convenience, we assume that the smallest diameter of the objects of  $F$  is one.

<sup>1</sup> An *object* is a point set in the plane, which is not necessarily connected.



■ **Figure 1** (a) A  $\frac{r_1}{r_2}$ -fat object. (b) The drawing of the geometric intersection graph  $G[F]$ .

The *intersection graph*  $G[F]$  of a set  $F$  of objects in  $\mathbb{R}^2$  is defined as the graph whose vertex set is  $F$  and two vertices are connected by an edge if and only if their corresponding objects intersect. The *drawing* of the intersection graph  $G[F]$  is a representation of  $G[F]$  in the plane such that the vertices lie on the centers of the objects of  $F$  and the edges are drawn as line segments. See Figure 1(b). We sometimes use an intersection graph  $G[F]$  and its drawing interchangeably if it is clear from the context.

In this paper, we focus on geometric intersection graphs of objects in the *plane* only. Because FEEDBACK VERTEX SET on unit ball graphs in  $\mathbb{R}^3$  has no subexponential-time algorithm parameterized by  $k$  unless ETH fails [16].

## 2.2 Tree Decomposition and Weighted Width

A *tree decomposition* of a graph  $G = (V, E)$  is defined as a pair  $(T, \beta)$ , where  $T$  is a tree and  $\beta$  is a mapping from nodes of  $T$  to subsets of  $V$  (called bags) with the following properties. Let  $\mathcal{B} := \{\beta(u) : u \in V(T)\}$  be the set of bags of  $T$ .

1. For any vertex  $u \in V$ , there is at least one bag in  $\mathcal{B}$  which contains  $u$ ;
2. For any edge  $(u, v) \in E$ , there is at least one bag in  $\mathcal{B}$  which contains both  $u$  and  $v$ .
3. For any vertex  $u \in V$ , the subset of bags of  $\mathcal{B}$  containing  $u$  forms a subtree of  $T$ .

The *width* of a tree decomposition is defined as the size of its largest bag minus one, and the *treewidth* of  $G$  is the minimum width among the tree decompositions of  $G$ . In this paper, as in [9], we use the notion of *weighted treewidth* introduced by [20]. Here, assume that each vertex  $v$  has its weight. The weight of each bag is defined as the sum of the weights of the vertices in the bag, and the *weighted width* of tree decomposition is defined as the maximum weight of the bags. The *weighted treewidth* of a graph  $G$  is the minimum weighted width among the tree decompositions of  $G$ .

## 2.3 $\kappa$ -Partition and $\mathcal{P}$ -Contraction

We use the concept of  $\kappa$ -partitions and  $\mathcal{P}$ -contractions introduced by de Berg et al. [9]. Let  $G[F] = (V, E)$  be a geometric intersection graph of a set  $F$  of similarly sized  $\alpha$ -fat objects. A  $\kappa$ -partition of  $G$  is a partition  $\mathcal{P} = (P_1, \dots, P_t)$  of  $V$  such that  $G[P_i]$  is connected and is the union of at most  $\kappa$  cliques for every partition class  $P_i$ . Given a  $\kappa$ -partition  $\mathcal{P}$ , we consider a graph obtained by contracting the vertices in the same partition class to a single vertex and removing all loops and parallel edges. We call the resulting graph the  $\mathcal{P}$ -contraction of  $G$  and denote it by  $G_{\mathcal{P}}$ . The weight of each vertex of  $G_{\mathcal{P}}$  is defined as  $\lceil \log |P_i| \rceil + 1$ , where  $P_i$  denotes the partition class of  $\mathcal{P}$  defining the vertex.

A *greedy partition* is a notion from De berg et al. [9]. For the given geometric intersection graph  $G[F]$ , they iteratively pick one geometric object, then construct a partition class which consist of the objects adjacent to the picked object. They denote the final partition by the greedy partition. Since all objects are  $\alpha$ -fat, the greedy partition is  $\kappa$ -partition of  $G[F]$  and moreover has maximum degree  $\delta$  for constants  $\kappa$  and  $\delta$ . Given a greedy partition  $\mathcal{P}$ , they showed that the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{n})$ . Moreover, they presented a  $2^{O(\sqrt{n})}$ -time algorithm for computing a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{n})$ .

## 2.4 Overview of Our Algorithm

Our algorithm consists of two steps: computing a (weighted) tree decomposition of  $G_{\mathcal{P}}$ , and then using a dynamic programming on the tree decomposition. As in [9], we first compute a greedy partition. Then we show that the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{k})$  if  $(G, k)$  is a **yes**-instance. To do this, we first show that the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{k})$  if  $G$  has  $O(\sqrt{k})$  vertices of degree at least three in Section 3. Then we show that  $G$  has  $O(\sqrt{k})$  vertices of degree at least three in Section 4.1.

Using this fact, we compute a constant approximation to the weighted treewidth of  $G_{\mathcal{P}}$ . If it is  $\omega(\sqrt{k})$ , we conclude that  $(G, k)$  is a **no**-instance immediately. Otherwise, we compute a feedback vertex set of size  $k$ , if it exists, using dynamic programming on a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{k})$ . The dynamic programming algorithm is described in Section 4.2.

## 3 Tree Decomposition and Weighted Treewidth

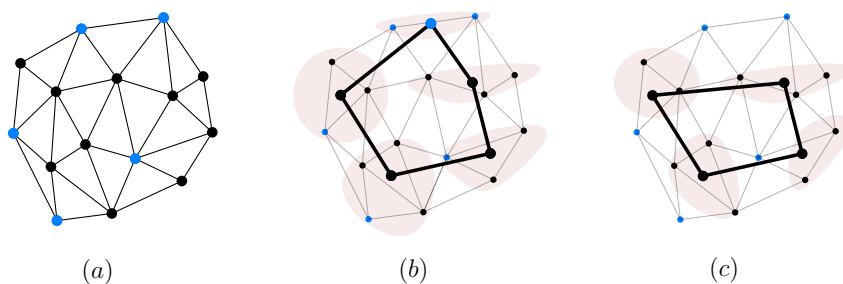
In this section, we present the first step of our algorithm: computing a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{|U|})$  for a greedy partition  $\mathcal{P}$  of a geometric intersection graph  $G$ , where  $U$  denotes the set of vertices of  $G$  of degree at least three in  $G$ . More precisely, we prove the following theorem.

► **Theorem 1.** *Let  $G$  be an intersection graph of similarly sized  $\alpha$ -fat objects with  $n$  vertices and  $m$  edges, and let  $\mathcal{P}$  be a greedy partition of  $G$ . Then the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{|U|})$ , where  $U$  denotes the set of vertices of  $G$  of degree at least three in  $G$ . Moreover, we can compute a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{|U|})$  in  $2^{O(\sqrt{|U|})}(n + m)$  time without using a geometric representation of  $G$ .*

In the following, let  $G$  be an intersection graph of similarly sized  $\alpha$ -fat objects,  $\mathcal{P}$  be a greedy partition of  $G$ , and  $U$  be the set of vertices of  $G$  of degree at least three in  $G$ . In Section 3.1, we prove the first part of the theorem, and in Section 3.2, we prove the second part of the theorem.

### 3.1 Weighted Treewidth of the $\mathcal{P}$ -contraction

We first show that the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{|U|})$ . To do this, we transform  $G$  into a planar graph  $H$ . As we did for  $G$ , we compute the  $\mathcal{P}$ -contraction of  $H$ , and remove degree-2 vertices of the  $\mathcal{P}$ -contractions of  $H$  and  $G$  in a specific way. In this way, we obtain  $\bar{H}_{\mathcal{P}}$  and  $\bar{G}_{\mathcal{P}}$  of complexity  $O(\sqrt{|U|})$ . Also, we assign the weight to each vertex of the resulting graphs. Then we transform a balanced separator of  $\bar{H}_{\mathcal{P}}$  of weight  $O(\sqrt{|U|})$  into a balanced separator of  $\bar{G}_{\mathcal{P}}$  of weight  $O(\sqrt{|U|})$ . Using a weighted balanced separator, we compute a tree decomposition of  $\bar{G}_{\mathcal{P}}$  of weighted width  $O(\sqrt{|U|})$ , and then transform it into a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{|U|})$ .



■ **Figure 2** (a) The Delaunay triangulation  $H$  of the vertex set of  $V(G)$ . The vertices of  $G$  of degree at most two are colored blue. (b)  $\mathcal{P}$  partitions  $V$  into five subsets. Each subset of  $\mathcal{P}$  is contained in a single pink region. Then the  $\mathcal{P}$ -contraction of  $H$  is the cycle consisting of five edges. (c) Each subset of  $\mathcal{P}$  consists of the black points contained in a single pink region.

**Delaunay triangulation and its contraction:  $H$  and  $H_{\mathcal{P}}$ .** Consider the Delaunay triangulation  $H$  of the point set  $V(G)$ . Here, we consider  $V(G)$  as the set of the centers of the objects defining  $G$ . We consider the Delaunay triangulation  $H$  as an edge-weighted plane graph such that the *length* of each edge is the Euclidean distance between their endpoints. The Delaunay triangulation  $H$  is a *5.08-spanner* of the complete Euclidean graph defined by  $V(G)$  [12]. That is, for any two points  $u$  and  $v$  of  $V(G)$ , the length of the shortest path in  $H$  between  $u$  and  $v$  is at most 5.08 times their Euclidean distance.

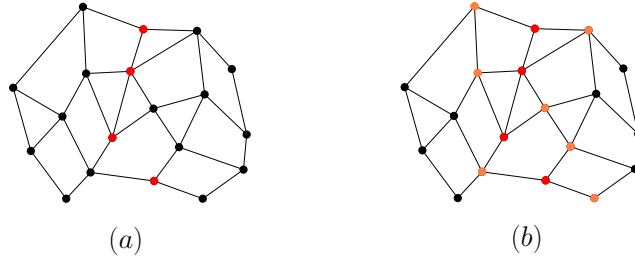
Notice that  $H$  might contain an edge which is not an edge of  $G$ . However, the following still holds as  $G$  is a subgraph of the complete Euclidean graph. For any edge  $(u, v)$  in  $G$ , there is a  $u$ - $v$  path in  $H$  consisting of  $5.08\|uv\|$  edges. Recall that  $\alpha$  is the measure for the fatness of the objects defining  $G$ , which is a constant. Let  $H_{\mathcal{P}}$  be the  $\mathcal{P}$ -contraction of  $H$ . Note that  $V(G_{\mathcal{P}}) = V(H_{\mathcal{P}})$ . For a vertex  $v$  of  $G_{\mathcal{P}}$  or  $H_{\mathcal{P}}$ , we let  $P(v)$  to denote the partition class of  $\mathcal{P}$  corresponding to  $v$ . See Figure 2(b).

**Partition  $\bar{\mathcal{P}}$  of  $U$ :  $\bar{G}_{\mathcal{P}}$  and  $\bar{H}_{\mathcal{P}}$ .** The size of  $G_{\mathcal{P}}$  and  $H_{\mathcal{P}}$  might be  $\Theta(n)$  even if  $|U|$  is small. Recall that  $U$  is the set of vertices of  $G$  of degree at least three in  $G$ . To obtain a balanced separator of  $G_{\mathcal{P}}$  of small weight, we compute a new graph  $\bar{G}_{\mathcal{P}}$  and  $\bar{H}_{\mathcal{P}}$  of size  $O(|U|)$  as follows.

Let  $\mathcal{P} = (P_1, \dots, P_t)$ , which is a partition of  $V(G)$ . Using this, we consider the partition  $\bar{\mathcal{P}} = (\bar{P}_1, \dots, \bar{P}_t)$  of  $U$  such that  $\bar{P}_i = P_i \cap U$ . Note that  $\bar{P}_i = \emptyset$  if every vertex of  $P_i$  has degree at most two in  $G$ . We first ignore the empty partition classes from  $\bar{\mathcal{P}}$ , and let  $\bar{G}_{\mathcal{P}}$  and  $\bar{H}_{\mathcal{P}}$  be the  $\bar{\mathcal{P}}$ -contraction of  $G$  and the  $\bar{\mathcal{P}}$ -contraction of  $H$ , respectively. We add several edges to  $\bar{G}_{\mathcal{P}}$  and  $\bar{H}_{\mathcal{P}}$  by considering the empty partition classes of  $\bar{\mathcal{P}}$  as follows. Since  $G[P_i]$  is connected and consists of  $\kappa$  cliques,  $G[P_i]$  is a simple cycle or a simple path if  $\bar{P}_i = \emptyset$ . Let  $P_{\emptyset}$  be the union of  $P_i$ 's for all indices with  $\bar{P}_i = \emptyset$ . Then each connected component of  $G[P_{\emptyset}]$  is also a simple cycle or a simple path. If the connected component is a simple cycle, no vertex in the component is connected by a vertex of  $U$  in  $G$ . Otherwise, the endpoints of the simple path, say  $p$  and  $q$ , are contained in  $U$ . In this case, we connect the vertices of  $\bar{G}_{\mathcal{P}}$  (and  $\bar{H}_{\mathcal{P}}$ ) by an edge unless this edge forms a loop. See Figure 2(c).

For a vertex  $\bar{v}$  of  $\bar{G}_{\mathcal{P}}$  or  $\bar{H}_{\mathcal{P}}$ , we let  $\bar{P}(\bar{v})$  be the partition class of  $\bar{\mathcal{P}}$  corresponding to  $\bar{v}$ . By construction,  $\bar{P}(\bar{v})$  is not empty for any vertex  $\bar{v}$ . Each vertex  $\bar{v}$  of  $\bar{G}_{\mathcal{P}}$  has a *g-weight*  $g(\bar{v}) = \lceil \log |\bar{P}(\bar{v})| \rceil + 1$ . Also, each vertex  $\bar{v}$  of  $\bar{H}_{\mathcal{P}}$  has a *h-weight*  $h(\bar{v}) = \sum_{\bar{u} \in N(\bar{v})} (\lceil \log |\bar{P}(\bar{u})| \rceil + 1)$ , where  $N(\bar{v})$  is the set of vertices  $\bar{u}$  of  $\bar{H}_{\mathcal{P}}$  such that there is a  $\bar{u}$ - $\bar{v}$  path in  $\bar{H}_{\mathcal{P}}$  consisting of at most  $t = (\delta + 1)(10.16(\alpha\gamma)^{-1})^2\pi$  vertices.





■ **Figure 3** (a) A balanced separator  $S$  of  $\bar{H}_{\mathcal{P}}$  consists of the red vertices. (b) A balanced separator  $S'$  of  $\bar{G}_{\mathcal{P}}$  consists of the red vertices and the orange vertices. The vertices whose distance from the red vertices are at most  $t = 1$  are colored with orange.

► **Observation 2.** A vertex  $\bar{v}$  of  $\bar{H}_{\mathcal{P}}$  has a constant degree, and thus the size of  $N(\bar{v})$  is  $O(1)$ .

**Balanced separator of  $\bar{H}_{\mathcal{P}}$  of small  $h$ -weights.** For a vertex-weighted graph  $G'$  and a subgraph  $H'$ , we denote the sum of the weights of the vertices of  $H'$  by the *weight* of  $H'$ . a subset  $S$  of  $V(G')$  is called a *balanced separator* of  $G'$  if the weight of each connected component of  $G' - S$  is at most  $2/3$  of the weight of  $G'$ . The *weight* of a balanced separator  $S$  of  $G'$  is the sum of the weights of the vertices of  $S$ . Djidjev [11] showed that a planar graph  $G'$  has a balanced separator of weight  $O(\sqrt{\sum_{v \in V} w(v)^2})$ , where  $w(v)$  is the weight of a vertex  $v$ .

The following lemma implies that  $\bar{H}_{\mathcal{P}}$  has a balanced separator of  $h$ -weight  $O(\sqrt{|U|})$ .

► **Lemma 3.** The sum of  $h(\bar{v})^2$  for all vertices  $\bar{v}$  of  $\bar{H}_{\mathcal{P}}$  is  $O(|U|)$ .

**Proof.** Let  $\bar{v}$  be a vertex of  $\bar{H}_{\mathcal{P}}$ . The  $h$ -weight of  $\bar{v}$  is the sum of  $g(\bar{u}) = (\lceil \log |\bar{P}(\bar{u})| \rceil + 1)$  for all vertices  $\bar{u}$  of  $N(\bar{v})$  by definition. By Cauchy-Schwarz inequality,  $(\sum_{\bar{u} \in N(\bar{v})} g(\bar{u}))^2$  is at most  $|N(\bar{v})|(\sum_{\bar{u} \in N(\bar{v})} g(\bar{u})^2)$ . Since the size of  $N(\bar{v})$  is at most a constant, say  $c$ , we have the following inequality.

$$\sum_{\bar{v} \in V} h(\bar{v})^2 \leq c \cdot \sum_{\bar{v} \in V} \sum_{\bar{u} \in N(\bar{v})} g(\bar{u})^2 = c \cdot \sum_{\bar{u} \in V} \sum_{\bar{v} \in N(\bar{u})} g(\bar{u})^2,$$

where  $V$  denotes the vertex set of  $\bar{H}_{\mathcal{P}}$ . Since  $\sum_{\bar{v} \in N(\bar{u})} g(\bar{u})^2 = |N(\bar{u})|g(\bar{u})^2 \leq c \cdot g(\bar{u})^2$ , we finally have the following.

$$\sum_{\bar{v} \in V} h(\bar{v})^2 \leq c^2 \cdot \sum_{\bar{u} \in V} g(\bar{u})^2 = c^2 \cdot \sum_{\bar{u} \in V} (\lceil \log |\bar{P}(\bar{u})| \rceil + 1)^2 = O\left(\sum_{\bar{u} \in V} |\bar{P}(\bar{u})|\right) = O(|U|).$$

Here, the second equality holds because  $2x \geq (\log x + 1)^2$  for all values  $x \geq 1$ . Also, the last equality holds because  $\bar{P}$  is a partition of  $U$ . ◀

**Balanced separator of  $\bar{G}_{\mathcal{P}}$  of small  $g$ -weight.** Let  $S$  be a balanced separator of  $\bar{H}_{\mathcal{P}}$  of  $h$ -weight  $O(\sqrt{|U|})$ . Let  $S'$  be the union of  $N(\bar{v})$  for all  $\bar{v} \in S$ . Recall that  $|V(\bar{G}_{\mathcal{P}})| = |V(\bar{H}_{\mathcal{P}})|$ . The sum of the  $g$ -weights of the vertices of  $S'$  is  $O(\sqrt{|U|})$  by definition of the  $g$ - and  $h$ -weights. Therefore, it suffices to show that the weight of each connected component of  $\bar{G}_{\mathcal{P}} - S'$  is at most  $2/3$  of the weight of  $\bar{G}_{\mathcal{P}}$ .

► **Lemma 4.** *For any edge  $(u, v)$  of  $G_{\mathcal{P}}$ , there is a  $u$ - $v$  path in  $H_{\mathcal{P}}$  consisting of  $t$  vertices.<sup>2</sup>*

**Proof.** Consider an edge  $(u, v)$  of  $G_{\mathcal{P}}$ . By construction, there is an edge  $(p, q)$  in  $G$  such that  $p \in P(u)$  and  $q \in P(v)$ . Recall that Delaunay triangulation is a 5.08-spanner of the complete Euclidean graph. Therefore, there exists a  $p$ - $q$  path  $\tau$  in  $H$  such that the sum of the lengths (Euclidean distance between two endpoints) of the edges is at most  $5.08\|pq\|$ .

We claim that  $\tau$  intersects at most  $(\delta + 1)(10.16(\alpha\gamma)^{-1})^2\pi$  partition classes of  $\mathcal{P}$ , where  $\delta$  is the maximum degree of  $G_{\mathcal{P}}$ ,  $\alpha$  is the measure for the fatness of the objects, and  $\gamma$  is the ratio of the largest and smallest diameters of the objects. Consider a *grid* of the plane, which is a partition of the plane into axis-parallel squares (called *cells*) with diameter  $1/\sqrt{2}$ . Recall that the smallest diameter of the objects defining  $G$  is one. For a cell  $\sigma$ , we denote the set of vertices of  $G$  contained in  $\sigma$  by  $P_{\sigma}$ . By definition,  $P_{\sigma}$  forms a clique in  $G$ . Since the maximum degree  $\delta$  of  $G_{\mathcal{P}}$  is constant, at most  $\delta + 1$  partition classes of  $\mathcal{P}$  has their vertices in  $P_{\sigma}$ . Note that every object is contained in a ball of radius  $(\alpha\gamma)^{-1}$ . Therefore,  $\|pq\| \leq 2(\alpha\gamma)^{-1}$ . Then, the Euclidean length of  $\tau$  is at most  $10.16(\alpha\gamma)^{-1}$ . This implies that  $\tau$  is contained in a ball of radius  $10.16(\alpha\gamma)^{-1}$  centered at  $p$ . In other words,  $\tau$  intersects at most  $(10.16(\alpha\gamma)^{-1})^2\pi$  cells. Since each cell intersects at most  $\delta + 1$  partition classes of  $\mathcal{P}$ ,  $\tau$  intersects  $(\delta + 1)(10.16(\alpha\gamma)^{-1})^2\pi$  partition classes. Let  $t = (\delta + 1)(10.16(\alpha\gamma)^{-1})^2\pi$ .

Then we consider a  $u$ - $v$  path in  $H_{\mathcal{P}}$  obtained by replacing each point in  $\tau$  to its contracted point in  $H_{\mathcal{P}}$ . This path now consists of  $t$  different vertices of  $H_{\mathcal{P}}$ , but it is not necessarily simple. We can obtain a simple path by removing duplicated subpaths that have same endpoints. Then we obtain a simple path of consisting of  $t$  vertices. This completes the proof. ◀

► **Lemma 5.** *For any edge  $(\bar{u}, \bar{v})$  of  $\bar{G}_{\mathcal{P}}$ , there is a  $\bar{u}$ - $\bar{v}$  path in  $\bar{H}_{\mathcal{P}}$  consisting of  $t$  vertices.*

**Proof.** We consider an edge  $(\bar{u}, \bar{v})$  of  $\bar{G}_{\mathcal{P}}$ . Let  $u$  and  $v$  be the vertices of  $G_{\mathcal{P}}$  with  $P(u) \cap U = \bar{P}(\bar{u})$  and  $P(v) \cap U = \bar{P}(\bar{v})$ . By construction, either there is an edge  $(u, v)$  in  $G_{\mathcal{P}}$ , or there is a  $u$ - $v$  path in  $G_{\mathcal{P}}$  such that no internal vertex is contained in  $U$ . We consider these two cases separately.

- **Case 1.  $(u, v)$  is an edge of  $G_{\mathcal{P}}$ .** In this case, there is a  $u$ - $v$  path  $\tau_1$  in  $H_{\mathcal{P}}$  consisting of  $t$  vertices by Lemma 4. Let  $I$  be the sequence of the indices of the partition classes of  $\mathcal{P}$  corresponding to the vertices of  $\tau_1$ . Then let  $I'$  be the subsequence of  $I$  consisting of the indices  $j$  with  $\bar{P}_j \neq \emptyset$ . If  $I = I'$ , then  $\bar{H}_{\mathcal{P}}$  has a  $\bar{u}$ - $\bar{v}$  path consisting of  $s$  vertices. Otherwise, since the internal vertices of  $\tau_1$  has degree at least two in  $H_{\mathcal{P}}$ , the vertices of  $\bar{H}_{\mathcal{P}}$  corresponding to  $\bar{P}(i)$  and  $\bar{P}(j)$  are connected by an edge for every consecutive indices  $i$  and  $j$  of  $I'$ . Therefore, there exists a  $\bar{u}$ - $\bar{v}$  path in  $\bar{H}_{\mathcal{P}}$  consisting of  $t$  vertices.
- **Case 2. There is a  $u$ - $v$  path  $\tau'$  in  $G_{\mathcal{P}}$  with no internal vertices in  $U$ .** Recall that  $V(G_{\mathcal{P}}) = V(H_{\mathcal{P}})$ . If  $\tau'$  entirely exists in  $H_{\mathcal{P}}$ , there is an edge  $(\bar{u}, \bar{v})$  in  $\bar{H}_{\mathcal{P}}$ , and thus we are done. If it is not the case, let  $(u', v')$  be an edge in  $\tau'$  which does not exist in  $H_{\mathcal{P}}$ . By Lemma 4, there is a  $u'$ - $v'$  path in  $H_{\mathcal{P}}$  consisting of  $t$  vertices. Since the internal vertices of  $\tau'$  have degree exactly two, the  $u'$ - $v'$  path contains both  $u$  and  $v$ . Therefore, the subgraph of the  $u'$ - $v'$  path lying between  $u$  and  $v$  is a  $u$ - $v$  path in  $H_{\mathcal{P}}$  consisting of at most  $t$  vertices. Then similarly to Case 1, we can show that there is a  $\bar{u}$ - $\bar{v}$  path in  $\bar{H}_{\mathcal{P}}$  consisting of  $t$  vertices.

Therefore, for an edge  $(\bar{u}, \bar{v})$  in  $\bar{G}_{\mathcal{P}}$ , there is a  $\bar{u}$ - $\bar{v}$  path in  $\bar{H}_{\mathcal{P}}$  consisting of at most  $t$  vertices. This completes the proof. ◀

<sup>2</sup> Recall that  $t = (\delta + 1)(10.16(\alpha\gamma)^{-1})^2\pi$ .

► **Lemma 6.** *The weight of each connected component of  $\tilde{G}_{\mathcal{P}} - S'$  is at most  $2/3$  of the weight of  $\tilde{G}_{\mathcal{P}}$ .*

**Proof.** We show that each connected component of  $\tilde{G}_{\mathcal{P}} - S'$  is a subset of a connected component of  $\tilde{H}_{\mathcal{P}} - S$ . This implies that each connected component of  $\tilde{G}_{\mathcal{P}} - S'$  consists of at most  $2|V(\tilde{G}_{\mathcal{P}})|/3$  vertices since  $S$  is a balanced separator of  $G_{\mathcal{P}}$ , and  $V(\tilde{G}_{\mathcal{P}}) = V(\tilde{H}_{\mathcal{P}})$ .

To show this, consider an edge  $(a, b)$  of  $\tilde{G}_{\mathcal{P}} - S'$ . We claim that  $a$  and  $b$  are contained in the same connected component of  $\tilde{H}_{\mathcal{P}} - S$ . By Lemma 5, there is an  $a$ - $b$  path in  $\tilde{H}_{\mathcal{P}}$  consisting of  $t$  vertices. If this path contains a vertex of  $S$ , then  $a$  and  $b$  are contained in  $S'$  by construction, which makes a contradiction. Therefore, an  $a$ - $b$  path in  $\tilde{H}_{\mathcal{P}}$  does not intersect  $S$ , which means that  $a$  and  $b$  are contained in the same connected component of  $\tilde{H}_{\mathcal{P}} - S$ .

Therefore, each connected component of  $\tilde{G}_{\mathcal{P}} - S'$  is a subset of the connected component of  $\tilde{H}_{\mathcal{P}} - S$ , and the lemma holds. ◀

**Tree decomposition of  $\tilde{G}_{\mathcal{P}}$ .** To construct a tree decomposition of  $\tilde{G}_{\mathcal{P}}$ , we observe that any subgraph of  $\tilde{G}_{\mathcal{P}}$  also has a balanced separator of small weight. Let  $A$  be a connected component of  $\tilde{G}_{\mathcal{P}} - S'$ . Then consider the subgraph  $\tilde{H}_{\mathcal{P}}[A]$  of  $\tilde{H}_{\mathcal{P}}$  induced by the vertices in  $A$ . Since  $\tilde{H}_{\mathcal{P}}[A]$  is planar, it has a balanced separator with small  $h$ -weight, and thus  $A$  has a balanced separator with small  $g$ -weight. Therefore, we can recursively compute a balanced separator for every connected component of  $\tilde{G}_{\mathcal{P}} - S'$ .

We compute a tree decomposition of each connected component of  $\tilde{G}_{\mathcal{P}} - S'$  recursively, and then we connect those trees by one additional empty bag, and add the separator to all bags of the resulting tree. Since the weight of the connected components decreases geometrically, the maximum weight of the bags is  $O(\sqrt{|U|})$ . Therefore, we can compute a tree decomposition  $(T, \beta)$  of  $\tilde{G}_{\mathcal{P}}$  of weighted width  $O(\sqrt{|U|})$ .

**Tree decomposition of  $G_{\mathcal{P}}$ .** The remaining step is computing a tree decomposition of  $G_{\mathcal{P}}$  from  $(T, \beta)$ . Then we start with the pair  $(T, \beta)$ , and then add several nodes (and bags) to  $T$  and add several vertices to the bags of  $\beta$  as follows.

Let  $Q$  be the set of vertices  $u$  of  $G_{\mathcal{P}}$  with  $P(u) \cap U \neq \emptyset$ , and let  $Q^c$  be the set of vertices of  $G_{\mathcal{P}}$  not in  $Q$ . For any vertex  $v$  of  $Q$ , there is a vertex  $\bar{v}$  in  $\tilde{G}_{\mathcal{P}}$  with  $P(v) \cap U = \bar{P}(\bar{v})$ , and thus  $\bar{v}$  is contained in bags of  $(T, \beta)$ . We replace  $\bar{v}$  with  $v$  in the bags of  $(T, \beta)$  containing  $v$ .

On the other hand, for a vertex  $v$  of  $Q^c$ ,  $P(v) \cap U = \emptyset$ , and thus no vertex of  $\tilde{G}_{\mathcal{P}}$  corresponds to  $v$ . Thus we are required to insert  $v \in Q$  into  $T'$ . Recall that every vertex in  $Q$  has degree one or two in  $G$ , and thus it also has degree one or two in  $G_{\mathcal{P}}$ . Also,  $G_{\mathcal{P}}$  is connected.<sup>3</sup> Therefore,  $G_{\mathcal{P}}[Q]$  consists of a number of simple paths and isolated vertices, all of which are connected to at most two vertices of  $Q$ . Thus, there are the following four cases.

- **Case 1.** An isolated vertex  $v$  is connected to only one vertex  $u \in Q$ . For a bag  $\beta(x)$  of  $(T, \beta)$  containing  $u$ , we add a leaf with bag  $\{u, v\}$  as a child of  $x$ .
- **Case 2.** An isolated vertex  $v$  is connected to exactly two vertices  $u, w \in Q$ . By construction, there is an edge  $(u, w)$  in  $\tilde{G}_{\mathcal{P}}$ . Then, there is a bag  $\beta(x)$  of  $T$  containing both  $\bar{u}$  and  $\bar{w}$ . We add a leaf with bag  $\{u, v, w\}$  as a child of  $x$ .
- **Case 3.** A simple  $v$ - $v'$  path  $(v_1, \dots, v_k)$  is connected to exactly one vertex  $u \in Q$ . Note that either  $v$  or  $v'$  is connected to  $u$ , but not both. Without loss of generality, we assume

<sup>3</sup> We assume that the input intersection graph  $G$  is connected, and thus  $G_{\mathcal{P}}$  is connected by construction.

that there is an edge  $(u, v)$  in  $G_{\mathcal{P}}$ . For a bag  $\beta(x)$  of  $T$  containing  $u$ , we add a node  $x_1$  with bag  $\{u, v\}$  as a child of  $x$ . For  $1 < t < k$ , we add a node  $x_t$  with bag  $\{v_t, v_{t+1}\}$  as a child of  $x_{t-1}$ .

- **Case 4.** A simple  $v$ - $v'$  path  $(v_1, \dots, v_k)$  is connected to two vertices  $u, w \in Q$ . In this case,  $v$  is connected to one of  $u$  and  $w$ , say  $u$ , and  $v'$  is connected to the other vertex, say  $w$ . Since  $\tilde{G}_{\mathcal{P}}$  has an edge  $(\bar{u}, \bar{w})$ , there is a bag  $\beta(x)$  of  $T$  that contains both  $u$  and  $w$ , where  $\bar{u}$  and  $\bar{w}$  are the vertices of  $\tilde{G}_{\mathcal{P}}$  corresponding to  $u$  and  $w$ , respectively. We add a node  $x_1$  with bag  $(u, w, v, v')$  as a child of  $x$ . For  $1 < t < k/2$ , we add a node  $x_t$  with bag  $\{v_t, v_{t+1}, v_{k-t}, v_{k-t+1}\}$  as a child of  $x_{t-1}$ .

In this way,  $(T, \beta)$  is a tree decomposition of  $G_{\mathcal{P}}$ . Now we analyze the weight of a bag of  $(T, \beta)$ . Since we replace  $\bar{v}$  with  $v$  for a vertex  $v$  of  $Q$  in the bags of  $(T, \beta)$ , the weight of a bag  $B$  of  $(T, \beta)$  consisting of vertices of  $Q$  increases. Let  $v$  be a vertex of  $Q$ , and let  $\bar{v}_i$  be the vertex of  $\tilde{G}_{\mathcal{P}}$  corresponding to  $v$ . Since  $P(v)$  is composed of at most  $O(1)$  cliques, it contains at most  $O(1)$  vertices of  $V \setminus U$ . Note that  $B$  contains  $O(\sqrt{|U|})$  vertices because the weighted width of the tree decomposition of  $\tilde{G}_{\mathcal{P}}$  is  $O(\sqrt{|U|})$ . Therefore, the weight of  $B$  increases by  $O(\sqrt{|U|})$  after replacing vertices of  $\tilde{G}_{\mathcal{P}}$  with vertices of  $G_{\mathcal{P}}$ . Thus, the weight of a bag consisting of vertices of  $Q$  is  $O(\sqrt{|U|})$ .

Now we show that the weight of a bag containing a vertex of  $Q^c$  is  $O(\sqrt{|U|})$ . Every vertex  $v \in Q^c$  is a contraction of simple paths or simple cycles of length at most  $O(1)$  in  $G$ . In particular, its weight in  $G_{\mathcal{P}}$  is  $O(1)$ . Since a bag containing a vertex of  $Q^c$  consists of at most two vertices of  $Q^c$  and at most two vertices of  $Q$ , the weight of the bag is  $O(\sqrt{|U|})$ .

### 3.2 Computing a Tree Decomposition of Small Weighted Width

We can compute a tree decomposition of weighted treewidth  $O(\sqrt{|U|})$  of  $G_{\mathcal{P}}$  using the algorithm proposed by de Berg et al. [9], which computes a tree decomposition of weighted width  $O(w)$  of  $G_{\mathcal{P}}$  assuming that the weighted treewidth of  $G_{\mathcal{P}}$  is  $w$ . This algorithm runs in  $2^{O(w)}(n+m)$  time. This algorithm works on the graph itself even if the geometric representation is unknown. In this section, we briefly describe how the algorithm proposed by de Berg et al. [9] for computing such a tree decomposition works.

A *blowup* of a vertex  $v$  by an integer  $t$  results in a graph where we replace a single vertex  $v$  into a clique of size  $t$ , in which we connect every vertex to the neighbor vertices of  $v$ . We denote the set of vertices in the clique by  $\mathcal{B}(v)$ . For the graph  $G_{\mathcal{P}}$ , we construct an unweighted graph  $G_{\mathcal{B}}$  by blowing up each vertex  $v$  by an integer  $\lceil g(v) \rceil$  where  $g(v) = \log |P(v)| + 1$ . We compute a tree decomposition of  $G_{\mathcal{B}}$ , denoted by  $(T_{\mathcal{B}}, \sigma_{\mathcal{B}})$ . Then we compute a tree decomposition of  $G_{\mathcal{P}}$  by using the same tree layout of  $T_{\mathcal{B}}$ . In particular, we add a vertex  $v \in G_{\mathcal{P}}$  to a bag if and only if the bag in  $T_{\mathcal{B}}$  contains all vertices of  $\mathcal{B}(v)$ . The Lemma 2.9 and Lemma 2.10 of [9] prove the correctness of this algorithm.

The remaining step is computing a tree decomposition of  $G_{\mathcal{B}}$ . We apply a constant-factor approximation algorithm for computing an optimal tree decomposition [8]. By Theorem 1 and Lemma 2.9 of [9], the treewidth of  $G_{\mathcal{B}}$  is  $O(\sqrt{|U|})$ . Therefore, the algorithm returns a tree decomposition of  $G_{\mathcal{B}}$  of width  $O(\sqrt{|U|})$ . This algorithm takes a  $2^{O(tw(G_{\mathcal{B}}))}n = 2^{O(\sqrt{|U|})}(n+m)$  time, and therefore, we can compute a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{|U|})$  without using the geometric representation of the graph. This completes the proof of Theorem 1.

## 4 Computing a Feedback Vertex Set

In this section, we present a  $2^{O(\sqrt{k})}(n+m)$ -time algorithm for finding a feedback vertex set of size  $k$  for the intersection graph  $G[F] = (V, E)$  of similarly sized fat objects in the plane.

To do this, we first show that  $G$  has  $O(k)$  vertices of degree at least three if it has a feedback vertex set of size at most  $k$  in Section 4.1. Therefore, by Theorem 1, for a greedy partition  $\mathcal{P}$  of  $G$ , the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{k})$  if  $(G, k)$  is a **yes**-instance.

To use this fact, we first compute such a greedy partition  $\mathcal{P}$  in  $O(n+m)$  time using the algorithm in [9]. Then we check if the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{k})$  in  $2^{O(\sqrt{k})}(n+m)$  time. If it is  $\omega(\sqrt{k})$ , we conclude that  $(G, k)$  is a **no**-instance immediately. Otherwise, we compute a feedback vertex set of size  $k$ , if it exists, using dynamic programming on a tree decomposition of  $G_{\mathcal{P}}$  of weighted width  $O(\sqrt{k})$ . De Berg et al. [9] presented a dynamic programming algorithm for FEEDBACK VERTEX SET which runs in  $2^{O(\sqrt{w})}(n+m)$  time, where  $w$  is the weighted treewidth of  $G_{\mathcal{P}}$ . We can use this algorithm, but for completeness, we briefly describe this algorithm in Section 4.2.

### 4.1 Weighted Treewidth for a Yes-Instance

We first remove all degree-1 vertices from  $G$  and denote it by  $G'$ . This reduction is feasible because no cycle encounters a degree-1 vertex. We next show that  $G'$  has  $O(k)$  vertices of degree at least three if it has a feedback vertex set of size  $k$ . We consider a partition of the plane into axis-parallel squares (called *cells*) with diameter  $1/\sqrt{2}$ , which we call the *grid*. Recall that we let the smallest diameter of the objects defining  $G$  be one. For a cell  $\sigma$ , let  $P(\sigma)$  be the set of the centers of the objects defining  $G$ . By definition,  $P(\sigma)$  forms a clique in  $G$ . We say a grid cell is a *neighbor* of another grid cell if the smallest distance between two points from the two cells is at most  $2\alpha\gamma$ , where  $\alpha$  is the measure for the fatness of the objects of  $F$ , and  $\gamma$  is the ratio of the largest and the smallest diameter of the objects. We say a grid cell  $\sigma$  is *heavy* if  $P(\sigma)$  consists of at least three points, and *light*, otherwise. Let  $P_h$  denote the set of vertices contained in the heavy cells, and let  $P_\ell$  denote the set of vertices contained in the light cells.

The following lemma implies that  $G_{\mathcal{P}}$  has a weighted treewidth of  $O(\sqrt{k})$  if it has a feedback vertex set of size  $k$ . The proof of the following lemma is inspired by the proof of [8, Lemma 9.1].

Note that the weighted treewidth of  $G$  is at most the weighted treewidth of  $G'$  plus one. Also,  $(G, k)$  is a **yes**-instance of FEEDBACK VERTEX SET if and only if  $(G', k)$  is a **yes**-instance of FEEDBACK VERTEX SET.

► **Lemma 7.**  *$G'$  has  $O(k)$  vertices of degree at least three if  $(G', k)$  is a **yes**-instance of FEEDBACK VERTEX SET, where  $G'$  is the graph obtained from  $G$  by removing all degree-1 vertices.*

**Proof.** If  $G'$  has a feedback vertex set of size  $k$ , the number of heavy cells is at most  $k$ , and the size of  $P_h$  is at most  $3k$ . This is because all but two of the vertices contained in the same cell must be contained in a feedback vertex set of  $G$ . In the following, we show that the number of vertices of  $P_\ell$  of degree at least three is at most  $O(k)$ .

To do this, consider the subgraph  $G_\ell$  of  $G'$  induced by  $P_\ell$ . We first observe that every vertex of  $G_\ell$  has degree at most  $4(\alpha\gamma)^2$  in  $G_\ell$ . (But note that its degree in  $G'$  might be larger than  $4(\alpha\gamma)^2$ .) Moreover,  $G_\ell$  also has a feedback vertex set of size  $k$ . Let  $X$  be a feedback vertex set of  $G_\ell$  of size  $k$ , and let  $Y$  be the set of vertices of  $G_\ell$  not contained in  $X$ . By definition, the subgraph  $G[Y]$  of  $G_\ell$  induced by  $Y$  is a forest. The number of vertices

of degree at least three is linear in the number of leaf nodes of the trees. Since  $G'$  has no degree-1 vertex, a leaf node of  $G[Y]$  is incident to a vertex of  $X \cup P_h$  in  $G'$ . Each vertex in  $X \cup P_h$  is incident to at most  $4(\alpha\beta)^2$  vertices of  $Y$ . Since  $|X \cup P_h|$  is  $O(k)$ , the total number of vertices of leaf nodes is  $O(k)$ , and thus, the total number of vertices of degree at least three of  $G_\ell$  (and  $G'$ ) is  $O(k)$ .  $\blacktriangleleft$

► **Lemma 8.** *For a greedy partition  $\mathcal{P}$ , the weighted treewidth of  $G_{\mathcal{P}}$  is  $O(\sqrt{k})$  if  $(G, k)$  is a *yes-instance* of FEEDBACK VERTEX SET.*

## 4.2 Dynamic Programming

By combining Lemma 7 and Theorem 1, we can obtain a tree decomposition of weighted width  $O(\sqrt{k})$  if a geometric intersection graph  $G$  has a feedback vertex set of size  $k$ . De Berg et al. [9] showed how to compute a feedback vertex set of size  $k$  in  $2^{O(\sqrt{w})}$  poly  $n$  time, where  $w$  denotes the weighted treewidth of  $G_{\mathcal{P}}$ . In this section, we briefly describe how this algorithm works.

Given a tree decomposition  $(T, \beta)$  of a graph  $G'$ , algorithm for solving FEEDBACK VERTEX SET computes for each bag  $\beta(t)$  and each subset  $S \subseteq \beta(t)$  together with the connectivity information  $\eta$  of the vertices of  $S$ , the maximum size  $c[t, S, \eta]$  of a feedback vertex set  $\hat{S}$  of  $G[t]$  with  $\hat{S} \cap \beta(t) = S$  satisfying the connectivity information  $\eta$ , where  $G[t]$  denotes the subgraph of  $G$  induced by the vertices appearing in bags in the subtree of  $T$  rooted at  $t$ .

Let  $t$  be a node of a tree decomposition  $(T, \beta)$  of  $G_{\mathcal{P}}$ . Recall that each vertex  $v$  of  $\beta(t)$  corresponds to the partition class  $P(v)$ . Let  $X_t = \bigcup_{v \in \beta(t)} P(v)$ . A feedback vertex set contains at least  $|C| - 2$  vertices of a clique  $C$ . Therefore, since from each partition class we can exclude  $O(1)$  vertices (at most two vertices from each clique), the number of subsets  $\hat{S}$  that need to be considered is at most

$$\prod_{v \in \beta(t)} |P(v)|^2 = \exp\left(\sum_{C \in \beta(t)} 2 \log |P(v)|\right) = 2^{O(\sqrt{k})}.$$

Therefore, we can track connectivity of these subsets by applying the rank-based approach of [3], which allows us to keep the number of equivalence relations considered single-exponential in  $O(\sqrt{k})$ . Assuming that we have  $c[t', \cdot, \cdot]$  for all descendants  $t'$  of  $t$  in  $T$  and their connectivity information, we can compute  $c[t, S, \eta]$  in  $2^{O(\sqrt{k})}$  time. Since the number of nodes in the tree decomposition is  $O(kn)$ , the total running time of the dynamic programming algorithm is  $2^{O(\sqrt{k})}(n + m)$ . Therefore, we have the following theorem.

► **Theorem 9.** *Given a intersection graph  $G$  of similarly sized fat objects in the plane (without its geometric representation), we can compute a feedback vertex set of size  $k$  in  $2^{O(\sqrt{k})}(n + m)$  time, if it exists.*

---

## References

- 1 Ann Becker and Dan Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167–188, 1996.
- 2 Sujoy Bhore, Paz Carmi, Sudeshna Kolay, and Meirav Zehavi. Parameterized study of Steiner tree on unit disk graphs. In *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, volume 162, pages 13:1–13:18, 2020.
- 3 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Information and Computation*, 243:86–111, 2015. 40th International Colloquium on Automata, Languages and Programming (ICALP 2013).



- 4 Marthe Bonamy and Łukasz Kowalik. A  $13k$ -kernel for planar feedback vertex set via region decomposition. *Theoretical Computer Science*, 645:25–40, 2016.
- 5 Sergio Cabello and Miha Ježič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015.
- 6 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: New measure and new structures. *Algorithmica*, 73(1):63–86, 2015.
- 7 Timothy M Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *Journal of Computational Geometry*, 10(2):3–20, 2019.
- 8 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshantov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 2015.
- 9 Mark de Berg, Hans L Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C van der Zanden. A framework for exponential-time-hypothesis-tight algorithms and lower bounds in geometric intersection graphs. *SIAM Journal on Computing*, 49(6):1291–1331, 2020.
- 10 Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and  $h$ -minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005.
- 11 Hristo N Djidjev. Weighted graph separators and their applications. In *Proceedings of the 5th Annual European Symposium on Algorithms (ESA 1997)*, pages 130–143. Springer, 1997.
- 12 David P Dobkin, Steven J Friedman, and Kenneth J Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990.
- 13 Fedor V. Fomin, Serge Gaspers, and Artem V. Pyatkin. Finding a minimum feedback vertex set in time  $O(1.7548^n)$ . In Hans L. Bodlaender and Michael A. Langston, editors, *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC 2006)*, volume 4169, pages 184–191, 2006.
- 14 Fedor V. Fomin, Daniel Lokshantov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80, pages 65:1–65:15, 2017.
- 15 Fedor V. Fomin, Daniel Lokshantov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. ETH-tight algorithms for long path and cycle on unit disk graphs. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164, pages 44:1–44:18, 2020.
- 16 Fedor V. Fomin, Daniel Lokshantov, and Saket Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1563–1575, 2012.
- 17 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. *Algorithmica*, 80(3):830–848, 2018.
- 18 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014.
- 19 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Proceedings of the 2nd International Conference on Parameterized and Exact Computation (IWPEC 2006)*, pages 154–165, 2006.
- 20 Frank van den Eijkhof, Hans L Bodlaender, and MC Arie Koster. Safe reduction rules for weighted treewidth. *Algorithmica*, 47(2):139–158, 2007.



# Streaming Algorithms for Graph $k$ -Matching with Optimal or Near-Optimal Update Time

Jianer Chen ✉

Department of Computer Science & Engineering, Texas A&M University, College Station, TX, USA

Qin Huang ✉

Department of Computer Science & Engineering, Texas A&M University, College Station, TX, USA

Iyad Kanj ✉

School of Computing, DePaul University, Chicago, IL, USA

Qian Li ✉

Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

Ge Xia ✉

Department of Computer Science, Lafayette College, Easton, PA, USA

---

## Abstract

We present streaming algorithms for the graph  $k$ -matching problem in both the insert-only and dynamic models. Our algorithms, while keeping the space complexity matching the best known upper bound, have optimal or near-optimal update time, significantly improving on previous results. More specifically, for the insert-only streaming model, we present a one-pass randomized algorithm that runs in optimal  $\mathcal{O}(k^2)$  space and has optimal  $\mathcal{O}(1)$  update time, and that, *w.h.p.* (with high probability), computes a maximum weighted  $k$ -matching of a weighted graph. Previously, the best upper bound on the update time was  $\mathcal{O}(\log k)$ , which was achieved by a deterministic streaming algorithm that however only works for unweighted graphs [16]. For the dynamic streaming model, we present a one-pass randomized algorithm that, *w.h.p.*, computes a maximum weighted  $k$ -matching of a weighted graph in  $\tilde{O}(Wk^2)$  space<sup>1</sup> and with  $\tilde{O}(1)$  update time, where  $W$  is the number of distinct edge weights. Again the update time of our algorithm improves the previous best upper bound  $\tilde{O}(k^2)$  [7]. Moreover, we prove that in the dynamic streaming model, any randomized streaming algorithm for the problem requires  $k^2 \cdot \Omega(W(\log W + 1))$  bits of space. Hence, both the space and update-time complexities achieved by our algorithm in the dynamic model are near-optimal. A streaming approximation algorithm for  $k$ -matching is also presented, whose space complexity matches the best known upper bound with a significantly improved update time.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Streaming, sublinear and near linear time algorithms

**Keywords and phrases** streaming algorithms, matching, parameterized algorithms, lower bounds

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.48

## 1 Introduction

Streaming algorithms for graph matching have been studied extensively, in which most of the work has been focused on approximating a maximum matching. A graph *stream*  $\mathcal{S}$  for an underlying graph  $G$  is a sequence of edge operations. In the *insert-only* streaming model, each operation is an edge-insertion, while in the *dynamic* streaming model each operation is either an edge-insertion or an edge-deletion (with a specified weight if  $G$  is weighted). The majority of the work on streaming algorithms for graph matching has been on the (simpler) insert-only model. More recently, streaming algorithms for graph  $k$ -matching

---

<sup>1</sup> The notation  $\tilde{O}()$  hides a poly-logarithmic factor in the input size.



(i.e., constructing a matching of  $k$  edges in an unweighted graph or a maximum weighted matching of  $k$  edges in a weighted graph), in both insert-only and dynamic models, have drawn increasing interests [7, 8, 9, 16].

The performance of streaming algorithms is measured by the limited memory (*space*) and the limited processing time per item (*update time*). For the space complexity, a lower bound  $\Omega(k^2)$  has been known for the graph  $k$ -matching problem on unweighted graphs for randomized streaming algorithms, even in the simpler insert-only model [7]. Nearly space-optimal streaming algorithms for graph  $k$ -matching have been developed [7].

The current paper will be focused on the update time of streaming algorithms for graph  $k$ -matching. While there has been much work on space complexity of streaming algorithms for graph matching, much less is known regarding the update time complexity of the problem. Note that the update time sometimes could be even more important than the space complexity [25], since the data stream can come at a very high rate. If the update processing does not catch the updating rate, the whole system may fail (see, e.g., [3, 34]).

We start with the insert-only model. We present a one-pass randomized streaming algorithm that constructs a maximum weighted  $k$ -matching in a weighted graph. Our algorithm runs in  $\mathcal{O}(k^2)$  space and has  $\mathcal{O}(1)$  update time, both are optimal. Our techniques rely on partitioning the graph (using hashing), and defining an auxiliary graph whose vertices are the different parts of the partition. The auxiliary graph is updated during the stream. By querying this auxiliary graph, the algorithm can compute a “compact” subgraph of size  $\mathcal{O}(k^2)$  that, *w.h.p.*, contains the edges of the desired  $k$ -matching. A maximum weighted  $k$ -matching can then be extracted from this compact subgraph.

Previously, Fafianie and Kratsch [16] studied kernelization streaming algorithms in the insert-only model. Their result implies a one-pass deterministic streaming algorithm for  $k$ -matching on unweighted graphs that uses  $\mathcal{O}(k^2)$  space and  $\mathcal{O}(\log k)$  update time. In comparison, our algorithm achieves the same space complexity but has optimal update time  $\mathcal{O}(1)$ . While improving the update time from  $\mathcal{O}(\log k)$  on the deterministic algorithm to  $\mathcal{O}(1)$  on randomized algorithms for unweighted graphs may not look surprising, our streaming algorithm with optimal space and update time for weighted graphs is a significant advance.

We then study steaming algorithms for graph  $k$ -matching in the dynamic model. We give a one-pass randomized streaming algorithm that, for a weighted graph  $G$  containing a  $k$ -matching, constructs a maximum weighted  $k$ -matching of  $G$  with probability at least  $1 - \frac{11}{20k^3 \ln(2k)}$ , and in case  $G$  does not contain a  $k$ -matching, reports correctly. The algorithm runs in  $\tilde{\mathcal{O}}(Wk^2)$  space and has  $\tilde{\mathcal{O}}(1)$  update time, where  $W$  is the number of distinct weights in the graph. This result directly implies a one-pass randomized streaming algorithm for unweighted  $k$ -matching running in  $\tilde{\mathcal{O}}(k^2)$  space with  $\tilde{\mathcal{O}}(1)$  update time.

In order to achieve the faster update time, we prove a structural result that can be useful in its own right for  $k$ -subset problems. Intuitively, the result states that, for any  $k$ -subset  $S \subseteq U$ , *w.h.p.* we can compute  $k$  subsets  $T_1, \dots, T_k$  of  $U$  that interact “nicely” with  $S$ . More specifically, (1) the sets  $T_i$ , for  $i \in [k]$ , are pairwise disjoint, (2)  $S$  is contained in their union  $\bigcup_{i \in [k]} T_i$ , and (3) each  $T_i$  contains exactly one element of  $S$ . We then apply the above result to obtain the sets  $T_i$  of vertices that *w.h.p.* induce the edges of the desired  $k$ -matching. Afterwards, we use  $\ell_0$ -sampling to select a smaller subset of edges induced by the vertices of the  $T_i$ ’s that *w.h.p.* contains the desired  $k$ -matching. From this smaller subset of edges, a maximum weighted  $k$ -matching can be extracted.

Employing lower bounds for communication complexity protocols, we prove that, modulo a poly-logarithmic function of the input size, the space complexity  $\tilde{\mathcal{O}}(Wk^2)$  achieved by our algorithm is optimal with respect to both  $k$  and  $W$ . More specifically, neither the linear

term  $W$  nor the quadratic function  $k^2$  in the space complexity of our algorithm for weighted  $k$ -matching can be improved/reduced (by more than a poly-logarithmic function). Given that our algorithms have  $\tilde{O}(1)$  update time, this implies that our algorithms are essentially near-optimal in terms of both space and update time complexities.

Chitnis et al. [7] proposed a streaming algorithm on the dynamic model for maximum matching. Under the promise that the cardinality of the maximum matching is not larger than  $k$  during the entire graph stream, their algorithm runs in space  $\tilde{O}(k^2)$  and has update time  $\tilde{O}(1)$  for unweighted graphs. The assumption that the cardinality of the maximum matching is at most  $k$  during the entire graph stream is essential for their techniques to work since it is used to upper bound the number of vertices of degree larger than or equal to  $10k$  by  $\mathcal{O}(k)$ , and the number of edges whose both endpoints have degree bounded by  $10k$  by  $\mathcal{O}(k^2)$ . They also developed a streaming algorithm that approximates the maximum matching for unweighted graphs. These two algorithms can be combined to construct a  $k$ -matching with update time  $\tilde{O}(k^2)$  and space  $\tilde{O}(k^2)$  for unweighted graphs. The algorithm for unweighted graphs can be extended to construct a maximum weighted  $k$ -matching for weighted graphs, which runs in space  $\tilde{O}(Wk^2)$  with update time  $\tilde{O}(k^2)$ . In comparison, our algorithm keeps the space complexity  $\tilde{O}(Wk^2)$  while has significantly improved update time  $\tilde{O}(1)$ .

A byproduct of our result is a one-pass streaming approximation algorithm that, for any  $\epsilon > 0$ , *w.h.p.* computes a  $k$ -matching that is within a factor of  $1 + \epsilon$  from a maximum weighted  $k$ -matching in  $G$ . The algorithm runs in  $\tilde{O}(k^2 \epsilon^{-1} \log W')$  space and has  $\tilde{O}(1)$  update time, where  $W'$  is the ratio of the maximum edge-weight to the minimum edge-weight in  $G$ . This result improves the update time complexity over the approximation result in [7], which has the same space complexity but has update time  $\tilde{O}(k^2)$ .

We observe that most work on weighted graph streams, including our current paper, assumes that the weight of an edge remains the same during the stream (see, e.g., [1, 2, 7, 20, 23]). To justify this assumption, we present an interesting lower bound result showing that, if this assumption is lifted, then the space complexity of the  $k$ -matching problem is at least linear in the size of the graph, and hence, can be much larger than the desirable space complexity for streaming algorithms.

The paper is organized as follows. Section 2 provides necessary definitions and a brief review on the related research. Improved streaming algorithms for graph  $k$ -matching in insert-only model and in dynamic model are presented and discussed in sections 3-5. Some lower bound results are give in section 6. Section 7 concludes with remarks.

## 2 Preliminaries

We refer to the following books for more detailed definitions [14, 15, 30]. We use “*u.a.r.*” as an abbreviation for “uniformly at random”. For an integer  $i$ , let  $[i]^-$  denote the set  $\{0, 1, \dots, i-1\}$ ,  $[i]$  the set  $\{1, \dots, i\}$ , and  $\lfloor i \rfloor$  the binary representation of  $i$ .

**Computational Model and Problem Definition.** In a parameterized graph streaming problem  $Q$ , we are given an instance of the form  $(\mathcal{S}, k)$ , where  $\mathcal{S}$  is graph stream of some underlying graph  $G$  and  $k \in \mathbb{N}$ , and we are asked to compute a solution for  $(\mathcal{S}, k)$  [9]. A  $k$ -matching in a graph  $G$  is a matching of  $k$  edges in  $G$ . We study the following problems:

- p-MATCHING: Given a graph stream  $\mathcal{S}$  of an unweighted graph  $G$  and a parameter  $k$ , compute a  $k$ -matching in  $G$  or report that no  $k$ -matching exists.
- p-WT-MATCHING: Given a graph stream  $\mathcal{S}$  of a weighted graph  $G$  and a parameter  $k$ , compute a  $k$ -matching of maximum weight in  $G$  or report that no  $k$ -matching exists.

We will assume that  $V(G) = [n]^-$ , and that the length of  $\mathcal{S}$  is polynomial in  $n$ . We will design parameterized streaming algorithms for the above problems. Our algorithms first sample a subgraph  $G'$  of the underlying graph  $G$  in the stream such that *w.h.p.*  $G'$  contains a desired  $k$ -matching of  $G$  if and only if  $G$  has one. In the case where the size of  $G'$  is a function of  $k$ , such algorithms are referred to as *kernelization streaming algorithms* [7]. We note that result in [7] also computes a subgraph containing the edges of the desired matching, without computing the matching itself, as there are efficient algorithms for extracting the desired matching from that subgraph [18].

**$\ell_0$ -Sampler.** Let  $\mathcal{S} = (i_1, \Delta_1), \dots, (i_p, \Delta_p), \dots$  be a stream of updates of an underlying vector  $\mathbf{x} \in \mathbb{R}^n$ , where  $i_j \in [n]$  and  $\Delta_j \in \mathbb{R}$ . The  $j$ -th update  $(i_j, \Delta_j)$  updates the  $i_j$ -th coordinate of  $\mathbf{x}$  by setting  $\mathbf{x}_{i_j} = \mathbf{x}_{i_j} + \Delta_j$ . Fix a parameter  $0 < \delta < 1$ . An  $\ell_0$ -sampler for  $\mathbf{x} \neq 0$  either fails with probability at most  $\delta$ , or conditioned on not failing, for any non-zero coordinate  $\mathbf{x}_j$  of  $\mathbf{x}$ , returns the pair  $(j, \mathbf{x}_j)$  with probability  $\frac{1}{\|\mathbf{x}\|_0}$ , where  $\|\mathbf{x}\|_0$  is the  $\ell_0$ -norm of  $\mathbf{x}$ , which is the same as the number of non-zero coordinates of  $\mathbf{x}$ . (We refer to [12].)

► **Lemma 1** (Follows from Theorem 2.1 in [7]). *Let  $0 < \delta < 1$  be a parameter. There exists an  $\ell_0$ -sampler algorithm that, given a dynamic graph stream, either returns FAIL with probability at most  $\delta$ , or returns an edge chosen u.a.r. from the edges of the stream that have been inserted and not deleted. This algorithm can be implemented using  $\mathcal{O}(\log^2 n \cdot \log(\delta^{-1}))$  bits of space and  $\tilde{O}(1)$  update time, where  $n$  is the number of vertices in the underlying graph.*

We give a brief review on the known work that is related to our current paper.

Most work on graph matching in the streaming model has focused on approximating a maximum matching (e.g., [4, 5, 19, 21, 22, 24, 26, 31]), with the majority of the work pertaining to the (simpler) insert-only model. The most relevant to ours are [7, 8, 9, 16], which studied parameterized streaming algorithms for the maximum matching problem.

Under the promise that the cardinality of the maximum matching at *every* instant of the stream is at most  $k$ , the authors of [8, 9] presented a one-pass dynamic streaming algorithm that *w.h.p.* computes a maximum matching in an unweighted graph stream. The algorithms given in [8, 9] run in  $\tilde{O}(k^2)$  space and the algorithm in [9] has  $\tilde{O}(k^2)$  update time.

The authors of [7] considered the problem of computing maximum matchings in the dynamic streaming model. For an unweighted graph  $G$ , under the promise that the cardinality of the maximum matching at every instant of the stream is at most  $k$ , a sketch-based algorithm is presented, which *w.h.p.* computes a maximum matching of  $G$ , runs in  $\tilde{O}(k^2)$  space, and has  $\tilde{O}(1)$  update time. They proved an  $\Omega(k^2)$  lower bound on the space complexity of any randomized algorithm for the parameterized maximum matching problem, even in the insert-only model, thus showing that the space complexity of their algorithm is optimal (modulo a poly-logarithmic factor). The algorithm for unweighted graphs has been extended to weighted graphs: under the same promise, there is an algorithm for computing a maximum weighted matching that runs in space  $\tilde{O}(k^2 W)$  and has  $\tilde{O}(1)$  update time, where  $W$  is the number of distinct edge weights. For unweighted graphs with larger matchings, an approximation algorithm is proposed [7]. Specifically, if the graph contains matchings of size larger than  $k$ , then for any  $1 \leq \alpha \leq \sqrt{k}$  and  $0 < \epsilon \leq 1$ , there exists an  $\tilde{O}(k^2 \alpha^{-3} \epsilon^{-2})$ -space algorithm that returns a matching of size at least  $\frac{(1-\epsilon)k}{2\alpha}$ . The algorithm has  $\tilde{O}(k^2 \alpha^{-2} \epsilon^{-2})$  update time.

Fafianie and Kratsch [16] studied kernelization streaming algorithms in the insert-only model for the NP-hard  $d$ -SET MATCHING problem (among others), which for  $d = 2$ , is equivalent to the  $k$ -matching problem on unweighted graphs. Their result implies a one-pass kernelization streaming algorithm for  $k$ -matching in unweighted graphs that computes a kernel of size  $\mathcal{O}(k^2 \log k)$ , runs in  $\mathcal{O}(k^2)$  space, and has  $\mathcal{O}(\log k)$  update time.

Chen et al. [6] studied algorithms for  $k$ -matching on the RAM model with limited computational resources, which is clearly very different from the streaming model. In order to translate their algorithm to the streaming model, it would require  $\Omega(nk)$  space and multiple passes. However, we remark that one of the steps of our algorithm in the insert-only model was inspired by the construction of reduced graphs introduced in [6].

Finally, there has been work on computing matchings in special graph classes, and with respect to parameters other than the cardinality of the matching (e.g., see [27, 28]).

### 3 Algorithms in Insert-Only Streaming Model

In this section, we give a streaming algorithm for P-WT-MATCHING, and hence for p-MATCHING as a special case, in the insert-only model. We start with some notations.

Given a weighted graph  $G = (V = [n]^-, E)$  along with a weight function  $wt : E(G) \rightarrow \mathbb{R}_{\geq 0}$ , and a parameter  $k$ , we define a new function  $\beta : E(G) \rightarrow \mathbb{R}_{\geq 0} \times [n]^- \times [n]^-$  as follows: for  $e = [u, v] \in E$ , where  $u < v$ , let  $\beta(e) = (wt(e), u, v)$ . Observe that  $\beta$  is injective.

Define a partial order relation  $\prec$  on  $E(G)$  as follows: for any two distinct edges  $e, e' \in E(G)$ ,  $e \prec e'$  if  $\beta(e)$  is lexicographically smaller than  $\beta(e')$ . For a vertex  $v \in V$  and an edge  $e$  incident to  $v$ , define  $\Gamma_v$  to be the sequence of edges incident to  $v$ , sorted in a decreasing order w.r.t.  $\prec$ . We say that  $e$  is the  $i$ -heaviest edge w.r.t.  $v$  if  $e$  is the  $i$ -th element in  $\Gamma_v$ .

Let  $f : V \rightarrow [4k^2]^-$  be a hash function, and let  $H$  be a subgraph of  $G$ . The function  $f$  partitions  $V(H)$  into a collection of subsets  $\mathcal{V} = \{V_0, V_1, \dots, V_{4k^2-1}\}$ , where each  $V_i$  consists of the vertices in  $V(H)$  that have the same image under  $f$ . A matching  $M$  in  $H$  is said to be *nice* w.r.t.  $f$  if no two vertices of  $M$  belong to the same  $V_i$  in  $\mathcal{V}$ . When the function  $f$  is clear from the context, we will simply say that “ $M$  is nice.” We define the *compact* subgraph of  $H$  under  $f$ , denoted  $Compact(H, f)$ , as the subgraph of  $H$  consisting of the edges  $e$  in  $H$  whose endpoints belong to different subsets  $V_i$  and  $V_j$  in  $\mathcal{V}$ , with  $i \neq j$ , and such that  $\beta(e)$  is the maximum over all edges between  $V_i$  and  $V_j$ . Finally, we define the *reduced compact* subgraph of  $H$  under  $f$ , denoted  $Red-Com(H, f)$ , by (1) for each pair  $(V_i, V_j)$  of subsets, selecting edges  $e \in Compact(H, f)$  with endpoints in  $V_i$  and  $V_j$  such that  $e$  is among the  $8k$  heaviest edges incident to vertices in  $V_i$  and among the  $8k$  heaviest edges incident to vertices in  $V_j$  (in both subsets, if there are not that many edges, then include all edges); and then (2) retaining from the selected edges in (1) the  $q = k(16k - 1)$  heaviest edges (again if there are not that many edges, include all edges). We have the following:

► **Lemma 2.** *The subgraph  $Compact(H, f)$  has a nice  $k$ -matching if and only if  $Red-Com(H, f)$  has a nice  $k$ -matching. If this is the case, then the weight of a maximum weighted nice  $k$ -matching in  $Compact(H, f)$  is equal to that in  $Red-Com(H, f)$ .*

**Proof.** Define the auxiliary weighted graph  $\Phi$  whose vertices are the subsets  $V_i$  in the collection  $\mathcal{V}$ , where  $i \in [4k^2]^-$ , such that there is an edge  $[V_i, V_j]$  in  $\Phi$  if some vertex  $u \in V_i$  is adjacent to some vertex  $v \in V_j$  in  $Compact(H, f)$ . We associate with edge  $[V_i, V_j]$  the value  $\beta([u, v])$  and associate the edge  $[u, v]$  with  $[V_i, V_j]$ . Obviously, there is one-to-one correspondence between the nice  $k$ -matchings in  $Compact(H, f)$  and the  $k$ -matchings in  $\Phi$ . Let  $\mathcal{H}$  be the subgraph of  $\Phi$  formed by selecting edges  $[V_i, V_j]$  such that  $[V_i, V_j]$  is among the  $8k$  heaviest edges incident to  $V_i$  and among the  $8k$  heaviest edges incident to  $V_j$ . Let  $\mathcal{H}'$  consist of the  $q = k(16k - 1)$  heaviest edges in  $\mathcal{H}$  (if  $\mathcal{H}$  has at most  $q$  edges, let  $\mathcal{H}' = \mathcal{H}$ ). Since there is a one-to-one correspondence between the nice  $k$ -matchings in  $Compact(H, f)$  and the  $k$ -matchings of  $\Phi$ , it suffices to prove the statement of the lemma with respect to matchings in  $\Phi$  and  $\mathcal{H}'$ : namely, if  $\Phi$  has a maximum weighted  $k$ -matching  $M$  then  $\mathcal{H}'$  has a maximum weighted  $k$ -matching of the same weight as  $M$ .

Suppose that  $\Phi$  has a maximum weighted  $k$ -matching  $M$ . Choose  $M$  such that the number of edges in  $M$  that remain in  $\mathcal{H}$  is maximized. We first show that all the edges in  $M$  remain in  $\mathcal{H}$ . Suppose not, then there is an edge  $[V_{i_0}, V_{i_1}] \in M$  such that  $[V_{i_0}, V_{i_1}]$  is not among the  $8k$  heaviest edges incident to one of its endpoints, say  $V_{i_1}$ . Since  $|V(M)| = 2k < 8k$ , it follows that there is a heaviest edge  $[V_{i_1}, V_{i_2}]$  incident to  $V_{i_1}$  such that  $\beta([V_{i_1}, V_{i_2}]) > \beta([V_{i_0}, V_{i_1}])$  and  $V_{i_2} \notin V_M$ . If  $[V_{i_1}, V_{i_2}] \in \mathcal{H}$ , then  $(M - [V_{i_0}, V_{i_1}]) + [V_{i_1}, V_{i_2}]$  is a maximum weighted  $k$ -matching of  $\Phi$  that contains more edges of  $\mathcal{H}$  than  $M$ , contradicting our choice of  $M$ . It follows that  $[V_{i_1}, V_{i_2}] \notin \mathcal{H}$ . Then,  $[V_{i_1}, V_{i_2}]$  is not among the  $8k$  heaviest edges incident to  $V_{i_2}$ . Now apply the above argument to  $V_{i_2}$  to select the heaviest edge  $[V_{i_2}, V_{i_3}]$  such that  $\beta([V_{i_2}, V_{i_3}]) > \beta([V_{i_1}, V_{i_2}]) > \beta([V_{i_0}, V_{i_1}])$  and  $V_{i_3} \notin V_M$ . By applying the above argument  $j$  times, we obtain a sequence of  $j$  vertices  $V_{i_1}, V_{i_2}, \dots, V_{i_j}$ , such that (1)  $\{V_{i_2}, \dots, V_{i_j}\} \cap V_M = \emptyset$ ; and (2)  $V_{i_a} \neq V_{i_b}$  for every  $a \neq b \in [j]$ , which is guaranteed by  $\beta([V_{i_a}, V_{i_{a+1}}]) < \beta([V_{i_{a+1}}, V_{i_{a+2}}]) < \dots < \beta([V_{i_{b-1}}, V_{i_b}])$  and  $[V_{i_a}, V_{i_{a+1}}]$  is the heaviest edge incident to  $V_{i_a}$  such that  $V_{i_{a+1}} \notin V_M$ . Since  $\Phi$  is finite, the above process must end at an edge  $e$  not in  $M$  and such that  $\beta(e)$  exceeds  $\beta([V_{i_0}, V_{i_1}])$ , contradicting our choice of  $M$ . Therefore,  $M \subseteq E(\mathcal{H})$ .

Now, choose a maximum weighted  $k$ -matching of  $\mathcal{H}$  that maximizes the number of edges retained in  $\mathcal{H}'$ . Without loss of generality, call it  $M$ . We prove that the edges of  $M$  are retained in  $\mathcal{H}'$ , thus proving the lemma. Suppose that this is not the case. Since each vertex in  $V(M)$  has degree at most  $8k$  and one of its edges must be in  $M$ , the number of edges in  $\mathcal{H}$  incident to the vertices in  $M$  is at most  $2k(8k - 1) + k = k(16k - 1) = q$ . It follows that there is an edge  $e$  in  $\mathcal{H}'$  whose endpoints are not in  $M$  and such that  $\beta(e)$  is larger than the  $\beta()$  value of some edge in  $M$ , contradicting our choice of  $M$ .  $\blacktriangleleft$

► **Lemma 3.** *Let  $f : V \rightarrow [4k^2]^-$  be a hash function, and let  $H$  be a subgraph of  $G$ . There is an algorithm **Alg-Reduce**( $H, f$ ) that constructs  $\mathcal{Red-Com}(H, f)$  and has both its time and space complexities bounded by  $\mathcal{O}(|H| + k^2)$ .*

We now present the streaming algorithm  $\mathcal{A}_{\text{Insert}}$  for P-WT-MATCHING. Let  $(\mathcal{S}, k)$  be an instance of p-WT-MATCHING, where  $\mathcal{S} = (e_1, wt(e_1)), \dots, (e_i, wt(e_i)), \dots$ . For  $i \in \mathbb{N}$ , let  $G_i$  be the subgraph of  $G$  consisting of the first  $i$  edges  $e_1, \dots, e_i$  of  $\mathcal{S}$ , and for  $j \leq i$ , let  $G_{j,i}$  be the subgraph of  $G$  whose edges are  $\{e_j, \dots, e_i\}$ ; if  $j > i$ , we let  $G_{j,i} = \emptyset$ . Let  $f$  be a hash function chosen u.a.r. from a universal set  $\mathcal{H}$  of hash functions mapping  $V$  to  $[4k^2]^-$ . The algorithm  $\mathcal{A}_{\text{Insert}}$ , after processing the  $i$ -th element  $(e_i, wt(e_i))$ , computes two subgraphs  $G_i^f$  and  $G_i^s$  defined as follows. For  $i = 0$ ,  $G_i^f = G_i^s = \emptyset$ . For  $i > 0$ , let  $\hat{i}$  be the largest multiple of  $q$  that is smaller than  $i$ , that is,  $i = \hat{i} + p$ , where  $0 < p \leq q$ ; and let  $i^*$  be the largest multiple of  $q$  that is smaller than  $\hat{i}$  if  $\hat{i} > 0$ , and 0 otherwise. The subgraph  $G_i^f$  is defined only when  $i$  is a multiple of  $q$ , and is defined recursively for  $i = j \cdot q > 0$  as  $G_i^f = \mathcal{Red-Com}(G_i^f \cup G_{i^*+1, \hat{i}})$ ; that is,  $G_i^f$  is the reduced compact subgraph of the graph consisting of  $G_i^f$  plus the subgraph consisting of the edges encountered after  $e_{i^*}$ , starting from  $e_{i^*+1}$  up to  $e_i$ . The subgraph  $G_i^s$  is defined as  $G_i^s = G_i^f \cup G_{i^*+1, i}$ ; that is,  $G_i^s$  consists of the previous (before  $i$ ) reduced compact subgraph plus the subgraph consisting of the edges starting after  $i^*$  up to  $i$ . We refer to Figure 1 for an illustration of the definitions of  $G_i^f$  and  $G_i^s$ .

► **Lemma 4.** *For each  $i \geq 1$ , if  $G_i$  contains a maximum weighted  $k$ -matching, then with probability at least  $1/2$ ,  $G_i^s$  contains a maximum weighted  $k$ -matching of  $G_i$ .*

**Proof.** Let  $M = \{[u_0, u_1], \dots, [u_{2k-2}, u_{2k-1}]\}$  be a maximum weighted  $k$ -matching in  $G_i$ , and let  $V_M = \{u_0, \dots, u_{2k-1}\}$ . Since  $f$  is a hash function chosen u.a.r. from a universal set  $\mathcal{H}$  of hash functions mapping  $V$  to  $[4k^2]^-$ , with probability at least  $1/2$ ,  $f$  is perfect w.r.t.  $V_M$  [11]. Now, suppose that  $f$  is perfect w.r.t.  $V_M$ . Thus,  $M$  is a nice matching (w.r.t.  $f$ ) in  $G_i$ . By the



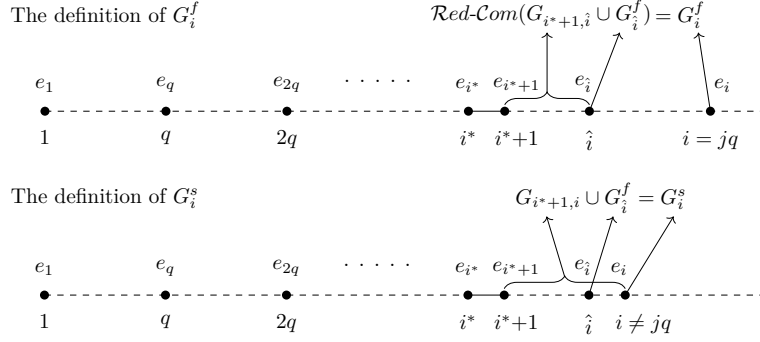


Figure 1 Illustration of the definitions of  $G_i^f$  and  $G_i^s$ .

definition of  $Compact(G_i, f)$ , there is a set  $M'$  of  $k$  edges  $M' = \{[u'_0, u'_1], \dots, [u'_{2k-2}, u'_{2k-1}]\}$  in  $Compact(G_i, f)$  such that  $\{f(u'_{2i}), f(u'_{2i+1})\} = \{f(u_{2i}), f(u_{2i+1})\}$  and  $\beta([u'_{2i}, u'_{2i+1}]) \geq \beta([u_{2i}, u_{2i+1}])$  for  $i \in [k]^-$ . It follows that  $wt([u'_{2i}, u'_{2i+1}]) \geq wt([u_{2i}, u_{2i+1}])$  for  $i \in [k]^-$ . Therefore,  $Compact(G_i, f)$  contains a maximum weighted  $k$ -matching of  $G_i$ , namely  $\{[u'_0, u'_1], \dots, [u'_{2k-2}, u'_{2k-1}]\}$ ; moreover, this matching is nice. By Lemma 2,  $Red-Com(G_i, f)$  contains a maximum weighted  $k$ -matching of  $Compact(G_i, f)$ .

Next, we prove that  $G_i^s$  contains a maximum weighted  $k$ -matching of  $G_i$ . If  $i \leq 2q$ , then  $G_i^s = G_{1,i} = G_i$  by definition, and hence  $G_i^s$  contains a maximum weighted  $k$ -matching of  $G_i$ . Suppose now that  $i > 2q$ . By definition,  $G_i^s = G_i^f \cup G_{i^*+1,i}$ . (Recall that, by definition,  $G_q^f = \emptyset$ ,  $G_{2q}^f = Red-Com(G_q^f \cup G_{1,q})$ ,  $G_{3q}^f = Red-Com(G_{2q}^f \cup G_{q+1,2q})$ ,  $\dots$ ,  $G_i^f = Red-Com(G_{i^*}^f \cup G_{i^*-q+1,i^*})$ .) For each  $j \geq 1$  that is multiple of  $q$ , let  $\mathcal{G}_j$  be the graph consisting of the edges that are in  $G_j^f \cup G_{j^*+1,\hat{j}}$  but are not kept in  $G_j^f$ . Consequently,  $(\bigcup_{q \leq j < \hat{i}, j \text{ is a multiple of } q} \mathcal{G}_j) \cup G_{i^*}^f = G_i^s$ . By the definition of  $Red-Com(G_i, f)$ , it is easy to verify that  $Red-Com(G_i, f)$  does not contain the edges in  $\mathcal{G}_j$ , for each  $j \geq 1$ . It follows that  $Red-Com(G_i, f)$  is a subgraph of  $G_i^s$ , and hence,  $G_i^s$  contains a maximum weighted  $k$ -matching of  $Red-Com(G_i, f)$ , and hence of  $Compact(G_i, f)$  by the above discussion. Since  $Compact(G_i, f)$  contains a maximum weighted  $k$ -matching of  $G_i$ ,  $G_i^s$  contains a maximum weighted  $k$ -matching of  $G_i$ . It follows that, with probability at least  $1/2$ ,  $G_i^s$  contains a maximum weighted  $k$ -matching of  $G_i$ . ◀

The algorithm  $\mathcal{A}_{Insert}$ , when queried at the end of the stream, either returns a maximum weighted  $k$ -matching of  $G$  or the empty set. To do so, at every instant  $i$ , it will maintain a subgraph  $G_i^s$  that will contain the edges of the desired matching, from which this matching can be extracted. To maintain  $G_i^s$ , the algorithm keeps track of the subgraphs  $G_{i-1}^s$ ,  $G_i^f$ , the edges  $e_{i^*+1}, \dots, e_i$ , and will use them in the computation of the subgraph  $G_i^s$  as follows. If  $i$  is not a multiple of  $q$ , then  $G_i^s = G_{i-1}^s + e_i$ , and the algorithm simply computes  $G_i^s$  as such. Otherwise (i.e.,  $i$  is a multiple of  $q$ ),  $G_i^s = G_i^f \cup G_{i^*+1,i}$ , and the algorithm uses  $G_i^f$  and  $G_{i^*+1,i} = \{e_{i^*+1}, \dots, e_i\}$  to compute and return  $G_i^s$ ; however, in this case (i.e.,  $i$  is a multiple of  $q$ ), the algorithm will additionally need to have  $G_i^f$  already computed, in preparation for the potential computations of subsequent  $G_j^s$ , for  $j \geq i$ . By Lemma 3, the subgraph  $G_i^f$  can be computed by invoking the algorithm **Alg-Reduce** in Lemma 3 on  $G_i^f \cup G_{i^*+1,\hat{i}}$ , which runs in time  $\mathcal{O}(q)$ . Note that both  $G_i^f$  and  $G_{i^*+1,\hat{i}}$  are available to  $\mathcal{A}_{Insert}$  at each of the steps  $\hat{i} + 1, \dots, i$ . Therefore, the algorithm will stagger the  $\mathcal{O}(q)$  many operations needed for the computation of  $G_i^f$  uniformly (roughly equally) over each of the steps  $\hat{i} + 1, \dots, i$ ,



yielding an  $\mathcal{O}(1)$  operations per step. Note that all the operations in **Alg-Reduce** can be explicitly listed, and hence, splitting them over an interval of  $q$  steps is easily achievable. Combining the above discussions and lemmas, we conclude with:

► **Lemma 5.** *The algorithm  $\mathcal{A}_{\text{Insert}}$  runs in space  $\mathcal{O}(k^2)$  and has update time  $\mathcal{O}(1)$ .*

► **Theorem 6.** *Let  $0 < \delta < 1$  be a parameter. There is an algorithm for P-WT-MATCHING such that, on input  $(S, k)$ , the algorithm outputs a matching  $M'$  satisfying that (1) if  $G$  contains a  $k$ -matching then, with probability at least  $1 - \delta$ ,  $M'$  is a maximum weighted  $k$ -matching of  $G$ ; and (2) if  $G$  does not contain a  $k$ -matching then  $M' = \emptyset$ . The algorithm runs in  $\mathcal{O}(k^2 \log \frac{1}{\delta})$  space and has  $\mathcal{O}(\log \frac{1}{\delta})$  update time. In particular, for any constant  $\delta$ , the algorithm runs in space  $\mathcal{O}(k^2)$  and has  $\mathcal{O}(1)$  update time.*

**Proof.** Run  $\lceil \log \frac{1}{\delta} \rceil$ -many copies of algorithm  $\mathcal{A}_{\text{Insert}}$  in parallel (i.e., using dove-tailing). Then, by the end of the stream, there are  $\lceil \log \frac{1}{\delta} \rceil$  copies of  $G_m^s$ , where  $m$  is the length of the stream. Let  $G'$  be the union of all the  $G_m^s$ 's produced by the runs of  $\mathcal{A}_{\text{Insert}}$ . If  $G'$  has a  $k$ -matching, let  $M'$  be a maximum weighted  $k$ -matching of  $G'$ ; otherwise, let  $M' = \emptyset$ .

By Lemma 4, if  $G_m$ , i.e.,  $G$ , contains a maximum weighted  $k$ -matching, with probability at least  $1/2$ , one copy of  $G_m^s$  contains a maximum weighted  $k$ -matching of  $G$ . Hence, with probability at least  $1 - (1/2)^{\lceil \log \frac{1}{\delta} \rceil} \geq 1 - \delta$ ,  $G'$  contains a maximum weighted  $k$ -matching of  $G$ . It follows that if  $G$  contains a maximum weighted  $k$ -matching  $M$  then, with probability at least  $1 - \delta$ ,  $G'$  contains a maximum weighted  $k$ -matching of the same weight as  $M$  and hence  $M'$  is a maximum weighted  $k$ -matching of  $G$ .

Observe that the graph  $G'$  is a subgraph of  $G$ . Therefore, statement (2) in the theorem clearly holds true. By Lemma 5, the above algorithm runs in space  $\mathcal{O}(k^2 \log \frac{1}{\delta})$  and has update time  $\mathcal{O}(\log \frac{1}{\delta})$ , thus completing the proof. ◀

## 4 The Toolkit

In this section, we prove a theorem that can be useful in its own right for subset problems, that is, problems in which the goal is to compute a  $k$ -subset  $S$  ( $k \in \mathbb{N}$ ) of some universe  $U$  such that  $S$  satisfies certain prescribed properties. Intuitively, the theorem states that, for any  $k$ -subset  $S \subseteq U$ , *w.h.p.* we can compute  $k$  subsets  $T_1, \dots, T_k$  of  $U$  that interact “nicely” with  $S$ . More specifically, (1) the sets  $T_i$ , for  $i \in [k]$ , are pairwise disjoint, (2)  $S$  is contained in their union  $\bigcup_{i \in [k]} T_i$ , and (3) each  $T_i$  contains exactly one element of  $S$ .

The above theorem will be used in Section 5 to design algorithms for p-MATCHING and p-WT-MATCHING in the dynamic streaming model. Intuitively speaking, the theorem will be invoked to obtain the sets  $T_i$  of vertices that *w.h.p.* induce the edges of the desired  $k$ -matching; however, these sets may not necessarily constitute the desired subgraph as they may not have “small” cardinalities. Sampling techniques will be used to select a smaller set of edges induced by the vertices of the  $T_i$ 's that *w.h.p.* contains the edges of the  $k$ -matching.

A family  $\mathcal{H}$  of hash functions, each mapping  $U$  to  $[r]^-$ , is called  $\kappa$ -wise independent if for any  $\kappa$  distinct keys  $x_1, x_2, \dots, x_\kappa \in U$ , and any  $\kappa$  (not necessarily distinct) values  $a_1, a_2, \dots, a_\kappa \in [r]^-$ , we have  $\Pr_{h \in \mathcal{H}}[h(x_1) = a_1 \wedge h(x_2) = a_2 \wedge \dots \wedge h(x_\kappa) = a_\kappa] = \frac{1}{r^\kappa}$ .

Let  $\mathbb{F}$  be a finite field. A  $\kappa$ -wise independent family  $\mathcal{H}$  of hash functions can be constructed as follows (See Construction 3.32 in [35]):  $\mathcal{H} = \{h_{a_0, a_1, \dots, a_{\kappa-1}} : \mathbb{F} \rightarrow \mathbb{F}\}$ , where  $h_{a_0, a_1, \dots, a_{\kappa-1}}(x) = a_0 + a_1x + \dots + a_{\kappa-1}x^{\kappa-1}$  for  $a_0, \dots, a_{\kappa-1} \in \mathbb{F}$ .

The following theorem is proved in [35], and will be used in our discussion.

► **Theorem 7** (Corollary 3.34 in [35]). *For every  $u, d, \kappa \in \mathbb{N}$ , there is a family of  $\kappa$ -wise independent functions  $\mathcal{H} = \{h : \{0, 1\}^u \rightarrow \{0, 1\}^d\}$  such that choosing a random function from  $\mathcal{H}$  takes space  $\mathcal{O}(\kappa \cdot (u + d))$ . Moreover, evaluating a function from  $\mathcal{H}$  takes time polynomial in  $u, d, \kappa$ .*

To prove our theorem, we proceed in two phases. We give an intuitive description of these two phases. In the first phase, we choose a hashing function  $f$  u.a.r. from an  $\mathcal{O}(\ln k)$ -wise independent set of hash functions, which hashes  $U$  to a set of  $d_1 = \mathcal{O}(k/\ln k)$  integers. We use  $f$  to partition the universe  $U$  into  $d_1$ -many subsets  $U_i$ . Afterwards, we choose  $d_1$  families  $F_0, \dots, F_{d_1-1}$  of hash functions, each containing  $d_2 = \mathcal{O}(\ln k)$  functions, chosen independently and u.a.r. from a universal set of hash functions. The family  $F_i, i \in [d_1]^-$ , will be used restrictively to map the elements of  $U_i$ . Since each family  $F_i$  is chosen from a universal set of hash function, for the subset  $S_i = S \cap U_i$ , w.h.p.  $F_i$  contains a hash function  $f_i$  that is perfect w.r.t.  $S_i$ ; that is, under the function  $f_i$  the elements of  $S_i$  are distinguished. This concludes the first phase of the process, which is described in **Algorithm 1**.

■ **Algorithm 1** : An algorithm for partitioning  $U$  and constructing families of hash functions.

---

**Input:**  $|U|, k \in \mathbb{N}$  where  $|U| > 1$

**Output:** A family of sets of hash functions

- 1: let  $u$  and  $d$  be the unique positive integers satisfying  $2^{u-1} < |U| \leq 2^u$  and  $2^{d-1} < \frac{k}{\ln k} \leq 2^d$
  - 2: choose  $f$  u.a.r. from  $\mathcal{H}$ , where  $\mathcal{H} = \{h : \{0, 1\}^u \rightarrow \{0, 1\}^d\}$  is a  $\lceil 12 \ln k \rceil$ -wise independent set of hash functions
  - 3: let  $\mathcal{H}'$  be a set of universal hash functions from  $U$  to  $[\lceil 13 \ln k \rceil^2]^-$
  - 4: let  $F_i$ , for  $i \in [2^d]^-$ , be a set of  $\lceil 8 \ln k \rceil$  hash functions chosen independently and u.a.r. from  $\mathcal{H}'$
  - 5: return  $\{f, F_0, \dots, F_{2^d-1}\}$
- 

In the second phase, we define a relation  $\mathcal{G}$  (from  $U$ ) that, for each  $x \in U$ , associates a set  $\mathcal{G}(x)$  of integers. This relation extends the hash functions in the  $F_j$ 's above by (1) ensuring that elements in different parts of  $U$  (w.r.t. the partitioning) are distinguished, in the sense that they are associated with subsets of integers that are contained in disjoint intervals of integers; and (2) maintaining the property that elements of the same part  $U_j$  that are distinguished under some function in  $F_j$  remain so under the extended relation. To do so, for each part  $U_j$ , we associate an “offset” and create a large gap between any two (consecutive) offsets; we will ensure that all the elements in the same  $U_j$  fall within the same interval determined by two consecutive offsets. To compute the set  $\mathcal{G}(x)$ , for an element  $x \in U_j$ , we start with an offset  $o_j$  that depends solely on  $U_j$  ( $o_j = j \cdot d_2 \cdot d_3$  in **Algorithm 2**), and consider every function in the family  $F_j$  corresponding to  $U_j$ . For each such function  $h_i$ , we associate an offset  $o'_i$  ( $o'_i = (i-1) \cdot d_3$  in **Algorithm 2**), and for  $x$  and that particular function  $h_i$ , we add to  $\mathcal{G}(x)$  the value  $g(j, i, x) = o_j + o'_i + h_i(x)$ . The above phase is described in **Algorithm 2**.

Now that the relations  $\mathcal{G}(x)$ , for  $x \in U$ , have been defined, we will show in the following theorem that, for any  $k$ -subset  $S$  of  $U$ , w.h.p. there exist  $k$  distinct elements  $i_0, \dots, i_{k-1}$ , such that their pre-images  $\mathcal{G}^{-1}(i_0), \dots, \mathcal{G}^{-1}(i_{k-1})$  are pairwise disjoint, contain all elements of  $S$ , and each pre-image contains exactly one element of  $S$ ; those pre-images serve as the desired sets  $T_i$ , for  $i \in [k]$ .

Consider **Algorithm 1** and **Algorithm 2**, and refer to them for the terminologies used in the subsequent discussions. For  $i \in [d_1 \cdot d_2 \cdot d_3]^-$ , define  $T_i = \{x \in U \mid i \in \mathcal{G}(x)\}$ . We define next two sequences of intervals, and prove certain properties about them, that will

■ **Algorithm 2** : An algorithm that defines the relation  $\mathcal{G}$  from  $U$  to  $[d_1 \cdot d_2 \cdot d_3]^-$ .

**Input:**  $x \in U$ ,  $k \in \mathbb{N}$ ,  $\{f, F_0, \dots, F_{d_1-1}\}$  is from Algorithm 1, where  $|F_0| = \dots = |F_{d_1-1}|$

**Output:** a set  $\mathcal{G}(x)$

- 1: let  $d_2 = |F_0| = \dots = |F_{d_1-1}|$  and  $d_3 = \lceil 13 \ln k \rceil^2$
- 2:  $\mathcal{G}(x) = \emptyset$
- 3: compute  $f(\lfloor x \rfloor)$  and let  $j$  be the integer such that  $\lfloor j \rfloor = f(\lfloor x \rfloor)$
- 4: **for**  $i = 1$  to  $d_2$  **do**
- 5:   let  $h_i$  be the  $i$ -th function in  $F_j$  (assuming an arbitrary ordering on  $F_j$ )
- 6:   let  $g(j, i, x) = j \cdot d_2 \cdot d_3 + (i - 1) \cdot d_3 + h_i(x)$  and let  $\mathcal{G}(x) = \mathcal{G}(x) \cup \{g(j, i, x)\}$
- 7: return  $\mathcal{G}(x)$

be used in the proof of Theorem 9. For  $q \in [d_1 \cdot d_2]^-$ , let  $I_q = \{r \mid q \cdot d_3 \leq r < (q + 1) \cdot d_3\}$ . For  $t \in [d_1]^-$ , let  $I'_t = \{r \mid t \cdot d_2 \cdot d_3 \leq r < t \cdot d_2 \cdot d_3 + d_2 \cdot d_3\}$ . Note that each interval  $I'_t$  is partitioned into the  $d_2$ -many intervals  $I_q$ , for  $q = t \cdot d_2, \dots, t \cdot d_2 + d_2 - 1$ .

► **Lemma 8.** *The following statements hold: (A) For any two distinct integers  $a, b \in I_q$ , where  $q \in [d_1 \cdot d_2]^-$ , we have  $T_a \cap T_b = \emptyset$ . (B) For  $t \in [d_1]^-$ , we have  $\mathcal{G}(U_t) \subseteq I'_t$ . Moreover, for any  $a \in I'_t, b \in I'_s$ , where  $s \neq t$ , we have  $T_a \cap T_b = \emptyset$ .*

► **Theorem 9.** *For any subset  $S \subseteq U$  of cardinality  $k \geq 2$ , with probability at least  $1 - \frac{4}{k^3 \ln k}$ , there exist  $k$  sets  $T_{i_0}, \dots, T_{i_{k-1}}$  such that: (1)  $|T_{i_j} \cap S| = 1$  for  $j \in [k]^-$ , (2)  $S \subseteq \cup_{j \in [k]^-} T_{i_j}$ , and (3)  $T_{i_j} \cap T_{i_l} = \emptyset$  for  $j \neq l \in [k]^-$ .*

**Proof.** For  $j \in [d_1]^-$ , let  $U_j$  be the set of elements in  $U$  whose image is  $\lfloor j \rfloor$  under  $f$  (defined in Step 2 of **Algorithm 1**), that is  $U_j = \{y \in U \mid f(\lfloor y \rfloor) = \lfloor j \rfloor\}$ . Clearly, the sets  $U_j$ , for  $j \in [d_1]^-$ , partition the universe  $U$ . We will show that, with probability at least  $1 - \frac{4}{k^3 \ln k}$ , there exist  $k$  sets  $T_{i_0}, \dots, T_{i_{k-1}}$  that satisfy conditions (1)–(3) in the statement of the theorem.

Let  $S \subseteq U$  be any subset such that  $|S| = k$ . For  $j \in [d_1]^-$  and  $y \in S$ , let  $X_{y,j}$  be the random variable defined as  $X_{y,j} = 1$  if  $f(\lfloor y \rfloor) = \lfloor j \rfloor$  and 0 otherwise. Let  $X_j = \sum_{y \in S} X_{y,j}$ , and  $S_j = \{y \in S \mid f(\lfloor y \rfloor) = \lfloor j \rfloor\}$ . Thus,  $|S_j| = X_j$ . Since  $f$  is  $\lceil 12 \ln k \rceil$ -wise independent, the random variables  $X_{y,j}$ , for  $y \in S$ , are  $\lceil 12 \ln k \rceil$ -wise independent and  $\Pr(X_{y,j} = 1) = \frac{1}{d_1}$ . Thus,  $E[X_j] = |S| \cdot \frac{1}{d_1}$ . Since  $d_1 = 2^d$  and  $2^{d-1} < \frac{k}{\ln k} \leq 2^d$  by definition, we have  $\frac{k}{\ln k} \leq d_1 < \frac{2k}{\ln k}$  and  $\frac{\ln k}{2} < E[X_j] \leq \ln k$ . Applying Theorem 2 in [33] with  $\mu = E[X_j]$  and  $\delta = \frac{12 \ln k}{E[X_j]} > 1$ , we get  $\Pr(X_j \geq (1 + \delta)E[X_j]) \leq e^{-E[X_j]\delta/3} = \frac{1}{k^4}$ . Since  $E[X_j] \leq \ln k$  and  $\delta = \frac{12 \ln k}{E[X_j]}$ , we have  $(1 + \delta)E[X_j] \leq 13 \ln k$ . Hence,  $\Pr(X_j \geq 13 \ln k) \leq \Pr(X_j \geq (1 + \delta)E[X_j]) \leq \frac{1}{k^4}$ . Let  $\mathcal{E}$  denote the event  $\bigwedge_{i \in [d_1]^-} (X_i \leq 13 \ln k)$ . By the union bound, we have  $\Pr(\mathcal{E}) \geq 1 - \frac{d_1}{k^4} \geq 1 - \frac{2}{k^3 \ln k}$ , where the last inequality holds since  $d_1 < 2k / \ln k$ .

Assume that event  $\mathcal{E}$  occurs, i.e., that  $|S_j| \leq 13 \ln k$  holds for  $j \in [d_1]^-$ . Consider Step 4 in **Algorithm 1**. Fix  $j \in [d_1]^-$ , and let  $E_j$  be the event that  $F_j$  does not contain any perfect hash function w.r.t.  $S_j$ . Let  $h$  be a hash function picked from  $\mathcal{H}'$  u.a.r. Since  $|S_j| \leq 13 \ln k$  (by assumption), by Theorem 11.9 in [11], with probability at least  $1/2$ ,  $h$  is perfect w.r.t.  $S_j$ . Since  $F_j$  consists of  $\lceil 8 \ln k \rceil$  hash functions chosen independently and u.a.r. from  $\mathcal{H}'$ , we have  $\Pr(E_j) \leq (1/2)^{\lceil 8 \ln k \rceil} < \frac{1}{k^4}$ . Applying the union bound, we have  $\Pr(\cup_{j \in [d_1]^-} E_j) \leq \frac{d_1}{k^4} < \frac{2}{k^3 \ln k}$ . Let  $\mathcal{E}'$  be the event that there exist  $d_1$  functions  $f_0, f_1, \dots, f_{d_1-1}$  such that  $f_j \in F_j$  and  $f_j$  is perfect w.r.t.  $S_j$ ,  $j \in [d_1]^-$ . Therefore,  $\Pr(\mathcal{E}') \geq \Pr(\mathcal{E})(1 - \Pr(\cup_{j \in [d_1]^-} E_j)) \geq 1 - \frac{4}{k^3 \ln k} + \frac{4}{k^6 \ln^2 k} \geq 1 - \frac{4}{k^3 \ln k}$ . Suppose that such a set  $\{f_0, \dots, f_{d_1-1}\}$  of functions exists. Let  $\eta(q)$  be the iteration number  $i$  in Step 5 of **Algorithm 2** during which  $f_q \in F_q$  is chosen, for  $q \in [d_1]^-$ . We define the following (multi-)set  $B$  as follows. For each  $q \in [d_1]^-$ , and for

element  $x \in S_q$ , add to  $B$  the element  $g(q, \eta(q), x)$  defined in Steps 5–6 of **Algorithm 2** (by  $\{f, f_0, \dots, f_{k-1}\}$ ). Observe that, by the definition of  $B$ , for every  $x \in S$ , there exists  $a \in B$  such that  $x \in T_a$ . We will show next that  $B$  contains exactly  $k$  distinct elements, and that, for any  $a \neq b \in B$ , it holds that  $T_a \cap T_b = \emptyset$ . The above will show that the sets  $\{T_a \mid a \in B\}$  satisfy conditions (1)–(3) of the theorem, thus proving the theorem.

It suffices to show that for any two distinct elements of  $S$ , the corresponding elements added to  $B$  are distinct. Let  $x_1$  and  $x_2$  be two distinct elements of  $S$ . Assume that  $x_1 \in S_j$  and  $x_2 \in S_l$ , where  $j, l \in [d_1]^-$ . We distinguish two cases based on whether or not  $j = l$ .

If  $j = l$ , we have  $g(j, \eta(j), x_1) = j \cdot d_2 \cdot d_3 + (\eta(j) - 1) \cdot d_3 + f_j(x_1)$  and  $g(j, \eta(j), x_2) = j \cdot d_2 \cdot d_3 + (\eta(j) - 1) \cdot d_3 + f_j(x_2)$ . Since  $f_j$  is perfect w.r.t.  $S_j$ , we have  $g(j, \eta(j), x_1) \neq g(j, \eta(j), x_2)$ . Moreover, both  $g(j, \eta(j), x_1)$  and  $g(j, \eta(j), x_2)$  are in  $I_{j \cdot d_2 + (\eta(j) - 1)}$  (since  $0 \leq h_j(x_1), h_j(x_2) < d_3$ ), where  $j \cdot d_2 + (\eta(j) - 1) \leq (d_1 - 1) \cdot d_2 + (d_2 - 1) \in [d_1 \cdot d_2]^-$ . By part (A) of Lemma 8, it holds that  $T_{g(j, \eta(j), x_1)} \cap T_{g(j, \eta(j), x_2)} = \emptyset$ .

Suppose now that  $j \neq l$ . By definition of  $S_j, S_l, U_j, U_l$ , we have  $S_j \subseteq U_j$  and  $S_l \subseteq U_l$ . Consequently,  $g(j, \eta(j), x_1) \in \mathcal{G}(U_j)$  and  $g(l, \eta(l), x_2) \in \mathcal{G}(U_l)$  hold. By part (B) of Lemma 8, we have  $\mathcal{G}(U_j) \subseteq I'_j$  and  $\mathcal{G}(U_l) \subseteq I'_l$ . Therefore,  $g(j, \eta(j), x_1) \neq g(l, \eta(l), x_2)$ . Moreover,  $T_{g(j, \eta(j), x_1)} \cap T_{g(l, \eta(l), x_2)} = \emptyset$  holds by part (B) of Lemma 8 as well.  $\blacktriangleleft$

**► Theorem 10.** *Algorithm 1 runs in space  $\mathcal{O}(k + (\log k)(\log |U|))$ , and Algorithm 2 runs in space  $\mathcal{O}(\log k)$  and in time polynomial in  $\log |U|$ .*

**Proof.** In **Algorithm 1**, since  $f$  is  $\lceil 12 \ln k \rceil$ -wise independent, by Theorem 7, storing  $f$  uses space  $\mathcal{O}(\ln k \cdot \max\{u, d\}) = \mathcal{O}((\log k)(\log |U|))$  (since  $k \leq |U|$ ). Storing a universal hash function uses  $\mathcal{O}(1)$  space, and thus storing  $\{F_0, \dots, F_{d_1-1}\}$  uses  $\mathcal{O}(d_1 \cdot d_2) = \mathcal{O}(k)$  space. Therefore, **Algorithm 1** can be implemented in space  $\mathcal{O}(k + (\log k)(\log |U|))$ .

For **Algorithm 2**, since  $\mathcal{G}(x)$  contains exactly  $d_2$  elements, storing  $\mathcal{G}(x)$  takes  $\mathcal{O}(d_2) = \mathcal{O}(\ln k)$  space. In Step 3, again by Theorem 7, computing  $f(\lfloor x \rfloor)$  takes time polynomial in  $\log |U|$  and  $\log k$ , since  $f$  is a  $\lceil 12 \ln k \rceil$ -wise independent hash function from  $\{0, 1\}^u$  to  $\{0, 1\}^d$ . Computing  $j$  in Step 3 takes time polynomial in  $d = \mathcal{O}(\log k)$  since  $f(\lfloor x \rfloor) \in \{0, 1\}^d$ . Therefore, Step 3 can be performed in time polynomial in  $\log |U|$  and  $\log k$ , and hence polynomial in  $\log |U|$  (since  $k \leq |U|$ ). Step 6 can be implemented in time polynomial in  $\log k$ , since  $|F_j| = \lceil 8 \ln k \rceil$ . Altogether, **Algorithm 2** takes time polynomial in  $\log |U|$ . This completes the proof.  $\blacktriangleleft$

## 5 Algorithms in Dynamic Streaming Model

In this section, we present results on p-MATCHING and p-WT-MATCHING in the dynamic streaming model. The algorithm uses the toolkit developed in the previous section, together with the  $\ell_0$ -sampling technique discussed in Section 2. We first give a high-level description of how the algorithm works.

Let  $\mathcal{S}$  be a graph stream of a weighted graph  $G = (V = [n]^-, E)$  along with the weight function  $wt : E(G) \rightarrow \mathbb{R}_{\geq 0}$ , and  $k$  be a parameter. Suppose  $G$  has  $W$  distinct weights. We will hash the vertices of the graph to a range  $R$  of size  $\mathcal{O}(k \log^2 k)$ . For each element  $(e = [u, v], wt(e), op) \in \mathcal{S}$ , where  $op$  is either insertion or deletion, we use the relation  $\mathcal{G}$ , discussed in Section 4, and compute the two sets  $\mathcal{G}(u)$  and  $\mathcal{G}(v)$ . For each  $i \in \mathcal{G}(u)$  and each  $j \in \mathcal{G}(v)$ , we associate an instance of an  $\ell_0$ -sampler primitive, call it  $\mathcal{C}_{i,j,wt(u,v)}$ , and update it according to the operation  $op$ . Recall that it is assumed that the weight of every edge does not change throughout the stream.

The solution computed by the algorithm consists of a set of edges created by invoking each of the  $\tilde{O}(Wk^2)$   $\ell_0$ -sampler algorithms to sample at most one edge from each  $\mathcal{C}_{i,j,w}$ , for each pair of  $i, j$  in the range  $R$  and each edge-weight of the graph stream.

The intuition behind the above algorithm (i.e., why it achieves the desired goal) is the following. Suppose that there exists a maximum weighted  $k$ -matching  $M$  in  $G$ , and let  $M = \{[u_0, u_1], \dots, [u_{2k-2}, u_{2k-1}]\}$ . By Theorem 9, *w.h.p.* there exist  $i_0, \dots, i_{2k-1}$  in the range  $R$  such that  $u_j \in T_{i_j}$ , for  $j \in [2k]^-$ , and such that the  $T_{i_j}$ 's are pairwise disjoint. Consider the  $k$   $\ell_0$ -samplers  $\mathcal{C}_{i_{2j}, i_{2j+1}, wt(u_{2j}, u_{2j+1})}$ , where  $j \in [k]^-$ . Then, *w.h.p.*, the  $k$  edges sampled from these  $k$   $\ell_0$ -samplers are the edges of a maximum weighted  $k$ -matching (since the  $T_{i_j}$ 's are pairwise disjoint) whose weight equals that of  $M$ .

■ **Algorithm 3** The streaming algorithm  $\mathcal{A}_{dynamic}$  in the dynamic streaming model.

---



---

**$\mathcal{A}_{dynamic}$ -Preprocess: The preprocessing algorithm**

---

**Input:**  $n = |V(G)|$  and a parameter  $k \in \mathbb{N}$

- 1: let  $\mathcal{C}$  be a set of  $\ell_0$ -samplers and  $\mathcal{C} = \emptyset$
  - 2: let  $\{f, F_0, F_1, \dots, F_{d_1-1}\}$  be the output of **Algorithm 1** on input  $(n, 2k)$
- 

**$\mathcal{A}_{dynamic}$ -Update: The update algorithm**

---

**Input:** An update  $(e = [u, v], wt(e), op) \in \mathcal{S}$ , where  $op$  is either insertion or deletion

- 1: let  $\mathcal{G}(u)$  be the output of **Algorithm 2** on input  $(u, 2k, \{f, F_0, F_1, \dots, F_{d_1-1}\})$
  - 2: let  $\mathcal{G}(v)$  be the output of **Algorithm 2** on input  $(v, 2k, \{f, F_0, F_1, \dots, F_{d_1-1}\})$
  - 3: **for**  $i \in \mathcal{G}(u)$  and  $j \in \mathcal{G}(v)$  **do**
  - 4:     **if**  $\mathcal{C}_{i,j,wt(uv)} \notin \mathcal{C}$  **then**
  - 5:         create the  $\ell_0$ -sampler  $\mathcal{C}_{i,j,wt(uv)}$
  - 6:     feed  $\langle [u, v], op \rangle$  to the  $\ell_0$ -sampler algorithm  $\mathcal{C}_{i,j,wt(u,v)}$  with parameter  $\delta$
- 

**$\mathcal{A}_{dynamic}$ -Query: The query algorithm after an update**

---

- 1: let  $E' = \emptyset$
  - 2: **for** each  $\mathcal{C}_{i,j,w} \in \mathcal{C}$  **do**
  - 3:     apply the  $\ell_0$ -sampler  $\mathcal{C}_{i,j,w}$  with parameter  $\delta$  to sample an edge  $e$
  - 4:     **if**  $\mathcal{C}_{i,j,w}$  does not FAIL **then** set  $E' = E' \cup \{e\}$
  - 5: return a maximum weighted  $k$ -matching in  $G' = (V(E'), E')$  if any; otherwise, return  $\emptyset$
- 

Choose  $\delta = \frac{1}{20k^4 \ln(2k)}$ . Let  $\mathcal{A}_{dynamic}$  be the algorithm consisting of the sequence of three subroutines/algorithms  **$\mathcal{A}_{dynamic}$ -Preprocess**,  **$\mathcal{A}_{dynamic}$ -Update**, and  **$\mathcal{A}_{dynamic}$ -Query**, where  **$\mathcal{A}_{dynamic}$ -Preprocess** is applied at the beginning of the stream,  **$\mathcal{A}_{dynamic}$ -Update** is applied after each operation, and  **$\mathcal{A}_{dynamic}$ -Query** is applied whenever the algorithm is queried for a solution after some update operation. Without loss of generality, and for convenience, we will assume that the algorithm is queried at the end of the stream  $\mathcal{S}$ , even though the query could take place after any arbitrary operation.

► **Lemma 11.** *Let  $M'$  be the matching obtained by applying the algorithm  $\mathcal{A}_{dynamic}$  with  **$\mathcal{A}_{dynamic}$ -Query** invoked at the end of  $\mathcal{S}$ . If  $G$  contains a  $k$ -matching then, with probability at least  $1 - \frac{11}{20k^3 \ln(2k)}$ ,  $M'$  is a maximum weighted  $k$ -matching of  $G$ .*

► **Theorem 12.** *The algorithm  $\mathcal{A}_{dynamic}$  outputs a matching  $M'$  such that (1) if  $G$  contains a  $k$ -matching then, with probability at least  $1 - \frac{11}{20k^3 \ln(2k)}$ ,  $M'$  is a maximum weighted  $k$ -matching of  $G$ ; and (2) if  $G$  does not contain a  $k$ -matching then  $M' = \emptyset$ . Moreover, the algorithm  $\mathcal{A}_{dynamic}$  runs in  $\tilde{O}(Wk^2)$  space and has  $\tilde{O}(1)$  update time.*

**Proof.** First, observe that  $G'$  is a subgraph of  $G$ , since it consists of edges sampled from subsets of edges in  $G$ . Therefore, statement (2) in the theorem clearly holds true. Statement (1) follows from Lemma 11. Next, we analyze the update time of algorithm  $\mathcal{A}_{dynamic}$ .

From **Algorithm 1** and **Algorithm 2**, we have  $d_1 = O(\frac{k}{\ln k})$ ,  $d_2 = O(\ln k)$ ,  $d_3 = O(\ln^2 k)$  and  $|F_i| = O(\ln k)$  for  $i \in [d_1]^-$ . Thus,  $|\mathcal{G}(u)| = O(\ln k)$  holds for all  $u \in V$ . For the update time, it suffices to examine Steps 1–6 of  **$\mathcal{A}_{dynamic}$ -Update** By Theorem 10, Steps 1–2 take time polynomial in  $\log n$ , which is  $\tilde{O}(1)$ . For Step 4, we can index  $\mathcal{C}$  using a sorted sequence of triplets  $(i, j, w)$ , where  $i, j \in [d_1 \cdot d_2 \cdot d_3]^-$  and  $w$  ranges over all possible weights. Since  $d_1 = O(\frac{k}{\ln k})$ ,  $d_2 = O(\ln k)$  and  $d_3 = O(\ln^2 k)$ , we have  $|\mathcal{C}| = O((d_1 \cdot d_2 \cdot d_3)^2 \cdot W) = O(Wk^2 \ln^4 k) = \tilde{O}(Wk^2)$ . Using binary search on  $\mathcal{C}$ , one execution of Step 4 takes time  $O(\log W + \log k)$ . Since  $|\mathcal{G}(u)| = O(\ln k)$  for every  $u \in V$ , and since by Lemma 1 an  $\ell_0$ -sampler algorithm has  $\tilde{O}(1)$  update time, Steps 3–6 take time  $O(\ln^2 k) \cdot (O(\log W + \log k) + \tilde{O}(1)) = \tilde{O}(1)$ . Therefore, the overall update time is  $\tilde{O}(1)$ .

Now, we analyze the space complexity of the algorithm. First, consider  **$\mathcal{A}_{dynamic}$ -Preprocess**. Obviously, Step 1 uses  $O(1)$  space. Steps 1–2 use space  $O(k + (\log k)(\log n))$  (including the space used to store  $\{f, F_0, \dots, F_{d_1-1}, \mathcal{C}\}$ ) by Theorem 10. Altogether,  **$\mathcal{A}_{dynamic}$ -Preprocess** runs in space  $O(k + (\log k)(\log n))$ . Next, we discuss  **$\mathcal{A}_{dynamic}$ -Update**. Steps 1–2 take space  $O(\ln k)$  by Theorem 10. Observe that the space used in Steps 3–6 is dominated by the space used by the set  $\mathcal{C}$  of  $\ell_0$ -samplers. Since  $\delta = \frac{1}{20k^4 \ln(2k)}$ , an  $\ell_0$ -sampler algorithm uses space  $O(\log^2 n \cdot \log k)$ , by Lemma 1. Since  $|\mathcal{C}| = O(Wk^2 \ln^4 k)$ , Steps 3–6 use space  $O(Wk^2 \log^2 n \log^5 k) = \tilde{O}(Wk^2)$ . Finally, consider  **$\mathcal{A}_{dynamic}$ -Query** The space in Steps 1–4 is dominated by the space used by  $\mathcal{C}$  and the space needed to store the graph  $G'$ , and hence  $E'$ . By the above discussion,  $\mathcal{C}$  takes space  $\tilde{O}(Wk^2)$ . Since at most one edge is sampled from each  $\ell_0$ -sampler instance and  $|\mathcal{C}| = O(Wk^2 \ln^4 k)$ , we have  $|E'| = |\mathcal{C}| = \tilde{O}(Wk^2)$ . Step 5 utilizes space  $O(|E'|)$  [17, 18]. Therefore,  **$\mathcal{A}_{dynamic}$ -Query** runs in space  $\tilde{O}(Wk^2)$ . It follows that the space complexity of  $\mathcal{A}_{dynamic}$  is  $\tilde{O}(Wk^2)$ . ◀

Using Theorem 12, and following the same approach in [7], we obtain the following:

► **Theorem 13.** *Let  $0 < \epsilon < 1$ . There exists an algorithm for  $p$ -WT-MATCHING that computes a matching  $M'$  such that (1) if  $G$  contains a maximum weighted  $k$ -matching  $M$ , then with probability at least  $1 - \frac{11}{20k^3 \ln(2k)}$ ,  $wt(M') > (1 - \epsilon)wt(M)$ ; and (2) if  $G$  does not contain a  $k$ -matching then  $M' = \emptyset$ . Moreover, the algorithm runs in  $\tilde{O}(k^2 \epsilon^{-1} \log W')$  space and has  $\tilde{O}(1)$  update time, where  $W'$  is the ratio of the max weight to min weight.*

**Proof.** For each edge  $e \in E$ , round  $wt(e)$  and assign it a new weight of  $(1 + \epsilon)^i$  such that  $(1 + \epsilon)^{i-1} < wt(e) \leq (1 + \epsilon)^i$ . Thus, there are  $O(\epsilon^{-1} \log W')$  distinct weights after rounding. By Theorem 12, the space and update time are  $\tilde{O}(k^2 \epsilon^{-1} \log W')$  and  $\tilde{O}(1)$  respectively, and the success probability is at least  $1 - \frac{11}{20k^3 \ln(2k)}$ . Now we prove that  $wt(M') > (1 - \epsilon)wt(M)$ .

Let  $e \in M$  and let  $e'$  be the edge sampled from the  $\ell_0$ -sampler that  $e$  is fed to. It suffices to prove that  $wt(e') > (1 - \epsilon)wt(e)$ . Assume that  $wt(e)$  is rounded to  $(1 + \epsilon)^i$ . Then,  $wt(e')$  is rounded to  $(1 + \epsilon)^j$  as well. If  $wt(e') \geq wt(e)$ , we are done; otherwise,  $(1 + \epsilon)^{i-1} < wt(e') < wt(e) \leq (1 + \epsilon)^i$ . It follows that  $wt(e') > (1 + \epsilon)^{i-1} \geq wt(e)/(1 + \epsilon) > (1 - \epsilon)wt(e)$ . ◀

The following theorem is a consequence of Theorem 12 (applied with  $W = 1$ ):

► **Theorem 14.** *There is an algorithm for  $p$ -MATCHING that computes a matching  $M'$  satisfying that (1) if  $G$  contains a  $k$ -matching then, with probability at least  $1 - \frac{11}{20k^3 \ln(2k)}$ ,  $M'$  is a  $k$ -matching of  $G$ ; and (2) if  $G$  does not contain a  $k$ -matching then  $M' = \emptyset$ . Moreover, the algorithm runs in  $\tilde{O}(k^2)$  space and has  $\tilde{O}(1)$  update time.*



## 6 Lower Bound

In this section, we discuss lower bounds on the space complexity of randomized streaming algorithms for  $p$ -MATCHING in the insert-only model (hence also in the dynamic model), and for  $p$ -WT-MATCHING in the dynamic model. These lower bound results, in conjunction with the algorithms given in the previous sections, show that the space complexity achieved by our algorithms is optimal (modulo a poly-logarithmic factor in the input size).

We will use the *one-way communication model* to prove lower bounds on the space complexity of randomized streaming algorithms for  $p$ -MATCHING and  $p$ -WT-MATCHING. In this model, there are two parties, Alice and Bob, each receiving  $x$  and  $y$  respectively, who wish to compute  $f(x, y)$ . Alice is permitted to send Bob a single message  $M$ , which only depends on  $x$  and Alice's random coins. Then Bob outputs  $b$ , which is his guess of  $f(x, y)$ . Here,  $b$  only depends on  $y, M$ , and Bob's random coins. We say the protocol computing  $f$  with success probability  $1 - \delta$  if  $\Pr(b = f(x, y)) \geq 1 - \delta$  for every  $x$  and  $y$ .

For  $p$ -MATCHING, we have the following theorem, which is implied from the space complexity lower-bound proof given in [7] for maximum matching.

► **Theorem 15** ([7]). *Any randomized streaming algorithm for  $p$ -MATCHING that, with probability at least  $2/3$ , computing a  $k$ -matching uses  $\Omega(k^2)$  bits.*

For  $p$ -WT-MATCHING, we start by defining the following problem:

PARTIAL MAXIMIZATION: Alice has a sequence  $a = \langle a_1, a_2, \dots, a_n \rangle$  of numbers, where each  $a_i \in [1, n^{1+\epsilon}]$ , and Bob has a subset  $T \subset [n]$ . Compute  $\max_{i \in [n] \setminus T} a_i$ .

Let  $X, Y, Z, Z_1, Z_2$  be random variables. Define the *Shannon entropy* of  $X$  as  $H(X) = \sum_x \Pr(X = x) \log\left(\frac{1}{\Pr(X=x)}\right)$ . Define the *conditional entropy* of  $Z_1$  given  $Z$  as  $H(Z_1 | Z) = \sum_z H(Z_1 | Z = z) \Pr(Z = z)$ , and the *mutual information* as  $I(Z_1; Z) = H(Z) - H(Z | Z_1)$ . Define the *conditional mutual information* of  $Z_1, Z_2$  given  $Z$  as  $I(Z_1; Z_2 | Z) = H(Z_1 | Z) - H(Z_1 | Z_2, Z)$ .  $X \rightarrow Y \rightarrow Z$  is said to form a *Markov chain* if the conditional distribution of  $Z$  depends only on  $Y$  and is conditionally independent of  $X$ .

► **Theorem 16.** *For any constant  $0 \leq \delta < 1$ , any randomized one-way communication protocol for PARTIAL MAXIMIZATION with success probability at least  $1 - \delta$  has communication complexity  $\Omega(n \log n)$  bits.*

**Proof.** The proof has a similar fashion as Augmented Indexing problem [29, 10]. Consider the case where each  $X_j$ , for  $j \in [n]$ , is picked uniformly at random from  $[(j-1) \cdot n^\epsilon + 1, j \cdot n^\epsilon]$ . Note that  $X_1 < X_2 < \dots < X_n$  and that  $H(X_j) = \epsilon \log n$  for each  $j \in [n]$ . For each  $j \in [n]$ , let  $T_j = [j+1, n]$  and let  $X'_j$  be Bob's guess of  $\max_{i \in [n] \setminus T_j} X_i = X_j$ . Let  $M$  be the message sent from Alice to Bob. Since  $\Pr(X'_j = X_j) \geq 1 - \delta$  and  $X_j \rightarrow (M, X_{j+1}, X_{j+2}, \dots, X_n) \rightarrow X'_j$  is a Markov chain, by Fano's Inequality [13], for all  $j \in [n]$ , we have  $H(X_j | M, X_{j+1}, \dots, X_n) \leq \delta \cdot \epsilon \log n + 1$ , and hence,

$$\begin{aligned} I(X_j; M | X_{j+1}, \dots, X_n) &= H(X_j | X_{j+1}, \dots, X_n) - H(X_j | M, X_{j+1}, \dots, X_n) \\ &= H(X_j) - H(X_j | M, X_{j+1}, \dots, X_n) \\ &\geq (1 - \delta)\epsilon \log n - 1, \end{aligned}$$

where the second equality holds because  $X_j, X_{j+1}, \dots, X_n$  are mutually independent. By Theorem 2.5.2 of [13],  $H(M) \geq I(X_1, X_2, \dots, X_n; M) = \sum_{j=1}^n I(X_j; M | X_{j+1}, \dots, X_n) = \Omega(n \log n)$ . Finally, by Theorem 2.6.4 of [13] the message  $M$  has at least  $2^{\Omega(n \log n)}$  possibilities, hence the length of the longest possible  $M$  is  $\Omega(n \log n)$ , completing the proof. ◀



Theorem 16, plus a reduction from the PARTIAL MAXIMIZATION problem to the p-WT-MATCHING problem with parameter value  $k = 1$ , gives directly the following result.

► **Theorem 17.** *Any randomized streaming algorithm for p-WT-MATCHING that has success probability at least  $2/3$  requires space  $k^2 \cdot \Omega(W(\log W + 1))$ .*

## 7 Concluding Remarks

In this paper, we presented streaming algorithms for the fundamental  $k$ -matching problem, for both unweighted and weighted graphs, and in both the insert-only and dynamic streaming models. While matching the best space complexity of known algorithms, which has been proved to be either optimal or near-optimal, our algorithms have much faster update times. For the insert-only model, our algorithm is optimal in both space and update time complexities. For the dynamic model, according to the new lower bounds we developed, our algorithms are near-optimal (i.e., optimal up to a poly-logarithmic factor) in both space and update time complexities. Our result for the weighted  $k$ -matching problem was achieved using a newly-developed structural result that is of independent interest. We believe that our results and techniques can have wider applicability for other fundamental graph problems.

Most work on weighted graph streams, including ours, assumes that the weight of an edge is unchanged in the stream [1, 2, 7, 20, 23]. We give an interesting observation below to justify this assumption and show that, if this assumption is lifted, then the space complexity of the  $k$ -matching problem can be much larger than the desirable space complexity for streaming algorithms. This lower bound is derived by a reduction from the following problem:

Given a data stream  $\mathcal{S}' = x_1, x_2, \dots, x_m$ , where  $x_i \in \{1, \dots, n'\}$ , let  $c_i = |\{j \mid x_j = i\}|$  be the number of occurrences of  $i$  in the stream  $\mathcal{S}'$ . Compute  $F_\infty = \max_{1 \leq i \leq n'} c_i$ .

► **Theorem 18** ([32]). *For data streams of length  $m$ , any randomized streaming algorithm computing  $F_\infty$  to within a  $(1 \pm 0.2)$  factor with probability  $2/3$  requires space  $\Omega(\min\{m, n'\})$ .*

Now we consider the p-WT-MATCHING problem in the more generalized dynamic streaming model, in which an instance of p-WT-MATCHING is given by a parameter  $k$  and a stream  $\mathcal{S} = (e_{i_1}, \Delta_1(e_{i_1})), \dots, (e_{i_j}, \Delta_j(e_{i_j})), \dots$  of updates of edge weights in the underlying graph  $G$ , where the update  $(e_{i_j}, \Delta_j(e_{i_j}))$  changes the current weight  $wt(e_{i_j})$  of edge  $e_{i_j}$  to  $wt(e_{i_j}) = wt(e_{i_j}) + \Delta_j(e_{i_j})$ , assuming  $wt(\cdot) = 0$  initially and  $wt(\cdot) \geq 0$  for all updates. This model generalizes the dynamic graph streaming model in [7].

► **Theorem 19.** *Under the more generalized dynamic streaming model, any randomized streaming algorithm that, with probability at least  $2/3$ , approximates the maximum weighted 1-matching of the graph to a factor of  $6/5$  uses space  $\Omega(\min\{m, \frac{(n-1)(n-2)}{2}\})$ .*

**Proof.** Given a data stream  $\mathcal{S}' = x_1, x_2, \dots, x_m$ , where each  $x_i \in \{1, \dots, n'\}$ , we define a graph stream  $\mathcal{S}$  for a weighted graph  $G$  on  $n$  vertices, where  $n$  satisfies  $(n-1)(n-2)/2 < n' \leq n(n-1)/2$ . Let  $V = \{0, \dots, n-1\}$  be the vertex-set of  $G$ . We first define a bijective function  $\chi : \{(i, j) \mid i < j \in [n]^-\} \rightarrow [\frac{n(n-1)}{2}]$ . Let  $\chi^{-1}$  be the inverse function of  $\chi$ . Then, we can translate  $\mathcal{S}'$  to a general dynamic graph streaming  $\mathcal{S}$  of underlying weighted graph  $G$  by corresponding with  $x_i$  the  $i$ -th element  $(\chi^{-1}(x_i), 1)$  of  $\mathcal{S}$ , for  $i \in [m]$ . Observe that computing  $F_\infty$  of  $\mathcal{S}'$  is equivalent to computing a maximum weighted 1-matching for the graph stream  $\mathcal{S}$  of  $G$ . Let  $uv$  be a maximum weighted 1-matching of  $\mathcal{S}$ , then  $\chi(uv)$  is  $F_\infty$  of  $\mathcal{S}'$ . By Theorem 18, it follows that any randomized approximation streaming algorithm that approximates the maximum weighted 1-matching of  $G$  to a  $\frac{6}{5}$ -factor with probability at least  $2/3$  uses space  $\Omega(\{m, \frac{(n-1)(n-2)}{2}\})$ , thus completing the proof. ◀

## References

- 1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems (PODS '12)*, pages 5–14, 2012.
- 2 KookJin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. In *Proceedings of the 32nd International Conference on Machine Learning (ICML '15)*, pages 2237–2246, 2015.
- 3 J. Alman and H. Yu. Faster update time for turnstile streaming algorithms. In *Proceedings of the 31st annual ACM-SIAM symposium on Discrete algorithms (SODA '20)*, pages 1803–1813, 2020.
- 4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 1723–1742, 2017.
- 5 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*, pages 1345–1364, 2016.
- 6 Jianer Chen, Ying Guo, and Qin Huang. Linear-time parameterized algorithms with limited local resources. *arXiv preprint*, 2020. [arXiv:2003.02866](https://arxiv.org/abs/2003.02866).
- 7 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the 27th annual ACM-SIAM symposium on Discrete algorithms (SODA '16)*, pages 1326–1344, 2016.
- 8 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. New streaming algorithms for parameterized maximal matching and beyond. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '15)*, pages 56–58, 2015.
- 9 Rajesh Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the 2015 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '15)*, pages 1234–1251, 2015.
- 10 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214. ACM, 2009.
- 11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
- 12 Graham Cormode and Donatella Firmani. A unifying framework for  $\ell_0$ -sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.
- 13 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006.
- 14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 15 Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, 2005.
- 16 Stefan Fafanie and Stefan Kratsch. Streaming kernelization. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science 2014 (MFCS '14)*, pages 275–286, 2014.
- 17 Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 434–443. SIAM, 1990.
- 18 Harold N. Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms*, 14(3), 2018.

- 19 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '12)*, pages 468–485, 2012.
- 20 Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *arXiv preprint*, 2012. [arXiv:1203.4900](https://arxiv.org/abs/1203.4900).
- 21 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '13)*, pages 1679–1697, 2013.
- 22 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '14)*, pages 734–751, 2014.
- 23 Michael Kapralov and David Woodruff. Spanners and sparsifiers in dynamic streams. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC '14)*, pages 272–281, 2014.
- 24 Christian Konrad and Adi Rosén. Approximating semi-matchings in streaming and in two-party communication. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP '13)*, pages 637–649, 2013.
- 25 K. Larsen, J. Nelson, and H. Nguyen. Time lower bounds for nonadaptive turnstile streaming algorithms. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing STOC'15*, pages 803–812. ACM, 2015.
- 26 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. *arXiv preprint*, to appear in *SODA '21*, 2020. [arXiv:2008.10062](https://arxiv.org/abs/2008.10062).
- 27 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. A linear-time algorithm for maximum-cardinality matching on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2820–2835, 2018.
- 28 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. *Algorithmica*, 82(12):3521–3565, 2020.
- 29 Peter B. Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.
- 30 Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, 2nd edition, 2017.
- 31 Ami Paz and Gregory Schwartzman. A  $(2+\epsilon)$ -approximation for maximum weight matching in the semi-streaming model. *ACM Transaction on Algorithms*, 15(2):18:1–18:15, 2019.
- 32 Tim Roughgarden. Communication complexity (for algorithm designers). *arXiv preprint*, 2015. [arXiv:1509.06257](https://arxiv.org/abs/1509.06257).
- 33 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.
- 34 M. Thorup and Y. Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.
- 35 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.



# Making Three out of Two: Three-Way Online Correlated Selection

Yongho Shin ✉

Department of Computer Science, Yonsei University, Seoul, South Korea

Hyung-Chan An<sup>1</sup> ✉ 

Department of Computer Science, Yonsei University, Seoul, South Korea

---

## Abstract

*Two-way online correlated selection (two-way OCS)* is an online algorithm that, at each timestep, takes a pair of elements from the ground set and irrevocably chooses one of the two elements, while ensuring negative correlation in the algorithm's choices. Whilst OCS was initially invented by Fahrback, Huang, Tao, and Zadimoghaddam to break a natural long-standing barrier in the edge-weighted online bipartite matching problem, it is an interesting technique on its own due to its capability of introducing a powerful algorithmic tool, namely negative correlation, to online algorithms. As such, Fahrback et al. posed two tantalizing open questions in their paper, one of which was the following: Can we obtain *n-way OCS* for  $n > 2$ , in which the algorithm can be given  $n > 2$  elements to choose from at each timestep?

In this paper, we affirmatively answer this open question by presenting a *three-way OCS*. Our algorithm uses two-way OCS as its building block and is simple to describe; however, as it internally runs two instances of two-way OCS, one of which is fed with the output of the other, the final output probability distribution becomes highly elusive. We tackle this difficulty by approximating the output distribution of OCS by a flat, less correlated function and using it as a safe “surrogate” of the real distribution. Our three-way OCS also yields a 0.5093-competitive algorithm for edge-weighted online matching, demonstrating its usefulness.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** online correlated selection, multi-way OCS, online algorithms, negative correlation, edge-weighted online bipartite matching

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.49

**Funding** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1C1C1008934).

**Acknowledgements** We thank the anonymous reviewers for their helpful comments.

## 1 Introduction

*Online correlated selection (OCS)* is an online algorithm that, at each timestep, takes a subset of the ground set as the input and irrevocably chooses a single element from the subset. When every input subset has cardinality  $n$ , we call it *n-way OCS* in particular. The aim of online correlated selection is to ensure a certain level of negative correlation in the choice made by the algorithm. For example, suppose we run a two-way OCS and afterwards specify  $m$  timesteps that contained some common element. The probability that this element was never chosen would be  $(1/2)^m$  if the algorithm made independent and uniformly random choices; the goal of two-way OCS is to reduce this probability by introducing negative correlations. (See Definition 4 for a full definition that quantifies the desired amount of reduction.)

---

<sup>1</sup> Corresponding author. Department of Computer Science, Yonsei University, 50 Yonsei-ro, Seodaemun-gu, Seoul 03722, South Korea.



OCS was first invented by Fahrbach, Huang, Tao, and Zadimoghaddam [10] to attack the edge-weighted online bipartite matching problem. Negative correlation has proven to be a very powerful technique in approximation algorithms design (see, e.g., [34, 2, 7] for a limited list of examples); this suggests that OCS as well bears high potential as a general tool in online algorithms design rather than as a specialized technique to solve a particular problem. This opportunity was also observed by the breakthrough paper of Fahrbach et al. [10] and recently exemplified by Huang, Zhang, and Zhang [23], who devised a certain variant of OCS called *panoramic OCS* to solve the AdWords problem with general bids.

In light of such value of OCS as an algorithmic tool, Fahrbach et al. [10] raised in their paper two follow-up questions that arise quite naturally: Can we improve the performance of their two-way OCS? Can we obtain an  $n$ -way OCS for  $n > 2$ ? This paper affirmatively answers the latter question. In this paper, we present a simple *three-way OCS* and analyze its performance. We will also show that our three-way OCS can be used to improve the previous competitive ratio of 0.5086 due to Fahrbach et al. to give a new 0.5093-competitive algorithm for edge-weighted online bipartite matching.

In fact, the construction itself of our three-way OCS is easy to describe. It internally executes two instances of two-way OCS. Upon arrival of a triple, we choose two of the three elements uniformly at random, and let the first two-way OCS choose one of them. We then pass its output, along with the element that was left out of the first OCS, to the second OCS. The second OCS chooses one of these two elements; this choice becomes the final output of this timestep.

In Section 3, we analyze the performance of our three-way OCS for a special case first: we bound the probability that a certain element, say  $u$ , is never chosen for  $k$  consecutive timesteps whose triples contain  $u$ . It is not that we require these  $k$  timesteps to be consecutive in the original input: they need to be consecutive in the subsequence of timesteps on which  $u$  appeared in the triple. By the definition of two-way  $\gamma$ -OCS, the probability that the second two-way OCS never chooses  $u$  is no greater than  $(1/2)^j(1 - \gamma)^{\max(j-1, 0)}$  for some constant  $\gamma$ , where  $j$  is the number of times  $u$  was passed to the second OCS during those  $k$  timesteps. Since this bound depends only on  $j$ , the question really reduces to determining (the probability distribution of)  $j$ .

In order for  $u$  to be passed to the second OCS, it needs to be either left out of the first OCS or output by it. It is easy to count how many times  $u$  is left out of the first OCS: this follows a binomial distribution. Therefore, the challenge is in counting the number of times  $u$  is output by the first OCS. Unfortunately, its probability distribution highly depends on the actual *input* to the first OCS, rather than the number of times  $u$  is shown to the first OCS. Nonetheless, the following observation is crucial in coping with this difficulty: the probability distribution of the number of times Fahrbach et al.'s two-way OCS chooses  $u$  is a unimodal symmetric distribution. Recall that the probability that the second OCS never chooses  $u$  is bounded by  $(1/2)^j(1 - \gamma)^{\max(j-1, 0)}$ , which is “nearly” convex. Therefore, even though we cannot exactly calculate the probability distribution of  $j$  without the full knowledge of the input, the above observation implies that we can instead use a “flatter” unimodal symmetric distribution in lieu of the actual distribution of  $j$ . Thanks to the near-convexity, this would give a valid upper bound on the probability. We formalize what a “flatter” distribution is by defining the notion of *central dominance* as follows.

► **Definition 1 (Central Dominance).** *Given two discrete symmetric probability distributions  $D_1$  and  $D_2$  on  $\{0, 1, \dots, x\}$  whose probability mass functions are  $p_1$  and  $p_2$ , respectively, we say  $D_1$  centrally dominates  $D_2$  if there exists  $z \in [0, \frac{x}{2}]$  such that, for any integer  $y \in [\frac{x}{2} - z, \frac{x}{2} + z]$ ,  $p_1(y) \geq p_2(y)$ , and for any integer  $y \in [0, \frac{x}{2} - z) \cup (\frac{x}{2} + z, x]$ ,  $p_1(y) \leq p_2(y)$ .*

We then construct our “surrogate” distribution that is centrally dominated by any possible probability distribution of  $j$ . This distribution depends only on the number of times the first OCS is given  $u$ . It is therefore much more amenable and allows us to obtain a bound on the probability that our three-way OCS never chooses  $u$  from the given consecutive triples.

In Section 4, we generalize this bound to the non-consecutive case, i.e., a disjoint set of consecutive subsequences of timesteps containing  $u$ . To obtain this bound, we perform a set of surgical operations that modify the input to the first OCS, which are designed to reduce negative correlation. These operations are inspired by those of Fahrbach et al. [10] that they used to obtain a similar generalization. In our case, however, we face a new obstacle: previously, it sufficed to bound only the probability that the two-way OCS never chooses a given element, since the output of that OCS was the final output. Our final output on the other hand is determined by the second OCS, and if our modification changes the output distribution of the first OCS, this may affect the output of the second OCS in an obscure way. We prove that a set of careful surgical operations can remove all correlations while ensuring that the bound is not affected. Once the correlations are removed, the general-case bound can be simply given as the product of our bounds from Section 3 for single subsequences.

► **Theorem 2** (simplified). *Consider a set of  $m$  disjoint consecutive subsequences of timesteps whose triples contain some element  $u$  of the ground set. Let  $k_1, \dots, k_m$  be the lengths of these subsequences. The probability that our three-way OCS never chooses  $u$  from these  $m$  subsequences is at most*

$$\prod_{i=1}^m \left[ \left( \frac{2}{3} \right)^{k_i} (1 - \delta_1)^{\max(k_i - 1, 0)} (1 - \delta_2)^{\max(k_i - 2, 0)} \right],$$

where  $\delta_1 = 0.0309587$  and  $\delta_2 = 0.0165525$ .

Our three-way OCS can be applied to edge-weighted and unweighted online bipartite matching [33].

## 1.1 Related Work

Introduced by Karp, Vazirani, and Vazirani [25], the unweighted online bipartite matching has been intensively and extensively studied with alternative proofs [16, 4, 8, 9] and under various settings including stochastic models [12, 29, 17, 32, 13, 20], fully online models [18, 19, 22], and general arrival models [14]. The study of edge-weighted online bipartite matching problem was initiated by Kalyanasundaram & Pruhs [24] and Khuller, Mitchell, & Vazirani [27], who independently considered this problem under the metric assumption. Feldman, Korula, Mirrokni, Muthukrishnan, and Pál [11] first investigated the edge-weighted version on arbitrary weights with free disposal, and more thorough understanding of this problem was achieved by subsequent work [28, 10]. Other variants and applications of online bipartite matching have been studied as well, including AdWords [31, 23], vertex-weighted version [1, 21, 13], stochastic or random arrival models [17, 26, 6, 20], and the windowed version [3]. We refer interested readers to the survey of Mehta [30].

**Recent related works.** Recently, after the authors independently obtained the present results but prior to announcing them, the authors learned that two closely related papers were announced on arXiv [15, 5].

The other open question raised by Fahrbach et al. [10] than the one answered by this paper was to improve two-way OCS. Gao, He, Huang, Nie, Yuan, and Zhong [15] addresses this question by giving a novel automata-based OCS, successfully departing from the previous



matching-based approach: their two-way OCS is a 0.167-OCS. In addition to this, they also give an improved primal-dual analysis and a variant of two-way OCS specifically adapted for edge-weighted online bipartite matching, yielding a 0.519-competitive algorithm. Finally, they consider a weaker relaxed notion of OCS called *semi-OCS*, where the probability bound holds only for consecutive prefixes. They provide a multi-way version of this semi-OCS that leads to a 0.593-competitive algorithm for unweighted/vertex-weighted online bipartite matching.

Blanc and Charikar [5] generalize Fahrbach et al.'s definition of OCS in two ways and give  $m$ -way OCS for any  $m$ . One of the two generalizations, called  $(F, m)$ -OCS, gives the probability bound specified as a discrete function rather than a fixed-form formula such as  $(1/2)^j(1 - \gamma)^{\max(j-1, 0)}$ . The other is called *continuous OCS*, which allows an element of the ground set to appear in a subset “to a fraction”, where the probability bound is now specified as a continuous function. Their continuous OCS along with their improved primal-dual analysis gives a 0.5368-competitive algorithm for edge-weighted online bipartite matching.

Considering these new results, an interesting question is whether the techniques from these papers and our independent result can together bring improvements in OCS or related problems such as online matching. In fact, since our framework treats the second two-way OCS as a black-box, any improved two-way OCS can be directly plugged into our three-way OCS. Combining the new two-way 0.167-OCS of Gao et al. [15] with our results, for example, immediately yields a 0.5132-competitive algorithm for edge-weighted online bipartite matching.

## 2 Preliminaries

In this section, we present some notation, definitions, and previous results to be used throughout this paper. Let us first introduce the definition of  $n$ -way OCS and two-way  $\gamma$ -OCS.

► **Definition 3** ( $n$ -way OCS). *Given a ground set, an  $n$ -way OCS is an online algorithm that, at each iteration, takes a subset of size  $n$  as the input and irrevocably chooses an element from the subset.*

For any element  $u$  of the ground set, we say a subsequence of subsets containing  $u$  is *consecutive* if every subset containing  $u$  that arrives between the first and last subsets of the subsequence is also in the subsequence. We also say that two or more subsequences of subsets containing  $u$  are *disjoint* if no two of these subsequences share a subset.

For example, suppose we are given  $\{u, v\}$ ,  $\{u, w\}$ ,  $\{v, w\}$ ,  $\{u, x\}$ , and  $\{u, y\}$  during five timesteps, in that order. Observe that  $(\{u, v\}, \{u, w\}, \{u, x\})$  is a consecutive subsequence containing  $u$ , but  $(\{u, v\}, \{u, x\}, \{u, y\})$  is not because  $\{u, w\}$  appears between  $\{u, v\}$  and  $\{u, x\}$ . Note also that  $(\{u, v\}, \{u, w\})$  and  $(\{u, y\})$  constitute a set of (two) disjoint consecutive subsequences containing  $u$ .

► **Definition 4** (Two-way  $\gamma$ -OCS). *A two-way  $\gamma$ -OCS is a two-way OCS such that, for any element  $u$  and a set of  $m$  disjoint consecutive subsequences of pairs containing  $u$  of lengths  $k_1, \dots, k_m$ , the probability that  $u$  never gets chosen by the OCS from any of the given subsequences is at most  $\prod_{i=1}^m (\frac{1}{2})^{k_i} (1 - \gamma)^{\max(k_i-1, 0)}$ .*

Fahrbach et al. [10] presents two versions of two-way OCS with varying performance guarantees. We will use both in later sections. For the sake of completeness, we present a full description of the two-way 1/16-OCS of Fahrbach et al. below.

**1/16-OCS.** We will define what is called the *ex-ante graph* first. Although the algorithm does not need to explicitly construct this graph, it helps simplify the presentation. The vertices of the ex-ante graph correspond to the input pairs. For each pair, say  $\{u, v\}$ , we introduce an edge between this pair and the immediately following pair that contains  $u$ , and likewise for  $v$ . For example, if an element  $u$  appears at timesteps 2, 4, and 7, pairs 2 and 4 are adjacent, and so are 4 and 7. This implies that every vertex has degree of at most 4. We annotate each edge with the common element that caused this edge. For example, if two pairs  $i = \{u, v\}$  and  $j = \{v, w\}$  are adjacent due to  $v$ , let  $(i, j)_v$  denote the edge (and its annotation). Note that the ex-ante graph may have parallel edges (with different annotations).

The algorithm samples exactly three random bits for each pair (or vertex). We will use them to define what is called the *ex-post graph* and further determine the output of the algorithm. The ex-post graph is on the same set of vertices. The first random bit is used to determine if the vertex becomes a “sender” or a “receiver”. If it becomes a sender, we use the second random bit to choose one of the two elements of the pair, and the sender will “want” to select the edge to the immediately following pair that shares the chosen element (if it exists). Similarly, if a vertex becomes a receiver, we use the second random bit to choose one element, and the receiver will “want” to select the edge to the immediately preceding pair that shares the element. Each edge of the ex-ante graph enters the ex-post graph if and only if both of its endpoints want to select the edge. Observe that the ex-post graph is a matching in the ex-ante graph.

Finally, the output of the algorithm is determined as follows. For every unmatched vertex, its output is determined solely by its third random bit. For each edge in the ex-post graph, we negatively correlate the choice of the two endpoints: we use the third random bit of the sender to determine the output of the sender, and the output of the receiver is determined so that the decision made for the shared element is the opposite. For example, if we have an edge annotated with  $u$  in the ex-post graph and the sender did not choose  $u$ , we choose  $u$  for the receiver; otherwise, we do not choose  $u$  for the receiver. The third random bit of the receiver is discarded.

► **Lemma 5** (Fahrbach et al. [10]). *This algorithm is a two-way  $\frac{1}{16}$ -OCS.*

This algorithm also has the following useful property.

► **Lemma 6** (Fahrbach et al. [10]). *Given a consecutive subsequence of pairs containing an element of length  $k$  input to this algorithm, the probability that there does not exist any edges in the subgraph of the ex-post graph induced by the consecutive pairs is at most  $(1 - 1/16)^{\max(k-1, 0)}$ .*

The other version is a  $\frac{13\sqrt{13}-35}{108}$ -OCS.

► **Lemma 7** (Fahrbach et al. [10]). *There exists a two-way  $\frac{13\sqrt{13}-35}{108}$ -OCS.*

In the *unweighted online bipartite matching problem*, we are given a bipartite graph  $G = (L \cup R, E)$  where we only know  $L$  in advance. Each vertex  $v \in R$  arrives one by one, and the edges adjacent with  $v$  are only then revealed. Upon each arrival of  $v$ , we irrevocably decide whether we match  $v$ , and if so, to which exposed adjacent vertex in  $L$  we match. The objective is to find a matching of the maximum size in  $G$ .

In the *edge-weighted online bipartite matching problem with free disposal*, we are given a bipartite graph  $G = (L \cup R, E)$  as well as an edge weight  $w_{uv} \geq 0$  for each  $(u, v) \in E$ , where we only know  $L$  in advance, again. Each vertex  $v \in R$  arrives one at a timestep, and the

edges adjacent with  $v$  and their edge weights are disclosed at that time. Upon each arrival of  $v$ , we decide whether we match  $v$ , and if so, to which adjacent vertex in  $L$  we match. We remark that  $v$  can be matched to a vertex  $u$  that is already matched, after disposing of the edge incident with  $u$  in the current matching (*free disposal*). The objective is to find a maximum-weight matching in  $G$ .

### 3 Three-Way Online Correlated Selection

In this section, we present our three-way OCS and analyze it by considering the special case where we are interested in a single consecutive subsequence of triples containing a common element. We will extend this to the general case in Section 4.

#### 3.1 Algorithm

Let  $\mathcal{A}$  and  $\mathcal{B}$  be two-way OCS algorithms whose random choices are independent. In particular, we choose Fahrback et al.'s 1/16-OCS as  $\mathcal{A}$  and  $\frac{13\sqrt{13}-35}{108}$ -OCS as  $\mathcal{B}$ . Let  $\gamma_{\mathcal{A}} := 1/16$  and  $\gamma_{\mathcal{B}} := \frac{13\sqrt{13}-35}{108}$ . We use the 1/16-OCS as  $\mathcal{A}$  due to its property that, in its ex-post graph, each edge's existence is determined by (the random bits of) its endpoints only.

Upon arrival of a triple, say  $\{u, v, w\}$ , we choose a pair uniformly at random out of  $\{\{u, v\}, \{u, w\}, \{v, w\}\}$ . We refer to this step as the *random pair choice phase*. Without loss of generality, suppose that  $\{u, v\}$  is chosen. We then input  $\{u, v\}$  to  $\mathcal{A}$  and let it choose one element from the pair. Let us say (without loss of generality again) that  $u$  is returned by  $\mathcal{A}$ . Now we let  $\mathcal{B}$  choose one element from  $\{u, w\}$ . The element chosen by  $\mathcal{B}$  is the final output of this iteration.

#### 3.2 Overview of the Analysis

The goal of Section 3 is to prove the following theorem.

► **Theorem 8.** *Consider a consecutive subsequence of timesteps whose triples contain some element  $u$  of the ground set. Let  $k$  be the length of the subsequence. The probability that our three-way OCS never chooses  $u$  during these  $k$  timesteps is at most*

$$\eta(k) := c_1 t_1^k + c_2 t_2^k - c_3 t_3^k - c_4 t_4^k$$

for some  $c_1 \approx 0.957795$ ,  $c_2 \approx 0.176756$ ,  $c_3 \approx 0.011047$ ,  $c_4 \approx 0.131738$ ,  $t_1 \approx 0.630024$ ,  $t_2 \approx 0.599919$ ,  $t_3 \approx 0.148345$ , and  $t_4 = 0.3125$ , which is bounded from above by

$$\left(\frac{2}{3}\right)^k (1 - \delta_1)^{\max(k-1, 0)} (1 - \delta_2)^{\max(k-2, 0)},$$

where  $\delta_1 = 0.0309587$  and  $\delta_2 = 0.0165525$ .

In what follows, we will fix an arbitrary set of random choices made during the random pair choice phase, and condition on them. Let  $x$  be the number of pairs containing  $u$  from the subsequence that are passed to  $\mathcal{A}$ . Observe that these pairs also form a consecutive subsequence to  $\mathcal{A}$ , and that the number of  $u$ 's that are left out of the first OCS is  $k - x$ .

For  $y = 0, \dots, x$ , let  $p(x, y)$  be the (conditional) probability that  $\mathcal{A}$  returns  $y$  number of  $u$ 's from these pairs. Then, we can see that the (conditional) probability that  $u$  never gets chosen from the given consecutive triples is bounded from above by

$$\sum_{y=0}^x p(x, y) \left(\frac{1}{2}\right)^{k-x+y} (1 - \gamma_{\mathcal{B}})^{\max(k-x+y-1, 0)} \quad (1)$$

by Lemma 7 and the fact that the pairs containing  $u$  passed to  $\mathcal{B}$  are also consecutive (see also Definition 4). To simplify the presentation, we introduce two shorthands: for a probability mass function  $p'$  defined on  $\{0, \dots, x\}$ , let

$$\theta(x, p') := \sum_{y=0}^x p'(y) \cdot \left(\frac{1}{2}\right)^y (1 - \gamma_{\mathcal{B}})^{y-1}; \text{ and} \tag{2}$$

$$\theta'(x, p') := \sum_{y=0}^x p'(y) \cdot \left(\frac{1}{2}\right)^y (1 - \gamma_{\mathcal{B}})^{\max(y-1, 0)}. \tag{3}$$

Observe that (1) is equal to  $\left(\frac{1-\gamma_{\mathcal{B}}}{2}\right)^{k-x} \theta(x, p(x, \cdot))$  if  $k > x$  and  $\theta'(k, p(k, \cdot))$  otherwise (if  $k = x$ ).

We would like to bound (1), but unfortunately,  $p(x, y)$  depends on the actual input to  $\mathcal{A}$ , not just  $x$ . Yet, we can circumvent this problem by exploiting the behavior of  $\mathcal{A}$ . Recall that  $\mathcal{A}$  constructs the ex-post graph (which is a matching in the ex-ante graph) and negatively correlate the two endpoints of each edge in the ex-post graph. Therefore, when we consider the subgraph of the ex-post graph induced by the given consecutive pairs, we can observe that

- for each edge in this subgraph, exactly one  $u$  is chosen from its two endpoints, and
- if a vertex is isolated in this subgraph,  $u$  is chosen with probability  $1/2$ , independently from the other vertices.

This observation implies that the probability distribution  $p(x, \cdot)$  is a unimodal symmetric distribution. Moreover, the more likely  $\mathcal{A}$  puts an edge in the ex-post graph, the pointier the distribution would be. Using this property, we will construct an imaginary probability distribution  $\{p^*(x, y)\}_{y=0, \dots, x}$  such that

- $p^*(x, y)$  is flatter than (or, formally, centrally dominated by) any distribution  $p$  that results from  $\mathcal{A}$  (see the proof of Lemma 10); and
- $p^*(x, y)$  only depends on  $x$ , not the input itself.

We will further demonstrate that, in (2) and (3), substituting  $p'$  with a centrally dominated distribution would only overestimate (2) and (3) (Lemma 9). Intuitively speaking, this is because  $(1/2)^y(1 - \gamma_{\mathcal{B}})^{y-1}$  and  $(1/2)^y(1 - \gamma_{\mathcal{B}})^{\max(y-1, 0)}$  are (nearly) convex functions.

Thus, the probability that our three-way OCS never chooses  $u$  from the given consecutive triples can finally be bounded from above by

$$\eta(k) := \sum_{x=0}^k \text{binom}\left(k, x, \frac{2}{3}\right) \left[ \sum_{y=0}^x p^*(x, y) \left(\frac{1}{2}\right)^{k-x+y} (1 - \gamma_{\mathcal{B}})^{\max(k-x+y-1, 0)} \right], \tag{4}$$

where  $\text{binom}(k, x, r)$  represents the probability of the binomial distribution for  $x$  successes out of  $k$  trials with probability  $r$  (Lemma 11).

### 3.3 Using a Centrally Dominated Distribution

Recall that  $k$  denotes the length of the subsequence, i.e., the number of triples, and  $x$  denotes the number of pairs containing  $u$  passed to  $\mathcal{A}$ . Let  $y$  be the number of  $u$ 's returned by  $\mathcal{A}$ ; then,  $\mathcal{B}$  receives a consecutive subsequence of pairs containing  $u$  of length  $k - x + y$ .

We remark that the probability that  $u$  is never chosen by  $\mathcal{B}$ , conditioned on the event that  $\mathcal{A}$  takes  $x$  pairs containing  $u$  and outputs  $y$  number of  $u$ 's, can be bounded by  $\left(\frac{1}{2}\right)^{k-x+y} \cdot (1 - \gamma_{\mathcal{B}})^{\max(k-x+y-1, 0)}$  from Lemma 7. Note that this bound depends only on the lengths  $k$ ,  $x$ , and  $y$ , not the actual input to  $\mathcal{B}$ .

For  $\mathcal{A}$ , on the other hand, we need to calculate the probability that exactly  $y$  number of  $u$ 's are chosen by  $\mathcal{A}$  from the given subsequence of pairs. However, this probability distribution highly depends on the input to  $\mathcal{A}$ ; therefore, we cannot conveniently fix a distribution parameterized by length  $x$  anymore. Nonetheless, we will show that there exists a bounding distribution, depending only on  $x$ , such that it yields a valid upper bound on the probability that  $u$  is never chosen by our entire algorithm from the given subsequence of triples.

### 3.3.1 Central Dominance

To construct the bounding distribution, we need the notion of central dominance. Recall the definition.

► **Definition 1** (Central Dominance). *Given two discrete symmetric probability distributions  $D_1$  and  $D_2$  on  $\{0, 1, \dots, x\}$  whose probability mass functions are  $p_1$  and  $p_2$ , respectively, we say  $D_1$  centrally dominates  $D_2$  if there exists  $z \in [0, \frac{x}{2}]$  such that, for any integer  $y \in [\frac{x}{2} - z, \frac{x}{2} + z]$ ,  $p_1(y) \geq p_2(y)$ , and for any integer  $y \in [0, \frac{x}{2} - z) \cup (\frac{x}{2} + z, x]$ ,  $p_1(y) \leq p_2(y)$ .*

Intuitively speaking, this definition indicates how flat a symmetric probability distribution is. Following is the lemma that formalizes why a flat distribution helps bound the probability that  $u$  never gets chosen from our three-way OCS. The definitions of  $\theta(\cdot, \cdot)$  and  $\theta'(\cdot, \cdot)$  can be found in (2) and (3), respectively. It is easy to intuitively see that the first half of the lemma should hold; yet,  $\theta'$  is only *nearly* convex and requires some work.

► **Lemma 9.** *Suppose we are given two discrete symmetric distributions  $D_1$  and  $D_2$  on  $\{0, 1, \dots, x\}$  whose probability mass functions are  $p_1$  and  $p_2$ , respectively. If  $D_1$  centrally dominates  $D_2$ , we have  $\theta(x, p_1) \leq \theta(x, p_2)$  and  $\theta'(x, p_1) \leq \theta'(x, p_2)$ .*

**Proof.** If  $x = 0, 1$ , it is trivial since there is only one possible symmetric distribution. Now we assume that  $x \geq 2$ . Given a discrete symmetric distribution with a probability mass function  $p$ , let us consider the following operation: For some  $w, z \in [0, \frac{x}{2}]$  such that  $w > z$ , we decrease  $p(\frac{x}{2} - z)$  and  $p(\frac{x}{2} + z)$  by some  $\epsilon > 0$ , and increase  $p(\frac{x}{2} - w)$  and  $p(\frac{x}{2} + w)$  by  $\epsilon$ , instead. We can observe that  $p$  still forms a valid probability distribution, and changes  $\theta(x, p)$  by

$$\Delta\theta(x, p) = \epsilon \left(\frac{1}{2}\right)^{x/2} (1 - \gamma_{\mathcal{B}})^{x/2-1} \left[ \left(\frac{1 - \gamma_{\mathcal{B}}}{2}\right)^w + \left(\frac{1 - \gamma_{\mathcal{B}}}{2}\right)^{-w} - \left(\frac{1 - \gamma_{\mathcal{B}}}{2}\right)^z - \left(\frac{1 - \gamma_{\mathcal{B}}}{2}\right)^{-z} \right].$$

Let  $\alpha := (1 - \gamma_{\mathcal{B}})/2$  and  $f(t) := \alpha^t + \alpha^{-t}$ . Since  $f'(t) = \ln \alpha \cdot (\alpha^t - \alpha^{-t})$ , we can verify that  $f(t)$  is increasing on  $t \geq 0$ . Hence, we can see that

$$\Delta\theta(x, p) = \epsilon \left(\frac{1}{2}\right)^{x/2} (1 - \gamma_{\mathcal{B}})^{x/2-1} (f(w) - f(z)) \geq 0.$$

Since  $D_1$  centrally dominates  $D_2$ , we can transform  $p_1$  into  $p_2$  by transferring the mass through a few times of the above operation with proper  $w$ 's,  $z$ 's, and  $\epsilon$ 's. For each application,  $\theta(x, p)$  only increases, yielding that  $\theta(x, p_1) \leq \theta(x, p_2)$ .

Proving the other statement is more subtle. Suppose we transform  $p_1$  into  $p_2$  through the same method. If we increase the probabilities of  $\frac{x}{2} - w$  and  $\frac{x}{2} + w$  where  $w < \frac{x}{2}$ , we can apply to the above analysis since the change of  $\theta'(x, p)$  is exactly the same as  $\Delta\theta(x, p)$ .

It remains to consider when the probabilities of 0 and  $x$  get increased by  $\epsilon$ . Assume that we instead decrease the probabilities of  $\frac{x}{2} - z$  and  $\frac{x}{2} + z$  where  $0 \leq z \leq \frac{x}{2} - 1$ . The change of  $\theta'(x, p)$  would be

$$\Delta\theta'(x, p) := \epsilon \left[ 1 + \left(\frac{1}{2}\right)^x (1 - \gamma_{\mathcal{B}})^{x-1} - \left(\frac{1}{2}\right)^{x/2-z} (1 - \gamma_{\mathcal{B}})^{x/2-z-1} - \left(\frac{1}{2}\right)^{x/2+z} (1 - \gamma_{\mathcal{B}})^{x/2+z-1} \right].$$

Let  $\beta := (1/2)^{x/2}(1 - \gamma_{\mathcal{B}})^{x/2-1}$  and  $g(z) := (2/(1 - \gamma_{\mathcal{B}}))^z$ . We then rewrite  $\Delta\theta'(x, p)$  as

$$\Delta\theta'(x, p) = \epsilon [1 + \beta^2(1 - \gamma_{\mathcal{B}}) - \beta(g(z) + 1/g(z))].$$

The partial derivative of  $\Delta\theta'(x, p)$  with respect to  $z$  is

$$\frac{\partial\Delta\theta'(x, p)}{\partial z} = -\epsilon\beta \ln\left(\frac{2}{1 - \gamma_{\mathcal{B}}}\right) \left[ \left(\frac{2}{1 - \gamma_{\mathcal{B}}}\right)^z - \left(\frac{1 - \gamma_{\mathcal{B}}}{2}\right)^z \right].$$

Note that  $\frac{\partial\Delta\theta'(x, p)}{\partial z} \leq 0$  for  $z \geq 0$ . Thus, we can bound  $\Delta\theta'(x, p)$  from below by plugging  $z = \frac{x}{2} - 1$ , i.e.,

$$\Delta\theta'(x, p) \geq \epsilon \left[ \frac{1}{2} - \frac{1 + \gamma_{\mathcal{B}}}{2} \left(\frac{1}{2}\right)^{x-1} (1 - \gamma_{\mathcal{B}})^{x-2} \right].$$

Since  $x \geq 2$ , we have

$$\Delta\theta'(x, p) \geq \epsilon \left[ \frac{1}{2} - \frac{1 + \gamma_{\mathcal{B}}}{4} \right] \geq 0,$$

where the last inequality comes from the fact that  $\gamma_{\mathcal{B}} < 1$ . ◀

### 3.3.2 Bounding Distribution

In this section, we construct the bounding distribution. As was noted earlier,  $\mathcal{A}$  negatively correlates the endpoints of every edge of the ex-post graph. Therefore, we can write the probability  $p(x, y)$  that the two-way OCS chooses precisely  $y$  number of  $u$ 's out of the given  $x$  consecutive pairs if we are given  $q \in \mathbb{R}_+^{\lfloor x/2 \rfloor + 1}$  defined as follows: for  $i = 0, \dots, \lfloor \frac{x}{2} \rfloor$ ,  $q_i$  is the probability that exactly  $i$  edges of the ex-post graph have both endpoints in the given subsequence. Note that  $\sum_{i=0}^{\lfloor x/2 \rfloor} q_i = 1$ . We have

$$p(x, y) = q_0 \cdot \binom{x}{y} \left(\frac{1}{2}\right)^x + q_1 \cdot \binom{x-2}{y-1} \left(\frac{1}{2}\right)^{x-2} + \dots + q_{\lfloor x/2 \rfloor} \cdot \binom{x-2\lfloor x/2 \rfloor}{y-\lfloor x/2 \rfloor} \left(\frac{1}{2}\right)^{x-2\lfloor x/2 \rfloor},$$

where  $\binom{b}{c} := 0$  if  $b < c$  or  $c < 0$ .

Note that  $q$  (and therefore in turn  $p$ ) depends on the actual input to  $\mathcal{A}$ . Given any probability vector  $q \in \mathbb{R}_+^{\lfloor x/2 \rfloor + 1}$ , let  $D(q)$  (or  $D(q_0, \dots, q_{\lfloor x/2 \rfloor})$ ) denote the probability distribution  $\{p(x, y)\}_{y=0, \dots, x}$ . It is easy to verify that, for all  $q$ ,  $D(q)$  is a valid probability distribution that is symmetric on  $y = \frac{x}{2}$ .

Now we construct the bounding distribution. For  $x = 0, \dots, k$ , let  $\alpha_x := (1 - \gamma_{\mathcal{A}})^{\max(x-1, 0)}$  for  $\gamma_{\mathcal{A}} = 1/16$ . We choose  $D(\alpha_x, 1 - \alpha_x, 0, \dots, 0)$  as our bounding distribution. Let  $p^*(x, \cdot)$  be the probability mass function of the bounding distribution, i.e., for each  $y = 0, \dots, x$ ,

$$p^*(x, y) := \begin{cases} \alpha_x \left(\frac{1}{2}\right)^x, & \text{if } y = 0 \text{ or } y = x, \\ \alpha_x \binom{x}{y} \left(\frac{1}{2}\right)^x + (1 - \alpha_x) \binom{x-2}{y-1} \left(\frac{1}{2}\right)^{x-2}, & \text{otherwise.} \end{cases}$$

Following is a key lemma to show that  $D(\alpha_x, 1 - \alpha_x, 0, \dots, 0)$  is a good choice of the centrally dominated distribution.

► **Lemma 10.** *Given a consecutive subsequence of pairs containing  $u$  of length  $x$  ( $0 \leq x \leq k$ ), let  $q \in \mathbb{R}_+^{\lfloor x/2 \rfloor + 1}$  be the probability vector realized by  $\mathcal{A}$ : i.e.,  $q_i$  is the probability that exactly  $i$  edges of the ex-post graph have both endpoints in the given subsequence. We then have  $\theta(x, p(x, \cdot)) \leq \theta(x, p^*(x, \cdot))$  and  $\theta'(x, p(x, \cdot)) \leq \theta'(x, p^*(x, \cdot))$ , where  $p(x, \cdot)$  follows the distribution  $D(q)$ .*

**Proof.** For  $x = 0, 1$ , it is easy to see that  $p(x, y) = p^*(x, y)$  for every possible  $y$  since  $q_0 = 1 = \alpha_x$ . For  $x \geq 2$ , we construct intermediate distributions as follows:

$$\begin{aligned} D_0 &:= D(q_0, 1 - q_0, 0, \dots, 0), \\ D_1 &:= D(q_0, q_1, 1 - q_0 - q_1, \dots, 0), \\ &\dots, \\ D_{\lfloor x/2 \rfloor - 1} &:= D(q_0, q_1, q_2, \dots, 1 - \sum_{i=0}^{\lfloor x/2 \rfloor - 1} q_i) = D(q). \end{aligned}$$

We claim that  $D_0$  centrally dominates  $D(\alpha_x, 1 - \alpha_x, 0, \dots, 0)$  and, for each  $j = 1, \dots, \lfloor \frac{x}{2} \rfloor - 1$ ,  $D_j$  centrally dominates  $D_{j-1}$ . Then, by repeatedly applying Lemma 9, we can prove the lemma.

Let us first prove that  $D_0$  centrally dominates  $D(\alpha_x, 1 - \alpha_x, 0, \dots, 0)$ . Since both distributions are symmetric, it is sufficient for us to consider  $y \in [0, \frac{x}{2}]$ . By Lemma 6, we have  $q_0 \leq \alpha_x$ . If  $q_0 = \alpha_x$ ,  $D_0$  is equivalent to  $D(\alpha_x, 1 - \alpha_x, 0, \dots, 0)$ , completing the proof. We now assume  $q_0 < \alpha_x$ . If  $y = 0$ ,  $D_0$  has the smaller probability. This implies that there must exist a point in  $[1, \frac{x}{2}]$  where  $D_0$  has the greater probability since both distributions are valid.

Let  $y^*$  be the smallest such point. We show that for any integer  $y \in [y^*, \frac{x}{2}]$ ,  $D_0$  has the greater probability. Let  $g(y) := (q_0 - \alpha_x) \binom{x}{y} \left(\frac{1}{2}\right)^x + (\alpha_x - q_0) \binom{x-2}{y-1} \left(\frac{1}{2}\right)^{x-2}$  be the subtraction of the probability of  $D_0$  from that of  $D(\alpha_x, 1 - \alpha_x, 0, \dots, 0)$  at point  $y$ . It suffices to prove that  $g(y)$  is non-decreasing on  $[y^*, \frac{x}{2}] \subseteq [1, \frac{x}{2}]$ . By rewriting the formula, we have  $g(y) = (\alpha_x - q_0) \binom{x}{y} \left(\frac{1}{2}\right)^x \left(\frac{y(x-y)}{4x(x-1)} - 1\right)$ , and it is not hard to see that the function is increasing on  $[1, \frac{x}{2}]$ .

A similar argument can be applied to show that, for each  $j = 1, \dots, \lfloor \frac{x}{2} \rfloor - 1$ ,  $D_j$  centrally dominates  $D_{j-1}$ . Indeed, if  $q_j = 1 - \sum_{i=0}^{j-1} q_i$ ,  $D_j$  is equivalent to  $D_{j-1}$ . We thus assume that  $q_j < 1 - \sum_{i=0}^{j-1} q_i$ . Observe that  $D_j$  has the same probabilities as  $D_{j-1}$  for  $y = 0, \dots, j-1$ , and has the smaller probability for  $y = j$ ; hence, there exists a point in  $[j+1, \frac{x}{2}]$  where  $D_j$  has the greater probability, and let  $y^*$  be the smallest such value. Again, let  $g(y)$  be the difference obtained by subtracting the probability of  $D_{j-1}$  from that of  $D_j$  at point  $y$ . We can write

$$\begin{aligned} g(y) &= \left(1 - \sum_{i=0}^j q_i\right) \left[ \binom{x-2j-2}{y-j-1} \left(\frac{1}{2}\right)^{x-2j-2} - \binom{x-2j}{y-j} \left(\frac{1}{2}\right)^{x-2j} \right] \\ &= \left(1 - \sum_{i=0}^j q_i\right) \binom{x-2j}{y-j} \left(\frac{1}{2}\right)^{x-2j} \left[ \frac{(y-j)(x-y-j)}{4(x-2j)(x-2j-1)} - 1 \right], \end{aligned}$$

implying that  $g(y)$  is increasing on  $[y^*, \frac{x}{2}] \subseteq [j+1, \frac{x}{2}]$ . ◀

We are now ready to bound the probability that our three-way OCS never chooses the element  $u$  from the consecutive subsequence of triples of length  $k$ . Let  $\text{binom}(k, x, r)$  be the probability mass function of the binomial distribution for  $x$  successes out of  $k$  trials with probability  $r$ , i.e.,  $\text{binom}(k, x, r) := \binom{k}{x} r^x (1-r)^{k-x}$ . Finally, we define  $\eta(k)$  as follows and it will be the desired bound.



$$\eta(k) := \sum_{x=0}^k \text{binom} \left( k, x, \frac{2}{3} \right) \left[ \sum_{y=0}^x p^*(x, y) \left( \frac{1}{2} \right)^{k-x+y} (1 - \gamma_{\mathcal{B}})^{\max(k-x+y-1, 0)} \right].$$

► **Lemma 11.** *Given a consecutive subsequence of triples containing  $u$  of length  $k$ , the probability that our three-way OCS never chooses  $u$  from the subsequence is at most  $\eta(k)$ .*

**Proof.** Conditioned on the pairs selected by the random pair choice phase, let  $x$  be the number of pairs containing  $u$  inserted into  $\mathcal{A}$ . Recall that these pairs form a consecutive subsequence. Let  $p(x, y)$  denote the probability that  $\mathcal{A}$  returns  $y$  number of  $u$ 's from this subsequence. Note that  $p$  follows  $D(q)$  for some  $q$ .

Note that, if  $\mathcal{A}$  returns  $y$  number of  $u$ 's,  $\mathcal{B}$  eventually takes  $k - x + y$  consecutive pairs that originate from the given consecutive triples. By Lemma 7, we can see that the probability that  $u$  is never selected from the given consecutive subsequences of triples is bounded from above by

$$\sum_{y=0}^x p(x, y) \left( \frac{1}{2} \right)^{k-x+y} (1 - \gamma_{\mathcal{B}})^{\max(k-x+y-1, 0)},$$

which can be rewritten as  $\left( \frac{1-\gamma_{\mathcal{B}}}{2} \right)^{k-x} \theta(x, p(x, \cdot))$  if  $x < k$  or  $\theta'(k, p(x, \cdot))$  if  $x = k$ . In any case, by Lemma 10, it can be further bounded by

$$\sum_{y=0}^x p^*(x, y) \left( \frac{1}{2} \right)^{k-x+y} (1 - \gamma_{\mathcal{B}})^{\max(k-x+y-1, 0)}.$$

It is noteworthy that this value depends only on the number  $x$  of  $u$ 's selected by the random pair choice phase. Observe that the probability that  $x$  pairs containing  $u$  are chosen from  $k$  consecutive triples by the random pair choice phase is exactly  $\text{binom}(k, x, 2/3)$ , completing our proof. ◀

It still remains to calculate the bounds on  $\eta(k)$  in the same form as stated in Theorem 8. We refer the interested readers to [33].

## 4 General Bound

We now extend our discussion to the general case and provide a bound for a set of disjoint consecutive subsequences. The following theorem states this bound.

► **Theorem 2 (restated).** *Consider a set of  $m$  disjoint consecutive subsequences of triples containing an element  $u$ . Let  $k_1, \dots, k_m$  be the lengths of these subsequences. The probability that our three-way OCS never chooses  $u$  from these  $m$  subsequences is at most  $\prod_{i=1}^m \eta(k_i)$ .*

Let  $\mathbf{k} = (k_1, \dots, k_m)$  be a vector whose  $i$ -th entry is the length of the  $i$ -th subsequence. In what follows, we fix (and condition upon) the choices made by the *random pair choice phase*. Let  $\mathbf{x} := (x_1, \dots, x_m)$  be the (now constant) vector whose  $i$ -th entry represents the number of pairs containing  $u$  from the  $i$ -th subsequence that are passed to  $\mathcal{A}$ . For  $\mathbf{y} := (y_1, \dots, y_m) \leq \mathbf{x}$ , let  $p(\mathbf{x}, \mathbf{y})$  be the conditional probability that  $\mathcal{A}$  chooses  $y_i$  number of  $u$ 's from the  $i$ -th subsequence of pairs for all  $i$ . Finally, let  $p_0$  be the probability that our three-way OCS never chooses  $u$ . By Lemma 7, we can observe that  $p_0$  is no greater than

$$\sum_{\mathbf{y} \leq \mathbf{x}} p(\mathbf{x}, \mathbf{y}) \prod_{i=1}^m \left( \frac{1}{2} \right)^{k_i - x_i + y_i} (1 - \gamma_{\mathcal{B}})^{\max(k_i - x_i + y_i - 1, 0)}. \tag{5}$$

To prove the desired bound, we will first present how we can modify the input to  $\mathcal{A}$  without ever decreasing (5). After finitely many modifications, we will be able to “decouple” the random choices across different subsequences.

Recall that each pair inserted into  $\mathcal{A}$  takes three independent random bits. The first random bit determines if the pair is a sender or a receiver. The second random bit decides which adjacent vertex to correlate. The third random bit selects an element to be output for this pair, unless the pair becomes matched as a receiver. Note that, with the first two random bits, the ex-post graph can be determined.

For each vertex  $j$  containing  $u$  in the ex-ante graph, let  $\text{pred}(j, u)$  and  $\text{pred}(j, -u)$  be the immediate predecessor of  $j$  linked by  $u$  and by the other element than  $u$ , respectively. Without loss of generality, let  $v$  be the other element of  $j$  than  $u$ . If we have  $j'$  and  $j$ , residing in different subsequences and  $j'$  appearing before  $j$ , such that

- (A)  $j' = \text{pred}(j, u)$  or  $j' = \text{pred}(j, -u)$  (or both); or
- (B) there exists  $\hat{j}$  such that
  - $\hat{j} = \text{pred}(j', u)$  or  $\hat{j} = \text{pred}(j', -u)$ ,
  - $\hat{j} = \text{pred}(j, -u)$ , and
  - $\hat{j}$  is not contained in any subsequence,

we call them *violations*. In particular, the first type of violations are called Type A violations and the other Type B. Our goal is to modify the input so that there are no violations while (5) never decreased.

For notational simplicity, we let  $\phi(y) := \left(\frac{1}{2}\right)^{k-x+y} (1 - \gamma_B)^{\max(k-x+y-1, 0)}$  for all  $y = 0, \dots, x$ . Observe that  $\phi(y)$  is decreasing over  $y$ . We can now consider (5) as the expected value of  $\prod_{i=1}^m \phi(y_i)$ .

**Removing Type A Violations.** Let us first consider the case when  $j' = \text{pred}(j, u) = \text{pred}(j, -u)$ . We build the new input to  $\mathcal{A}$  by inserting  $\bar{j}_1 = (u, \star)$  and  $\bar{j}_2 = (v, \diamond)$  right before  $j$  is input where  $\star$  and  $\diamond$  are elements appearing nowhere else and  $\star \neq \diamond$ .

Let us fix some random bits of  $\mathcal{A}$ : one can think of this as further conditioning on those bits. In particular, we will fix the first two random bits of every pair except for  $\bar{j}_1$  and  $\bar{j}_2$ .

Suppose for now that the random bits dictate that neither  $(j', j)_u$  nor  $(j', j)_v$  is present in the ex-post graph of the original input. We claim that the probability distribution  $p$  conditioned on this event stays the same even after the modification and therefore the “contribution” to the expectation (5) is not affected either (or in other words, the conditional expectation remains equal).

We will show a stronger statement that the marginal probability distribution of the pairs in the subsequences stays the same. (Then the probability distribution  $p$  will automatically be the same.) Note that the ex-post graph of the two inputs will look almost identical, except for possible addition of some edges incident with  $\bar{j}_1$  or  $\bar{j}_2$ . Therefore, the output choice of every pair in the subsequences will be determined by the same random bit in both inputs, except that, it may be the case that  $j$  was not adjacent with any pair in the ex-post graph of the original input but becomes a receiver of  $\bar{j}_1$  (or  $\bar{j}_2$ ) after the modification. In this case, the output choice of  $j$  was determined by the third random bit of  $j$  before but by that of  $\bar{j}_1$  (or  $\bar{j}_2$ ) now. However, the only pair in the subsequence whose output choice is determined by the third random bit of  $\bar{j}_1$  (or  $\bar{j}_2$ ) is  $j$ . Moreover, the third random bits of the pairs are i.i.d.; hence, the marginal distribution of the pairs in the subsequences stays the same, as was claimed.

Now consider the remaining case where  $j'$  and  $j$  are adjacent in the ex-post graph of the original input. In this case, we will show that the conditional expectation cannot decrease (as opposed to staying the same). To ease the argument, we will fix more random bits (and

argue that the conditional expectation cannot decrease in all cases): we fix the third random bits of all pairs, except for  $j'$ ,  $j$ ,  $\bar{j}_1$ , and  $\bar{j}_2$ . Let  $z'$  be the number of  $u$ 's chosen from the subsequence containing  $j'$ , except for  $j'$  itself, in the original input. (Since we did not fix the third random bit of  $j'$ , we cannot determine what the output choice of  $j'$  is.) Similarly, let  $z$  be the number of  $u$ 's chosen from the subsequence containing  $j$ , except for  $j$  itself.

In the original input, note that the output choice of  $j'$  and  $j$  are both determined by the third random bit of  $j'$ . That is, with probability  $1/2$ ,  $z'$  and  $z+1$  respectively are the number of  $u$ 's chosen from the subsequences containing  $j'$  and  $j$ , and with probability  $1/2$ ,  $z'+1$  and  $z$  are. In the modified input, however, while we do not know whether  $j'$  is adjacent with  $\bar{j}_1$  or  $\bar{j}_2$  in the ex-post graph,  $\bar{j}_1$  and  $\bar{j}_2$  can each be adjacent with at most one pair in the ex-post graph. Therefore, in the marginal distribution of the pairs in the subsequences, the output choice of  $j'$  is independent from all other pairs, including  $j$ . This shows that the number of  $u$ 's chosen from the subsequence containing  $j'$  in the modified input is  $z'$  with probability  $1/2$  and  $z'+1$  with probability  $1/2$ , and independently from that, the number of  $u$ 's from the subsequence containing  $j$  is  $z$  with probability  $1/2$  and  $z+1$  with probability  $1/2$ .

Let us now calculate the increase of the conditional expectation of  $\prod_{i=1}^m \phi(y_i)$ , but since we are only interested in its sign, we will ignore the common terms, i.e.,  $\phi(y_i)$ 's for the subsequences other than ones containing  $j$  and  $j'$ . Note that

$$\begin{aligned} & \frac{1}{4} [\phi(z')\phi(z) + \phi(z')\phi(z+1) + \phi(z'+1)\phi(z) + \phi(z'+1)\phi(z+1)] \\ & - \frac{1}{2} [\phi(z')\phi(z+1) + \phi(z'+1)\phi(z)] \\ = & \frac{1}{4} (\phi(z') - \phi(z'+1))(\phi(z) - \phi(z+1)) \geq 0, \end{aligned}$$

where the inequality follows from the fact that  $\phi(y)$  is decreasing over  $y$ . This shows our claim.

Let us now consider the case where  $j' = \text{pred}(j, u) \neq \text{pred}(j, -u)$  or vice versa. In this case, we insert  $\bar{j} = (u, \star)$  or  $\bar{j} = (v, \star)$  right before  $j$ , where  $\star$  again is a unique element. A similar argument shows that we can show that (5) can only increase in the new input.

**Removing Type B Violations.** We build the new input to  $\mathcal{A}$  by inserting  $\bar{j} = (v, \star)$  right before  $j$ , where  $\star$  is a new unique element. Let us fix the first two random bits of every pair (including  $\bar{j}$ ). We claim that the marginal distributions of the pairs in the subsequences are identical in both inputs.

Consider the ex-post graph of the two inputs. They will be almost identical only with the following possible differences:

- $(\hat{j}, j)$  may be only in the original ex-post graph;
- $(\hat{j}, \bar{j})$  or  $(\bar{j}, j)$  may be only in the new ex-post graph.

For each pair  $x$  in the subsequences, the ex-post graph determines, among the third random bits of all pairs, which one determines the output choice of  $x$ .

Suppose  $(\hat{j}, j)$  is in the original ex-post graph. In this case,  $j$  is the only pair in the subsequences whose output choice is determined by the third random bit of  $\hat{j}$ . (Recall that  $\hat{j}$  is not in any subsequence.) In the modified input, if  $(\bar{j}, j)$  is in the ex-post graph,  $j$  will be the only pair in the subsequences whose output choice is determined by the third random bit of  $\bar{j}$ . If  $(\bar{j}, j)$  is not in the ex-post graph,  $j$  will be the only pair whose output choice is determined by  $j$ . Note that the output choice of every other pair in the subsequences is determined by the same random bit in both inputs. Again, since the third random bits of the pairs are i.i.d., this shows that the marginal distribution stays the same.

## 49:14 Making Three out of Two: Three-Way Online Correlated Selection

Suppose now  $(\hat{j}, j)$  is not in the original ex-post graph. Let us focus on  $j$ , since it is the only pair whose output choice may be determined by a different random bit in the modified input. If  $j$  was adjacent with another pair in the original ex-post graph, the edge will remain in the new ex-post graph, too, in which case there is nothing to prove. Otherwise, it may be the case that  $(\bar{j}, j)$  may be newly introduced as an edge in the ex-post graph, but this only means that  $j$  will be the only pair in the subsequences whose output choice is determined by the third random bit of  $\bar{j}$  (instead of  $j$ ). The marginal distribution therefore stays the same.

**Further Modification.** We can remove all Type A and Type B violations after finitely many modifications of the input. We can interpret  $\mathcal{A}$  as an algorithm that first constructs the ex-ante graph using the first two random bits and then determines the output using only the third random bits. Our next modification will delete some edges directly from the ex-ante graph rather than modifying the input pairs. (You could alternatively think of this edge being “disabled,” which can never appear in the ex-post graph even if it is chosen by the first two random bits of the pairs.) We will now show how we can modify this input graph without affecting (5). Suppose we have some pair  $j'$  and  $j$  such that

- $j'$  appears before  $j$ ,
- $j'$  and  $j$  are adjacent in the ex-ante graph, and
- $j$  is not in any subsequence.

If we delete this edge from the ex-ante graph, this can change the output choice of only  $j$  in  $\mathcal{A}$ 's output. (Imagine we fix all random bits, and we can easily observe this fact.) Therefore, we can safely delete all such edges without affecting the marginal distribution of the pairs in the subsequences.

Once we remove all such edges, the following argument shows that no two pairs in different subsequences can belong to the same connected component of the ex-ante graph: since two pairs that are not in any subsequences cannot be adjacent, the only way of having two pairs from different subsequences in the same connected component is by having a direct edge in-between (which would be a Type A violation) or a length-2 path whose “midpoint” is a predecessor of the two pairs (which would be a Type B violation: note that, in the definition of Type B violation,  $\hat{j}$  cannot be  $\text{pred}(j, u)$  because  $j'$  appears before  $j$ ).

**Conclusion.** Let  $p'(\mathbf{x}, \mathbf{y})$  be the probability that  $\mathcal{A}$  chooses  $y_i$  number of  $u$ 's from the  $i$ -th subsequence of pairs for all  $i$  in the *modified* input. We have so far shown that

$$p_0 \leq \sum_{\mathbf{y} \leq \mathbf{x}} p'(\mathbf{x}, \mathbf{y}) \prod_{i=1}^m \left(\frac{1}{2}\right)^{k_i - x_i + y_i} (1 - \gamma_{\mathcal{B}})^{\max(k_i - x_i + y_i - 1, 0)}.$$

For each  $i$ , let  $p'_i(x_i, y_i)$  be the probability that  $\mathcal{A}$  chooses  $y_i$  number of  $u$ 's from the  $i$ -th subsequence of pairs. Since  $\mathcal{A}$  may introduce negative correlation only on those pair of vertices that are adjacent in the ex-ante graph, output choices made across different connected components of the ex-ante graph will be independent. This implies that  $p'(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^m p'_i(x_i, y_i)$ , yielding

$$\begin{aligned} p_0 &\leq \sum_{\mathbf{y} \leq \mathbf{x}} p'(\mathbf{x}, \mathbf{y}) \prod_{i=1}^m \left(\frac{1}{2}\right)^{k_i - x_i + y_i} (1 - \gamma_{\mathcal{B}})^{\max(k_i - x_i + y_i - 1, 0)} \\ &= \sum_{\mathbf{y} \leq \mathbf{x}} \prod_{i=1}^m p'_i(x_i, y_i) \left(\frac{1}{2}\right)^{k_i - x_i + y_i} (1 - \gamma_{\mathcal{B}})^{\max(k_i - x_i + y_i - 1, 0)} \end{aligned}$$

$$\begin{aligned}
&= \prod_{i=1}^m \sum_{y_i=0}^{x_i} p'_i(x_i, y_i) \left(\frac{1}{2}\right)^{k_i-x_i+y_i} (1-\gamma_{\mathcal{B}})^{\max(k_i-x_i+y_i-1,0)} \\
&\leq \prod_{i=1}^m \sum_{y_i=0}^{x_i} p^*(x_i, y_i) \left(\frac{1}{2}\right)^{k_i-x_i+y_i} (1-\gamma_{\mathcal{B}})^{\max(k_i-x_i+y_i-1,0)},
\end{aligned}$$

where the last inequality follows from the proof of Lemma 11. Since  $p^*(x, \cdot)$  depends only on  $x$ , rather than the actual input, the bound on  $p_0$  we obtain depends only on  $\mathbf{x}$ . Let  $\text{binom}(\mathbf{k}, \mathbf{x}, r) := \prod_{i=1}^m \text{binom}(k_i, x_i, r)$ . Now the probability that our three-way OCS never chooses  $u$  from the given subsequences is no greater than

$$\begin{aligned}
&\sum_{\mathbf{x} \leq \mathbf{k}} \text{binom}\left(\mathbf{k}, \mathbf{x}, \frac{2}{3}\right) \left[ \prod_{i=1}^m \sum_{y_i=0}^{x_i} p^*(x_i, y_i) \left(\frac{1}{2}\right)^{k_i-x_i+y_i} (1-\gamma_{\mathcal{B}})^{\max(k_i-x_i+y_i-1,0)} \right] \\
&= \sum_{\mathbf{x} \leq \mathbf{k}} \left[ \prod_{i=1}^m \text{binom}\left(k_i, x_i, \frac{2}{3}\right) \right] \left[ \prod_{i=1}^m \sum_{y_i=0}^{x_i} p^*(x_i, y_i) \left(\frac{1}{2}\right)^{k_i-x_i+y_i} (1-\gamma_{\mathcal{B}})^{\max(k_i-x_i+y_i-1,0)} \right] \\
&= \prod_{i=1}^m \sum_{x_i=0}^{k_i} \text{binom}\left(k_i, x_i, \frac{2}{3}\right) \left[ \sum_{y_i=0}^{x_i} p^*(x_i, y_i) \left(\frac{1}{2}\right)^{k_i-x_i+y_i} (1-\gamma_{\mathcal{B}})^{\max(k_i-x_i+y_i-1,0)} \right] \\
&= \prod_{i=1}^m \eta(k_i).
\end{aligned}$$

This completes the proof of Theorem 2.

## 5 Application to Online Bipartite Matching

Our three-way OCS can be applied to online bipartite matching problems. While the algorithm and the factor-revealing LP had to be generalized to a higher dimension, our algorithm and analysis are still analogous to Fahrback et al. [10]. We refer the interested readers to [33].

► **Theorem 12.** *There exists a 0.5096-competitive algorithm for unweighted online bipartite matching.*

► **Theorem 13.** *There exists a 0.5093-competitive algorithm for edge-weighted online bipartite matching with free disposal.*

---

### References

- 1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Symposium on Discrete Algorithms (SODA)*, pages 1253–1264. SIAM, 2011.
- 2 Arash Asadpour, Michel X Goemans, Aleksander Mądry, Shayan Oveis Gharan, and Amin Saberi. An  $O(\log n / \log \log n)$ -approximation algorithm for the asymmetric traveling salesman problem. *Operations Research*, 65(4):1043–1061, 2017.
- 3 Itai Ashlagi, Maximilien Burq, Chinmoy Dutta, Patrick Jaillet, Amin Saberi, and Chris Sholley. Edge weighted online windowed matching. In *Conference on Economics and Computation (EC)*, pages 729–742, 2019.
- 4 Benjamin E. Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *SIGACT News*, 39(1):80–87, 2008.
- 5 Guy Blanc and Moses Charikar. Multiway online correlated selection. *arXiv preprint*, 2021. To appear in Symposium on Foundations of Computer Science (FOCS 2021). [arXiv:2106.05579](https://arxiv.org/abs/2106.05579).

- 6 Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. New algorithms, better bounds, and a novel model for online stochastic matching. In *European Symposium on Algorithms (ESA)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 7 Chandra Chekuri and Jan Vondrák. Randomized pipage rounding for matroid polytopes and applications. *CoRR*, abs/0909.4348, 2009. [arXiv:0909.4348](#).
- 8 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *Symposium on Discrete Algorithms (SODA)*, pages 101–107. SIAM, 2013.
- 9 Alon Eden, Michal Feldman, Amos Fiat, and Kineret Segal. An economics-based analysis of ranking for online bipartite matching. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 107–110. SIAM, 2021.
- 10 Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. In *Symposium on Foundations of Computer Science (FOCS)*, pages 412–423. IEEE, 2020.
- 11 Jon Feldman, Nitish Korula, Vahab Mirrokni, Shanmugavelayutham Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *International Workshop on Internet and Network Economics (WINE)*, pages 374–385. Springer, 2009.
- 12 Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan. Online stochastic matching: Beating  $1-1/e$ . In *Symposium on Foundations of Computer Science (FOCS)*, pages 117–126. IEEE, 2009.
- 13 Buddhima Gamalath, Sagar Kale, and Ola Svensson. Beating greedy for stochastic bipartite matching. In *Symposium on Discrete Algorithms (SODA)*, pages 2841–2854. SIAM, 2019.
- 14 Buddhima Gamalath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *Symposium on Foundations of Computer Science (FOCS)*, pages 26–37. IEEE, 2019.
- 15 Ruiquan Gao, Zhongtian He, Zhiyi Huang, Zipei Nie, Bijun Yuan, and Yan Zhong. Improved online correlated selection. *arXiv preprint*, 2021. To appear in Symposium on Foundations of Computer Science (FOCS 2021). [arXiv:2106.04224](#).
- 16 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Symposium on Discrete Algorithms (SODA)*, pages 982–991. SIAM, 2008.
- 17 Bernhard Haeupler, Vahab S Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In *International Workshop on Internet and Network Economics (WINE)*, pages 170–181. Springer, 2011.
- 18 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *Journal of the ACM*, 67(3):1–25, 2020.
- 19 Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *Symposium on Discrete Algorithms (SODA)*, pages 2875–2886. SIAM, 2019.
- 20 Zhiyi Huang and Xinkai Shu. Online stochastic matching, poisson arrivals, and the natural linear program. In *Symposium on Theory of Computing (STOC)*, pages 682–693. ACM, 2021.
- 21 Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating  $1-1/e$  with random arrivals. *ACM Transactions on Algorithms*, 15(3):1–15, 2019.
- 22 Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Fully online matching ii: Beating ranking and water-filling. In *Symposium on Foundations of Computer Science (FOCS)*, pages 1380–1391. IEEE, 2020.
- 23 Zhiyi Huang, Qiankun Zhang, and Yuhao Zhang. Adwords in a panorama. In *Symposium on Foundations of Computer Science (FOCS)*, pages 1416–1426. IEEE, 2020.
- 24 Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

- 25 Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Symposium on Theory of Computing (STOC)*, pages 352–358. ACM, 1990.
- 26 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *European Symposium on Algorithms (ESA)*, pages 589–600. Springer, 2013.
- 27 Samir Khuller, Stephen G Mitchell, and Vijay V Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.
- 28 Nitish Korula, Vahab S Mirrokni, and Morteza Zadimoghaddam. Bicriteria online matching: Maximizing weight and cardinality. In *International Conference on Web and Internet Economics (WINE)*, pages 305–318. Springer, 2013.
- 29 Vahideh H Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Mathematics of Operations Research*, 37(4):559–573, 2012.
- 30 Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.
- 31 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM*, 54(5):22, 2007.
- 32 Aranyak Mehta, Bo Waggoner, and Morteza Zadimoghaddam. Online stochastic matching with unequal probabilities. In *Symposium on Discrete Algorithms (SODA)*, pages 1388–1404. SIAM, 2015.
- 33 Yongho Shin and Hyung-Chan An. Making three out of two: Three-way online correlated selection. *arXiv preprint*, 2021. [arXiv:2107.02605](https://arxiv.org/abs/2107.02605).
- 34 Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *Symposium on Foundations of Computer Science (FOCS)*, pages 588–597. IEEE, 2001.





# Tight Competitive Analyses of Online Car-Sharing Problems

**Ya-Chun Liang** ✉

Department of Industrial Engineering and Engineering Management,  
National Tsing Hua University, Hsinchu, Taiwan

**Kuan-Yun Lai** ✉

Department of Industrial Engineering and Engineering Management,  
National Tsing Hua University, Hsinchu, Taiwan

**Ho-Lin Chen** ✉

Department of Electrical Engineering,  
National Taiwan University, Taipei, Taiwan

**Kazuo Iwama** ✉

Department of Industrial Engineering and Engineering Management,  
National Tsing Hua University, Hsinchu, Taiwan

---

## Abstract

The car-sharing problem, proposed by Luo, Erlebach and Xu in 2018, mainly focuses on an online model in which there are two locations: 0 and 1, and  $k$  total cars. Each request which specifies its pick-up time and pick-up location (among 0 and 1, and the other is the drop-off location) is released in each stage a fixed amount of time before its specified start (i.e. pick-up) time. The time between the booking (i.e. released) time and the start time is enough to move empty cars between 0 and 1 for relocation if they are not used in that stage. The model, called  $kS2L-F$ , assumes that requests in each stage arrive sequentially regardless of the same booking time and the decision (accept or reject) must be made immediately. The goal is to accept as many requests as possible. In spite of only two locations, the analysis does not seem easy and the (tight) competitive ratio (CR) is only known to be 2.0 for  $k = 2$  and 1.5 for a restricted value of  $k$ , i.e., a multiple of three. In this paper, we remove all the holes of unknown CR's; namely we prove that the CR is  $\frac{2k}{k+\lceil k/3 \rceil}$  for all  $k \geq 2$ . Furthermore, if the algorithm can delay its decision until all requests have come in each stage, the CR is improved to roughly  $4/3$ . We can take this advantage even further, precisely we can achieve a CR of  $\frac{2+R}{3}$  if the number of requests in each stage is at most  $Rk$ ,  $1 \leq R \leq 2$ , where we do not have to know the value of  $R$  in advance. Finally we demonstrate that randomization also helps to get (slightly) better CR's.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Car-sharing, Competitive analysis, On-line scheduling, Randomized algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.50

**Funding** This work was supported by MOST Taiwan under Grants 110-2223-E-007-001, 110-2811-E-007-507 and KAKENHI Japan under Grant 16H02782.

## 1 Introduction

Our problem in this paper is the *online car-sharing problem*. In car-sharing (not only for cars, but also for other resources like bikes and shuttle-buses), there are several service stations in the city, for instance in residential areas and downtown, at popular sightseeing spots, and so on. Customers can make a request with a pick-up time and place and a drop-off time and place. The decision for accepting or rejecting a request should be made in an online fashion and we want to maximize the profit by accepting as many requests as possible. Relocation of (unused) resources is usually possible with a much smaller or even negligible costs. (It is seen occasionally that a truck is carrying bikes for this purpose.) Theoretical studies of this problem have started rather recently and turned out to be nontrivial even for two locations.



© Ya-Chun Liang, Kuan-Yun Lai, Ho-Lin Chen, and Kazuo Iwama;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

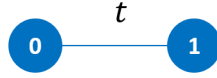
Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 50; pp. 50:1–50:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Model.** We basically follow the problem setting of previous studies by Luo et al. [7–10] with main focus to that of two locations and  $k$  servers (i.e. cars). The two locations are denoted by 0 and 1 and  $k(\geq 2)$  servers are initially located at location 0. The travel time from 0 to 1 and 1 to 0 is the same, denoted by  $t$ . The problem for  $k$  servers and two locations is called the  $kS2L$  problem for short.



■ **Figure 1** The car-sharing problem with two locations.

We denote the  $i$ -th request by  $r_i = (\tilde{t}_i, t_i, p_i)$  which is specified by the *release time* or the *booking time*  $\tilde{t}_i$ , the *start time*  $t_i$ , and the *pick-up location*  $p_i \in \{0, 1\}$  (the *drop-off location* is  $1 - p_i$ ). If  $r_i$  is accepted, the server must pick up the customer at  $p_i$  at time  $t_i$  and drop off the customer at  $1 - p_i$  at time  $t_i + t$ . Suppose for each  $r_i$ ,  $t_i$  is an integer multiple of the travel time between location 0 and 1, i.e.,  $t_i = vt$  for some  $v \in \mathbb{N}$ . We assume that  $t_i - \tilde{t}_i$  is equal to a fixed value  $a$ , where  $a \geq t$  for all requests. Without loss of generality, assume  $a = t$ . Then we are only interested in a discrete-time stage, denoted by  $0, 1, 2, \dots$

Each server can only serve one request at a time. Serving a request yields a fixed positive profit  $y$ . A server used for a request with  $p_i = 0$  ( $p_i = 1$ , resp.) cannot be used for a request with  $p_{i+1} = 0$  ( $p_{i+1} = 1$ , resp.) in the next stage. We allow *empty movements*, i.e., a server can be moved from one location to the other without serving any request. An empty movement spends time  $t$ , but takes no cost. The goal of the  $kS2L$  problem is to maximize the total profit by serving a set of online requests. Note that the performance of an online scheduling algorithm is typically evaluated by *competitive analysis*. More precisely, the quality of an online algorithm is measured by the worst case ratio, called *competitive ratio* (CR), which is defined to be the fraction of the profit of the offline optimal algorithm over that of the online algorithm. The offline algorithm is aware of all requests in advance. If an online algorithm is randomized, we use expected values for the output of the online algorithm. The online algorithm is called  $1/\delta$ -competitive, if for any instances, the profit of the algorithm is at least  $\delta$  times the offline optimal profit. So far the current model is exactly the same as  $kS2L-F$  in [7–10], although no randomized cases were discussed in these papers.

**Simultaneous Decision Model.** Recall that in the  $kS2L$  model, two (or more) inputs with *the same booking time* still have an order. Thus we can equivalently think that if  $r_1, \dots, r_d$  are requests with booking time  $t$ , they are coming later than  $t - 1$  and before or at  $t$ , one by one. Each of them should get a decision (accept or reject) immediately before the next request. The adversary can change  $r_i$  after looking at the response of the online algorithm against  $r_1, \dots, r_{i-1}$ .

This setting sounds reasonable as an online model, but the following question seems also natural; what if requests with the same booking time come exactly at the same time, the online player can see all of them and can make decisions all together simultaneously at the booking moment (equivalently the requests arrive in the same fashion as above but the player can delay his/her online decisions until the booking moment). In this study we also consider this new model, denoted by  $kS2L-S$ . We further extend the model, assuming that the number of requests with the same booking time is at most  $Rk$  for some constant  $R$ . We call the generalized model  $RkS2L-S$ . Notice that having more than  $k$  requests at the same location with the same booking time never helps. Therefore, we only need to study the range

$0 \leq R \leq 2$  and  $kS2L-S$  means the special case that  $R = 2$ . Our algorithm for  $RkS2L-S$  is *adaptive* in the sense that it automatically accommodates the value of  $R$ , which does not have to be known in advance.

**Prior work.** The car-sharing problem has received a considerable amount of attention in recent years. Luo et al. [7] studied the problem with a single server and two locations with both fixed booking time and variable booking time. Here “variable booking time” essentially means that requests with start time  $t$  may come after requests with start time  $t - 1$ . They gave lower bounds on the CR for both fixed and variable booking time under the positive empty movement assumption. Later, Luo et al. [8] studied the car-sharing problem with two servers and two locations, i.e.  $2S2L$ . They considered only the problem with fixed booking time and proposed an online algorithm which can achieve a tight bound of two. Luo et al. [9] studied the car-sharing problem with  $k$  servers and two locations, for both fixed booking time ( $kS2L-F$ ) and variable booking time ( $kS2L-V$ ). Namely they showed the CR is at least 1.5 for all  $k$  and at most 1.5 for  $k = 3i$  for  $kS2L-F$  and at least  $5/3$  for all  $k$  and at most  $5/3$  for  $k = 5i$  for  $kS2L-V$ . Very recently, Luo et al. [10] studied the car-sharing problem on a star network with  $k$  servers as well as two types of travel time: a unit travel time and an arbitrary travel time.

In comparison with the online setting, Böhmová et al. [3] considered the offline car-sharing problem in which all input requests are known in advance. The objective is to minimize the number of vehicles while satisfying all the requests. The offline (i.e. static) problem can be solved in polynomial time. Another closely related problem is the on-line dial-a-ride problem (OLDARP), where objects are transported between given points in a metric space. The problem has been studied widely. The goal is to minimize the total makespan [1, 2] or the maximum flow time [6]. Christman et al. [4] studied a variation of OLDARP where each request yields a revenue. Yi et al. [12] studied another variation of OLDARP where each request has a deadline, having a similar flavor as car sharing.

**Our contribution.** Recall that the tight CR of  $kS2L-F$  is 1.5 for  $k = 3i$  [9] and 2 for  $k = 2$  [8], but open for other  $k$ 's. In this paper, we show that it is  $\frac{2k}{k+\lceil k/3 \rceil}$  for all  $k \geq 2$  and 1.5 for all  $k \geq 2$  if randomization is allowed. For  $kS2L-S$  that allows the online player to delay its decision, it is shown that we can indeed take this advantage. Namely the tight CR for  $kS2L-S$  is  $\frac{2k}{k+\lceil k/2 \rceil}$  for all  $k \geq 2$  and  $4/3$  for all  $k \geq 2$  if randomization is allowed. For  $RkS2L-S$  (we can assume  $1 \leq R \leq 2$  without loss of generality), it is shown that the CR is strictly improved if  $R < 2$ , namely the tight CR (for randomized algorithms) is improved to  $(2 + R)/3$ . Note that if  $R = 1.1$  (the number of requests at each stage exceeds  $k$  by at most 10%), the CR becomes at most 1.034.

The basic idea of our algorithms is “greedy” and “balanced”. Both notions have already appeared in [9], but our implementation of them is significantly different from theirs. More importantly, our analysis is completely new; namely we use a simple mathematical induction (augmented by two interesting parameters other than the profit itself) while a classification of request types was used in [9].

The merit of our new analysis is demonstrated more clearly in  $kS2L-S$  than in the original  $kS2L-F$ . Therefore we present the results for  $kS2L-S$  first and then those for  $kS2L-F$ ; the deterministic case in Section 2 and the randomized case in Section 3.  $RkS2L-S$  is discussed in Section 4, where we introduce two magic numbers calculated from the number of requests in each stage. Finally all matching lower bounds are given in Section 5.

■ **Table 1** Overview of known and new results.

Problem	Booking Time	Start Time	The Cost Of Empty Move	Types of algorithms	Lower Bound	Upper Bound	Reference
2S2L	Fixed	$t_i$	$c = y$	Deterministic	—	1	MFCS'18 [8]
2S2L	Fixed	$t_i$	0	Deterministic	2	2	MFCS'18 [8]
2S2L	Fixed	$t_i$	$0 < c < y$	Deterministic	2	2	MFCS'18 [8]
$k$ S2L-F	Fixed	$t_i = vt$ for $v \in \mathbb{N}$	0	Deterministic	1.5	$1.5(k = 3i, i \in \mathbb{N})$	ISAAC'18 [9]
$k$ S2L-F	Fixed	$t_i = vt$ for $v \in \mathbb{N}$	0	Deterministic	$\frac{2k}{k+\lfloor k/3 \rfloor}$	$\frac{2k}{k+\lfloor k/3 \rfloor}$	this paper
$k$ S2L-F	Fixed	$t_i = vt$ for $v \in \mathbb{N}$	0	Randomized	1.5	1.5	this paper
$k$ S2L-V	Variant	$t_i = vt$ for $v \in \mathbb{N}$	0	Deterministic	1.5	$1.5(k = 3i, i \in \mathbb{N})$	ISAAC'18 [9]
$k$ S2L-V	Variant	$t_i = vt$ for $v \in \mathbb{N}$	0	Deterministic	5/3	$5/3(k = 5i, i \in \mathbb{N})$	ISAAC'18 [9]
$k$ S2L-S	Fixed	$t_i = vt$ for $v \in \mathbb{N}$	0	Deterministic	$\frac{2k}{k+\lfloor k/2 \rfloor}$	$\frac{2k}{k+\lfloor k/2 \rfloor}$	this paper
$k$ S2L-S	Fixed	$t_i = vt$ for $v \in \mathbb{N}$	0	Randomized	4/3	4/3	this paper
$Rk$ S2L-S ( $1 \leq R \leq 2$ )	Fixed	$t_i = vt$ for $v \in \mathbb{N}$	0	Randomized	$(2+R)/3$	$(2+R)/3$	this paper

## 2 Deterministic algorithms

As mentioned in the previous section, we first discuss the basic GBA that works for the  $k$ S2L-S model and then its accept/reject version that works for the original  $k$ S2L-F model. The analysis for the former will carry over to that of the latter pretty well. The following table summarizes our notations which are used in the rest of the paper.

### Notation

$k$	The number of total servers
$(0, 1)$ :	Requests from location 0 to 1
$(1, 0)$ :	Requests from location 1 to 0
$Il_i$ :	The number of $(0,1)$ 's requested in stage $i$ with start time $i$
$Ir_i$ :	The number of $(1,0)$ 's requested in stage $i$ with start time $i$
$Gl_i$ :	The number of $(0,1)$ 's accepted by the algorithm in stage $i$
$Gr_i$ :	The number of $(1,0)$ 's accepted by the algorithm in stage $i$
$Gf_i$ :	The number of servers not used, i.e., $k - Gr_i - Gl_i$
$Ol_i$ :	The number of $(0,1)$ 's accepted by OPT in stage $i$
$Or_i$ :	The number of $(1,0)$ 's accepted by OPT in stage $i$
$Of_i$ :	The number of servers not used, i.e., $k - Or_i - Ol_i$

Suppose there are  $x$  and  $x'$  servers at location 0 and  $y$  and  $y'$  servers at location 1 at time  $i$ , where  $x$  servers will serve  $(0,1)$ 's with start time  $i$  but  $x'$  servers are not used. Similarly,  $y$  servers will serve  $(1,0)$ 's but  $y'$  servers not. Then at time  $i + 1$  we can use  $x$  servers for  $(1,0)$ 's and  $y$  servers for  $(0,1)$ 's. Furthermore,  $x' + y'$  servers are available for requests of both directions since the requests with start time  $i + 1$  come at time  $i$  and we can move  $x' + y'$  servers to whichever locations as we like. Now we introduce *stages* and say we have  $x$  ( $= Gl_i$ ) servers at location 1,  $y$  ( $= Gr_i$ ) servers at location 0, and  $x' + y'$  ( $= Gf_i$ ) “floating” servers in stage  $i + 1$ . We also denote the *server allocation* at time  $i + 1$  as  $[y, x' + y', x]$  using the new notation  $[-, -, -]$ . Note that we need to have artificial  $Gr_0, Gl_0$  and  $Gf_0$  for stage 1 whose values are 0, 0, and  $k$ , respectively.

Now the idea of this new greedy balanced algorithm can be illustrated as follows. Let  $k = 100$  and suppose requests in stage 1 are  $(Il_1, Ir_1) = (100, 100)$ . As described above, the server allocation in stage 1 is  $[0, 100, 0]$ , and hence accepted requests,  $(Gl_1, Gr_1)$ , can be anything like  $(100, 0)$ ,  $(75, 25)$  or  $(0, 100)$ . However, if  $(100, 0)$  is selected, then the server allocation in stage 2 is  $[0, 0, 100]$  and the adversary would send  $(Il_2, Ir_2) = (100, 0)$  for

stage 2, by which no servers are available for the online player. Since the almighty adversary can select  $(0,100)$  in stage 1 and he/she can accept all the requests in stage 2, the CR would be 2. Thus one can easily see that the best thing an algorithm can do is to accept  $(Gl_1, Gr_1) = (50, 50)$  in stage 1 to secure a CR of 1.5. This is the notion of “Balanced”. Note that requests denoted by  $(Il_2, Ir_2) = (100, 0)$  have booking time 1, but a booking time of requests (and when they actually come) is no longer important once we know  $(Il_i, Ir_i)$  and the server allocation  $[Gr_{i-1}, Gf_{i-1} = k - Gl_{i-1} - Gr_{i-1}, Gl_{i-1}]$  in stage  $i$ . Thus we will simply use “requests for stage  $i$ ” without mentioning their booking time.

What if  $(Il_1, Ir_1) = (60, 20)$ ? In this case,  $(Gl_1, Gr_1) = (60, 20)$  is the best, i.e., the strategy is a simple “Greedy” one. If  $(Il_1, Ir_1) = (100, 30)$ , our selection is  $(Gl_1, Gr_1) = (70, 30)$ , namely “Greedy” but as “Balanced” as possible. Algorithm 1 realizes this idea almost as it is and it will also be a core of all the subsequent algorithms in this paper.

■ **Algorithm 1** GBA( $k$ ): Greedy Balanced Algorithm.

---

**Input:**  $Il_i$  and  $Ir_i$  are the numbers of  $(0,1)$ 's and  $(1,0)$ 's in stage  $i$ , respectively. An integer  $k$  is the number of total servers. The server allocation at the beginning of stage  $i$  is  $[Gr_{i-1}, Gf_{i-1}, Gl_{i-1}]$ , namely  $Gr_{i-1}$  and  $Gl_{i-1}$  servers at locations 0 and 1, respectively and  $Gf_{i-1} = k - Gl_{i-1} - Gr_{i-1}$  floating servers.

**Output:**  $Gl_i$  and  $Gr_i$  are the numbers of accepted  $(0,1)$ 's and  $(1,0)$ 's, respectively.

```

1: if  $Gr_{i-1} + Gf_{i-1} \leq \lfloor k/2 \rfloor$  or  $Il_i \leq \lfloor k/2 \rfloor$  then
2:    $Gl_i \leftarrow \min\{Il_i, Gr_{i-1} + Gf_{i-1}\}$ ;  $Gr_i \leftarrow \min\{Ir_i, Gl_{i-1} + Gf_{i-1}, k - Gl_i\}$ ;
3: else
4:   if  $Gl_{i-1} + Gf_{i-1} \leq \lfloor k/2 \rfloor$  or  $Ir_i \leq \lfloor k/2 \rfloor$  then
5:      $Gr_i \leftarrow \min\{Ir_i, Gl_{i-1} + Gf_{i-1}\}$ ;  $Gl_i \leftarrow \min\{Il_i, Gr_{i-1} + Gf_{i-1}, k - Gr_i\}$ ;
6:   else
7:      $Gr_i \leftarrow \lfloor k/2 \rfloor$ ;  $Gl_i \leftarrow \lceil k/2 \rceil$ ;
8:   end if
9: end if
10: return  $Gl_i$  and  $Gr_i$ 

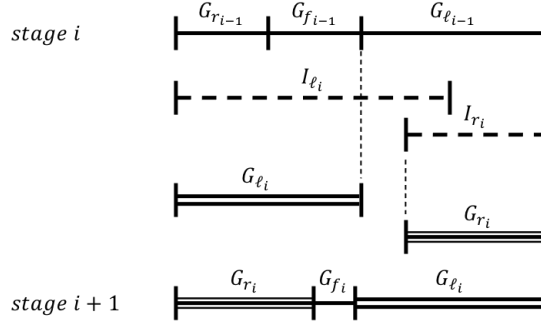
```

---

Recall that floating servers in stage  $i$  are actually sit at location 0 or 1 at time  $i - 1$ , say 30 ones at location 0 and 10 at location 1 among 40 floating ones. So if we need 20 floating servers in stage  $i$  at location 1, we need to move 10 servers from location 0 to 1 using the duration from time  $i - 1$  to  $i$ . However, we do not describe this empty movement in our algorithms since it is easily seen and its cost is free in our model.

Observe that the greedy part of Algorithm 1 appears in lines 1 and 2 for  $(0,1)$ 's and in lines 4 and 5 for  $(1,0)$ 's. If the condition in line 1 is met, then we accept  $(0,1)$ 's until we have exhausted the servers or the  $(0,1)$ 's available in this stage. After that we accept as many  $(1,0)$ 's as possible. It works similarly for lines 4 and 5. If neither the condition in line 1 nor the one in line 4 is met, we just split the requests almost evenly in line 7. The following theorem shows that GBA achieves the optimal  $\frac{4}{3}$ -competitiveness for all even  $k$  and approaches this value when  $k$  is a large odd number. In the rest of this paper, we use ALG to denote an online algorithm and OPT an offline optimal scheduler in general.

► **Theorem 1.** *GBA is a  $1/\delta$ -competitive algorithm for  $kS2L-S$  for any  $k \geq 2$ , where  $\delta = \frac{k + \lfloor k/2 \rfloor}{2k}$ .*



■ **Figure 2** Server allocation in GBA.

**Proof.** In order to prove the theorem, we consider the following six key values:

$$A_i = \sum_{j=1}^i (Gr_j + Gl_j), \quad B_i = \sum_{j=1}^i (Or_j + Ol_j),$$

$$X_i = A_i + Gr_i + Gf_i, \quad Y_i = B_i + Or_i + Of_i,$$

$$U_i = A_i + Gl_i + Gf_i, \quad V_i = B_i + Ol_i + Of_i.$$

Our goal is to bound  $A_i$  by  $B_i$ . To do so, it is popular to use a potential function for competitive analysis, which is typically the difference between configurations of ALG and OPT. In our present case, it may be the difference between server allocations of GBA and OPT. It turns out, however, that this configuration difference or a similar one is unlikely to work since we still have a freedom for server selection which is not controlled by this difference strongly. Instead we introduce four parameters,  $X_i$ ,  $Y_i$ ,  $U_i$  and  $V_i$ , which play a key role in our proof. Note that  $X_i$  and  $Y_i$  denote the total revenue of GBA and OPT respectively for the first  $i+1$  stages assuming that the adversary tries to penalize the algorithm choice by introducing  $k$   $(0,1)$ 's in stage  $i+1$ ; the last two values,  $U_i$  and  $V_i$ , denote the total revenue of GBA and OPT respectively for the first  $i+1$  stages assuming that the adversary tries to penalize the algorithm choice by introducing  $k$   $(1,0)$ 's in stage  $i+1$ . Intuitively, GBA balances the accepted requests in both directions and guarantees that the CR's in these two instances ( $\frac{Y_i}{X_i}$  and  $\frac{V_i}{U_i}$ , respectively) are not too large. It turns out that taking care of these two extreme instances is sufficient to keep the CR low for all instances.

In order to prove that the algorithm is  $1/\delta$ -competitive, we show that the set of the following inequalities (i) to (iii), denoted by  $S(n)$ ,

$$(i) A_n \geq \delta B_n, \quad (ii) X_n \geq \delta Y_n, \quad (iii) U_n \geq \delta V_n$$

hold for every  $n$  by induction.

For the base case,  $n = 0$ , we have  $A_0 = B_0 = Gr_0 = Gl_0 = Or_0 = Ol_0 = 0$  and  $Gf_0 = Of_0 = k$ . Thus the three inequalities hold since  $\delta \leq 1$ .

Now the main part of the proof is proving  $S(n)$  assuming, as stated in the induction hypothesis, that  $S(j)$  holds for all  $0 \leq j \leq n-1$ . Note that we can rewrite  $A_i$ ,  $B_i$ ,  $X_i$  and so on as follows:

$$A_i = A_{i-1} + Gl_i + Gr_i, \quad B_i = B_{i-1} + Ol_i + Or_i,$$

$$X_i = A_{i-1} + k + Gr_i, \quad Y_i = B_{i-1} + k + Or_i,$$

$$U_i = A_{i-1} + k + Gl_i, \quad V_i = B_{i-1} + k + Ol_i.$$



Since  $Ol_i \leq \min\{Il_i, Or_{i-1} + Of_{i-1}\}$  and  $Or_i \leq \min\{Ir_i, Ol_{i-1} + Of_{i-1}\}$ , the following lemma is obvious, but will be used frequently.

► **Lemma 2.**  $Ol_i \leq Il_i$ ,  $Ol_i \leq Or_{i-1} + Of_{i-1}$ ,  $Or_i \leq Ir_i$ , and  $Or_i \leq Ol_{i-1} + Of_{i-1}$ .

Now we are ready to prove the theorem. Suppose line 2 is executed. Then the following (L1) or (L2) holds for the value of  $G\ell_i$  and (R1), (R2) or (R3) for the value of  $Gr_i$ . Similarly if line 5 is executed, (R1) or (R2) holds for  $Gr_i$  and (L1), (L2) or (L3) for  $G\ell_i$ .

$$\begin{aligned} \text{(L1)} \quad G\ell_i &= Il_i, \quad \text{(L2)} \quad G\ell_i = Gr_{i-1} + Gf_{i-1}, \quad \text{(L3)} \quad G\ell_i = k - Gr_i (\geq \lfloor k/2 \rfloor), \\ \text{(R1)} \quad Gr_i &= Ir_i, \quad \text{(R2)} \quad Gr_i = G\ell_{i-1} + Gf_{i-1}, \quad \text{(R3)} \quad Gr_i = k - G\ell_i (\geq \lfloor k/2 \rfloor). \end{aligned}$$

Note that the condition “ $\geq \lfloor k/2 \rfloor$ ” in (L3) and (R3) comes from the conditions in lines 1 and 4, respectively. Now we consider stage  $n$  and show that if (L1), (L2) or (L3) holds, the induction (iii) holds, if (R1), (R2) or (R3) hold, the induction (ii) holds and if any one of the nine combinations  $\{(L1), (L2), (L3)\} \times \{(R1), (R2), (R3)\}$  holds, (i) holds. First suppose (L1) holds. Then since  $Ol_n \leq Il_n$  by Lemma 2

$$\begin{aligned} U_n &= A_{n-1} + k + G\ell_n = A_{n-1} + k + Il_n, \\ V_n &= B_{n-1} + k + Ol_n \leq B_{n-1} + k + Il_n. \end{aligned}$$

Thus (iii) is true by the induction hypothesis on (i). Similarly for (L2), i.e., by Lemma 2

$$\begin{aligned} U_n &= A_{n-1} + k + G\ell_n = A_{n-1} + k + Gr_{n-1} + Gf_{n-1}, \\ V_n &= B_{n-1} + k + Ol_n \leq B_{n-1} + k + Or_{n-1} + Of_{n-1}. \end{aligned}$$

Thus (iii) is proved by the hypothesis on (ii). Finally for (L3),

$$\begin{aligned} U_n &= A_{n-1} + k + G\ell_n \geq A_{n-1} + k + \lfloor k/2 \rfloor, \\ V_n &= B_{n-1} + k + Ol_n \leq B_{n-1} + k + k, \end{aligned}$$

then use the hypothesis on (i) to claim (iii).

The proof that (R1) or (R2) or (R3) implies (ii) is similar and omitted.

Finally we show that (i) follows from any combination. Observe that (L1) and (R1) obviously implies (i) since no algorithms accept more requests than requested. All combinations including (L3) or (R3) are also obvious since GBA accepts  $k$  requests. Similarly for (L2) and (R2) when  $Gf_{i-1} = 0$  (otherwise impossible). The remaining cases are (L1) and (R2), and (L2) and (R1). For the former, by Lemma 2

$$\begin{aligned} A_n &= A_{n-1} + G\ell_n + Gr_n = A_{n-1} + Il_n + G\ell_{n-1} + Gf_{n-1}, \\ B_n &= B_{n-1} + Ol_n + Or_n \leq B_{n-1} + Il_n + Ol_{n-1} + Of_{n-1}, \end{aligned}$$

and we can use the hypothesis on (iii). (L2) and (R1) is similar and omitted.

What remains is the case that line 7 is executed. Observe that line 7 gives us  $G\ell_n \geq \lfloor k/2 \rfloor$ ,  $Gr_n \geq \lfloor k/2 \rfloor$  and  $G\ell_n + Gr_n = k$ . We have already shown that the first one implies (iii), the second one (ii) and the third one means all the servers accept requests and is obviously enough for (i). Thus the theorem is proved. ◀

As seen in GBA and its analysis, a dangerous situation for the online player is that ALG accepts too many requests of one direction when it is possible. If ALG knows the total number of requests in each direction in advance, we can avoid this situation rather easily. Now we discuss  $kS2L-F$ , in which ALG does not know the total number of requests in advance. A simple and apparent solution is to stop accepting requests of one direction when its number gets to some value, even if more requests of that direction are coming and could

be accepted. In the next algorithm, ARGBA, we set this value as  $2k/3$ . It then turns out, a little surprisingly, that the analysis for Theorem 1 is also available for the new algorithm almost as it is.

■ **Algorithm 2** ARGBA( $k$ ): Accept or reject GBA.

**Input:** The server location is  $[Gr_{i-1}, Gf_{i-1} = k - Gl_{i-1} - Gr_{i-1}, Gl_{i-1}]$  at the beginning of this stage  $i$ . Requests are coming sequentially, each of which,  $r$ , is (0,1) or (1,0).  $k$  is the total number of servers.

**Output:** Immediate accept or reject for  $r$ .  $Gl_i$  and  $Gr_i$  for the next server allocation.

```

1:  $Gl_i \leftarrow 0$ ;  $Gr_i \leftarrow 0$ ,  $Al_i \leftarrow 0$ ;  $Ar_i \leftarrow 0$  ( $Al_i$  ( $Ar_i$ , resp.) is the number of (0,1)'s ((1,0)'s, resp.) received in this stage so far.)
2: while a new request  $r$  comes do
3:   if  $r$  is (0,1) then
4:      $Al_i \leftarrow Al_i + 1$ ;
5:     if  $Al_i < Gr_{i-1} + Gf_{i-1}$  and  $Al_i < 2k/3$  and  $Gl_i + Gr_i < k$  then
6:       accept  $r$ ;  $Gl_i \leftarrow Gl_i + 1$ ;
7:     else
8:       reject  $r$ ;
9:     end if
10:  else (namely,  $r$  is (1,0))
11:     $Ar_i \leftarrow Ar_i + 1$ ;
12:    if  $Ar_i < Gl_{i-1} + Gf_{i-1}$  and  $Ar_i < 2k/3$  and  $Gl_i + Gr_i < k$  then
13:      accept  $r$ ;  $Gr_i \leftarrow Gr_i + 1$ ;
14:    else
15:      reject  $r$ ;
16:    end if
17:  end if
18: end while
19: return  $Gl_i$  and  $Gr_i$ 

```

► **Theorem 3.** ARGBA is a  $1/\delta$ -competitive algorithm for  $kS2L-F$  for any  $k \geq 2$ , where  $\delta = \frac{k + \lfloor k/3 \rfloor}{2k}$ .

**Proof.** Observe that once a (0,1) (similarly for (1,0)) is rejected, then subsequent ones are all rejected. Let  $X_i$  and  $Y_i$  be the last (0,1) and (1,0), respectively, that are accepted and  $Gl_i$  and  $Gr_i$  be their numbers after lines 6 and 10, respectively. Also, let  $Il_i$  and  $Ir_i$  be the total numbers of (0,1)'s and (1,0)'s in stage  $i$ , respectively (only used for analysis). Suppose the last accepted (0,1) has gone through the conditions in line 5. Then, one can see that the next (0,1), if any, is blocked by one of these conditions and thus one of the following four conditions, (L1) through (L4), is met. Similarly for (1,0)'s, one of (R1) through (R4) is met.

$$\begin{aligned}
& \text{(L1) } Gl_i = Il_i, \quad \text{(L2) } Gl_i = Gr_{i-1} + Gf_{i-1}, \quad \text{(L3) } Gl_i = \lceil 2k/3 \rceil, \\
& \quad \quad \quad \text{(L4) } Gl_i + Gr_i = k \text{ and } Gl_i \geq \lfloor k/3 \rfloor \text{ and } Gr_i \geq \lfloor k/3 \rfloor, \\
& \text{(R1) } Gr_i = Ir_i, \quad \text{(R2) } Gr_i = Gl_{i-1} + Gf_{i-1}, \quad \text{(R3) } Gr_i = \lceil 2k/3 \rceil, \\
& \quad \quad \quad \text{(R4) } Gl_i + Gr_i = k \text{ and } Gl_i \geq \lfloor k/3 \rfloor \text{ and } Gr_i \geq \lfloor k/3 \rfloor.
\end{aligned}$$

Note that the lower bound condition in (L4) and (R4) is correct since otherwise the second condition in line 5 or 11 should have been met before.

Now consider stage  $n$ . In a way similar to the proof of Theorem 1, we can show that one of (L1) to (L4) (with subscript  $n$  replacing  $i$ ) implies the induction (iii). In fact, the reason is exactly the same for (L1) and (L2) as before. Using  $G\ell_i \geq \lfloor k/3 \rfloor$  in (L4) we have

$$\begin{aligned} U_n &= A_{n-1} + k + G\ell_n \geq A_{n-1} + k + \lfloor k/3 \rfloor, \\ V_n &= B_{n-1} + k + O\ell_n \leq B_{n-1} + k + k, \end{aligned}$$

and then use the hypothesis on (i) to claim (iii) (recall that our target CR is relaxed to  $\frac{2k}{k + \lfloor k/3 \rfloor}$ ). (L3) obviously implies  $G\ell_i \geq \lfloor k/3 \rfloor$ .

Also we can show, though omitted, that one of (R1) to (R4) (with subscript  $n$  replacing  $i$ ) implies the induction (ii).

We can furthermore show that any one of the 16 combinations of (L1) to (L4) and (R1) to (R4) implies (i): If a combination includes one of (L3), (L4), (R3) and (R4), then (i) is obvious since ARGBA accepts at least  $\lceil 2k/3 \rceil$  requests in this stage. So we only have to consider the four combinations ((L1) or (L2), and (R1) or (R2)), and these cases already appeared in the proof of Theorem 1, which concludes the proof. ◀

### 3 Randomized Algorithms

Notice that the CR of GBA is 2 when  $k = 2$ . The reason is simple, i.e., the existence of the ceiling function, namely if we can accept a fractional request, our CR would be  $4/3$ . Of course it is impossible to accept a request by one third, but it is possible to accept that request with probability  $1/3$ , which has the same effect as accepting it by one third in terms of an expected number.

We define the following function for probabilistic rounding. Let  $x$  be a (possibly fractional) non-negative number. Then define

$$\text{prrd}(x) = \begin{cases} \lceil x \rceil & \text{with probability } x - \lfloor x \rfloor \\ \lfloor x \rfloor & \text{with probability } 1 - (x - \lfloor x \rfloor) \end{cases}$$

For instance,  $\text{prrd}(3.3)$  is 4 with probability 0.3 and 3 with probability 0.7.  $\text{prrd}(3)$  is always 3. Note  $E[\text{prrd}(x)] = x$ . Now we are ready to introduce the Probabilistic GBA.

■ **Algorithm 3** PrGBA( $k$ ): Probabilistic GBA.

---

1: The same as Algorithm 1 except that line 7 is replaced as follows:  $Gr_i \leftarrow \text{prrd}(k/2)$  and  $G\ell_i \leftarrow k - Gr_i$

---

▶ **Theorem 4.** *PrGBA is a  $4/3$ -competitive algorithm for  $kS2L$ -S for any  $k \geq 2$ .*

**Proof.** Observe the induction in the proof of the deterministic case. The base case is fine with  $\delta = 3/4$ , and we can keep using this  $3/4$  unless line 7 is executed. Since the expected value of  $Gr_i$  and  $G\ell_i$  are both  $k/2$  when the modified line 7 is executed, we can remove the ceiling sign from the description of the algorithm. Thus our new CR is  $2k/(k + k/2) = 4/3$ . ◀

■ **Algorithm 4** PrARGBA( $k$ ): Probabilistic ARGBA.

---

1: ARGBA is modified as follows: Suppose the three conditions in line 5 are all met and  $0 < 2k/3 - A\ell^{j-1} < 1$ . Then accept  $r_i^j$  and increase  $G\ell_i$  if  $\text{prrd}(2k/3 - A\ell^{j-1}) = 1$ , otherwise reject it. Similarly for line 11.

---

► **Theorem 5.** *PrARGBA is a 1.5-competitive algorithm for  $kS2L-F$  for any  $k \geq 2$ .*

**Proof.** The same idea as the proof of Theorem 4. If this modified part is executed, the expected value of  $Gl_i$  is  $2k/3$  and hence the expected value of  $Gr_i$  is  $k/3$  if  $Gl_i + Gr_i = k$ . Thus the worst case of deterministic ARGBA,  $Gl_i = \lceil 2k/3 \rceil$  and  $Gr_i = \lfloor k/3 \rfloor$ , can be avoided. ◀

#### 4 Adaptive GBA

$kS2L-S$  gives the online player the advantage of an advance knowledge of the number of requests in the current stage. GBA does exploit this advantage, but not fully. Suppose  $(Il_1, Ir_1) = (50, 100)$ . GBA accepts the same number, 50, of  $(0,1)$ 's and  $(1,0)$ 's in stage 1. Then the adversary sends  $(Il_2, Ir_2) = (100, 0)$ , resulting in that only 50  $(0,1)$ 's can be accepted by GBA in stage 2, but 100  $(0,1)$ 's by OPT which could accept 100  $(1,0)$ 's in stage 1. Thus the CR in these two steps is  $4/3$ .

Now what about accepting roughly 28.57  $(0,1)$ 's and 71.43  $(1,0)$ 's in stage 1 (recall we can handle fractional numbers due to randomized rounding)? Then the best the adversary can do is to provide  $(Il_2, Ir_2) = (100, 0)$  or  $(0, 50)$ , in both of which the CR is  $200/171.43 \approx 150/128.57 \approx 1.17$ , significantly better than 1.5 of GBA. This is the basic idea of our new algorithm, AGBA. These key values 28.57 and 71.43 are denoted by  $\alpha_i$  and  $\beta_i$ , respectively. The ultimate goal of AGBA is to accept exactly  $\alpha_i$   $(0,1)$ 's and  $\beta_i$   $(1,0)$ 's while the ultimate goal of GBA was to accept  $k/2$   $(0,1)$ 's and  $k/2$   $(1,0)$ 's. If this goal is unachievable, to be figured out from the values of  $Il_i, Ir_i, Gl_{i-1}, Gr_{i-1}$  and  $Gf_{i-1}$ , both algorithms simply turn greedy.

■ **Algorithm 5** AGBA( $k$ ): Adaptive GBA.

**Input:** The same as GBA

**Output:** The same as GBA

```

1: if  $R_i = (Il_i + Ir_i)/k \geq 1$  then
2:    $\alpha_i = \frac{(1-R_i)k+3Il_i}{2+R_i}; \beta_i = \frac{(1-R_i)k+3Ir_i}{2+R_i};$ 
3: else
4:    $\alpha_i = Il_i; \beta_i = Ir_i;$ 
5: end if
6: if  $Gr_{i-1} + Gf_{i-1} < \alpha_i$  then
7:    $Gl_i \leftarrow Gr_{i-1} + Gf_{i-1}; Gr_i \leftarrow \min\{Ir_i, Gl_{i-1}\};$ 
8: else
9:   if  $Gl_{i-1} + Gf_{i-1} < \beta_i$  then
10:     $Gl_i \leftarrow \min\{Il_i, Gr_{i-1}\}; Gr_i \leftarrow Gl_{i-1} + Gf_{i-1};$ 
11:   else
12:     $Gl_i \leftarrow \text{prrd}(\alpha_i);$ 
13:     $Gr_i \leftarrow \beta_i;$ 
14:   end if
15: end if
16: return  $Gl_i$  and  $Gr_i$ 

```

Suppose  $Il_i + Ir_i$  is at most  $kR_i$  for stage  $i$ . As mentioned in a moment, we do not lose generality if  $R_i$  is restricted to  $1 \leq R_i \leq 2$ , under which the CR of AGBA is bounded by  $(2+R)/3$ , where  $R$  is the maximum value of  $R_i$  in the whole stages. We do not know  $R$  in advance and AGBA does not have to, either.

The competitive analysis is given by the following theorem. Note that if the input in stage  $i$  includes more than  $k$  (0,1)'s, we can select an arbitrary subset of size  $k$  and similarly for (1,0)'s. This guarantees that  $R \leq 2$  and the case that  $Il_i + Ir_i < k$  is covered by  $R = 1$ . Thus the restriction of  $R$ ,  $1 \leq R \leq 2$ , makes sense. Also, note that  $\alpha_i + \beta_i = k$  whenever  $R_i \geq 1$ . AGBA uses  $R_i$  but not  $R$ .  $R = \max\{R_i\}$  by definition and so  $\delta \leq \delta_i$  (where  $\delta_i$  is the local value of  $\delta$  in this stage, see its definition at Lemma 8) for all  $i$ .

► **Theorem 6.** *Suppose the number of requests is limited to at most  $Rk$  in all stages for some  $R$  such that  $1 \leq R \leq 2$  and  $Rk$  is an integer. Then AGBA solves  $RkS2L-S$  and is  $1/\delta$ -competitive, where  $\delta = 3/(2 + R)$ .*

**Proof.** Because of the probabilistic rounding used in lines 12, the expected values of  $G\ell_i$  and  $Gr_i$  are  $\alpha_i$  and  $\beta_i$ , respectively, if this part is executed. We need two lemmas; the second one illustrate a tricky nature of  $\alpha_i$  and  $\beta_i$ .

► **Lemma 7.** *For any  $R_i \geq 0$ ,  $\alpha_i \leq Il_i$  and  $\beta_i \leq Ir_i$ . Also,  $\alpha_i + \beta_i \leq k$ .*

**Proof.** A simple calculation of the formulas in line 2 is enough if the condition  $Il_i + Ir_i \geq k$  is met. Otherwise it is also obvious by line 4. ◀

► **Lemma 8.** *Let  $\delta_i = \min\{1, 3/(2 + R_i)\}$ . Then  $k + \beta_i = \delta_i(k + Ir_i)$  and  $k + \alpha_i = \delta_i(k + Il_i)$ .*

**Proof.** If  $R_i < 1$  then  $\delta_i = 1$ ,  $\alpha_i = Il_i$  and  $\beta_i = Ir_i$ , so the lemma is obviously true. Otherwise, a simple calculation:

$$k + \alpha_i = k + \frac{(1 - R_i)k + 3Il_i}{2 + R_i} = \frac{3k + 3Il_i}{2 + R_i} = \delta_i(k + Il_i).$$

Similarly for the other. ◀

The basic strategy of the proof is the same as Theorem 1. We consider the following four conditions for the values of  $G\ell_i$  and  $Gr_i$ . Suppose line 7 is executed. Then  $G\ell_i$  satisfies (L2) and  $Gr_i$  satisfies (R1). Similarly if line 10 is executed, (L1) and (R2) hold.

$$(L1) G\ell_i = \min\{Il_i, Gr_{i-1}\}, (L2) G\ell_i = Gr_{i-1} + Gf_{i-1},$$

$$(R1) Gr_i = \min\{Ir_i, G\ell_{i-1}\}, (R2) Gr_i = G\ell_{i-1} + Gf_{i-1}.$$

Now consider stage  $n$ . It is shown that (L1) or (L2) implies (iii) of the induction. For (L2), the analysis is the same as before and omitted. For (L1), using Lemma 2 for  $V_n$ , we have

$$U_n = A_{n-1} + k + G\ell_n = A_{n-1} + k + \min\{Il_n, Gr_{n-1}\},$$

$$V_n = B_{n-1} + k + Ol_n \leq B_{n-1} + k + Il_n.$$

If  $\min\{Il_n, Gr_{n-1}\} = Il_n$ , then we are done using the hypothesis (i). Otherwise recall that the condition of line 9,  $G\ell_{n-1} + Gf_{n-1} < \beta_n$ , is met. So we have  $Gr_{n-1} = k - (G\ell_{n-1} + Gf_{n-1}) \geq k - \beta_n \geq \alpha_n$  (by Lemma 7 for the last inequality) and thus we can use Lemma 8 and the hypothesis on (i) to claim (iii). The proof that (R1) or (R2) implies (ii) is very similar and omitted.

Next we prove that each of the four combinations implies (i). (L1) and (R1), and (L2) and (R2) are obvious since AGBA is as efficient as OPT or accepts  $k$  requests (recall  $\min\{Il_n, Gr_{n-1}\} \geq \alpha_n$  and  $\min\{Ir_n, G\ell_{n-1}\} \geq \beta_n$  mentioned above and by Lemma 7). For (L1) and (R2), using Lemma 2 and  $\alpha_n \leq Il_n$ , we have

$$A_n = A_{n-1} + G\ell_n + Gr_n = A_{n-1} + \min\{I\ell_n, Gr_{n-1}\} + G\ell_{n-1} + Gf_{n-1},$$

$$B_n = B_{n-1} + O\ell_n + Or_n \leq B_{n-1} + I\ell_n + O\ell_{n-1} + Of_{n-1}.$$

Thus the hypothesis on (iii) or  $Gr_{n-1} + G\ell_{n-1} + Gf_{n-1} = k$  implies that  $A_n \geq \delta B_n$ . (L2) and (R1) are similar.

The remaining case is the one that lines 12 and 13 are executed. If  $G\ell_n = \alpha_n$ , we have  $U_n = A_{n-1} + k + \alpha_n$  and thus we can use Lemma 8 to claim (iii) as shown above. Similarly for  $Gr_n$  and (ii). (i) is obvious since  $G\ell_i + Gr_i = k$ , completing the proof. ◀

## 5 CR Lower Bounds

The CR's given so far are all tight. In this section we prove matching lower bounds for  $kS2L-S$ , for  $kS2L-F$ , for  $kS2L-F$  with randomization, and for  $RkS2L-S$  with randomization (including for  $kS2L-S$  with randomization as a special case).

► **Theorem 9.** *No deterministic online algorithms for the  $kS2L-S$  problem can achieve a CR of less than  $\frac{2k}{k+\lceil k/2 \rceil}$ .*

**Proof.** Let  $\mathcal{A}$  be any deterministic algorithm. The adversary requests  $k$  (0,1)'s and  $k$  (1,0)'s in stage 1.  $\mathcal{A}$  accepts  $k_\ell$  (0,1)'s and  $k_r$  (1,0)'s. If  $k_\ell \leq \lfloor k/2 \rfloor$ , then the adversary requests  $k$  (1,0)'s (and zero (0,1)'s) in stage 2. The profit of  $\mathcal{A}$  is  $k_\ell + k_r$  in stage 1, and at most  $(k - k_\ell - k_r) + k_\ell$  in stage 2. Therefore, the total profit of  $\mathcal{A}$  is at most  $k + k_\ell \leq k + \lfloor k/2 \rfloor$ . The profit of OPT is  $2k$ , and the theorem is proved. If  $k_\ell > \lfloor k/2 \rfloor$ , then  $k_r \leq \lfloor k/2 \rfloor$ . Now the adversary requests  $k$  (0,1)'s in stage 2. The profit of  $\mathcal{A}$  and OPT are exactly the same as above and we may omit the rest of calculation. Thus the bound is tight. ◀

► **Theorem 10.** *No deterministic online algorithms for the  $kS2L-F$  problem can achieve a CR of less than  $\frac{2k}{k+\lceil k/3 \rceil}$ .*

**Proof.** Let  $\mathcal{A}$  be any deterministic algorithm. The basic idea is similar to [9]. The adversary gives  $k$  (0,1)'s (sequentially) for stage 1. If  $\mathcal{A}$  accepts at most  $\lfloor 2k/3 \rfloor$  ones, then the adversary stops his/her requests and the game ends. Thus the CR is at least  $\frac{k}{\lfloor 2k/3 \rfloor}$ . Otherwise, if  $\mathcal{A}$  accepts  $\lfloor 2k/3 \rfloor$  or more, then the adversary gives another  $k$  (1,0)'s for stage 1 and  $k$  (0,1)'s for stage 2. Since  $\mathcal{A}$  has accepted at least  $\lfloor 2k/3 \rfloor$  (0,1)'s in stage 1,  $\mathcal{A}$  cannot use those servers for the (0,1)'s for stage 2. Hence  $\mathcal{A}$  can accept at most  $k$  requests in stage 1 and at most  $k - \lfloor 2k/3 \rfloor$  requests in stage 2, meaning at most  $2k - \lfloor 2k/3 \rfloor = k + \lfloor k/3 \rfloor$  requests in total. OPT can accept  $k$  (1,0)'s in stage 1 and  $k$  (0,1)'s in stage 2, i.e.,  $2k$  in total. Thus the CR is at least  $\frac{2k}{k+\lceil k/3 \rceil}$ . Since  $\frac{2k}{k+\lceil k/3 \rceil} \leq \frac{k}{\lfloor 2k/3 \rfloor}$  for all  $k$  (this can be verified by checking for  $k = 3j$ ,  $k = 3j + 1$  and  $k = 3j + 2$ ), the theorem is proved. ◀

► **Theorem 11.** *No randomized online algorithms for the  $kS2L-F$  problem can achieve a CR of less than 1.5.*

**Proof.** The proof is almost the same as that of Theorem 10. Since the adversary has the full information (other than random values) of ALG, he/she can compute the expected value of ALG's output. So what we have to do is just removing the floor and ceiling signs from the previous proof and considering the resulting numbers as expected values. The proof is complete since the previous deterministic OPT has a profit of at least  $2k$ . ◀

► **Theorem 12.** *No randomized online algorithms for the  $RkS2L-S$  problem can achieve a CR of less than  $\frac{2+R}{3}$ .*

**Proof.** Let  $\mathcal{A}$  be any randomized algorithm. The adversary requests  $\lfloor Rk/2 \rfloor$  (0,1)’s and  $\lceil Rk/2 \rceil$  (1,0)’s in stage 1 ( $\lfloor Rk/2 \rfloor + \lceil Rk/2 \rceil = Rk$  by the integrality condition).  $\mathcal{A}$  accepts  $k\ell$  (0,1)’s and  $kr$  (1,0)’s. Let  $\alpha = \frac{(1-R)k + 3\lfloor \frac{Rk}{2} \rfloor}{2+R}$  and if  $E[k\ell] \leq \alpha$  (note that the adversary has the full information of  $\mathcal{A}$ , so it can compute  $E[k\ell]$ ), then the adversary requests  $k$  (1,0)’s (and zero (0,1)’s) in stage 2. The profit of  $\mathcal{A}$  is at most

$$k + E[k\ell] \leq k + \alpha = k + \frac{(1-R)k + 3\lfloor \frac{Rk}{2} \rfloor}{2+R} = \frac{3(k + \lfloor \frac{Rk}{2} \rfloor)}{2+R}.$$

The profit of OPT is  $k + \lfloor Rk/2 \rfloor$ , and the theorem is proved. If  $E[k\ell] > \alpha$  then let  $\beta = \frac{(1-R)k + 3\lceil \frac{Rk}{2} \rceil}{2+R}$ , and it is easy to see that  $\alpha + \beta = k$ . Hence we have  $E[kr] \leq \beta$  because  $E[k\ell] + E[kr] = E[k\ell + kr] \leq k$ . Now the adversary requests  $k$  (0,1)’s. The profit of  $\mathcal{A}$  and OPT are exactly the same as above by replacing  $\lfloor \frac{Rk}{2} \rfloor$  with  $\lceil \frac{Rk}{2} \rceil$ . We may omit the rest of calculation. Note that although the two input instances provide a tight lower bound for the competitive ratio, applying Yao’s Minimax theorem on any probability distribution over these two input instances does not provide the same tight bound. ◀

## 6 Concluding Remarks

We have presented a different greedy and balanced algorithm with a new analysis which fully exploits the notion of “floating servers,” and the arguments of the mathematical induction using supplementary parameters  $X_i$  and  $U_i$  (and their OPT counterparts). We believe that the analysis technique is powerful for future studies of many extensions of the car sharing problem, which would be worthwhile to investigate for practical purposes. In particular, relaxation of the rental period condition should be challenging and important.

---


### References

- 1 Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS’00)*, pages 639–650, 2000.
- 2 Antje Bjelde, Yann Disser, Jan Hackfeld, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Kevin Schewior, Miriam Schlöter, and Leen Stougie. Tight bounds for online tsp on the line. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’17)*, pages 994–1005, 2017. doi:10.1137/1.9781611974782.63.
- 3 Kateřina Böhmová, Yann Disser, Matúš Mihalák, and Rastislav Šrámek. Scheduling transfers of resources over time: Towards car-sharing with flexible drop-offs. In *Proceedings of the 12th Latin American Symposium on Theoretical Informatics (LATIN’16)*, volume 9644 of *LNCS*, pages 220–234, 2016.
- 4 Ananya Christman, William Forcier, and Aayam Poudel. From theory to practice: maximizing revenues for on-line dial-a-ride. *Journal of Combinatorial Optimization*, 35(2):512–529, 2018.
- 5 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.
- 6 Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the online dial-a-ride problem. In *Proceedings of the Third International Conference on Approximation and Online Algorithms (WAOA’05)*, pages 258–269, 2005. doi:10.1007/11671411\_20.



- 7 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-sharing between two locations: Online scheduling with flexible advance bookings. In *24th International Computing and Combinatorics Conference (COCOON'18)*, volume 10976 of *LNCS*, pages 242–254, 2018.
- 8 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing between Two Locations: Online Scheduling with Two Servers. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS'18)*, volume 117 of *LIPICs*, pages 50:1–50:14, 2018. doi:10.4230/LIPICs.MFCS.2018.50.
- 9 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Online Scheduling of Car-Sharing Requests Between Two Locations with Many Cars and Flexible Advance Bookings. In *29th International Symposium on Algorithms and Computation (ISAAC'18)*, volume 123 of *LIPICs*, pages 64:1–64:13, 2018. doi:10.4230/LIPICs.ISAAC.2018.64.
- 10 Kelin Luo, Thomas Erlebach, and Yinfeng Xu. Car-Sharing on a Star Network: On-Line Scheduling with  $k$  Servers. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS'19)*, volume 126 of *LIPICs*, pages 51:1–51:14, 2019. doi:10.4230/LIPICs.STACS.2019.51.
- 11 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 222–227. IEEE Computer Society, 1977.
- 12 Fanglei Yi and Lei Tian. On the online dial-a-ride problem with time-windows. In *International Conference on Algorithmic Applications in Management (AAIM'05)*, pages 85–94. Springer, 2005.

# Skeletons and Minimum Energy Scheduling

Antonios Antoniadis ✉ 

University of Twente, Enschede, The Netherlands

Gunjan Kumar ✉

National University of Singapore, Singapore

Nikhil Kumar ✉

Hasso Plattner Institute Potsdam, Germany

---

## Abstract

Consider the problem where  $n$  jobs, each with a release time, a deadline and a required processing time are to be feasibly scheduled in a single- or multi-processor setting so as to minimize the total energy consumption of the schedule. A processor has two available states: a *sleep state* where no energy is consumed but also no processing can take place, and an *active state* which consumes energy at a rate of one, and in which jobs can be processed. Transitioning from the active to the sleep does not incur any further energy cost, but transitioning from the sleep to the active state requires  $q$  energy units. Jobs may be preempted and (in the multi-processor case) migrated.

The single-processor case of the problem is known to be solvable in polynomial time via an involved dynamic program, whereas the only known approximation algorithm for the multi-processor case attains an approximation factor of 3 and is based on rounding the solution to a linear programming relaxation of the problem. In this work, we present efficient and combinatorial approximation algorithms for both the single- and the multi-processor setting. Before, only an algorithm based on linear programming was known for the multi-processor case. Our algorithms build upon the concept of a *skeleton*, a basic (and not necessarily feasible) schedule that captures the fact that some processor(s) must be active at some time point during an interval. Finally, we further demonstrate the power of skeletons by providing a 2-approximation algorithm for the multiprocessor case, thus improving upon the recent breakthrough 3-approximation result. Our algorithm is based on a novel rounding scheme of a linear-programming relaxation of the problem which incorporates skeletons.

**2012 ACM Subject Classification** Theory of computation → Scheduling algorithms

**Keywords and phrases** scheduling, energy-efficiency, approximation algorithms, dynamic programming, combinatorial algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.51

**Related Version** *Full Version:* <https://arxiv.org/abs/2107.07800>

**Funding** *Gunjan Kumar:* This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13].

## 1 Introduction

Energy consumption is one of the most important aspects of computing environments as supported, for example, by the fact that data centers already account for more than 1% of global electricity demand, and are forecast to reach 8% by 2030 [10]. With that in mind, modern hardware increasingly incorporates power-management capabilities. However, in order to take full advantage of these capabilities algorithms in general, and scheduling algorithms in particular, must take energy consumption into consideration – on top of the classical algorithm complexity measures of time and space.



© Antonios Antoniadis, Gunjan Kumar, and Nikhil Kumar;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 51; pp. 51:1–51:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work we study *power-down mechanisms* which are one of the most popular power-management techniques available on modern hardware. In the most basic setting, the processor (or device) can reside either in an *active* state in which processing can take place, or in a *sleep* state of negligible energy-consumption in which no processing can take place. Since transitioning the processor from the sleep to the active state requires energy, the scheduler would like to satisfy all the processing requirements, while making use of the sleep state as efficiently as possible. To give some intuition, it is preferable to reside in the sleep state for fewer but longer time-intervals than frequently switching between the two states.

A bit more formally, consider a set of  $n$  jobs, each with a release time, a deadline and a processing-time requirement, to be feasibly scheduled on either a single or a multi-processor system that is equipped with a powerdown mechanism. When the (each) processor resides in the active state it consumes energy at a rate of one, whereas it does not consume any energy when in the sleep state. Transitioning from the sleep state to the active state requires a constant amount of energy (the *wake-up cost*), whereas transitioning from the active state to the sleep state is free of charge (this is w.l.o.g., since positive energy cost could be folded onto the wake-up cost). Jobs can be preempted and (in the multi-processor setting) migrated, but every job can be processed on at most one machine at any given time. The objective is to produce a feasible schedule (assuming that such a schedule exists) which consumes the minimum amount of energy. In Graham's notation, and with  $E$  being the appropriate energy function, the problems we study can be denoted as  $1|r_j; \bar{d}_j; \text{pmtn}|E$  and  $m|r_j; \bar{d}_j; \text{pmtn}|E$  respectively.

The problem was first stated in its *single-processor* version by Irani et al. [9] along with an  $O(n \log n)$ -time 2-approximation algorithm for it. The algorithm, called *Left-to-Right* greedily keeps the processor at its current state for as long as possible. The problem plays a central role in the area of energy efficient algorithms [8], and the exact complexity of it was only resolved by Baptiste et al. [5] (and a bit earlier for the special case of unit-processing-time jobs [4]), who gave an exact polynomial-time algorithm for the problem. Their algorithm is based on a dynamic programming approach, which at least for the arbitrary processing-time case is rather involved, and obtains a running time of  $O(n^5)$ .

When considering the *multi-processor* setting, it is unclear how to adapt the aforementioned dynamic programming approach to the *multi-processor* setting, while enforcing that a job does not run in parallel to itself. Additionally, several structural properties that had proven useful in the analysis of the single-processor setting do not carry over to the multiprocessor setting. Only very recently, an approximation algorithm for the problem that attains a non-trivial approximation guarantee was presented by Antoniadis et al. [2]. Their algorithm is based on carefully rounding a relaxation of a linear programming formulation for the problem and has an approximation ratio of 3. They also show that their approach gives an LP-based 2-approximation algorithm for the single-processor case. We note that whether the problem is NP-hard or not in the multiprocessor-setting remains a major open question.

## 1.1 Formal Problem Statement and Preliminaries

Consider a set of jobs  $\{j_1, j_2, \dots, j_n\}$ ; job  $j_i$  has release time  $r_i$ , deadline  $d_i$  and a processing requirement of  $p_i$ , where all these quantities are non-negative integers. Let  $r_{\min}$  and  $d_{\max}$  be the earliest release time and furthest deadline of any job; it is no loss of generality to assume  $r_{\min} = 0$  and  $d_{\max} = D$ . For  $t \in \mathbb{Z}_{\geq 0}$ , let  $[t, t+1]$  denote the  $t^{\text{th}}$  *time-slot*. Let  $I = [t, t']$ ,  $t, t' \in \mathbb{Z}_{\geq 0}$ ,  $t < t'$  be an *interval*. The length of  $I$ , denoted by  $|I|$  is  $t' - t$ . Finally, we use  $t \in I = [a, b]$  to denote  $a \leq t \leq b$  and call interval  $[r_i, d_i]$  the *span* of job  $i$ .

Two intervals  $I_1 = [a_1, b_1]$  and  $I_2 = [a_2, b_2]$  *overlap* if there is a  $t$  such that  $t \in I_1$  and  $t \in I_2$ . Thus two intervals which are right next to each other would also be considered overlapping. Intervals which do not overlap are considered *disjoint*.  $I_1$  is *contained* in  $I_2$ , denoted  $I_1 \subseteq I_2$ , if  $a_2 \leq a_1 < b_1 \leq b_2$  and it is *strictly contained* in  $I_2$ , denoted  $I_1 \subset I_2$ , if  $a_2 < a_1 < b_1 < b_2$ .

At any time-slot, a processor can be in one of two states: the *active*, or the *sleep* state. For each time-slot that a processor is in the active state, it requires one unit of power whereas no power is consumed in the sleep state. However,  $q$  units of energy (called *wake up* energy) are expended when the processor transitions from the sleep to the active state. In its active state, the processor can either process a job (in which case we refer to it as being *busy*) or just be *idle*. On the other hand the processor cannot perform any processing while in the sleep state. Note that whenever a period of inactivity is at least  $q$  time-slots long then it is worthwhile to transition to the sleep state, whereas if it is less than  $q$  time-slots long then it is preferable to remain in the active state.

A processor can process at most one job in any time-slot and a job cannot be processed on more than one processor in a time-slot. However, job preemption and migration are allowed, i.e., processing of a job can be stopped at any time and resumed later on the same or on a different processor. A job  $j_i$  must be processed for a total of  $p_i$  time-slots within the interval  $[r_i, d_i]$ . Any assignment of jobs to processors and time slots satisfying the above conditions is called a (feasible) *schedule*. We assume that the processor is initially in the sleep state. Therefore, the energy consumed by a schedule is the total length of the intervals during which the processor is active plus  $q$  times the number of intervals in which the processor is active. The objective of the problem is to find a schedule which consumes minimum energy.

We will use  $P$  to denote the sum of all the processing times, ie.  $P = \sum_{i=1}^n p_i$ . We will use  $OPT$  to denote a fixed optimal solution and  $Q$  to denote the total wake up cost incurred by  $OPT$ . Given a solution  $S$  (not necessarily feasible), by *maximal gaps* of  $S$ , we will refer to intervals  $[a, b]$  such that there are no active time slots in  $[a, b]$  and processor is active at time slots  $a - 1$  and  $b$ . An interval  $[a, b]$  of  $S$  is *active* if the processor is active in all time slots in  $[a, b]$  and inactive in time slots  $a - 1$  and  $b$ .

Given a single processor instance, a schedule is called a *skeleton* if for all jobs  $j_i$ , there is at least one active interval overlapping with the span of  $j_i$ . In other words, there must be at least one active time slot in  $[r_i - 1, d_i + 1]$ . Note that a skeleton need not be a feasible solution. A *minimum cost skeleton* is a skeleton of minimal energy consumption over all skeletons.

In the following we let  $(x)^+$  to stand for  $x$  if  $x \geq 0$  and 0 otherwise. Due to space constraints omitted proofs are deferred to the appendix.

## 1.2 Our Contribution

We study the problem in both the single-processor and the multi-processor setting. Our core technical contribution is that of introducing the concept of minimum cost skeletons. We then employ this concept to design combinatorial and efficient approximation algorithms for single and multi-processor setting. Finally we demonstrate how skeletons can also be useful in strengthening the LP-formulation of the problem with additional constraints by giving the first known  $(2 + \epsilon)$ -approximation algorithm for the multi-processor case. More formally, our contribution is based on the following.

### Single Processor

We begin by introducing the notion of skeletons for single-processor instances, and presenting a simple dynamic programming algorithm for computing minimum cost skeletons in Section 2. Roughly speaking, a skeleton is a (not necessarily feasible) solution which overlaps with the span of every job. Apart from providing a lower bound for the cost of the optimal solution  $OPT$ , a skeleton has useful structural properties that allow us to convert it into a feasible solution without much overhead. The first result in which we demonstrate this, is the following.

► **Theorem 1.** *There exists an  $O(n \log n)$ -time algorithm that computes a solution of cost at most  $OPT + P$ .*

The algorithm produces a solution of cost at most  $OPT + P$ , where  $OPT$  is the cost of optimal solution and  $P$  is the sum of processing times. Since  $P \leq OPT$  this implies another 2-approximation algorithm, thus matching algorithm Left-to-Right of [9] in both running time and approximation guarantee.

We further build upon the ideas of Section 2 in Section 3 to give a  $O(n \log n)$ -time algorithm that also builds upon a minimum cost skeleton in order to compute a solution of cost at most  $OPT + P/(\alpha - 1) + Q(2\alpha + 1)$ . Here  $Q$  is the total wake up cost incurred by the optimal solution and  $\alpha$  is any real number greater than 1. We show how this result can be used in order to obtain a  $O(n \log^2 n)$  algorithm that computes a near optimal solution when  $P \gg Q$  (which in most scenarios is the practically relevant case) or  $Q \gg P$ :

► **Theorem 2.** *Let  $t = \max\{P/Q, Q/P\}$ . Then there exists a  $O(n \log^2 n)$  time algorithm which computes a solution of cost at most  $OPT(1 + 8t^{-1/2})$ .*

Note, that this implies a  $(1 + \epsilon)$ -approximation algorithm when  $t \geq 8/\epsilon^2$ , whereas Left-to-Right by Irani et al. [9] remains a 2-approximation algorithm even in that case.

Finally, we give an algorithm that is also an  $35/18 \approx 1.944$ -approximation algorithm in  $O(n \log n)$ -time, improving upon the greedy 2-approximation algorithm of Irani et al. [9]:

► **Theorem 3.** *There is a  $\frac{35}{18}$  approximation algorithm for the single processor case in  $O(n \log n)$  time.*

Although (as already mentioned) Baptiste et al. [5] present an exact algorithm for the problem, it is based on a rather involved dynamic program. In contrast, our algorithms have as their main advantages that they are combinatorial in nature, simple to implement, it has an improved approximation-ratio compared to all other non-exact algorithms for the problem and even obtains a near-optimal solution for the interesting and practically relevant case of  $P \gg Q$  in near linear time.

### Multiple Processors

Although the notion of a skeleton does not naturally extend to the multi-processor setting, it is possible to define skeletons for that setting so as to capture the same intuition. In Section 4 we do exactly that before presenting a polynomial-time algorithm for computing minimum-cost multi-processor skeletons. In Section 5, we give an algorithm to convert any skeleton into a feasible solution while increasing the cost by a factor of at most 6:

► **Theorem 4.** *There exists a combinatorial 6-approximation algorithm for the multi-processor case of the problem.*

This implies the first combinatorial constant-factor approximation algorithm for the multiprocessor case. The arguments required in its analysis are however much more delicate and involved than the single processor case and heavily build upon the tools developed in [2].

Finally, in Section 6 we further demonstrate the power of skeletons by using them to develop a  $(2+\epsilon)$ -approximation algorithm for the multi-processor case. Thus we improve upon the recent breakthrough 3-approximation from [2]. We note, that obtaining any non-trivial approximation guarantee in the multiprocessor setting has been a long standing open problem (see [5]).

► **Theorem 5.** *There exists a  $(2 + \epsilon)$ -approximation algorithm for the problem on parallel machines.*

More specifically, we are able to strengthen the linear program used in [2] with additional constraints that are guided by the definition of skeletons. We note that it is unclear whether the rounding scheme used in [2] can take advantage of these new constraints. To that end we devise a novel rounding technique, and also show that the considered linear program has an integrality gap of at most 2. In other words, we show the following result.

► **Theorem 6.** *There exists a pseudo-polynomial time 2-approximation algorithm for deadline scheduling on parallel processors.*

Theorem 6 when combined with standard arguments, which were already presented in [2], then implies Theorem 5.

### 1.3 Further Related Work

The single-processor setting has also been studied in combination with the other popular power-management mechanism of *speed-scaling* where the processor can additionally vary its speed while in the active state where the power consumption grows convexly in the speed. This allows for increased flexibility, as in some cases it may be beneficial to spend some more energy by increasing the speed in the active state in order to incur larger savings by transitioning the processor to the sleep state for longer periods of inactivity. This technique is commonly referred to as *race to idle* in the literature. The combined problem is known to be **NP**-hard [1, 11] and to admit a fully polynomial time approximation scheme (FPTAS) [3].

Finally, the problem of minimizing the number of gaps in the schedule, i. e., the number of contiguous intervals during which the processor is idle. Note that with respect to exact solutions our problem generalizes that of minimizing the number of gaps. Chrobak et al. [6] present a simple  $O(n^2 \log n)$ -time, 2-approximation algorithm for the problem of minimizing the number of gaps on single processor with unit-processing times. Demaine et al. [7] give an exact algorithm for the problem of minimizing the number of gaps in the multi-processor setting with unit-processing-times.

## 2 Computing Minimum Cost Skeletons for a Single Processor

The goal of this section is to prove Theorem 1. Any solution to the minimum cost skeleton problem can be seen as a (non-overlapping) set of active intervals separated by a set of maximal gaps. In this spirit one interpretation of a skeleton is that of a set of active intervals which overlaps with the span of every job. An equivalent definition for a skeleton is a set of maximal gaps such that the span of no job is properly contained inside any maximal gap: if there is at least one active time-slot in  $[r_i - 1, d_i + 1]$  then the span of job  $i$  is not contained in a maximal gap and vice versa. As we will see, this alternative viewpoint will be useful in designing a near linear time algorithm for computing a minimum cost skeleton.

► **Definition 7. Gap Skeleton Problem.** Find a set of maximal gaps  $G_1, G_2, \dots, G_k$  such that for each job  $j_i$ ,  $[r_i - 1, d_i + 1] \not\subseteq G_p$  for  $p = 1, 2, \dots, k$  and the quantity  $\sum_{i=1}^k (|G_i| - q)^+$  is maximized.

Observe that we may w.l.o.g. restrict ourselves to input instances with at least two jobs, and may only consider minimum cost skeletons, the leftmost interval of which begins at  $d_{min}$  and the rightmost active interval ends at  $r_{max}$  (or else we can transform them to a skeleton satisfying the property without increasing their cost). We call such skeletons *nice*. For the purpose of computing a minimum cost skeleton, we shall restrict our attention to nice skeletons only. The maximal gaps of a nice skeleton form a feasible solution to the gap skeleton problem. By construction, the sum of costs of a nice skeleton and the corresponding gap skeleton is exactly equal to  $r_{max} - d_{min} + q$ . Hence, the problem of finding a minimum cost skeleton is in fact equivalent to finding a maximum cost gap skeleton. We now show how to compute a maximum gap skeleton by using dynamic programming.

Without loss of generality, we may also assume that maximal gaps in any maximum cost gap skeleton start and end at one of the points in  $T = \cup_{i=1}^n \{r_i, d_i\}$  (otherwise we could increase the length of maximal gaps, without thereby decreasing the cost of the solution). A maximal gap  $[x, y]$  is called *right maximal* if there exists a  $j_i$  such that  $d_i = y$  and  $r_i > x$ . Without loss of generality, we may assume that there exists an optimal solution to maximum gap skeleton problem in which all the gaps are right maximal (we can always convert any given optimal solution to one containing only right maximal gaps). Let us rename  $T = \cup_{i=1}^n \{r_i, d_i\}$  to  $T = \{t_1, t_2, \dots, t_{2n}\}$  such that  $t_1 \leq t_2 \leq \dots \leq t_{2n}$ . Observe that for any  $t \in T$ , at most one right maximal gap can have its left endpoint at  $t$  and hence there can be a total of at most  $2n$  right maximal gaps.

We can list all the right maximal gaps in  $O(n \log n)$  time as follows: we sort all the  $r_i$ 's in  $O(n \log n)$  time and then for each  $t \in T$ , we compute the first  $r_i$  to the right of  $t$ . Then  $[t, d_i]$  is the unique right maximal gap starting at  $t$ . Since for each  $t$ , the above can be done in  $O(\log n)$  time (by using binary search), all the right maximal gaps can be computed in  $O(n \log n)$  time.

Let  $T_i = \{t_i, t_{i+1}, \dots, t_{2n}\}$ ,  $i = 1, \dots, 2n$  and  $A[i]$  be the maximum value of the gap skeleton problem when restricted to  $T_i$ . By the discussion above, we may restrict our attention to only the right maximal gaps. Observe that  $A[2n] = 0$  and  $A[1]$  gives the value of maximum cost gap skeleton.  $A$  satisfies the following recurrence: let  $g = [t_i, t_j]$  be the right maximal gap starting at  $t_i$ . In the optimal solution, either a right maximal gap starts at  $t_i$  or it doesn't, giving  $A[i] = \max\{(t_j - t_i - q)^+ + A[j + 1], A[i + 1]\}$ . Using the above recurrence,  $A[1], \dots, A[2n]$  can be computed in  $O(n)$  time. Hence, by the equivalence of the two problems, a skeleton with minimum cost can be computed in  $O(n \log n)$  time.

► **Theorem 8.** A minimum cost skeleton can be computed in  $O(n \log n)$  time.

Since any feasible solution to the minimum energy scheduling problem is also a skeleton, the following follows:

► **Observation 9.** The minimum cost skeleton has value at most  $OPT$ .

In Lemma 10 we show how to convert a skeleton into a feasible solution in  $O(n \log n)$  time with an additional cost of at most  $P$ . Along with Theorem 8 and Observation 9, this completes the proof of Theorem 1

► **Lemma 10.** Let  $S$  be any feasible skeleton and  $P_S$  be the maximum total volume of jobs that can be feasibly processed in it. Then we can convert  $S$  into a feasible solution  $S'$  with an additional cost of  $P - P_S$  in  $O(n \log n)$  time.



### 3 Improved Approximation Algorithms

We further develop the ideas introduced in the last section to give fast and improved approximation algorithm for the minimum energy scheduling problem. The main insight is to compute a minimum cost skeleton after scaling the wake up cost and then using Lemma 10 to find a feasible solution. As we will show, this leads to near optimal solutions in case  $P \gg Q$  or  $P \ll Q$ .

Let  $\alpha > 1$  be a real number and  $S_\alpha$  be the minimum cost skeleton obtained by scaling the wake up cost by  $\alpha$ .  $S_\alpha$  can be computed in  $O(n \log n)$  time by Theorem 8. Let  $F_\alpha$  be the solution obtained by converting  $S_\alpha$  into a feasible solution using Lemma 10. The following theorem bounds the cost of  $F_\alpha$  in terms of  $P, Q$  and  $\alpha$ .

► **Theorem 11.** *The cost of  $F_\alpha$  is at most  $OPT + 2(\alpha + 1)Q + P/(\alpha - 1)$ .*

**Proof.** Let  $OPT$  be a fixed optimal solution to the original instance. We abuse notation and also use  $OPT$  to denote the cost of the solution. To bound the cost of  $F_\alpha$ , we will first convert  $OPT$  into  $S_\alpha$  in a series of steps, while carefully accounting for changes in the cost, and the total volume of jobs that can be processed. Since  $F_\alpha$  can be obtained from  $S_\alpha$  by using Lemma 10 with a further increase in cost equal to the missing volume, this will allow us to obtain the desired bound. Let  $g = [a, b]$  be any maximal gap in  $OPT$ . We first show that at most two active intervals of  $S_\alpha$  overlap with  $g$ . The proof of this claim follows a similar proof found in Irani et al. [9].

▷ **Claim 12.** At most two active intervals of  $S_\alpha$  overlap with any maximal gap  $g$  of  $OPT$ .

Let  $g$  be a maximal gap of  $OPT$  and  $I_1, I_2$  be the two intervals of  $S_\alpha$  that overlap with  $g$  (it is of course possible that either one or both of these intervals are empty). Recall that  $|I_1 \cap g|$  and  $|I_2 \cap g|$  denote the length of overlap between  $g$  and  $I_1, I_2$  respectively.

▷ **Claim 13.**  $|I_1 \cap g| \leq \alpha q$  and  $|I_2 \cap g| \leq \alpha q$ .

**Proof.** Let  $I_1 \cap g = [a, b]$ . For the sake of contradiction, suppose that  $|I_1 \cap g| > \alpha q$ . Observe that span of no job is strictly contained inside  $I_1 \cap g$ , ie. for no  $j_k, [r_k, d_k] \subset I_1 \cap g$  (otherwise job  $j_k$  would not be scheduled in  $OPT$  at all). This implies that  $S_\alpha$  remains a valid skeleton even if we make all the time slots in  $[a, b]$  inactive. This operation decreases the total length of active intervals in  $S_\alpha$  by  $|a - b| > \alpha q$  while introducing at most one new active interval. This implies that  $S_\alpha$  is not the optimal skeleton with wake up cost equal to  $\alpha q$ , a contradiction (as making all time slots in  $[a, b]$  will give a lower cost skeleton). Hence,  $|I_1 \cap g| \leq \alpha q$ . A similar argument shows that  $|I_2 \cap g| \leq \alpha q$ . ◁

We first transform  $OPT$  so that its active intervals contain the active intervals of  $S_\alpha$ . More formally:

▷ **Claim 14.** We can modify  $OPT$  while increasing the cost by at most  $2(\alpha + 1)Q$  so that for any active interval  $I_\alpha \in S_\alpha$ , there exists an active interval  $I_o \in OPT$  such that  $I_\alpha \subseteq I_o$ .

**Proof.** We transform every maximal gap  $g$  of  $OPT$  as follows. Let  $I_1, I_2$  be two active intervals of  $S_\alpha$  that overlap with  $g$ . We add (active) intervals  $I_1 \cap g$  and  $I_2 \cap g$  to  $OPT$ . The additional cost incurred is  $2q + |I_1 \cap g| + |I_2 \cap g| \leq 2q + \alpha q + \alpha q = 2(\alpha + 1)q$ . The condition of the claim clearly holds after we perform the above transformation for every maximal gap of  $OPT$ . The total cost incurred over all maximal gaps is  $2(\alpha + 1)Q$  and the claim follows. ◁

Let  $OPT_1$  be the solution obtained after the modification in Claim 14. Let  $OPT_1 \setminus S_\alpha$  denote the set of active intervals formed by taking the difference of active time slots in  $OPT_1$  and  $S_\alpha$ . Let  $G_\alpha$  be the set of all maximal gaps of  $S_\alpha$ . By Claim 14, for every active interval  $I \in OPT_1 \setminus S_\alpha$ , there exists a maximal gap  $g_\alpha \in G_\alpha$  such that  $I \subseteq g_\alpha$ . We convert  $OPT_1$  into  $S_\alpha$  by removing all the active intervals in  $OPT_1 \setminus S_\alpha$ . To bound the cost of  $OPT_1$ , we need to consider the following possibilities:

- $I \subset g_\alpha$ : In this case, removing  $I$  doesn't create any new active intervals in  $OPT_1$ . Hence, decrease in the cost of  $OPT_1$  is equal to the length of  $I$ . Also, the reduction in the total volume of jobs that can be scheduled in  $OPT_1$  is at most  $|I|$ .
- $I = g_\alpha$ : In this case, removing  $I$  creates a new interval in  $OPT_1$ . Hence, total cost of  $OPT_1$  decreases by  $|I| - q$ . Also, reduction in the total volume of jobs that can be scheduled in  $OPT_1$  is at most  $|I|$ .

Let  $A$  the set of intervals such that  $I \in OPT_1 \setminus S_\alpha = g_\alpha$ . We need the following bound on the cardinality of  $A$  to finish the proof.

▷ **Claim 15.**  $q|A| \leq P/(\alpha - 1)$ .

*Proof.* Recall that we compute  $S_\alpha$  by scaling the wake up cost by  $\alpha$ , hence any maximal gap of  $S_\alpha$  has length at least  $\alpha q$ . Since for any interval  $I \in A$ , there exists a maximal gap  $g_\alpha \in S_\alpha$  such that  $I = g_\alpha$ , we have  $|I| \geq \alpha q$  for any  $I \in A$ . We now show that there can't be more than  $q$  idle time slots in  $I$  in any feasible schedule of  $OPT$ .

Fix a feasible schedule of jobs in  $OPT$ . Since  $I = g_\alpha$  and  $S_\alpha$  is a feasible skeleton, there is no  $k$  such that span of  $j_k$  is strictly contained in  $I$ . Let  $J_1$  be the set of jobs which overlap with the left end point of  $I$  and  $J_2$  be the set of jobs which overlap only with the right end point of  $I$ . Note it is possible that some job is included in both  $J_1$  and  $J_2$ . By shifting the jobs in  $J_1$  to as far left as possible and those in  $J_2$  to as far right as possible, we may assume that all the idle slots in  $I$  appear contiguously. If the length of this idle interval is more than  $q$ , then its removal decreases the cost of the solution without affecting the feasibility, thus contradicting the fact that  $OPT$  is a minimum cost solution. Hence, there cannot be more than  $q$  idle slots in any  $I \in A$  in any feasible schedule of  $OPT$ . This implies that at least  $|I| - q$  volume of jobs is processed in  $I$  in any feasible schedule of  $OPT$ . Hence,  $P \geq \sum_{I \in A} (|I| - q) \geq (\alpha - 1)q|A|$  and the statement of the claim follows. ◁

We are now ready to bound the cost of the final solution. By the case analysis above, the cost of  $S_\alpha$  is at most  $OPT_1 - \sum_{I \in OPT_1 \setminus S_\alpha} |I| + q|A| \leq OPT + 2(\alpha + 1)Q - \sum_{I \in OPT_1 \setminus S_\alpha} |I| + q|A|$ . The total volume of jobs that can be processed in  $S_\alpha$  is at least  $P - \sum_{I \in OPT_1 \setminus S_\alpha} |I|$ . Hence, using Lemma 10 to convert  $S_\alpha$  into a feasible solution gives that the total cost of  $S_\alpha$  is at most  $OPT + 2(\alpha + 1)Q - \sum_{I \in OPT_1 \setminus S_\alpha} |I| + q|A| + \sum_{I \in OPT_1 \setminus S_\alpha} |I| = OPT + 2(\alpha + 1)Q + q|A| \leq OPT + 2(\alpha + 1)Q + P/(\alpha - 1)$ . This completes the proof of the theorem. ◀

We are now ready to prove the two main theorems of this section. We start with Theorem 2.

**Proof of Theorem 2.** We construct a series of solutions,  $F_0, F_1, F_2, F_4, \dots, F_{2^{\lceil \log \lceil n \rceil}}$  as follows:  $F_0$  is the solution given by Theorem 1 of cost at most  $OPT + P$ . For each  $i \geq 1$ , we construct  $F_i$  by setting  $\alpha = \sqrt{P/iq}$  in Theorem 11. Our solution  $F$  is obtained by taking the minimum cost solution among all the  $F_i$ 's. Since, each of  $F_i$ 's can be constructed in  $O(n \log n)$  time,  $F$  can be constructed in  $O(n \log^2 n)$  time. We now show that  $F$  has the desired approximation guarantee. First note that  $OPT \geq P + Q$  and  $t \geq 1$ . If  $P \leq Q$ , then  $F_0$  has cost at most  $OPT + P = OPT(1 + \frac{P}{OPT}) \leq OPT(1 + \frac{P}{P+Q}) \leq$

$OPT(1 + t^{-1}) \leq OPT(1 + 8t^{-1/2})$ . Now consider the case when  $P > Q$ . One can verify that for  $\alpha \geq 1$ ,  $f(\alpha) = Q(2\alpha + 2) + P/(\alpha - 1)$  has a unique minimum at  $\alpha^* = \sqrt{P/2Q} + 1$ . We must have used exactly one  $\alpha \in [\alpha^*, \sqrt{2\alpha^*}]$  to construct one of the  $F_i$ 's. Since,  $f(\alpha)$  has a unique minimum in  $\alpha \geq 1$ , it follows that it is increasing in  $[\alpha^*, \sqrt{2\alpha^*}]$ . Hence, there exists an  $F_i$  with cost no more than guaranteed by setting  $\alpha = 2\alpha^*$  in Theorem 11. A straightforward calculation shows that  $f(2\alpha^*) \leq 8\sqrt{PQ}$ . Hence,  $F$  has cost at most  $OPT + 8\sqrt{PQ} \leq OPT(1 + \frac{8\sqrt{PQ}}{P+Q}) \leq OPT(1 + \frac{8\sqrt{t}}{t+1}) \leq OPT(1 + 8t^{-1/2})$ . ◀

Finally, we give a  $O(n \log n)$  algorithm that has a performance guarantee better than 2.

**Proof of Theorem 3.** We construct two solutions: first one of cost at most  $OPT + P$  (by using Theorem 1) and second one of cost at most  $OPT + 8Q + P/2$  (by using Theorem 11). In case  $P \leq 17Q$ , the first solution has cost at most  $OPT + P \leq OPT + 17(P + Q)/18 \leq OPT \cdot 35/18$ . In case  $P > 17Q$ , the second solution has cost at most  $OPT + 8Q + P/2 \leq OPT + 8(1/18) \cdot OPT + 1/2 \cdot OPT = (1 + 8/18 + 1/2) \cdot OPT = OPT \cdot 35/18$ . ◀

## 4 Skeletons for Parallel Processors

In this section, we extend the idea of skeletons from the single processor setting to the multi-processor one. We design an efficient and combinatorial algorithm for finding the minimum cost skeleton. In the next section we then show how this skeleton can be used to design a combinatorial approximation algorithm for the multi-processor setting. Let  $T$  be as defined in the last section, i. e.  $T = \cup_{i=1}^n \{r_i, d_i\}$ . For any  $t_i, t_j \in T$  such that  $t_i < t_j$ , let  $l(t_i, t_j)$  be the maximum number of processors that can be blacked out in  $[t_i, t_j]$ . More formally,  $l(t_i, t_j)$  is the maximum number of processors so that there exists a feasible schedule using at most on  $m - l(t_i, t_j)$  many processors at any timeslot  $t \in [t_i, t_j]$ . Equivalently, at some time  $t \in [t_i, t_j]$ , at least  $m - l(t_i, t_j)$  processors must be active in any feasible solution. We are now ready to define skeleton for the multi-processor case.

► **Definition 16.** *A set of active intervals (not necessarily feasible) is called a **skeleton** if for any time interval  $[t_i, t_j]$ , there exists a  $t \in [t_i, t_j)$  such that at least  $m - l(t_i, t_j)$  processors are active at timeslot  $t$ .*

By the definition of  $l(t_i, t_j)$  and the definition of multi-processor skeletons, it directly follows that every feasible solution is also a skeleton. Hence, the cost of the optimal skeleton is a lower bound on the cost of an optimal solution.

We note that all  $l(t_i, t_j)$ 's can be computed in polynomial time: there are  $O(n^2)$  possible pairs  $(t_i, t_j)$  and for each pair, the value  $l(t_i, t_j)$  can be computed in  $O(F \log m)$  time by using binary search ( $F$  here denotes the time needed to check feasibility of an instance, which can also be done in polynomial time as we will see in the next section). Therefore, the total time required to compute all  $l(t_i, t_j)$ 's is  $O(Fn^2 \log m)$ .

### 4.1 Computing Minimum Cost Skeleton for Parallel Processors

We show how an optimal multi-processor skeleton can be computed by combining up to  $m$  many distinct single-processor skeletons. To that end, let  $\mathcal{I}_k$  be the set of all tuples  $(t_i, t_j)$  such that  $m - l(t_i, t_j) \geq k$ . Each of  $\{I_k\}_{k=1}^m$  can be thought of as defining an instance of the minimum skeleton problem for a single processor as follows: for each  $I = [a, b] \in \mathcal{I}_k$ , we have a job with release time  $a$ , deadline  $b$  and a unit processing requirement. We can compute the minimum skeleton  $S_k$  for each  $k = 1, 2, \dots, m$  using Theorem 8. It remains to show that  $\{S_k\}_{k=1}^m$  is indeed the desired optimal skeleton.

► **Lemma 17.**  $\{S_k\}_{k=1}^m$  is an optimal skeleton for the multi-processor case.

**Proof.** Let  $O$  be the optimal multi-processor skeleton for an arbitrary given instance. Without loss of generality, we may assume that  $O$  has a laminar structure, i.e. each active interval on processor  $k+1$  is a subset of some active interval on processor  $k$ . Let  $O_k$  be the set of active intervals on processor  $k$  in the optimal solution and  $l_i$  be the total length of active intervals on processor  $i$ . We consider  $O = \{O_k\}_{k=1}^m$  such that it is lexicographically maximal with respect to the  $m$ -tuple  $(l_1, l_2, \dots, l_m)$ , and it differs from  $\{S_k\}_{k=1}^m$  in least number of processors among those lexicographically maximal ones. If  $O_k = S_k$  for  $1 \leq k \leq m$ , then the lemma follows. For the sake of contradiction, let us assume that  $O_k \neq S_k$ , for some  $k$ .

▷ **Claim 18.**  $O_r$  is a feasible skeleton for the single processor instance  $\mathcal{I}_r$ ,  $1 \leq r \leq m$ .

**Proof.** Suppose the statement of the claim doesn't hold. Then there exists an interval  $[t_i, t_j]$  such that  $m - l(t_i, t_j) \geq r$  but there is no active interval overlapping  $[t_i, t_j]$  in  $O_r$ . Since the optimal solution  $O$  is laminar, there is no active interval overlapping  $[t_i, t_j]$  for any  $O_i$ ,  $r \leq i \leq m$ . This implies that the number of active interval at any time in  $[t_i, t_j]$  in  $O$  is at most  $r - 1 < m - l(t_i, t_j)$ . This contradicts the feasibility of the optimal solution and the claim follows. ◁

▷ **Claim 19.** The solution obtained by replacing the intervals on processor  $k$  in the optimal solution, i.e.  $O_k$  by  $S_k$  is a feasible multi-processor skeleton.

**Proof.** Suppose the statement of the claim doesn't hold. Then there exists an interval  $[t_i, t_j]$  such that some active interval in  $O_k$  overlaps with  $[t_i, t_j]$  but no active interval in  $S_k$  overlaps with  $[t_i, t_j]$ . Since the optimal solution  $O$  is laminar, this implies that some active interval in  $O_\ell$  overlaps with  $[t_i, t_j]$  for any  $1 \leq \ell \leq k$ . Hence,  $m - l(t_i, t_j) \geq k$ , which implies that some active interval in  $S_k$  must overlap with  $[t_i, t_j]$ . This contradicts our assumption and the claim follows. ◁

Since  $O_k$  is a feasible solution to  $\mathcal{I}_k$  and  $S_k$  is an optimal solution for  $\mathcal{I}_k$ , cost of  $S_k$  is at most the cost of  $O_k$ . Hence, replacing  $O_k$  by  $S_k$  gives a feasible skeleton without increasing the cost. The new solution as constructed above is laminar as well, otherwise we could move active time slots from a higher numbered processor to a lower numbered processor, contradicting our assumption that the optimal solution is the largest in lexicographical ordering  $(l_1, l_2, \dots, l_m)$ . Thus we have obtained a different optimal solution which matches  $\{S_i\}_{i=1}^m$  on more processors. This contradicts our choice of the optimal skeleton and the lemma follows. ◀

## 5 Converting a Minimum Cost Skeleton into a Feasible Solution

As argued in Lemma 17, the cost of the obtained optimal skeleton is at most the cost of an optimal solution. However, the optimal skeleton may not be a feasible solution. In this section we show how to overcome this by transforming the optimal skeleton  $\{S_k\}_{k=1}^m$  into a feasible solution while increasing its energy cost by at most a factor 6. This transformation consists of two phases: the *extension phase* and the *tripling phase*. In the following subsections we describe each one of them in more detail.

## 5.1 Extension Phase

The extension phase of the transformation is inspired by a similar transformation performed in [2]. For the sake of completeness we give a brief and high-level description of the required terminology and results and refer the interested reader to [2] for the details.

We begin by introducing the notions of *forced volume* and of *deficiency*:

► **Definition 20** ([2]). *The forced volume of a job  $j_i$  with respect to an interval  $[a, b]$ , is defined as  $\text{fv}(j_i, [a, b]) := \max\{0, p_i - (|[r_i, d_i] \setminus [a, b]|)\}$ . Let  $\mathcal{D}$  be a set of disjoint intervals. The forced volume of job  $j_i$  with respect to  $\mathcal{D}$  is defined as  $\text{fv}(j_i, \mathcal{D}) := \max\{0, p_i - |[r_i, d_i] - \sum_{D \in \mathcal{D}} |D \cap [r_i, d_i]|\}$ .*

Intuitively,  $\text{fv}(j_i, [a, b])$  is the minimum volume of  $j_i$  that must be processed during  $[a, b]$  in any feasible schedule, and  $\text{fv}(j_i, \mathcal{D})$  is the amount of volume that must be processed within the intervals of  $\mathcal{D}$  in any feasible schedule.

► **Definition 21** ([2]). *Let  $\mathcal{D}$  be a set of disjoint intervals, and  $\mathcal{I} = \{I_1, I_2, \dots, I_k\}$  be a set of not necessarily disjoint intervals with the property, that for any time-point  $t$ ,  $m_t := |\{i \in \mathcal{I} : I_i \cap t \neq \emptyset\}| \leq m$  holds. Furthermore let  $\mathcal{J}$  be a set of jobs. The deficiency of  $\mathcal{D}$  with respect to  $\mathcal{I}$  and  $\mathcal{J}$ , denoted by  $\text{def}(\mathcal{D}, \mathcal{I}, \mathcal{J})$ , is the non-negative difference between the sum of the forced volume of all jobs of  $\mathcal{J}$  with respect to  $\mathcal{D}$  and the total volume that can be processed in  $\mathcal{D}$  within  $\mathcal{I}$ . Thus*

$$\text{def}(\mathcal{J}, \mathcal{D}, \mathcal{I}) = \max \left( 0, \sum_{j \in \mathcal{J}} \text{fv}(j, \mathcal{D}) - \sum_{t: [t, t+1] \subseteq \mathcal{D}} m_t \right).$$

In [2], a decision problem called **deadline-scheduling-on-intervals** was introduced. More formally, problem **deadline-scheduling-on-intervals** takes as input  $k$  (not necessarily disjoint) supply-intervals  $\mathcal{I} = \{I_1, I_2, \dots, I_k\}$  and the set  $\mathcal{J}$  of jobs (each with a release-time, a deadline and processing volume), and asks whether the jobs of  $\mathcal{J}$  can be feasibly scheduled on  $\mathcal{I}$ . In [2] a polynomial-time algorithm *DSI-ALG* was presented that decides **deadline-scheduling-on-intervals**. Furthermore, in case the input instance is infeasible, *DSI-ALG* returns a *minimal* set of intervals  $\mathcal{D} = \{D_1, D_2, \dots, D_\ell\}$  of maximum deficiency with respect to  $\mathcal{I}$ . The following theorem follows from Section 4 in [2] (more specifically the first statement appears in [2] as Theorem 4.1, the last one as Claim 4.2, and the polynomial-time algorithm is described and analyzed throughout Section 4):

► **Theorem 22** ([2]). *An instance of **deadline-scheduling-on-intervals** is feasible iff no set of disjoint intervals has positive deficiency. Additionally, there is a polynomial-time algorithm that decides if a given instance to **deadline-scheduling-on-intervals** is feasible and if it is not, then a minimal set of disjoint intervals of maximum deficiency with respect to the instance is returned. Furthermore, increasing the volume of supply intervals at any time point in the minimal set of maximum deficiency by one unit decreases the maximum deficiency by one unit.*

Finally, [2] presents a polynomial time algorithm *EXT-ALG* which extends a supply interval  $I' \in \mathcal{I}$  (a property that we will use later, is that  $I'$  is chosen so that it overlaps some  $D' \in \mathcal{D}$  without containing  $D'$ , i.e,  $D' \not\subseteq I'$  but  $D' \cap I' \neq \emptyset$ ) by one time slot so as to decrease the total deficiency of the set of intervals of maximum deficiency  $\mathcal{D}$  by one. This is repeated until the resulting set of supply intervals becomes feasible. Since the maximal deficiency at the beginning was at most the total processing time  $P$  of all jobs, the total increase in energy consumption by extending the supply intervals could also only have been at most  $P$ .

The extension phase consists of repeatedly extending the intervals of  $\{S_k\}_{k=1}^m$  via algorithm EXT-ALG until this is not possible anymore. Assume that at this point  $\{S_k\}_{k=1}^m$  has been extended to an interval set  $\mathcal{I}$ . The extension phase thus terminates either because  $\mathcal{I}$  is a feasible instance for  $\mathcal{J}$  (and by Theorem 22 no set of disjoint intervals has positive deficiency with respect to  $\mathcal{I}$  and  $\mathcal{J}$ ) or because the minimal set  $\mathcal{D}$  of maximum deficiency returned by DSI-ALG does not contain any interval  $D'$  such that  $I' \cup D' \neq \emptyset$  and  $I' \not\supseteq D'$  holds for some  $I' \in \mathcal{I}$ . In the later case  $\mathcal{I}$  is still not feasible, and a further transformation (described in the next subsection) is required. The extension phase as stated now is pseudo-polynomial but can be carried out in polynomial time by using standard techniques (see [2] for more details). From the argument from [2] as well as the discussion above, the following lemma follows:

► **Lemma 23.** *The energy-cost of the schedule  $\mathcal{I}$  differs from that of  $\{S_k\}_{k=1}^m$  by at most an additive factor of  $P$ . Furthermore  $\mathcal{I}$  is either feasible, or contains no interval  $I' \in \mathcal{I}$  that overlaps but does not contain an interval from the minimal set of intervals of maximum deficiency  $\mathcal{D}$ .*

## 5.2 Tripling Phase

In case the extension phase terminated with an infeasible solution, then, by Lemma 23, there is no interval  $I' \in \mathcal{I}$  such that  $I'$  overlaps some interval  $D' \in \mathcal{D}$  without containing it. In that case, we need to perform the tripling phase, in which we carefully power on further machines at specific times so as to make the instance feasible. Let  $m_t$  be the number of machines active at time  $t$  in  $\mathcal{I}$ . We create a new solution  $\mathcal{I}'$  by setting  $m'_t = \min\{3m_t, m\}$ . In Lemma 24, we show that  $\mathcal{I}'$  is a feasible solution to the original instance. By construction, the total cost of intervals in  $\mathcal{I}'$  is at most thrice that of  $\mathcal{I}$ . From the above discussion and Lemma 23, it follows that the algorithm consisting of the tripling and the extension phase is a 6-approximation algorithm. In other words Theorem 4 follows.

► **Lemma 24.**  *$\mathcal{I}'$  is a feasible solution to the original instance.*

## 6 A 2-Approximation Algorithm for Multiple Processors

In this section we prove Theorem 5 thus improving upon the recent 3-approximation algorithm of [2]. To achieve this, we introduce additional constraints to the linear programming (LP) relaxation of [2] and devise a new rounding scheme to harness the power of new constraints. In the remainder of the section, we prove that our rounding technique gives a pseudo-polynomial time 2-approximation algorithm – thus also showing an upper bound of 2 on the integrality-gap of the LP. By using standard arguments, this directly implies  $(2 + \epsilon)$ -approximation algorithm in polynomial time (see [2] for more details).

We now give a brief description of the LP relaxation of [2]. For every possible interval  $I \subseteq [0, D]$ , there is an associated variable  $x_I$ ,  $0 \leq x_I \leq m$  which indicates the number of times  $I$  is picked in the solution. The objective is to minimize the total energy consumption, ie.  $\sum_I x_I (|I| + q)$ .  $m_t$  denotes the number of processors that are active during time slot  $t$  (or equivalently total capacity of active intervals in time slot  $t$ ) and  $f(i, t)$  denotes the volume of job  $i$  that is processed in time slot  $t$ . The constraints of the LP are self explanatory; the interested reader is referred to [2] for the details.

In the following we describe the additional constraints (in bold). Recall that for any interval  $[a, b]$ ,  $l(a, b) \in \mathbb{Z}_{\geq 0}$  is the maximum number of machines that can remain inactive throughout interval  $[a, b]$  without affecting the feasibility of the instance (see Section 4). This implies that at least  $m - l(a, b)$  active intervals are overlapping with  $[a, b]$  in any feasible schedule. We add a constraint capturing this fact for every  $[a, b] \subseteq [0, D]$ .



$$\begin{aligned}
& \text{minimize} && \sum_I (|I| + q)x_I \\
& \text{subject to} && \\
& m_t = && \sum_{I: [t, t+1] \in I} x_I && 0 \leq t < D \\
& m_t \geq && \sum_{i: r_i \leq t \leq d_i - 1} f(i, t) && 0 \leq t < D \\
& p_i = && \sum_{t=r_i}^{d_i-1} f(i, t) && 0 \leq i \leq n \\
& \sum_{I: [a, b] \cap I \neq \emptyset} x_I \geq && m - l(a, b) && 0 \leq a < b \leq D \\
& f(i, t) \in && [0, 1] && \forall i, t \\
& x_I, m_t \in && [0, m] && \forall t, I \subseteq [0, D]
\end{aligned}$$

Suppose the optimum fractional solution to the linear program has value  $f$ . For reasons that will become apparent later, we would like to use an optimum solution maximizing the value  $\sum_t \min(m_t, 1)$ . In order to compute such a solution we solve a second linear program that is based on the previous one, as follows: we introduce a new set of variables  $y_t$  and additional constraints  $y_t \leq m_t$  and  $0 \leq y_t \leq 1$  for each time slot  $t$ . By adding constraint  $\sum_I x_I (|I| + q) = f$  we enforce that the resulting solution has energy cost equal to  $f$  and is therefore also optimal. Finally we set the objective function to maximize  $\sum_t y_t$ .

Let  $F = \{I : x_I > 0\}$  be the optimal fractional solution after solving the second LP. Let  $\epsilon = \gcd_{i \in F}(x_I)$ . We create  $x_I/\epsilon$  copies of each  $I \in F$  to assume that all intervals in  $F$  have the same  $x_I$  value (note that  $F$  is a multiset). If there exist  $[a, b], [c, d], a < c < b < d$  with  $x_{[a, b]}, x_{[c, d]} = \epsilon$ , we replace them by  $[a, d], [b, c]$  with  $x_{[a, d]}, x_{[b, c]} = \epsilon$ . It is easily verified that this process doesn't affect the feasibility of the solution. This process is repeated until no such pair of intervals remain in the instance. We therefore assume from now on, that the intervals in  $F$  are non-crossing. We would like to stress that the above is done only for the ease of analysis and during the course of the proof, it will be clear that we don't actually need to do this.

We partition the time slots in  $[0, D]$  into *blocks* and *non-blocks* as follows. A duration  $[a, b]$  is called a block iff  $m_t \in [0, 1)$  for all  $a \leq t \leq b - 1$  and  $m_{a-1}, m_b \geq 1$ . A duration  $[a, b]$  is called a non-block iff  $m_t \in [1, m]$  for all  $a \leq t \leq b - 1$  and  $m_{a-1}, m_b < 1$ . The following lemma leverages the new constraints and lower bounds the total weight of intervals of  $F$  contained in a non-block. The proof of this lemma crucially uses the fact that  $F$  is an optimum solution maximizing the value  $\sum_t \min(m_t, 1)$ .

► **Lemma 25.** *Let  $N = [a, b]$  be a non-block. If there exists a  $t \in [a, b - 1]$  such that  $m_t > 1$ , then  $\sum_{I: I \cap [a, b] \neq \emptyset} x_I \geq 2$ .*

**Proof.** By noting that a non-block is a maximal contiguous set of time intervals with  $m_t \geq 1$ , intervals in  $F$  are laminar and  $m_t > 1$  for some  $t \in [a, b]$ , there must exist an  $I \in F$  such that  $I \subseteq [a, b]$ . Let  $F'$  be the solution obtained by replacing  $I = [l, r]$  by  $I' = [l + 1, r]$  in  $F$ . Since  $F$  is a fractional solution with minimum cost,  $F'$  must be infeasible. Let  $\mathcal{D} = \{D_1, D_2, \dots, D_\ell\}$  be the *minimal* set of disjoint intervals of maximum deficiency (with respect to  $\mathcal{I}$ ) as returned by Theorem 22 and  $J'$  be the set of jobs such that  $fv(j_i, \mathcal{D}) > 0$  for all  $j_i \in J'$ . Note that the deficiency of  $\mathcal{D}$  with respect to the current solution is  $\epsilon$ . Let  $m'_t$  be defined with respect to  $F'$ .

Suppose there exists a  $t$  such that  $[t, t + 1] \subseteq D \in \mathcal{D}$ ,  $[t - 1, t]$  is part of a non-block and  $[t, t + 1]$  is part of a block (or vice versa). This implies that  $m'_t < 1$  and  $m'_{t-1} \geq 1$ . Since  $m'_{t-1} > m'_t$ , there exists an  $I' \in F'$  which ends at  $t$ . If we extend  $I'$  to the right by 1 unit, deficiency would decrease by  $\epsilon$  (by Theorem 22) and we will obtain a feasible



solution different from  $F$ , with greater value of  $\sum_t \min(m_t, 1)$  (as extending  $J'$  implies setting  $m'_t = m_t + \epsilon > m_t$ ). Hence, no  $D \in \mathcal{D}$  overlaps with a block and non-block simultaneously. Thus we can partition the intervals in  $\mathcal{D}$  depending on whether they are contained in a block or a non-block. Let  $\mathcal{D}_{NB} \subseteq \mathcal{D}$  and  $\mathcal{D}_B \subseteq \mathcal{D}$  be the intervals of  $\mathcal{D}$  contained in blocks and non-blocks respectively.

Let  $D_1, D_2 \in \mathcal{D}$  be such that  $D_1, D_2$  do not belong to the same block or the same non-block. Suppose there exists a  $j_i \in J'$  such that  $[r_i, d_i] \cap D_1 \neq \emptyset$  and  $[r_i, d_i] \cap D_2 \neq \emptyset$ . Then there must exist a  $t$  such that  $[t, t+1] \subseteq [r_i, d_i]$  and  $m_t < 1$ . Since  $fv(j_i, \mathcal{D} \cup [t, t+1]) = fv(j_i, \mathcal{D}) + 1$  and  $m_t < 1$ , we have that the deficiency of  $\mathcal{D} \cup [t, t+1]$  is strictly more than the deficiency of  $\mathcal{D}$ . This contradicts the fact that  $\mathcal{D}$  has maximum deficiency and hence no job in  $J'$  overlaps with distinct  $D_i, D_j$ . Therefore jobs in  $J'$  can be partitioned according to the block or non-block they overlap with. Let  $J'_N \subseteq J'$  be the set of jobs with positive forced volume overlapping with the non-block  $N$  and  $\mathcal{D}_N \subseteq \mathcal{D}_{NB}$  be the set of intervals of  $\mathcal{D}$  overlapping with  $N$ .

Observe that  $\mathcal{D}_N$  must contain the time slot  $[l, l+1]$  and hence is non-empty (recall that  $[l, r]$  was replaced by  $[l+1, r]$  in  $F$  to obtain  $F'$ ). Also,  $J'_N \neq \emptyset$ , otherwise  $\mathcal{D} \setminus \mathcal{D}_N$  would have been the minimal set with maximum deficiency. Hence,  $def(J'_N, \mathcal{D}_N, F') > 0$ . Let  $x$  be the left end point of leftmost interval in  $\mathcal{D}_N$  and  $y$  be the right end point of the rightmost interval in  $\mathcal{D}_N$ . Since  $m'_t \geq 1$  for all  $[t, t+1] \in [x, y] \subseteq N$  and  $def(J'_N, \mathcal{D}_N, F') > 0$ , there must exist a time slot  $t \in [x, y-1]$  such that  $m_t \geq 2$  in any integer feasible solution. This implies that  $m - l(x, y) \geq 2$  and hence  $\sum_{I: I \cap [a, b] \neq \emptyset} x_I \geq m - l(a, b) \geq 2$ .  $\blacktriangleleft$

### Rounding Scheme

We next convert/round  $F$  into an integral solution in a series of steps. We will denote the three intermediate solutions by  $F_1, F_2, F_3$  and the corresponding  $m'_t$ s as  $m_t^1, m_t^2, m_t^3$ . Let  $T_N, T_B$  be the set of time slots in non-blocks and blocks respectively. Let  $P_F$  be the total available capacity in  $F$  (ie.  $P = \sum_t m_t$ ) and  $P_B, P_N$  be the total available capacity in the blocks and non-blocks respectively. Let  $Q_F$  be the total wake up cost incurred by  $F$ , ie.  $Q_F = q \sum_I x_I$ . Then  $Cost(F) = P_F + Q_F$  and  $P_F = P_B + P_N$ . For each non-block  $N = [a, b]$  such that  $m_t > 1$  for some  $t \in [a, b-1]$ , we add an additional supply interval  $[a, b]$  with  $x_{[a, b]} = 1$  to  $F$  and call this solution  $F_1$ .

$\triangleright$  **Claim 26.**  $Cost(F_1) \leq Cost(F) + P_N + Q_F$ .

*Proof.*  $F_1$  is constructed by adding an interval  $[a, b]$  with  $x_{[a, b]} = 1$  for a non-block  $[a, b]$  if  $m_t > 1$  for some  $t \in [a, b]$ . For each time slot  $t$  in a non-block, we have  $m_t \geq 1$  and hence the total length of new intervals added is at most  $\sum_{[t, t+1] \subseteq T_N} m_t \leq P_N$ . If we add an additional interval for a non-block  $[a, b]$ , then  $\sum_{I: I \cap [a, b] \neq \emptyset} x_I \geq 2$  (by Lemma 25). Since the intervals in  $F$  are non-crossing, the above implies that the total weight of intervals which are completely contained in  $[a, b]$  is at least 1. Thus the wakeup cost of each new interval can be charged to the wakeup cost of intervals completely contained inside the corresponding non-block, and the total additional wake up cost incurred is no more than  $Q_F$ .  $\blacktriangleleft$

We convert  $F_1$  into  $F_2$  by deleting the portions of existing intervals in  $F$  such that for each time slot  $t \in T_N$ , we have the property  $m_t^2 = \lfloor m_t^1 \rfloor$ . This operation might increase the total wake up cost, but since the processing cost gets decreased by at least as much, the overall cost of the solution does not increase. This allows us to state the following.

$\triangleright$  **Claim 27.**  $Cost(F_2) \leq Cost(F) + P_N + Q_F$ . Also,  $m_t^2$  is an integer and  $m_t^2 \geq m_t$  for each time slot  $t \in T_N$ .

We now describe our third transformation. As discussed earlier, we may again assume that all the intervals in  $F_2$  have a weight of exactly  $\epsilon$  and are non-crossing. Consider the single machine instance  $J_S = J_N \cup J_B$ , where  $J_B = \{j_i | j_i \in J, [r_i, d_i] \subseteq T_B\}$  consists of jobs in  $J$  which are completely contained inside some block and  $J_N$  consists of additional jobs defined as follows: for each time slot  $t \in T_N$ , there is a job  $j_t$  with release time  $t$ , deadline  $t + 1$  and a processing requirement of 1. We now pick a subset  $F'_2$  of intervals in  $F_2$  which form a feasible solution to the following LP relaxation of the minimum cost skeleton.

$$\begin{aligned} \min \quad & \sum_I x_I(q + |I|) \\ & \sum_{I:t \in I} x_I \leq 1 \quad \forall t \in [0, D] \\ & \sum_{I:I \cap [r_i, d_i] \neq \emptyset} x_I \geq 1 \quad \forall i \in [1, n] \\ & x_I \geq 0 \quad \forall I \subseteq [a, b] \end{aligned}$$

$F'_2 \subseteq F_2$  is the set of intervals of maximum total length such that the total weight of intervals containing any particular time slot is at most 1. It is worth noting that any interval containing a time slot of some block is a part of  $F'_2$ . The proof of the following claim is deferred to the full version.

▷ **Claim 28.**  $F'_2$  is a feasible fractional skeleton for  $J_S$ .

In the full version, we also show that the LP relaxation for the minimum cost skeleton is exact. Hence, there exists an integer skeleton  $J_S$  of cost no more than  $F'_2$ . Then our solution  $F_3$  is  $(F \setminus F'_2) \cup S$ . Observe that  $\text{Cost}(F_3) \leq \text{Cost}(F_2)$  and  $m_t^3$  is an integer for every time slot  $t$ . We may therefore assume that  $x_I = 1$  for each  $I \in F_3$ . We now describe the final phase our algorithm, where we convert  $F_3$  into a feasible solution by extending some existing intervals using *EXT-ALG* (see Section 5). If  $F_3$  is a feasible solution, our algorithm terminates. Otherwise we find a disjoint minimal set of intervals of maximum deficiency  $\mathcal{D} = \{D_1, D_2, \dots, D_k\}$  guaranteed by Theorem 22. In each subsequent iteration, we use *EXT-ALG* to extend an interval of  $F_3$  by 1 unit, thereby reducing the maximum deficiency by 1. Claim 29 shows that if the current solution is infeasible and it is not possible to extend an interval to reduce the deficiency, then the original instance is infeasible. Hence the extension phase of the algorithm terminates with a feasible solution.

▷ **Claim 29.** Let  $F_{curr}$  be the current solution. If  $F_{curr}$  is infeasible and for all  $I \in F_{curr}$  the following is true: if  $I \cap D_i \neq \emptyset$ , then  $D_i \subseteq I$ , then the original instance is infeasible.

The deficiency at the start of the extension phase can be at most  $P_B$  as  $m_t^3 \geq m_t$  for  $t \in T_N$ . Since we decrease the deficiency by 1 in each iteration, there can be at most  $P_B$  iterations of the extension phase. In each step we increase the cost of the solution by 1, hence cost of the final solution is at most  $\text{Cost}(F_3) + P_B \leq \text{Cost}(F) + P_N + Q_F + P_B = \text{Cost}(F) + P + Q_F \leq 2\text{Cost}(F)$ . This shows that the integrality gap of the LP relaxation is at most 2. To compute  $F_1, F_2, F_3$ , we only need the value of  $m_t$ 's and don't need to create multiple copies of intervals in our solution. Thus our rounding algorithm can be implemented in pseudo-polynomial time and this completes the proof of Theorem 6.

---

**References**


---

- 1 Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms*, 10(2):9:1–9:31, 2014.
- 2 Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *SODA*, pages 2758–2769. SIAM, 2020.
- 3 Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1102–1113, 2015.
- 4 Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 364–367. Society for Industrial and Applied Mathematics, 2006.
- 5 Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial-time algorithms for minimum energy scheduling. *ACM Trans. Algorithms*, 8(3):26:1–26:29, 2012.
- 6 Marek Chrobak, Uriel Feige, Mohammad Taghi Hajiaghayi, Sanjeev Khanna, Fei Li, and Seffi Naor. A greedy approximation algorithm for minimum-gap scheduling. *Journal of Scheduling*, 20(3):279–292, 2017.
- 7 Erik D. Demaine, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. *J. Sched.*, 16(2):151–160, 2013.
- 8 Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.
- 9 Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4):41, 2007.
- 10 Nicola Jones. How to stop data centres from gobbling up the world’s electricity. <https://www.nature.com/articles/d41586-018-06610-y>, 2018.
- 11 Gunjan Kumar and Saswata Shannigrahi. On the NP-hardness of speed scaling with sleep state. *Theor. Comput. Sci.*, 600:1–10, 2015.

# Machine Covering in the Random-Order Model

Susanne Albers ✉

Department of Computer Science, Technische Universität München, Germany

Waldo Gálvez ✉ 🏠 

Institute of Engineering Sciences, Universidad de O'Higgins, Rancagua, Chile

Maximilian Janke ✉

Department of Computer Science, Technische Universität München, Germany

---

## Abstract

In the Online Machine Covering problem jobs, defined by their sizes, arrive one by one and have to be assigned to  $m$  parallel and identical machines, with the goal of maximizing the load of the least-loaded machine. Unfortunately, the classical model allows only fairly pessimistic performance guarantees: The best possible deterministic ratio of  $m$  is achieved by the Greedy-strategy, and the best known randomized algorithm has competitive ratio  $\tilde{O}(\sqrt{m})$  which cannot be improved by more than a logarithmic factor.

Modern results try to mitigate this by studying semi-online models, where additional information about the job sequence is revealed in advance or extra resources are provided to the online algorithm. In this work we study the Machine Covering problem in the recently popular *random-order* model. Here no extra resources are present, but instead the adversary is weakened in that it can only decide upon the input set while jobs are revealed uniformly at random. It is particularly relevant to Machine Covering where lower bounds are usually associated to highly structured input sequences.

We first analyze Graham's Greedy-strategy in this context and establish that its competitive ratio decreases slightly to  $\Theta\left(\frac{m}{\log(m)}\right)$  which is asymptotically tight. Then, as our main result, we present an improved  $\tilde{O}(\sqrt[4]{m})$ -competitive algorithm for the problem. This result is achieved by exploiting the extra information coming from the random order of the jobs, using sampling techniques to devise an improved mechanism to distinguish jobs that are relatively large from small ones. We complement this result with a first lower bound showing that no algorithm can have a competitive ratio of  $O\left(\frac{\log(m)}{\log \log(m)}\right)$  in the random-order model. This lower bound is achieved by studying a novel variant of the Secretary problem, which could be of independent interest.

**2012 ACM Subject Classification** Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

**Keywords and phrases** Machine Covering, Online Algorithm, Random-Order, Competitive Analysis, Scheduling

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.52

**Related Version** *Full Version:* <https://arxiv.org/pdf/2110.09161.pdf>

**Funding** Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.

*Waldo Gálvez:* This project was carried out when the author was a postdoctoral researcher at the Department of Computer Science of the Technical University of Munich in Germany.

## 1 Introduction

We study the *Machine Covering* problem, a fundamental load balancing problem where  $n$  jobs have to be assigned (or scheduled) onto  $m$  identical parallel machines. Each job is characterized by a non-negative size, and the goal is to maximize the smallest machine load. This setting is motivated by applications where machines consume resources in order to work, and the goal is to keep the whole system active for as long as possible. Machine Covering has found additional applications in the sequencing of maintenance actions for



© Susanne Albers, Waldo Gálvez, and Maximilian Janke;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 52; pp. 52:1–52:16

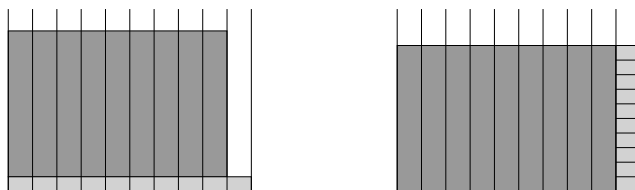
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

aircraft engines [20] and in the design of robust Storage Area Networks [41]. The offline problem, also known as Santa-Claus or Max-Min Allocation Problem, received quite some research interest, see [44, 10, 6] and references therein. In particular, the problem is known to be strongly NP-hard but to allow for a Polynomial-Time Approximation Scheme (PTAS) [44].

This paper focuses on the online version of the problem, where jobs arrive one by one and must be assigned to some machine upon arrival. Lack of knowledge about future jobs can enforce very bad decisions in terms of the quality of the constructed solutions: In a classical lower bound sequence,  $m$  jobs of size 1 arrive first to the system and they must be assigned to  $m$  different machines by a competitive deterministic online algorithm. Then subsequent  $m - 1$  jobs of size  $m$  arrive, which make the online algorithm perform poorly, see Figure 1. Indeed, the best possible deterministic algorithm achieves a competitive ratio of  $m$  [44], and if randomization is allowed, the best known competitive ratio is  $\tilde{O}(\sqrt{m})$ , which is best possible up to logarithmic factors [7]. The corresponding lower bound also uses that the online algorithm cannot schedule the first  $m$  jobs correctly, at least not with probability exceeding  $\frac{1}{\sqrt{m}}$ .



■ **Figure 1** The instance showing that no deterministic algorithm is better than  $m$ -competitive for Machine Covering. To the left, the best possible solution that an online algorithm can construct achieves minimum load 1. To the right, the optimal minimum load is  $m$ .

Such restrictive facts have motivated the study of different semi-online models that provide extra information [7, 14, 34, 37] or extra features [41, 42, 22, 16] to the online algorithm.

This work studies the Online Machine Covering problem in the increasingly popular *random-order* model. In this model, jobs are still chosen worst possible by the adversary but they are presented to the online algorithm in a uniformly random order. The random-order model derives from the Secretary Problem [13, 35] and has been applied to a wide variety of problems such as generalized Secretary problems [32, 8, 33, 18, 9], Scheduling problems [39, 38, 2, 3], Packing problems [30, 31, 5, 4], Facility Location problems [36] and Convex Optimization problems [25] among others. See also [26] for a survey chapter. It is particularly relevant to Machine Covering, where hard instances force online algorithms to make an irredeemable mistake right on the first  $m$  jobs due to some hidden large job class at the end.

Indeed, we show that the competitive ratio of Graham’s Greedy-strategy improves from  $m$  to  $O\left(\frac{m}{\log(m)}\right)$ , and that this is asymptotically tight. We also develop an  $\tilde{O}(\sqrt[3]{m})$ -competitive algorithm, providing evidence that known hardness results rely on “pathological” inputs, and complement it by proving that no algorithm can be  $O\left(\frac{\log(m)}{\log \log(m)}\right)$ -competitive in this model.

## 1.1 Related Results

The most classical Scheduling problem is *Makespan Minimization* on parallel and identical machines. Here, the goal is dual to Machine Covering; one wants to minimize the maximum load among the machines. This problem is strongly NP-hard and there exists a PTAS [27]. The online setting received considerable research attention, and already in 1966 Graham

showed that his famous Greedy-strategy is  $(2 - 1/m)$ -competitive. A long line of research [21, 11, 29, 1, 19] starting in the 1990s lead to the currently best competitive ratio of 1.9201 due to Fleischer and Wahl [19]. Regarding lower bounds, again after a sequence of results [17, 12, 24] the current best one is 1.88 [40].

The landscape for Online Machine Covering differs considerably from Online Makespan Minimization as discussed before, which has motivated the study of semi-online models to deal with the implied hard restrictions. If the value of the optimal minimum load of the instance is known in advance, Azar et al. have shown that a simple greedy algorithm already is  $(2 - 1/m)$ -competitive and that no algorithm can attain a competitive ratio better than  $7/4$  [7]. These bounds were improved by Ebenlendr et al. to  $11/6$  and 1.791 respectively [14]. In the *bounded migration* model, whenever a job of size  $p$  arrives, older jobs of total size at most  $\beta \cdot p$  can be reassigned to different machines. Sanders et al. [41] provide a 2-competitive algorithm for  $\beta = 1$ . Later results [42, 22] study the interplay between improved competitive ratios and larger values of  $\beta$ . Another semi-online model provides the online algorithm with a *reordering buffer*, which is used to rearrange the input sequence “on the fly”. Epstein et al. [16] provide a  $(H_{m-1} + 1)$ -competitive algorithm using a buffer of size  $m - 1$ , and show that this ratio cannot be improved for any sensible buffer size. These and many more semi-online models have also been studied for Makespan Minimization, see the survey in [15] and references therein.

The first Scheduling result in the random-order model is due to Osborn and Torng [39]. They establish that the Greedy-strategy for Makespan Minimization does not achieve a competitive ratio better than 2 for general  $m$ . Recently, [2, 3] show that for Makespan Minimization the random-order model allows for better performance guarantees than the classical model. Molinaro [38] has studied the Online Load Balancing problem with the objective to minimize general  $l_p$ -norms of the machine loads, providing an algorithm that returns solutions of expected norm  $(1 + \varepsilon)\text{OPT} + O\left(\frac{p}{\varepsilon}(m^{1/p} - 1)\right)$  in the random-order model, where OPT denotes the optimal norm. Göbel et al. [23] have studied Average Weighted Completion Time Minimization on one machine in the random-order model. Their competitive ratio is logarithmic in the input length  $n$  for general job sizes and constant if all jobs have size 1. To the best of our knowledge, no previous result is known for Online Machine Covering in the random-order model.

## 1.2 Our Contribution

We first establish that the Greedy-strategy is  $\Theta\left(\frac{m}{\log(m)}\right)$ -competitive in the random-order model. This is only a tiny, albeit significant improvement compared to worst-case orders. Since the bound is tight, more refined strategies that make particular use of the characteristics of the random-order model are required. The analysis also gives first intuitions about what these characteristics are and also about the techniques used to analyze the main algorithm.

The following theorem summarizes the central result of this paper.

► **Theorem 1.** *There exists a  $\tilde{O}(\sqrt[4]{m})$ -competitive algorithm for the online Machine Covering problem in the random-order model.*

In the classical online Machine Covering problem, difficult instances are usually related to the inability of distinguishing “small” and “large” jobs induced by a lack of knowledge about large job classes hidden at the end of the sequence. Figure 1 depicts the easiest example on which deterministic schedulers cannot perform well as they cannot know that the first  $m$  jobs are tiny. Azar and Epstein [7] ameliorate this by maintaining a randomized threshold, which is used to distinguish small and large job sizes. They have to correctly classify up to  $m$  large

jobs with constant probability while controlling the total size of incorrectly classified small jobs, which leads to their randomized competitive ratio of  $\tilde{O}(\sqrt{m})$ . However, their lower bound shows that general randomized algorithms are still unable to schedule the first  $m$  jobs correctly with probability exceeding  $\frac{1}{\sqrt{m}}$ , again due to relevant job classes being hidden at the end of the input.

Random-order arrival makes such hiding impossible. This already helps the Greedy-strategy as now large jobs in the input are evenly distributed instead of being clustered at the end. Our  $\tilde{O}(\sqrt[4]{m})$ -competitive algorithm enhances the path described previously by making explicit use of the no-hiding-feature; it determines those large jobs the adversary would have liked to hide. Information about large job sizes is, as is common in Secretary problems, estimated in a sampling phase, which returns a threshold distinguishing all except for  $\sqrt{m}$  of the large jobs. This reduction by a square root carries over to the competitive ratio: We now can allow to misclassify these remaining  $\sqrt{m}$  jobs with a higher probability, which in turn leads to a better classification of small jobs and a better competitive ratio of  $\tilde{O}(\sqrt[4]{m})$ .

We complement the upper bound of  $\tilde{O}(\sqrt[4]{m})$  with a lower bound of  $\omega\left(\frac{\log(m)}{\log\log(m)}\right)$  for the competitive ratio in the random-order model. Lower bounds in the random-order model are usually considered hard to devise since one cannot hide larger pieces of input. Instead of hiding large job classes, we figuratively make them hard to distinguish by adding noise. To this end, we study a novel variant of the Secretary problem, the *Talent Contest* problem, where the goal is to find a good but not too good candidate (or secretary).

More in detail, we want to pick the  $K$ -th best among a randomly permuted input set of candidates. Unlike classical Secretary problems (or the more general Postdoc problem [43]), we may pick several candidates as long as they are not better than the  $K$ -th candidate. Furthermore, we interview candidates  $t$  times and make a decision at each arrival. In this setting, information gained by earlier interviews helps the decisions required in later ones. It can be proven that the expected number of times the desired candidate can be correctly identified relates to the ability of distinguishing exactly the  $m - 1$  largest jobs from a Machine Covering instance in the random-order model, and hence bounding the aforementioned expected value allows us to obtain the desired hardness result.

### 1.3 Organization of the paper

In Section 2 we provide the required definitions and tools, and then in Section 3 analyze the Greedy-algorithm in the context of random-order arrival. In Section 4 we present our main algorithm and its analysis. Section 5 then introduces the Talent Contest Problem and concludes with a lower bound for the best competitive ratio in the random-order model. Due to space constraints, some proofs from Section 5 are deferred to the full version of the paper.

## 2 Preliminaries

In this section we introduce the main definitions and tools that are used along this work.

In the Machine Covering problem, we are given  $n$  jobs  $\mathcal{J} = \{J_1, \dots, J_n\}$ , specified by their non-negative sizes  $p_i$ , which are to be assigned onto  $m$  parallel and identical machines. The *load*  $l_M$  of a machine  $M$  is the sum of the sizes of all the jobs assigned to it. The goal is to maximize the minimum load among the machines, i.e. to maximize  $\min_M l_M$ .

To an online algorithm, jobs are revealed one-by-one and each has to be assigned permanently and irrevocably before the next one is revealed. Formally, given the symmetric group  $S_n$  on  $n$  elements, each permutation  $\sigma \in S_n$  defines the order in which the elements of



$\mathcal{J}$  are revealed, namely  $\mathcal{J}^\sigma = (J_{\sigma(1)}, \dots, J_{\sigma(n)})$ . Classically, the performance of an online algorithm  $A$  is measured in terms of competitive analysis. That is, if we denote the minimum machine load of  $A^1$  on  $\mathcal{J}^\sigma$  by  $A(\mathcal{J}^\sigma)$  and the minimum machine load an optimal offline algorithm may achieve by  $\text{OPT}(\mathcal{J})$  (which is independent of the order  $\sigma$ ), one is interested in finding a small (*adversarial*) *competitive ratio*  $c = \sup_{\mathcal{J}} \sup_{\sigma \in S_n} \frac{\text{OPT}(\mathcal{J})}{A(\mathcal{J}^\sigma)}$ .

In the random-order model, the job order is chosen uniformly at random. We consider the permutation group  $S_n$  as a probability space under the uniform distribution. Then, given an input set  $\mathcal{J}$  of size  $n$ , we charge  $A$  *random-order cost*  $A^{\text{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma \sim S_n} [A(\mathcal{J}^\sigma)] = \frac{1}{n!} \sum_{\sigma \in S_n} A(\mathcal{J}^\sigma)$ . The *competitive ratio in the random-order model* of  $A$  is  $c = \sup_{\mathcal{J}} \frac{\text{OPT}(\mathcal{J})}{A^{\text{rom}}(\mathcal{J})}$ . Throughout this work we will assume that  $n$  is known to the algorithm; this assumption is common in the literature and it can be proven that it does not help in the adversarial setting, see the full version of the article for details. When clear from the context, we will omit the dependency on  $\mathcal{J}$ .

Given  $0 \leq i \leq n$ , let  $P_i = P_i[\mathcal{J}]$  refer to the size of the  $i$ -th largest job in  $\mathcal{J}$ . Given  $i \geq 1$ , we define  $L_i := \sum_{j \geq i} P_j$  to be the total size of jobs smaller than  $P_i$ . Note that the terminology 'smaller' uses an implicit tie breaker since there may be jobs of equal sizes.

### 3 Properties and Analysis of the Greedy-strategy

We now proceed with some useful properties of Graham's Greedy-strategy. Recall that this algorithm always schedules an incoming job on some least loaded machine breaking ties arbitrarily. The following two lemmas recall useful standard properties of the algorithm, which will help us later to restrict ourselves to simpler special instances.

► **Lemma 2.** *The minimum load achieved by the Greedy-strategy is at least  $P_m$ .*

**Proof.** If the  $m$  largest jobs get assigned to different machines, the bound holds directly. On the other hand, if one of the  $m$  largest jobs  $J_j$  gets assigned to a machine that already had one of the  $m$  largest jobs  $J_{j'}$ , of size  $P_{j'}$ , then at the arrival of  $J_j$  the minimum load was at least  $P_{j'} \geq P_m$ , concluding the claim as the minimum load does not decrease through the iterations. ◀

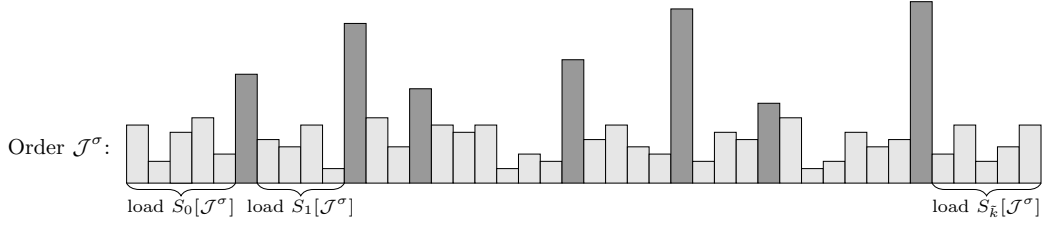
► **Lemma 3.** *The minimum load achieved by the Greedy-strategy is, for each  $i \in \{1, \dots, m\}$ , bounded below by  $\frac{L_i}{m} - P_i$ .*

**Proof.** For each machine  $M$ , let  $J_{\text{last}}(M)$  be the last job assigned to machine  $M$ . Let  $\mathcal{J}_{\text{last}}$  be the set of all these last jobs. If  $ALG$  denotes the minimum load achieved by the Greedy-strategy, then the load of each machine  $M$  in the schedule is at most  $ALG + J_{\text{last}}(M)$  as jobs are iteratively assigned to a least loaded machine. Now remove the  $i - 1$  largest jobs in  $\mathcal{J}_{\text{last}}$  from the solution and let  $\tilde{L}$  denote the total size of the remaining jobs in  $\mathcal{J}_{\text{last}}$ . Then the total size of all remaining jobs is at most  $m \cdot ALG + \tilde{L}$ . On the other hand, since we only removed  $i - 1$  jobs, the total size of the remaining jobs is at least  $L_i$ . Since  $\tilde{L} \leq (m - i + 1) \cdot P_i$ , we have that  $L_i \leq m \cdot ALG + (m - i + 1) \cdot P_i$ . Hence, the minimum load achieved by the Greedy-strategy is at least  $\frac{L_i}{m} - \frac{m-i+1}{m} P_i \geq \frac{L_i}{m} - P_i$ . ◀

With these tools we can now prove that the Greedy-strategy has a competitive ratio of at most  $O\left(\frac{m}{H_m}\right)$  in the random-order model, where  $H_m = \sum_{i=1}^m \frac{1}{i}$  denotes the  $m$ -th harmonic number. We also describe a family of instances showing that this analysis is asymptotically tight.

<sup>1</sup> In case  $A$  is randomized, we then refer to the expected minimum load.

<sup>2</sup> Using the convention that  $0/0 = 1$  and  $a/0 = \infty$  for  $a > 0$ .



■ **Figure 2** A possible order  $\mathcal{J}^\sigma$  for an instance with  $\tilde{k} < m$  large (dark) jobs. They partition the sequence into sets of small (light) jobs, each one having total size  $S_i(\mathcal{J}^\sigma)$ ,  $i = 0, \dots, \tilde{k}$ .

► **Theorem 4.** *The Greedy-strategy is  $(2 + o_m(1)) \frac{m}{H_m}$ -competitive in the random-order model.*

**Proof.** Thanks to Lemma 2 we can assume that  $P_m \leq \frac{H_m}{2m} \text{OPT}$ . We say that a job is *large* if its size is larger than  $\frac{H_m}{2m} \text{OPT}$ , otherwise it is small. Let  $\tilde{k} < m$  be the number of large jobs in the instance. Note that  $L_{\tilde{k}+1} \geq (m - \tilde{k}) \text{OPT}$  since in the optimal solution at most  $\tilde{k}$  machines receive large jobs. Using Lemma 3 we may assume  $\tilde{k} \geq m - H_m$ , as otherwise we are done.

For a given order  $\mathcal{J}^\sigma$  and  $0 \leq i \leq \tilde{k}$ , let  $S_i(\mathcal{J}^\sigma)$  be the total size of small jobs preceded by precisely  $i$  large jobs with respect to  $\mathcal{J}^\sigma$  (see Figure 2). We will prove that the minimum load achieved by the Greedy-strategy is at least  $\min \left\{ \sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i} - \frac{H_m}{2m} \text{OPT}, \frac{H_m}{2m} \right\} \text{OPT}$ .

A machine is said to be *full* once it receives a large job, and notice that we can assume that no full machine gets assigned further jobs as otherwise the minimum load would be already at least  $\frac{H_m}{2m} \text{OPT}$ . Consider the set of small jobs that arrive to the system before the first large job in the sequence. At this point the average load of the machines is exactly  $\frac{S_0(\mathcal{J}^\sigma)}{m}$  and, since the upcoming large job gets assigned to a least loaded machine, the average load of the remaining machines is still at least  $\frac{S_0(\mathcal{J}^\sigma)}{m}$ . Now the upcoming small jobs that arrive before the second large job in the sequence get assigned only to these non-full machines, whose average load is now at least  $\frac{S_0(\mathcal{J}^\sigma)}{m} + \frac{S_1(\mathcal{J}^\sigma)}{m-1}$ . Since the following large job gets assigned to the least loaded of these machines, the average load of the remaining ones is still lower-bounded by this quantity. By iterating this argument, it can be seen that the average load of the machines containing only small jobs is at least  $\frac{1}{m-\tilde{k}} \sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i}$ . Since in a Greedy-strategy the load of two machines cannot differ by more than the size of the largest job assigned to them, we conclude that the minimum load achieved by the algorithm is at least  $\sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i} - \frac{H_m}{2m} \text{OPT}$ .

Now let us understand the random variable  $S(\mathcal{J}^\sigma) = \sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i}$ . If we pick  $\sigma \sim S_n$  uniformly at random and consider the number  $s(J)$  of large jobs that precede any fixed small job  $J$ , this number is uniformly distributed between 0 and  $\tilde{k}$ . Then we can rewrite  $S(\mathcal{J}^\sigma) = \sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i} = \sum_{J \text{ small}} \frac{p_J}{m-s(J)}$ . Since  $\mathbf{E}[\frac{1}{m-s(J)}] = \sum_{i=0}^{\tilde{k}} \frac{1}{\tilde{k}+1} \frac{1}{m-i} = \frac{H_m - H_{m-\tilde{k}-1}}{\tilde{k}+1}$  we get by linearity of expectation that

$$\begin{aligned} \mathbf{E}[S(\mathcal{J}^\sigma)] &= \sum_{J \text{ small}} p_J \frac{H_m - H_{m-\tilde{k}-1}}{\tilde{k}+1} \\ &= \frac{H_m - H_{m-\tilde{k}-1}}{\tilde{k}+1} L_{\tilde{k}+1} \\ &\geq \frac{(m - \tilde{k})(H_m - H_{m-\tilde{k}-1})}{\tilde{k}+1} \frac{L_{\tilde{k}+1}}{m - \tilde{k}}. \end{aligned}$$

Notice that  $\frac{L_{\tilde{k}+1}}{m-\tilde{k}}$  is a lower bound for  $\text{OPT}$ , which we will use later. The first factor is decreasing as a function of  $\tilde{k} \leq m-1$ , and consequently the expression is at least  $\frac{H_m}{m}$ . Thus

$$\mathbf{E}[S(\mathcal{J}^\sigma)] \geq \frac{H_m}{m} \cdot \frac{L_{\tilde{k}+1}}{m-\tilde{k}}.$$

We also get  $\text{Var} \left[ \frac{1}{m-s(J)} \right] \leq \mathbf{E} \left[ \left( \frac{1}{m-s(J)} \right)^2 \right] = \sum_{i=0}^{\tilde{k}} \frac{1}{\tilde{k}+1} \frac{1}{(m-i)^2} \leq \frac{1}{\tilde{k}+1} \sum_i \frac{1}{i^2} \leq \frac{1}{\tilde{k}+1} \frac{\pi^2}{6}$ . Using that  $\tilde{k} \geq m - H_m$ , we broadly bound  $\tilde{k}$  via  $\tilde{k} + 1 \geq \frac{m}{2} \geq \frac{m(m-\tilde{k})^2}{2H_m^2}$ . Substituting this in the previous bound yields  $\text{Var} \left[ \frac{1}{m-s(J)} \right] \leq \frac{H_m^2}{m} \frac{1}{(m-\tilde{k})^2} \frac{\pi^2}{3}$ . Moreover, for two small jobs  $J_i$  and  $J_j$  we have  $\text{Cov} \left[ \frac{1}{m-s(J_i)}, \frac{1}{m-s(J_j)} \right] \leq \left( \text{Var} \left[ \frac{1}{m-s(J_i)} \right] \text{Var} \left[ \frac{1}{m-s(J_j)} \right] \right)^{1/2} \leq \frac{H_m^2}{m} \frac{1}{(m-\tilde{k})^2} \frac{\pi^2}{3}$ . For  $i \neq j$  this bound is pessimistic. The correlation between  $s(J_i)$  and  $s(J_j)$  is positive but tiny. We use this covariance to bound  $\text{Var}[S(\mathcal{J}^\sigma)] = \sum_{J \text{ small}} \frac{p_J}{m-s(J)}$ .

$$\begin{aligned} \text{Var}[S(\mathcal{J}^\sigma)] &= \sum_{J_i, J_j \text{ small}} p_i p_j \text{Cov} \left[ \frac{1}{m-s(J_i)}, \frac{1}{m-s(J_j)} \right] \\ &\leq \sum_{J_i \text{ small}} p_i \cdot \sum_{J_j \text{ small}} p_j \cdot \frac{H_m^2}{m} \frac{1}{(m-\tilde{k})^2} \frac{\pi^2}{3} \\ &\leq \frac{\pi^2}{3} \frac{H_m^2}{m} \left( \frac{L_{\tilde{k}+1}}{m-\tilde{k}} \right)^2. \end{aligned}$$

The last inequality uses again that  $L_{\tilde{k}+1} \geq (m-\tilde{k})\text{OPT}$ . Hence, the standard deviation  $\text{SD}[S(\mathcal{J}^\sigma)]$  is at most  $C^{3/2} \frac{L_{\tilde{k}+1}}{m-\tilde{k}}$  for  $C = \sqrt[3]{\frac{\pi^2}{3} \frac{H_m^2}{m}} = \Theta \left( \sqrt[3]{\frac{(\log(m))^2}{m}} \right)$ . Chebyshev's inequality, which allows to bound the probability of deviating from the mean in terms of the standard deviation, yields

$$\begin{aligned} \mathbf{P} \left[ S(\mathcal{J}^\sigma) \leq \left( \frac{H_m}{m} - C \right) \text{OPT} \right] &\leq \mathbf{P} \left[ S(\mathcal{J}^\sigma) \leq \left( \frac{H_m}{m} - C \right) \frac{L_{\tilde{k}+1}}{m-\tilde{k}} \right] \\ &\leq \mathbf{P} \left[ |\mathbf{E}[S(\mathcal{J}^\sigma)] - S(\mathcal{J}^\sigma)| \geq C^{-1/2} \cdot \sigma[S(\mathcal{J}^\sigma)] \right] \\ &\leq C. \end{aligned}$$

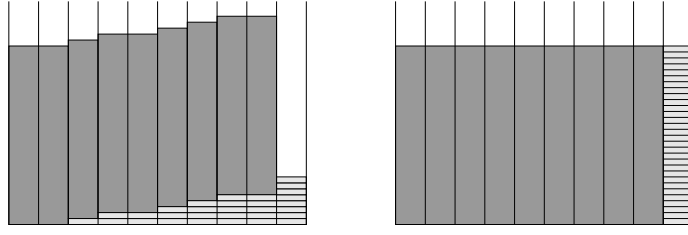
We conclude that with probability  $1 - C$  the minimum load achieved by the Greedy-strategy exceeds  $\min \left\{ S(\mathcal{J}^\sigma) - \frac{H_m}{2m}, \frac{H_m}{2m} \right\} \text{OPT} \geq \frac{H_m}{m} \left( \frac{1}{2} - C \right) \text{OPT}$ . Thus its competitive ratio in the random-order model is at most  $(1 - C) \frac{m}{H_m} \frac{1}{\left( \frac{1}{2} - C \right)} = (2 + o_m(1)) \frac{m}{H_m}$ . ◀

► **Theorem 5.** *The Greedy-strategy is not better than  $\frac{m}{H_m}$ -competitive in the random-order model.*

**Proof.** Consider the following instance for  $\varepsilon > 0$ : Job set  $\mathcal{J}$  consists of  $m - 1$  jobs of size 1 and  $\frac{1}{\varepsilon}$  jobs of size  $\varepsilon$ . It is not difficult to see that  $\text{OPT} = 1$  by assigning the jobs of size 1 to different machines and the jobs of size  $\varepsilon$  together on the remaining machine. Following a similar approach as the one above, interpreting the jobs of size 1 as large and the rest as small, we can prove that for any given order  $\mathcal{J}^\sigma$ , the expected minimum load achieved by the Greedy-strategy is at most  $\sum_{i=0}^{m-1} \frac{S_i(\mathcal{J}^\sigma)}{m-i} + \varepsilon m$ . If now  $\sigma \sim S_n$  is chosen uniformly at random, then the minimum load achieved by the Greedy-strategy is at most  $\sum_{i=0}^{m-1} \frac{1}{m(m-i)} + \varepsilon m = \frac{H_m}{m} + \varepsilon m$ . In particular, the competitive ratio in the random-order model of the Greedy-strategy is at most  $1 / \left( \frac{H_m}{m} + \varepsilon m \right)$  which approaches  $\frac{m}{H_m}$  as  $\varepsilon \rightarrow 0$ . ◀

#### 4 An $\tilde{O}(\sqrt[4]{m})$ -competitive algorithm

We now describe an improved competitive algorithm for the problem further exploiting the extra features of the random-order model. More in detail, we devise a sampling-based method to classify relatively large jobs (with respect to OPT). After this, we run a slight adaptation



■ **Figure 3** A comparison of the solution returned by the Greedy-strategy (left) and the optimal solution (right) for some order  $\sigma$  of the instance defined in Theorem 4. Dark jobs have size 1 and the remaining jobs have size  $\varepsilon > 0$ .

of the algorithm *Partition* due to Azar and Epstein [7] in order to distinguish large jobs that our initial procedure could not classify, leading to strictly better approximation guarantees (see Algorithm 1 for a description). We will assume w.l.o.g. that job sizes are rounded down to powers of 2, which induces an extra multiplicative factor of at most 2 in the competitive ratio.

#### 4.1 Simple and Proper Inputs

Given an input sequence  $\mathcal{J}$ , we call any job  $J$  *large* if its size is larger than  $\frac{\text{OPT}[\mathcal{J}]}{100\sqrt[4]{m}}$ , and we call it *small* otherwise. Let  $k = k[\mathcal{J}]$  be the number of large jobs in  $\mathcal{J}$ . Let  $L_{\text{small}} = L_{k+1}$  be the total size of small jobs. The following definition allows to recognize instances where the Greedy-strategy performs well, see Proposition 7.

► **Definition 6.** We call the input set  $\mathcal{J}$  *simple* if either

- The set  $\mathcal{J}$  has size  $n < m$ ,
- There are at least  $m$  large jobs, i.e.  $k \geq m$ , or
- There are at most  $m - \frac{\sqrt[4]{m^3}}{50}$  large jobs, i.e.  $k \leq m - \frac{\sqrt[4]{m^3}}{50}$ .

Note that the first condition is mostly included for ease of notation; the third condition implies that sequences with  $n < m - \frac{\sqrt[4]{m^3}}{50}$  are simple, which is good enough for our purposes but somewhat clumsy to refer to.

► **Proposition 7.** The Greedy-strategy achieves minimum load at least  $\frac{\text{OPT}}{100\sqrt[4]{m}}$  on simple inputs.

**Proof.** We consider each case from Definition 6 separately:

- If the instance has less than  $m$  jobs,  $\text{OPT} = 0$  and every algorithm is optimal.
- If there are at least  $m$  large jobs in the instance, then the minimum load achieved by the Greedy algorithm is at least  $P_m \geq \frac{\text{OPT}}{100\sqrt[4]{m}}$  thanks to Lemma 2.
- If there are at most  $m - \frac{\sqrt[4]{m^3}}{50}$  large jobs, then  $L_{k+1} \geq \frac{\sqrt[4]{m^3}}{50}\text{OPT}$  as there are at least  $\frac{\sqrt[4]{m^3}}{50}$  machines without large jobs in the optimal solution. If we apply Lemma 3 with  $i = k + 1$ , the minimum load achieved by the Greedy algorithm is at least  $\frac{\text{OPT}}{100\sqrt[4]{m}}$ . ◀

For an input set  $\mathcal{J}$  which is not simple, let  $d := \lceil \log_2(m - k) \rceil$ . Note that  $0 \leq d \leq \lceil \frac{3}{4} \log(m) \rceil$ . We say that such an instance  $\mathcal{J}$  is *proper* (of degree  $d$ ).

Algorithm 1 guesses a value  $t$  with probability  $\Omega(1/\log(m))$  to address in a different way simple instances (case  $t = -1$ ) and proper instances of degree  $t$  for each  $0 \leq t \leq \lceil \frac{3}{4} \log(m) \rceil$ , at the expense of an extra logarithmic factor in the competitive ratio. By Proposition 7, simple instances are sufficiently handled by the Greedy-strategy. From now on we will focus on proper instances of degree  $d$  and show that the corresponding case when  $t = d$  in Algorithm 1 returns a  $O(\sqrt[4]{m})$ -approximate solution.

■ **Algorithm 1** The online Algorithm for Random-Order Machine Covering.

---

**Input:** Job sequence  $\mathcal{J}^\sigma$ ,  $m$  identical parallel machines.

- 1: Guess  $t \in \{-1, 0, 1, \dots, \lceil \frac{3}{4} \log(m) \rceil\}$  uniformly at random.
  - 2: **if**  $t = -1$ , **then** Run the Greedy-strategy and **return** the computed solution.
  - 3: **else** Partition the machines into  $2^t$  *small* machines and  $m - 2^t$  *large* machines.
- 

*Phase 1 – Sampling.*

---

- 4: Schedule the first  $n/8$  jobs iteratively into a least loaded large machine.
  - 5: Let  $P^\uparrow$  be the  $\left(\frac{m-2^t}{8} - \frac{\sqrt{m}}{2}\right)$ -th largest job size among these first  $n/8$  jobs.
- 

*Phase 2 – Partition.*

---

- 6:  $\tau \leftarrow 0$
  - 7: **for**  $j = \frac{n}{8} + 1, \dots, n$  **do**
  - 8:     **if**  $p_{\sigma(j)} \geq P^\uparrow$ , **then** Schedule job  $J_{\sigma(j)}$  onto a least loaded large machine.
  - 9:     **if**  $p_{\sigma(j)} > \tau$ , **then** Update  $\tau$  to  $p_{\sigma(j)}$  with probability  $\frac{1}{9 \cdot 2^t \sqrt{m}}$ .
  - 10:    **if**  $p_{\sigma(j)} \leq \tau$ , **then** Schedule job  $J_{\sigma(j)}$  onto a least loaded small machine.
  - 11:    **if**  $p_{\sigma(j)} > \tau$ , **then** Schedule job  $J_{\sigma(j)}$  onto a least loaded large machine.
  - 12: **end for**
  - 13: **return** the computed solution.
- 

## 4.2 Algorithm for Proper Inputs of Degree $d$

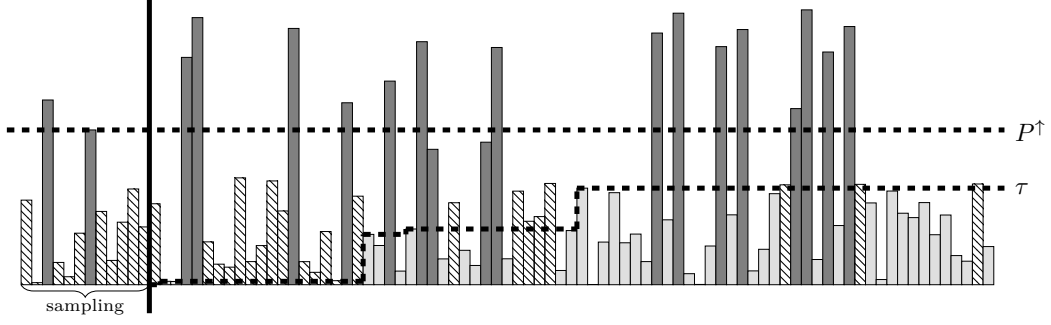
Let  $\mathcal{J}$  be proper of degree  $d$  and assume without loss of generality that its size  $n$  is divisible by 8 (an online algorithm can always simulate up to 7 extra jobs of size 0 to reduce to this case). The algorithm, assuming  $d$  is guessed correctly, chooses  $2^d$  *small* machines  $\mathcal{M}_{\text{small}}$ , while the other machines  $\mathcal{M}_{\text{large}}$  are called *large machines*. The algorithm will always assign the incoming job either to a least loaded small or to a least loaded large machine, the only choice it has to make is to which set of machines the job will go. Theoretically, the goal would be to assign all large jobs to large machines and all small ones to small machines according to our definition. Large and small jobs, unfortunately, cannot be distinguished by an online algorithm with certainty. Instead, we have to use randomization and expect a small error. We aim for a small one-sided error, meaning that we want to avoid misclassifying large jobs at all cost while incorrectly labeling very few small jobs as large.

The algorithm starts with a *sampling phase*: the first  $\frac{n}{8}$  jobs will be used for sampling purposes, and since we yet lack good knowledge about what should be considered large, these jobs will all be assigned to large machines. Let  $P^\uparrow$  be the element of rank  $\frac{m-2^d}{8} - \frac{\sqrt{m}}{2}$  among these elements. The following lemma shows that  $P^\uparrow$  can be used as a threshold to distinguish most of the large jobs from small ones.

► **Lemma 8.** *For proper sequences, it holds that  $\mathbf{P}[P_{k-8\sqrt{m-2^d}} \geq P^\uparrow \geq P_k] \geq \frac{1}{3}$ .*

**Proof.** Let  $n_{\text{large}}$  be the number of large jobs among the first  $\frac{n}{8}$  jobs in the sequence. Notice that  $n_{\text{large}}$  obeys an hypergeometric distribution with parameters  $N = n$  (size of the total population),  $K = k$  (number of elements with the desired property) and  $r = \frac{n}{8}$  (size of the sample). This implies that  $\mathbf{E}_{\sigma \sim S_n}[n_{\text{large}}] = \frac{k}{8} \geq \frac{m-2^d}{8}$ . Moreover

$$\text{Var}[n_{\text{large}}] = \frac{n}{8} \cdot \frac{k}{n} \cdot \frac{n-k}{n} \cdot \frac{7n}{8(n-1)} = \frac{7}{64} \cdot \frac{k(n-k)}{n-1} < \frac{m}{8}.$$



■ **Figure 4** The classification of large (dark) and small (light and dashed) jobs. During sampling, all small jobs are misclassified (dashed ones). Threshold  $P^\uparrow$  classifies large jobs, while threshold  $\tau$  classifies small jobs. Jobs in between are conservatively classified as large, since misclassifying large jobs is fatal. Increasing  $\tau$  due to a small job is a helpful event as less small jobs will be misclassified. On the other hand, increasing  $\tau$  due to a large job below  $P^\uparrow$  is a fatal event as large jobs will be misclassified. The choice of  $P^\uparrow$  ensures that fatal events are unlikely to happen.

If we use Cantelli's inequality, a one-sided version of Chebyshev's inequality, then

$$\mathbf{P}[P^\uparrow < P_k] = \mathbf{P}\left[n_{\text{large}} < \frac{m-2^d}{8} - \frac{\sqrt{m}}{2}\right] \leq \mathbf{P}\left[n_{\text{large}} < \mathbf{E}[n_{\text{large}}] - \frac{\sqrt{m}}{2}\right] \leq \frac{m/8}{\frac{m}{8} + \frac{m}{4}} = \frac{1}{3}.$$

Consider now  $n'_{\text{large}}$  to be the number of the  $k - 8\sqrt{m} - 2^d$  largest jobs among the  $\frac{n}{8}$  first jobs in the sequence. Similarly as before,  $n'_{\text{large}}$  obeys an hypergeometric distribution with parameters  $N = n$ ,  $K = k - 8\sqrt{m} - 2^d$  and  $r = \frac{n}{8}$ , which implies that  $\mathbf{E}_{\sigma \sim S_n}[n'_{\text{large}}] = \frac{k-2^d-8\sqrt{m}}{8} < \frac{m-2^d}{8} - \sqrt{m}$ . Here we use that  $k \leq m$ . Furthermore  $\text{Var}[n'_{\text{large}}] = \frac{n}{8} \cdot \frac{k-8\sqrt{m}-2^d}{n}$ .  $\frac{n-k+8\sqrt{m}+2^d}{n} \cdot \frac{7n}{8(n-1)} \leq \frac{m}{8}$ . Then, using again Cantelli's inequality, we obtain that

$$\begin{aligned} \mathbf{P}[P_{k-8\sqrt{m}-2^d} < P^\uparrow] &= \mathbf{P}\left[n'_{\text{large}} \geq \frac{m-2^d}{8} - \frac{\sqrt{m}}{2}\right] \\ &\leq \mathbf{P}\left[n'_{\text{large}} \geq \mathbf{E}[n'_{\text{large}}] + \frac{\sqrt{m}}{2}\right] \leq \frac{m/8}{\frac{m}{8} + \frac{m}{4}} = \frac{1}{3}. \end{aligned}$$

We conclude that  $\mathbf{P}[P_{k-8\sqrt{m}-2^d} \geq P^\uparrow \geq P_k] = 1 - \mathbf{P}[P_{k-8\sqrt{m}-2^d} < P^\uparrow] - \mathbf{P}[P^\uparrow < P_k] \geq 1/3$ . ◀

After the aforementioned sampling phase is finished and  $P^\uparrow$  is known, the algorithm will enter a *partition phase*. If a job now has size at least  $P^\uparrow$ , it is assigned to the large machines as it will be large with high probability. The large jobs below this value are the most difficult to assign. To this end, we define a *threshold value*  $\tau$ , which we initialize to 0. If the incoming job has size at most  $\tau$  we simply assign it to a least loaded small machine, but whenever we encounter a job  $J$  of size  $p > \tau$ , we set  $\tau = p$  with probability  $\frac{1}{9 \cdot 2^d \sqrt{m}}$ . If now  $p \leq \tau$ , which could happen if we just increased  $\tau$ , we schedule  $J$  on the least loaded small machine. Else,  $J$  is assigned to the least loaded machine in  $\mathcal{M}_{\text{large}}$  (see Figure 4 for a depiction of the procedure). As the following lemma shows, this procedure distinguishes all the large jobs with constant probability.

► **Lemma 9.** *All large jobs are scheduled onto large machines with constant probability.*

**Proof.** Let us assume that  $P_{k-8\sqrt{m}-2^d} \geq P^\dagger \geq P_k$ . Now, the statement of the lemma can only be wrong if we decided to increase  $\tau$  when encountering some large job of size less than  $P^\dagger$ . By assumption there are less than  $8\sqrt{m} + 2^d$  such jobs. The desired probability is thus at least

$$\left(1 - \frac{1}{9 \cdot 2^d \sqrt{m}}\right)^{8\sqrt{m}+2^d} \geq 1 - \frac{8\sqrt{m} + 2^d}{9 \cdot 2^d \sqrt{m}} \geq 1 - \frac{8}{9 \cdot 2} - \frac{1}{9\sqrt{m}} > \frac{4}{9}.$$

The first inequality is Bernoulli's inequality, the second one uses that  $d \geq 1$ . The lemma follows by multiplying with the probability from Lemma 8.  $\blacktriangleleft$

We call an input sequence *orderly* if it satisfies the properties of Lemmas 8 and 9. The following lemma shows that the total size of misclassified small jobs can be, in expectation, bounded from above. This proof is an adaptation of one of the results from Azar and Epstein [7], which we present for the sake of completeness.

► **Lemma 10.** *The expected total size of small jobs scheduled onto small machines is at least  $\frac{31}{800\sqrt[4]{m}}L_{\text{small}}$ , even when conditioned on the input sequence being orderly.*

**Proof.** Let  $L'_{\text{small}}$  be the random variable corresponding to the size of small jobs assigned to large machines in the sampling phase. Since jobs appear in random order, we have that

$$\mathbf{E}_{\sigma \sim S_n}[L'_{\text{small}}] = \frac{1}{n!} \sum_{J_i \text{ small}} \frac{n}{8} \cdot (n-1)! \cdot p_i = \frac{1}{8}L_{\text{small}}.$$

Let us now bound the total size of misclassified small jobs in the partition phase. We will define, for a given set of  $18 \cdot 2^d \sqrt[4]{m}$  small jobs of the same size  $p_i$  (recall that jobs are rounded down to powers of 2), the following event: after  $9 \cdot 2^d \sqrt[4]{m}$  jobs from the set arrived,  $\tau$  is at least  $p_i$ . If this were not the case,  $\tau$  was never updated at any of these  $9 \cdot 2^d \sqrt[4]{m}$  first jobs albeit being smaller than  $p_i$ . The probability for this is at most

$$\left(1 - \frac{1}{9 \cdot 2^d \sqrt[4]{m}}\right)^{9 \cdot 2^d \sqrt[4]{m}} \leq e^{-\frac{1}{\sqrt[4]{m}}} \leq 1 - \frac{1}{2\sqrt[4]{m}},$$

where we used the fact that  $e^{-x} \leq 1 - \frac{x}{2}$  if  $0 \leq x \leq 1$ . This implies that the probability of the previous event occurring is at least  $\frac{1}{2\sqrt[4]{m}}$ .

Let  $\mathcal{S}$  be the set of small jobs remaining after the sampling phase. We will partition  $\mathcal{S}$  into batches of  $18\sqrt[4]{m}$  jobs of the same size. There will be jobs that are not assigned to any batch because there are not enough jobs of the same size to complete it, but the total size of these jobs is at most

$$18 \cdot 2^d \sqrt[4]{m} \sum_{i \geq 1} \frac{\text{OPT}}{100\sqrt[4]{m} \cdot 2^i} \leq \frac{18 \cdot 2^{d-1} \cdot \text{OPT}}{25} \leq \frac{18L_{\text{small}}}{25},$$

where the last inequality holds as there are at least  $2^{d-1}$  machines in the optimal solution without large jobs. For each of the batches, the probability of assigning at least half of its size to small machines is bounded below by the probability of the previously described event occurring, which is at least  $\frac{1}{2\sqrt[4]{m}}$ . Hence, the expected total size of small jobs assigned to small machines is at least

$$\frac{1}{2\sqrt[4]{m}} \cdot \frac{1}{2} \sum_{J_i \in \mathcal{S}} p_i \geq \frac{1}{4\sqrt[4]{m}} \cdot \left(1 - \frac{1}{8} - \frac{18}{25}\right) L_{\text{small}} = \frac{31}{800\sqrt[4]{m}} L_{\text{small}}.$$

To observe that we can condition on the sequence being orderly, it suffices to note that the arguments work for every way to fix  $P^\dagger$  and that they do not make any assumptions on  $\tau$  being increased at large jobs.  $\blacktriangleleft$



Putting all the previous ingredients together we can conclude the following proposition.

► **Proposition 11.** *The previously described algorithm is  $O(\sqrt[d]{m})$ -competitive in the random-order model for the case of proper inputs of degree  $d$ .*

**Proof.** By assumption, all  $k \geq m - 2^d$  large jobs are scheduled onto large machines. A lower bound of  $P_k = \frac{\text{OPT}}{100 \sqrt[d]{m}}$  for the minimum load achieved in the large machines follows then from Lemma 2. By Lemma 3, the minimum load among small machines is at least  $\frac{31}{800 \sqrt[d]{m}} \frac{L_{\text{small}}}{2^d} - \frac{\text{OPT}}{100 \sqrt[d]{m}}$ . The proposition follows from observing that  $L_{\text{small}} \geq 2^{d-1} \text{OPT}$  since the optimal solution contains at least  $m - k \geq 2^{d-1}$  machines with only small jobs. ◀

### 4.3 The Final Algorithm

As discussed before, our final algorithm first guesses whether the instance is simple or proper of degree  $d$ . Then we apply the appropriate algorithm, the Greedy-strategy or the previously described algorithm for the right degree. Since there are  $O(\log(m))$  many possibilities, this guessing induces an extra logarithmic factor on the final competitive ratio, which concludes the proof of Theorem 1 restated below.

► **Theorem 1.** *There exists a  $\tilde{O}(\sqrt[d]{m})$ -competitive algorithm for the online Machine Covering problem in the random-order model.*

## 5 A Lower Bound for the Random-Order Model

The main difficulty for Online Machine Covering algorithms, including our main result, is to tell large jobs apart from the largest small jobs. In this section we prove that doing so is, to a certain extent, inherently hard. The main difference to adversarial models is that hardness is not obtained through withholding information but rather through obscuring it. This relates to some studied variants of the classical Secretary Problem such as the Postdoc Problem [43] but requires additional features particularly catered to our needs.

### 5.1 The Talent Contest Problem

Consider the following selection problem: To a yearly talent show contest  $n$  candidates apply. To appeal to a general audience, we try to exclude the best candidates because an imperfect performance is more entertaining, but we also want to have at least an appropriate candidate who can be presented as the winner. To do so, each candidate will participate in  $t$  trials (each trial is considered as an arrival) and we must decide for every arrival if we mark the candidate or not, meaning that we consider her to be the  $K$ -th best candidate or worse. The global order in which candidates arrive for trials is uniformly distributed, thus at later trials we have much more information to go by. Our final goal is to maximize the number of trials for which we successfully marked the  $K$ -th best candidate without marking any better candidate.

Formally, the *Talent Contest* problem is specified by three parameters  $K$ ,  $n$  and  $t$ , where  $K \leq n$ . Candidates have pairwise different non-negative valuations  $v_1, v_2, \dots, v_n$ , and each candidate arrives  $t$  times; the arrival order is chosen uniformly at random. The valuation of each candidate is revealed when the candidate arrives for the first time. We may decide to mark each arrival or not, though once the next candidate arrives such marking decision is permanent. For each value  $1 \leq h \leq t$  we get one point if we marked the  $h$ -th arrival of the  $K$ -th best candidate, but not the  $h$ -th arrival of any better candidate. In

particular, we can get up to  $t$  points in total, one for each value of  $h$ . Let  $P(K, t, n)$  be the expected number of points the optimal online algorithm scores given the three values  $K, t$  and  $n$ . Similar to the classical Secretary problem, we are mostly interested in the limit case  $P(K, t) = \lim_{n \rightarrow \infty} P(K, t, n)$ .

We require for our desired results one extra technical definition. Given  $\lambda \geq 1$ , we call the valuations of candidates  $\lambda$ -steep if all candidate valuations are guaranteed to be at least by a factor  $\lambda$  apart, i.e.  $\min_{v_i > v_j} \frac{v_i}{v_j} \geq \lambda$ . It is possible to prove the following bound on the expected value  $P(K, t)$ , whose proof we defer to the full version of the paper.

► **Lemma 12.** *It holds that  $P(K, t) \leq \frac{\zeta(t/2)(t+1)^{t/2}}{2\pi\sqrt{K}}$ , where  $\zeta$  is the Riemann Zeta Function. This bound still holds if we restrict ourselves to  $\lambda$ -steep valuations for some  $\lambda \geq 1$ .*

Roughly speaking, the proof relies on the fact that if an algorithm manages to perform relatively well in the Talent Contest problem, then it could be used to guess the value of a binomially distributed random variable. For the latter problem, difficulty can be directly established. However, the proof involves more technical aspects as some few irregular orders do not allow this reduction and have to be excluded beforehand. The property of  $\lambda$ -steepness is ensured by choosing  $\mu$  sufficiently large and applying the transformation  $v \mapsto \mu^v$  to each valuation.

## 5.2 Reduction to Machine Covering

It is possible to show that the Talent Contest problem and the Online Machine Covering problem in the random-order model are related as the following lemma states.

► **Lemma 13.** *Given  $K$  and  $t$ , let  $m = (K - 1) \cdot t + 1$ . No (possibly randomized) algorithm for Machine Covering in the random-order model on  $m$  machines can be better than  $\frac{t}{P(K,t)+1}$ -competitive.*

**Proof.** Let  $\lambda > t$ . Consider any instance of the Talent Contest problem with  $\lambda$ -steep valuations. We will treat the arrival sequence of candidates as a job sequence, where each arrival corresponds to a job of size given by the valuation of the corresponding candidate. We call the  $m - 1 = t(K - 1)$  jobs corresponding to the arrivals of the  $K$  largest candidates *large*. The  $t$  jobs corresponding to the next candidate are called *medium*. Notice that the size of a medium job is at most  $\frac{OPT}{t}$  as evidenced by the schedule that assigns each large job on a separate machine and the  $t$  medium jobs onto the single remaining machine. Jobs which are neither large nor medium have total size at most  $t \sum_{i=1}^{\infty} \lambda^{-i} \frac{OPT}{t} = \frac{OPT}{\lambda-1}$  and are thus called *small*. They will become negligible for  $\lambda \rightarrow \infty$ .

Consider an online algorithm  $\mathcal{A}_{MC}$  for Machine Covering in the random-order model. We will derive an algorithm  $\mathcal{A}_{TC}$  for the Talent Contest problem as follows:  $\mathcal{A}_{TC}$  marks each job that gets assigned to a machine that already contains a job of the same size. Let  $P$  be the number of points this strategy gets. We will first show that the schedule of  $\mathcal{A}_{MC}$  contains a machine which has at most  $P + 1$  medium jobs and no big job. For this we consider any fixed input order, and if  $\mathcal{A}_{MC}$  is randomized, we consider any fixed outcome of its coin tosses.

For  $2 \leq i \leq t$ , let  $w_i$  be an indicator variable that is 1 if  $\mathcal{A}_{TC}$  gains a point for the  $i$ -th arrival, i.e. if it marks the  $i$ -th arrival of the  $K$ -th best candidate but not the  $i$ -th arrival of a better candidate;  $w_i = 0$  otherwise. Let also  $r_i$  be an indicator variable that is 1 if  $\mathcal{A}_{TC}$  marked the  $i$ -th arrival of the  $K$ -th best candidate but still loses due to also marking the  $i$ -th arrival of a better candidate. Finally, let  $\mathcal{M}_{\text{small}}$  be the machines which do not receive a large job in the schedule of  $\mathcal{A}_{MC}$ , and let  $Z_{\text{med}}$  be the average number of medium jobs on machines in  $\mathcal{M}_{\text{small}}$ . Our intermediate goal is to show that  $Z_{\text{med}} < 2 + \sum_{i=2}^t w_i$ .

Since there are only  $m - 1$  large jobs, of which at least  $\sum_{i=2}^t r_i$  are scheduled on a machine already containing a large job, we have that  $|\mathcal{M}_{\text{small}}| \geq 1 + \sum_{i=2}^t r_i$ . Let  $d \geq 0$  such that  $|\mathcal{M}_{\text{small}}| = 1 + d + \sum_{i=2}^t r_i$ . Now, observe that  $\sum_{i=2}^t (w_i + r_i)$  counts the number of medium jobs that are placed on a machine already containing a medium job. Thus, the number of medium jobs on machines in  $\mathcal{M}_{\text{small}}$  is at most  $|\mathcal{M}_{\text{small}}| + \sum_{i=2}^t (w_i + r_i) = 1 + d + \sum_{i=2}^t (w_i + 2r_i)$ . Now we can bound the average number of medium jobs on  $\mathcal{M}_{\text{small}}$ , namely  $Z_{\text{med}} \leq \frac{1+d+\sum_{i=2}^t (w_i+2r_i)}{1+d+\sum_{i=2}^t r_i}$ . Let us assume that  $Z_{\text{med}} \geq 2$ ; then the term on the right hand side increases if we set  $d$  and all  $r_i$  to zero, and we obtain that  $Z_{\text{med}} \leq 1 + \sum_{i=2}^t w_i$  and thus  $Z_{\text{med}} < 2 + \sum_{i=2}^t w_i$ . To derive this inequality we assumed that  $Z_{\text{med}} \geq 2$  but it is trivially true if  $Z_{\text{med}} < 2$ .

Now, let  $z_{\text{med}}$  be the least number of medium jobs on a machine in  $\mathcal{M}_{\text{small}}$ . Then  $z_{\text{med}} \leq Z_{\text{med}} < 2 + \sum_{i=2}^t w_i$ . Since  $z_{\text{med}}$  and the right hand side are both integers, it holds that  $z_{\text{med}} \leq 1 + \sum_{i=2}^t w_i$ . Since  $\sum_{i=2}^t w_i = P$ , the number of points obtained by algorithm  $\mathcal{A}_{TC}$  for the Talent Contest problem, we have shown that the schedule of  $\mathcal{A}_{MC}$  contains a machine  $M$  with at most  $P + 1$  medium jobs and no large one as desired.

As argued before, each medium job has size at most  $\frac{\text{OPT}}{t}$  and the small jobs have total size at most  $\frac{\text{OPT}}{\lambda-1}$  in total. Thus, machine  $M$  has load at most  $\frac{P+1}{t}\text{OPT} + \frac{\text{OPT}}{\lambda-1}$ . In conclusion, the expected load of the least loaded machine in the schedule of  $\mathcal{A}_{MC}$  is at most  $\frac{P(K,t)+1}{K}\text{OPT} + \frac{\text{OPT}}{\lambda-1}$ , given a worst-case input for the Talent Contest Problem. This concludes the proof by taking  $\lambda \rightarrow \infty$ .  $\blacktriangleleft$

By setting  $K = (t+1)^t$  and combining this with the lower bound in Lemma 12, we obtain the following general lower bound.

► **Theorem 14.** *The competitive ratio of no online algorithm for Machine Covering in the random-order model, deterministic or randomized, is better than  $\frac{\lfloor e^{W(\ln(m))} \rfloor - 1}{1.16+o(1)}$ . Here,  $W(x)$  is the Lambert  $W$ -function, i.e. the inverse to  $x \mapsto xe^x$ . In particular, no algorithm can be  $O\left(\frac{\log(m)}{\log \log(m)}\right)$ -competitive for Online Machine Covering in the random-order model.*

**Proof.** Let  $K = (t+1)^t$ . Then Lemma 12 yields  $P(K, t) \leq \frac{\zeta(t/2)}{2\pi} \leq 1.16+o(1)$ . By Lemma 13 no algorithm can be better than  $\left(\frac{t}{1.16+o(1)}\right)$ -competitive for  $m = (K-1) \cdot t + 1 < (t+1)^{t+1}$ . We can always add a few jobs of large enough size so that the lower bound extends to larger numbers of machines. The theorem follows since the inverse function of  $x \mapsto (t+1)^{t+1}$  is  $t \mapsto e^{W(\ln(m))} - 1$ ; the second part uses the identity  $W(x) \geq \log(x) - \log \log(x) + \omega(1)$ , see [28].  $\blacktriangleleft$

---

## References

- 1 Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999. doi:10.1137/S0097539797324874.
- 2 Susanne Albers and Maximilian Janke. Scheduling in the random-order model. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168, pages 68:1–68:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.68.
- 3 Susanne Albers and Maximilian Janke. Scheduling in the secretary model. In *41st Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2021. To appear.
- 4 Susanne Albers, Arindam Khan, and Leon Ladewig. Best fit bin packing with random order revisited. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 170, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.7.

- 5 Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021. doi:10.1007/s00453-021-00801-2.
- 6 Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM J. Comput.*, 39(7):2970–2989, 2010. doi:10.1137/080723491.
- 7 Yossi Azar and Leah Epstein. On-line machine covering. *Journal of Scheduling*, 1(2):67–77, 1998. doi:10.1002/(SICI)1099-1425(199808)1:2<67::AID-JOS6>3.0.CO;2-Y.
- 8 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop (APPROX/RANDOM)*, volume 4627, pages 16–28. Springer, 2007. doi:10.1007/978-3-540-74208-1\_2.
- 9 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Matroid secretary problems. *J. ACM*, 65(6):35:1–35:26, 2018. doi:10.1145/3212512.
- 10 Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40. ACM, 2006. doi:10.1145/1132516.1132522.
- 11 Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995. doi:10.1006/jcss.1995.1074.
- 12 Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994. doi:10.1016/0020-0190(94)00026-3.
- 13 Eugene B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Math. Dokl.*, 4, 1962.
- 14 Tomás Ebenlendr, John Noga, Jirí Sgall, and Gerhard J. Woeginger. A note on semi-online machine covering. In *Approximation and Online Algorithms, Third International Workshop (WAOA)*, volume 3879, pages 110–118. Springer, 2005. doi:10.1007/11671411\_9.
- 15 Leah Epstein. A survey on makespan minimization in semi-online environments. *J. Sched.*, 21(3):269–284, 2018. doi:10.1007/s10951-018-0567-z.
- 16 Leah Epstein, Asaf Levin, and Rob van Stee. Max-min online allocations with a reordering buffer. *SIAM J. Discret. Math.*, 25(3):1230–1250, 2011. doi:10.1137/100794006.
- 17 Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.
- 18 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple  $O(\log \log(\text{rank}))$ -competitive algorithm for the matroid secretary problem. *Math. Oper. Res.*, 43(2):638–650, 2018. doi:10.1287/moor.2017.0876.
- 19 Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms, 8th Annual European Symposium (ESA)*, volume 1879, pages 202–210. Springer, 2000. doi:10.1007/3-540-45253-2\_19.
- 20 Donald K. Friesen and Bryan L. Deuermeier. Analysis of greedy solutions for a replacement part sequencing problem. *Math. Oper. Res.*, 6(1):74–87, 1981. doi:10.1287/moor.6.1.74.
- 21 Gábor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham’s list scheduling. *SIAM J. Comput.*, 22(2):349–355, 1993. doi:10.1137/0222026.
- 22 Waldo Gálvez, José A. Soto, and José Verschae. Symmetry exploitation for online machine covering with bounded migration. *ACM Trans. Algorithms*, 16(4):43:1–43:22, 2020. doi:10.1145/3397535.
- 23 Oliver Göbel, Thomas Kesselheim, and Andreas Tönnis. Online appointment scheduling in the random order model. In *Algorithms, 23rd Annual European Symposium (ESA)*, volume 9294, pages 680–692. Springer, 2015. doi:10.1007/978-3-662-48350-3\_57.
- 24 Todd Gormley, Nick Reingold, Eric Torng, and Jeffery R. Westbrook. Generating adversaries for request-answer games. In *11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 564–565. ACM/SIAM, 2000.

- 25 Anupam Gupta, Ruta Mehta, and Marco Molinaro. Maximizing profit with convex costs in the random-order model. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.71.
- 26 Anupam Gupta and Sahil Singla. Random-order models. In *Beyond the Worst-Case Analysis of Algorithms*, pages 234–258. Cambridge University Press, 2020. doi:10.1017/9781108637435.015.
- 27 Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. doi:10.1145/7531.7535.
- 28 Abdolhossein Hoorfar and Mehdi Hassani. Inequalities on the lambert w function and hyperpower function. *J. Inequal. Pure and Appl. Math*, 9(2):5–9, 2008.
- 29 David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20(2):400–430, 1996. doi:10.1006/jagm.1996.0019.
- 30 Claire Kenyon. Best-fit bin-packing with random order. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–364. ACM/SIAM, 1996.
- 31 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. *SIAM J. Comput.*, 47(5):1939–1964, 2018. doi:10.1137/15M1033708.
- 32 Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 630–631. SIAM, 2005.
- 33 Oded Lachish.  $O(\log \log \text{rank})$  competitive ratio for the matroid secretary problem. In *55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 326–335. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.42.
- 34 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1859–1877. SIAM, 2020. doi:10.1137/1.9781611975994.114.
- 35 D. V. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society*, 10(1):39–51, March 1961.
- 36 Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 426–431. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959917.
- 37 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020. doi:10.1017/9781108637435.037.
- 38 Marco Molinaro. Online and random-order load balancing simultaneously. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1638–1650. SIAM, 2017. doi:10.1137/1.9781611974782.108.
- 39 Christopher J. Osborn and Eric Torng. List’s worst-average-case or WAC ratio. *J. Sched.*, 11(3):213–215, 2008. doi:10.1007/s10951-007-0019-7.
- 40 John F. Rudin III. *Improved bounds for the on-ligne scheduling problem*. PhD thesis, The University of Texas at Dallas, 2001.
- 41 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009. doi:10.1287/moor.1090.0381.
- 42 Martin Skutella and José Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math. Oper. Res.*, 41(3):991–1021, 2016. doi:10.1287/moor.2015.0765.
- 43 Robert J. Vanderbei. The postdoc variant of the secretary problem. Unpublished. URL: <http://www.princeton.edu/~rvdb/tex/PostdocProblem/PostdocProb.pdf>.
- 44 Gerhard J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20(4):149–154, 1997. doi:10.1016/S0167-6377(96)00055-7.

# Nearly-Tight Lower Bounds for Set Cover and Network Design with Deadlines/Delay

Noam Touitou

Tel Aviv University, Israel

---

## Abstract

---

In network design problems with deadlines/delay, an algorithm must make transmissions over time to satisfy connectivity requests on a graph. To satisfy a request, a transmission must be made that provides the desired connectivity. In the deadline case, this transmission must occur inside a time window associated with the request. In the delay case, the transmission should be as soon as possible after the request's release, to avoid delay cost.

In FOCS 2020, frameworks were given which reduce a network design problem with deadlines/delay to its classic, offline variant, while incurring an additional competitiveness loss factor of  $O(\log n)$ , where  $n$  is the number of vertices in the graph. Trying to improve upon this loss factor is thus a natural research direction.

The frameworks of FOCS 2020 also apply to *set cover with deadlines/delay*, in which requests arrive on the elements of a universe over time, and the algorithm must transmit sets to serve them. In this problem, a universe of sets and elements is given, requests arrive on elements over time, and the algorithm must transmit sets to serve them.

In this paper, we give nearly tight lower bounds for set cover with deadlines/delay. These lower bounds imply nearly-tight lower bounds of  $\Omega(\log n / \log \log n)$  for a few network design problems, such as node-weighted Steiner forest and directed Steiner tree. Our results imply that the frameworks in FOCS 2020 are essentially optimal, and improve quadratically over the best previously-known lower bounds.

**2012 ACM Subject Classification** Theory of computation; Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

**Keywords and phrases** Network Design, Deadlines, Delay, Online, Set Cover

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.53

## 1 Introduction

### Network Design with Deadlines

In network design problems with deadlines, one is given a set of connectivity requests. Each connectivity request has an associated time window, starting with a release time and ending with a deadline. The input also contains items, with associated costs; usually, these items are some characteristic of a given graph, such as the edges or the nodes. A solution consists of a set of *transmissions*, taking place at various points in time, where each transmission is of some set of items. Such a solution is feasible if for every connectivity request, there exists a transmission which occurs within the request's time window, and provides the connectivity desired by the request.

This general description captures many network design problems. Two concrete examples are node-weighted Steiner forest and directed Steiner tree, both of which are considered in this paper.

- In *node-weighted Steiner forest with deadlines*, the items are the nodes of a given graph, with associated costs. A connectivity request is of some pair of terminal nodes in the graph, demanding that a connection be made between these nodes. A set of items (i.e.



© Noam Touitou;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 53; pp. 53:1–53:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



- nodes) which satisfies a request must contain a path connecting the request's terminal nodes; such a set must be transmitted within the request's time window to satisfy that request.
- In *directed Steiner tree with deadlines*, the items are the edges of a directed graph, with associated costs. A unique node in the graph is designated as the root of the graph; each connectivity request is of some terminal node, demanding its connection to the root node. A set of items (i.e. edges) which satisfies a request must contain a directed path connecting the request's terminal node to the root. Such a satisfying set must be transmitted within the request's time window for the solution to be feasible.

Network design with deadlines has been studied in the online setting, in which the algorithm constructs a solution as time advances, deciding at each point in time whether (and what) to transmit at that time. Each request is revealed to the algorithm at the request's release time. In line with previous work, we consider the clairvoyant model, where all parameters of the request are revealed at release time (in particular its deadline).

A more general model is *network design with delay*. In this model, each request has a (nondecreasing, continuous) delay function in lieu of a deadline. Each request can be served by a satisfying transmission after its release time. However, in addition to the cost of transmissions, a solution must also pay the *delay cost* of each request – the value of its delay function at the time of the earliest transmission which satisfies it (after its release time). This model can easily be seen to generalize the deadline model.

In [8], a framework for network design problems with deadlines is presented. This framework is in fact a polynomial-time reduction from online network design with deadlines to offline, classic network design: given a  $\gamma$ -approximation algorithm for the offline problem, the framework yields an  $O(\gamma \log |U|)$ -competitive algorithm for the online problem with deadlines, where  $U$  is the set of items. For most problems, in which the items are either the nodes or edges of a given simple graph  $G = (V, E)$ , we have that  $\log |U| = O(\log n)$  – we only consider such problems in this paper. A similar framework for network design with delay is also given in [8], also based on a reduction with  $O(\log n)$  loss<sup>1</sup>.

In [8], a lower bound of  $\Omega(\sqrt{\log n})$  is given on the competitiveness of any (randomized) algorithm for specific network design problems (node-weighted Steiner tree and directed Steiner tree). This implies that *every* reduction incurs a loss factor of  $\Omega(\sqrt{\log n})$  (indeed, this information-theoretic lower bound applies even when we allow exponential time, which admits a 1-approximation for the offline problem). This leaves a quadratic gap between the upper and lower bounds; a natural research question would be to close this gap. In this paper, we give a negative answer to this question; in fact, we show that the framework of [8] is essentially tight.

### Set Cover with Deadlines

The lower bounds of  $\Omega(\sqrt{\log n})$ -competitiveness for node-weighted Steiner tree and directed Steiner tree stem from a lower bound for the problem of set cover with deadlines, which is a special case of both problems. In the problem of set cover with deadlines, one is given a universe which consists of a set of elements  $E$  and a collection  $S$  of subsets from  $E$ , where the sets have costs. The input contains a set of requests, such that each request has an

---

<sup>1</sup> The reduction for the delay case given in [8] is to the prize-collecting offline problem, a similar offline problem which is almost always approximable to the same degree as the classic offline problem (up to some small constant).



associated element in  $E$  and a time window (release time and deadline). A solution consists of a set of transmissions, each of which occurs at some point in time, and consists of some set in  $S$ . A solution is feasible if the associated element of every request belongs to the set of some transmission which occurs during the request's time window. While not classically considered a network design problem, the set cover with deadlines problem does conform to the model of this paper: the sets are the items of the problem, and the requests are of specific elements.

Set cover with deadlines is indeed a special case of both node-weighted Steiner forest with deadlines and directed Steiner tree with deadlines, as seen by folklore reductions. In both reductions, the set cover input is reduced to graph comprising a root node, “set” nodes and “element” nodes, where the root node must connect to “element” nodes through the “set” nodes; see [8] for a full description of these reductions.

The best known lower bounds for competitiveness in the set cover with deadlines problem, described in [3], are  $\Omega(\sqrt{\log \ell})$  and  $\Omega(\sqrt{\log m})$ , where  $\ell$  and  $m$  are the number of elements and the number of sets in the universe, respectively. In this paper, we improve these lower bounds to  $\Omega\left(\frac{\log \ell}{\log \log \ell}\right)$  and  $\Omega\left(\frac{\log m}{\log \log m}\right)$  respectively.

## 1.1 Our Results

In this paper, we present lower bounds for three network design problems with deadlines which are tight up to a log log factor. For set cover with deadlines or delay, where  $\ell$  and  $m$  are the number of elements and sets respectively, we prove the following theorem.

► **Theorem 1.1.** *There exist  $\Omega\left(\frac{\log \ell}{\log \log \ell}\right)$  and  $\Omega\left(\frac{\log m}{\log \log m}\right)$  lower bounds on the competitiveness on any randomized online algorithm for set cover with deadlines (or delay).*

For node-weighted Steiner tree with deadlines and directed Steiner tree with deadlines or delay, for a graph with  $n$  vertices, we prove the following theorems.

► **Theorem 1.2.** *There exists an  $\Omega\left(\frac{\log n}{\log \log n}\right)$  lower bound on the competitiveness on any randomized online algorithm for node-weighted Steiner tree with deadlines (or delay).*

► **Theorem 1.3.** *There exists an  $\Omega\left(\frac{\log n}{\log \log n}\right)$  lower bound on the competitiveness on any randomized online algorithm for directed Steiner tree with deadlines (or delay).*

The lower bounds of Theorems 1.1–1.3 are nearly tight – the framework of [8] implies  $O(\log n)$ -competitive algorithms for both node-weighted Steiner forest with deadlines and directed Steiner tree with deadlines, as well as  $O(\log m)$ - and  $O(\log \ell)$ -competitive algorithms for set cover with deadlines<sup>2</sup>.

We prove the lower bounds of Theorems 1.1–1.3 for the deadline case only; as deadlines are a special case of delay, the lower bounds apply to the delay model as well.

## 1.2 Our Techniques

The lower bounds of this paper stem from describing a sequence of adversaries  $(\mathcal{A}_0, \mathcal{A}_1, \dots)$  for set cover with deadlines. As we advance in this sequence, we describe adversaries that use a larger universe, and force a worse competitive ratio on the online algorithm. The growth rate of the universe ( $\ell$  and  $m$ ) together with the growth rate of the competitive ratio yields the desired lower bound.

<sup>2</sup> Note that the  $O(\log \ell)$ -competitive algorithm is not a direct application of the framework of [8], but requires a simple observation.

We construct each adversary  $\mathcal{A}_i$  recursively from the description of  $\mathcal{A}_{i-1}$ . The universe of  $\mathcal{A}_i$ , the elements  $E_i$  and sets  $S_i$ , consists of multiple copies of  $E_{i-1}$  and  $S_{i-1}$ , but these copies are interleaved in a way that creates increasing difficulty for the algorithm with each iteration. As for requests,  $\mathcal{A}_i$  makes recursive calls to  $\mathcal{A}_{i-1}$  on copies of  $E_{i-1}$  (which appear inside  $E_i$ ) which have the structure of  $\mathcal{A}_{i-1}$  locally – specifically, restricting the sets of  $S_i$  to this copy of  $E_{i-1}$  yields a copy of  $S_{i-1}$ .

While the recursive nature of the construction is similar to the adversary in [3], the underlying idea is different: in [3], the main idea of the adversary was to present the algorithm with two options for sets, an “expensive” option (which serves some additional long-term requests) and a “cheap” option (which only serves the most urgent requests). The idea was to exploit the fact that the algorithm doesn’t know whether investing in preparation for the future would pay off. However, our construction relies on a different principle – we present the algorithm with *many* different options for sets (the number of which grows logarithmically with the universe), the costs of which are identical (i.e. there is no cheap option). The online algorithm will either pick some small number of these options (in which case it would probably miss the correct one), or pick many of these options (which would be very expensive compared to the optimal solution).

### 1.3 Related Work

Classic online variants for network design problems have been studied extensively in the past. In such variants, the requests arrive over a sequence, and the items are bought rather than transmitted, such that a bought item can be used until the end of the input sequence. Some such problems were studied in [26, 23, 9, 29, 24, 1].

The relationship between this classic online and network design with deadlines is interesting: the clairvoyant deadlines model, considered in this paper, is more closely related to the offline problem (as shown in [8]), and can be much easier than the classic online variant. However, the *nonclairvoyant* deadlines model, where the deadline becomes known only at the end of the request’s time window, is as hard as the classic online variant: the reduction showing this for set cover appears in [3].

The online set cover with deadlines/delay problem was first presented by Carrasco *et al.* [19], who gave tight upper- and lower-bounds of logarithmic competitiveness in the number of requests in the input. In [3], bounds referring to the size of universe (sets and elements) were given, including an  $O(\log \ell \log m)$ -competitive algorithm. This upper bound of  $O(\log \ell \log m)$  applies to the nonclairvoyant setting. As nonclairvoyant set cover with deadlines/delay generalizes the classic online set cover, this algorithm is thus optimal for the class of polynomial, randomized algorithms, conditioned on  $\text{NP} \subsetneq \text{BPP}$  [28].

A specific network design problem which was previously considered is facility location with deadlines/delay. In this problem, requests arrive on the nodes of a graph, and a transmission consists of a facility at some node, to which some pending requests are connected. In [7],  $O(\log^2 n)$ -competitive randomized algorithms for facility location with deadlines/delay were presented, which only worked for the uniform case (i.e. identical facility opening costs), where  $n$  is the number of nodes in the graph. These results were then improved to deterministic  $O(\log n)$ -competitive algorithms for both deadlines and delay in [8].

Another network design problem with deadlines/delay is multilevel aggregation, which is in fact Steiner tree with deadlines/delay where the underlying metric space is a tree. This problem was first presented by Bienkowski *et al.* [10], as a generalization to the previously studied TCP acknowledgement [20, 27, 17] and joint replenishment [18, 15, 11] problems. The algorithm of Bienkowski *et al.* had competitiveness which was exponential in the depth of the tree  $D$ . This was first improved to  $O(D)$  for the deadline case by Buchbinder *et al.* [16] and then to  $O(D^2)$  for the general delay case by [7].

Other problems with deadlines/delay, outside of network design, have also seen significant interest. The  $k$ -server problem with delay was presented in [5], and studied in [14, 7, 25]. Interestingly, this problem is related to the network design problem of multilevel aggregation, as discussed in [7].

Min-cost perfect matching with delays, presented in [21] is another such problem. This problem is only tractable for specific delay functions (e.g. linear and concave), and is studied in [2, 22, 21, 4, 12, 13, 6].

## 2 Preliminaries

Since our results for node-weighted Steiner forest and directed Steiner tree stem from our result for set cover through a folklore reduction, we only provide a formal description for set cover with deadlines.

### Set cover with deadlines

In the set cover with deadlines problem, one is given a universe which consists of elements  $E$  and a collection of sets  $S$ , such that  $s \subseteq E$  for every  $s \in S$ . In addition, each set  $s \in S$  has a cost  $c(s)$ . Additionally, the input contains a set of requests  $Q$ . Each request  $q \in Q$  has an associated element  $e_q \in E$ , a release time  $r_q$  and a deadline  $d_q$  (such that  $r_q < d_q$ ).

A solution for the input is some collection of (instantaneous) transmissions  $\{T_1, \dots, T_k\}$  at various points in time, such that each transmission is of some set  $s_T$ . The cost of this solution is  $\sum_{i=1}^k c(s_{T_i})$ , i.e. the sum of costs of transmitted sets. Note that a set can be transmitted more than once in a solution; in this case, the set's cost is incurred more than once. A solution is *feasible* if for each request  $q \in Q$ , there exists a transmission  $T$  at some time between  $r_q$  and  $d_q$  such that  $e_q \in s_T$ . In words, a set containing the element requested by  $q$  must be transmitted within  $q$ 's time window.

Rigorously, the events at time  $t$  occur in the following order: first, any transmitted sets at  $t$  serve pending requests; then, any request  $q$  with  $r_q = t$  is released. Thus, in order to serve a request  $q$ , a transmission must take place in the half-open time window  $(r_q, d_q]$ . We remark that the results of this paper hold for any choice of order, and that our specific order is chosen for ease of presentation. (Indeed, our lower bound can be stated with distinct release/deadline times, which would bypass this issue.)

### The online setting

An online algorithm receives the universe (i.e.  $E$ ,  $S$  and  $\{c(s)\}_{s \in S}$ ) up front, while the requests  $Q$  are revealed to the algorithm as time advances. Specifically, each request  $q \in Q$  appears to the online solution only upon its release time  $r_q$ . We consider the *clairvoyant* model, in which all parameters of the request  $q$  are revealed at  $r_q$  (in particular its deadline  $d_q$ ).

At any point in time, the algorithm is allowed to make a transmission  $T$  of any set  $s$  (thus incurring a cost of  $c(s)$ ).

## 3 Lower Bound for Set Cover

In this section, we describe and analyze an oblivious adversary for set cover with deadlines, which we then use to prove Theorems 1.1–1.3.

We define the sequence  $(\ell_i)_{i=0}^\infty$  recursively by  $\ell_0 := 1$  and by  $\ell_i := (6i + 1)\ell_{i-1}$  for  $i > 0$ . Similarly, we define the sequence  $(m_i)_{i=0}^\infty$  recursively by  $m_0 := 1$  and by  $m_i := 4i \cdot m_{i-1}$  for  $i > 0$ . We prove the following lemma.

► **Lemma 3.1.** *For every index  $i$ , there exists an oblivious adversary  $\mathcal{A}_i$  which generates a set cover with deadlines instance on a universe with  $\ell_i$  elements and  $m_i$  sets, such that any deterministic algorithm is at least  $\Omega(i)$ -competitive against  $\mathcal{A}_i$ .*

Lemma 3.1, together with Yao's principle and some folklore reductions, is later used to prove Theorems 1.1–1.3.

### 3.1 The Set Cover Adversary

To prove Lemma 3.1, we now introduce the oblivious adversary  $\mathcal{A}_i$ . This adversary  $\mathcal{A}_i$  provides a universe with elements  $E_i$  and set collection  $S_i$ . The adversary always provides unweighted instances, i.e. the cost of every set in  $S_i$  is always 1.

#### The Universe of $\mathcal{A}_i$

In the base case of  $i = 0$ , we have a universe of a single element, ( $E_0 = \{e\}$ ) and a single set ( $S_0 = \{s\}$ ).

For  $i > 0$ , we construct the universe of  $\mathcal{A}_i$  recursively from the universe of  $\mathcal{A}_{i-1}$ . Define  $b_i := 2i$ . The set  $E_i$  contains copies of elements from  $E_{i-1}$ . Specifically, each element  $e \in E_{i-1}$  has the following copies:

1. The copy  $e^s$  (define  $E_{i-1}^s := \{e^s | e \in E_{i-1}\}$ ). This copy is called the *special* copy.
2. The  $b_i$  copies  $e^{a,1}, \dots, e^{a,b_i}$  (for every  $j$ , define  $E_{i-1}^{a,j} := \{e^{a,j} | e \in E_{i-1}\}$ ). These are called the *ancillary* copies of elements.
3. The  $b_i$  copies  $e^{p,1}, \dots, e^{p,b_i}$  (for every  $j$ , define  $E_{i-1}^{p,j} := \{e^{p,j} | e \in E_{i-1}\}$ ). These are called the *positive* copies of elements.
4. The  $b_i$  copies  $e^{n,1}, \dots, e^{n,b_i}$  (for every  $j$ , define  $E_{i-1}^{n,j} := \{e^{n,j} | e \in E_{i-1}\}$ ). These are called the *negative* copies of elements.

The elements of the instance of  $\mathcal{A}_i$  are defined as

$$E_i := E_{i-1}^s \cup \left( \bigcup_{j=1}^{b_i} E_{i-1}^{a,j} \right) \cup \left( \bigcup_{j=1}^{b_i} E_{i-1}^{p,j} \right) \cup \left( \bigcup_{j=1}^{b_i} E_{i-1}^{n,j} \right)$$

We can now observe that  $|E_i| = (3b_i + 1)|E_{i-1}| = (6i + 1)|E_i|$ . Since  $|E_0| = 1$ , for every  $i$  we have  $|E_i| = \ell_i$ , as required by Lemma 3.1.

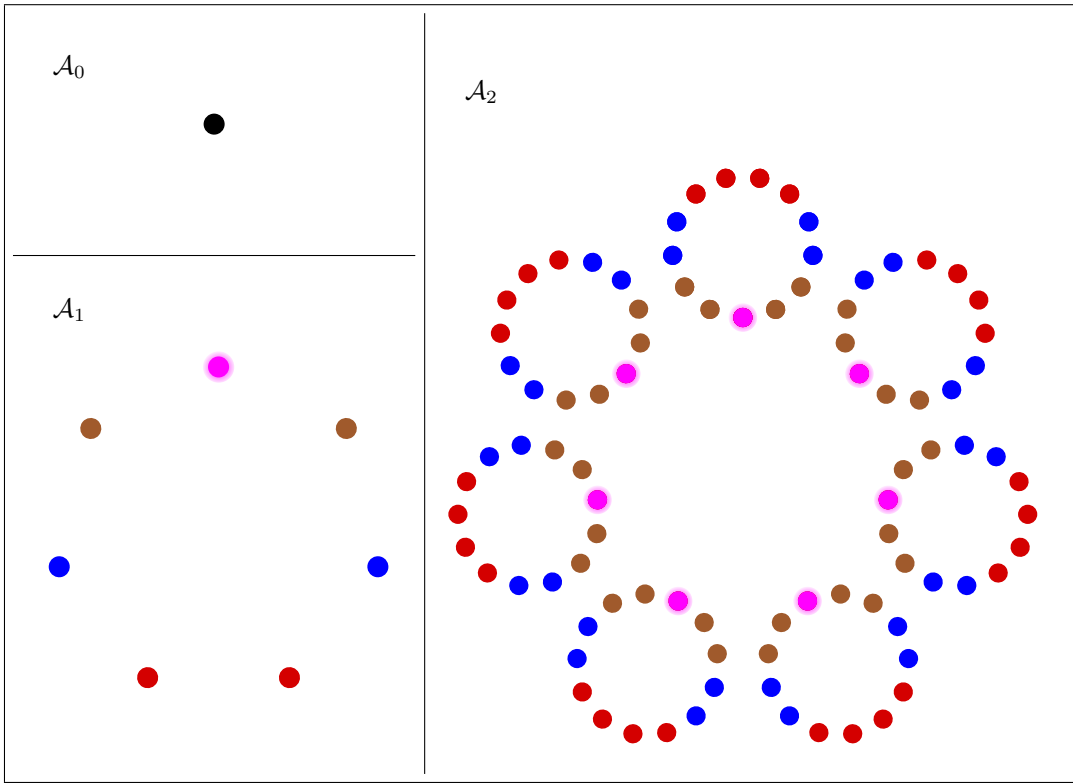
Figure 1 shows the elements of  $\mathcal{A}_0$  (upper left),  $\mathcal{A}_1$  (lower left) and  $\mathcal{A}_2$  (right). Each element in  $\mathcal{A}_1$  and  $\mathcal{A}_2$  is a copy of an element from the universe of the previous adversary; each copy is special (purple), ancillary (brown), positive (blue) or negative (red).

As for the sets, each set  $s \in S_{i-1}$  has the following copies in  $S_i$ :

1. The  $b_i$  copies  $s^{p,1}, \dots, s^{p,b_i}$ , such that

$$s^{p,j} = \bigcup_{e \in s} \{e^s, e^{a,j}, e^{p,j}\}$$

(for each  $j$ , we define  $S_{i-1}^{p,j} := \{s^{p,j} | s \in S_{i-1}\}$ ). These copies are called the *positive* copies of sets.



■ **Figure 1** The Elements of  $\mathcal{A}_0$ ,  $\mathcal{A}_1$  and  $\mathcal{A}_2$ .

2. The  $b_i$  copies  $s^{n,1}, \dots, s^{n,b_i}$ , such that

$$s^{n,j} = \bigcup_{e \in s} \left( \{e^{n,j}\} \cup \left( \bigcup_{j' \neq j} \{e^{p,j'}\} \right) \right)$$

(for each  $j$ , we define  $S_{i-1}^{n,j} := \{s^{n,j} \mid s \in S_{i-1}\}$ ). These copies are called the *negative* copies of sets.

Note that holds that  $|S_i| = 2b_i \cdot |S_{i-1}| = 4i \cdot |S_{i-1}|$ . Combined with the fact that  $|S_0| = 1$ , for every  $i$  we have  $|S_i| = m_i$ , as required by Lemma 3.1.

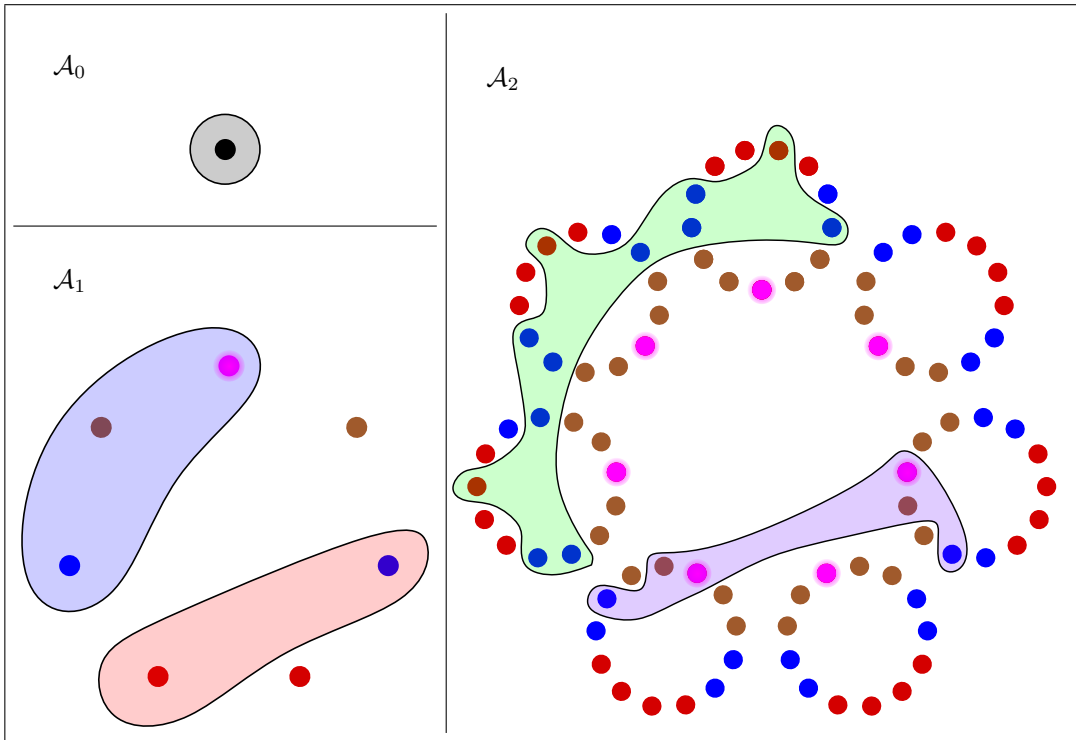
Figure 2 shows some (not all) of the sets in  $\mathcal{A}_0$ ,  $\mathcal{A}_1$  and  $\mathcal{A}_2$ . The gray set in  $S_0$  is the only set of that universe. The blue and red sets in  $S_1$  are positive and negative copies of the gray set, respectively. The purple set in  $S_2$  is a positive copy of the red set. The green set in  $S_2$  is a negative copy of the blue set.

### Request sequence of $\mathcal{A}_i$

We now describe the sequence of requests generated by  $\mathcal{A}_i$ .

#### Recursive calls to $\mathcal{A}_{i-1}$

For  $i > 0$ , the adversary  $\mathcal{A}_i$  relies on recursive calls to  $\mathcal{A}_{i-1}$ . Note that the elements  $E_i$  comprise copies of  $E_{i-1}$  ( $E_{i-1}^s, E_{i-1}^{p,j}$ , et cetera). Thus, for any copy  $E'_{i-1}$  of  $E_{i-1}$ , it is well-defined to call  $\mathcal{A}_{i-1}$  on  $E'_{i-1}$ : we release a request on a copy  $e' \in E'_{i-1}$  of  $e \in E_{i-1}$  whenever a request is released on  $e$  by  $\mathcal{A}_{i-1}$ .



■ **Figure 2** Some Example Sets in  $\mathcal{A}_0$ ,  $\mathcal{A}_1$  and  $\mathcal{A}_2$ .

For abbreviation, we use the superscript denoting the copy of the elements on the adversary. For example, calling  $\mathcal{A}_{i-1}^{\mathbf{P},j}$  means calling  $\mathcal{A}_{i-1}$  on the copy  $E_{i-1}^{\mathbf{P},j}$ .

**Adversary time bounds**

As we would like to pack multiple recursive  $\mathcal{A}_{i-1}$  adversaries into one timeline, we would like to bound each  $\mathcal{A}_{i-1}$  by some time interval, so that we can charge the algorithm for solving each  $\mathcal{A}_{i-1}$  disjointly.

Define  $T_i$  for  $i \geq 0$  recursively by  $T_0 = 1$  and  $T_i = 2b_i \cdot T_{i-1}$ . Informally,  $T_i$  defines a time interval which contains  $\mathcal{A}_i$ ; formally, each request  $q$  which  $\mathcal{A}_i$  can possibly release has  $[r_q, d_q] \subseteq [0, T_i]$  (this can be verified for the construction of  $\mathcal{A}_i$  we describe next).

**Requests of  $\mathcal{A}_i$**

For the case that  $i = 0$ , the adversary releases at time 0 a single request  $q$  on the single element in  $E_0$ , such that  $d_q = 1$ .

For the case of  $i > 0$ , the adversary  $\mathcal{A}_i$  is given in Procedure 1; we now provide a verbal description of that procedure. The adversary  $\mathcal{A}_i$  uses  $b_i$  phases, each of which takes  $2T_{i-1}$  time, and makes two recursive calls to  $\mathcal{A}_{i-1}$ . In the beginning of each phase  $k$ , “background” requests are released on the elements in some positive copies of  $E_{i-1}$ , with deadlines at the end of the phase (i.e.  $2T_{i-1}$  time after release). These requests are released on copies of  $E_{i-1}$  whose index is in some set  $M$ , where  $M$  initially consists of all  $b_i$  indices of positive copies of  $E_{i-1}$ . In the first half of each phase, i.e. the first  $T_{i-1}$  time units,  $\mathcal{A}_i$  calls  $\mathcal{A}_{i-1}^{\mathbf{S}}$  (and waits  $T_{i-1}$  time for its completion). Then,  $\mathcal{A}_i$  chooses an index  $j_k$  to remove from  $M$ , and calls  $\mathcal{A}_{i-1}^{\mathbf{N},j_k}$  which occurs in the second half of the phase.

### Intuitive Explanation

The intuition behind this construction of  $\mathcal{A}_i$  is the following. When the online algorithm handles the first recursive call in a phase, it has some choices to make. First, it has the standard choices to make when addressing  $\mathcal{A}_{i-1}$  – that is, which sets from  $S_{i-1}$  to transmit. In addition, for each set in  $S_{i-1}$  that it wishes to transmit, it must choose the appropriate copy from  $S_i$ . Note that the only copies of a set  $s \in S_{i-1}$  which can be used to solve  $\mathcal{A}_{i-1}^s$  are the positive copies, i.e.  $s^{\mathbf{P}:j}$  for some index  $j$ ; moreover, since this first recursive call is on the special copies, *all* choices of  $j$  are possible. (The purpose of the special copies is exactly this: to be at the intersection of positive copies of sets, and force the algorithm to make a choice.)

However, transmitting these positive copies of sets serves an additional purpose – serving the background requests on  $M$ . For this reason, the online algorithm should choose to transmit  $s^{\mathbf{P}:j}$  for some  $j \in M$ . But note that for earlier phases, the set  $M$  is rather large – its size is  $\Omega(i)$ . The size of  $M$  prohibits the algorithm from transmitting every positive copy in  $M$  of every set – otherwise, the algorithm would incur a great cost. Instead, the algorithm must focus on a small subset of  $M$ , which leaves many background requests unsatisfied.

The optimal solution, on the other hand, is provided a “shortcut”: it uses the second recursive call of the phase, which is to  $\mathcal{A}_{i-1}^{\mathbf{n}:j_k}$ , to serve all background requests except for those on  $E_{i-1}^{\mathbf{P}:j_k}$ . The requests on  $E_{i-1}^{\mathbf{P}:j_k}$  are served by the optimal solution in the first recursive call to  $\mathcal{A}_{i-1}^s$ , where the solution only transmits copies from  $S_{i-1}^{\mathbf{P}:j_k}$ . Note that such efficient handling of the background requests cannot be achieved by the online algorithm, since it does not have knowledge of  $j_k$  during the first half of phase  $k$ .

Near the end of a phase, when the online algorithm has (with high probability) a large amount of pending background requests to serve, the algorithm must incur the cost of an (offline) set cover for those pending requests; the only purpose of ancillary element copies is to keep the cost of this offline cover large, which ensures high costs for the algorithm. (Specifically, the ancillary element copies ensure that each positive set copy  $s$  has an element unique to  $s$ , which forces  $s$  to be part of the offline set cover; this is used in Proposition 3.4.)

#### ■ Procedure 1 Adversary $\mathcal{A}_i$ .

---

```

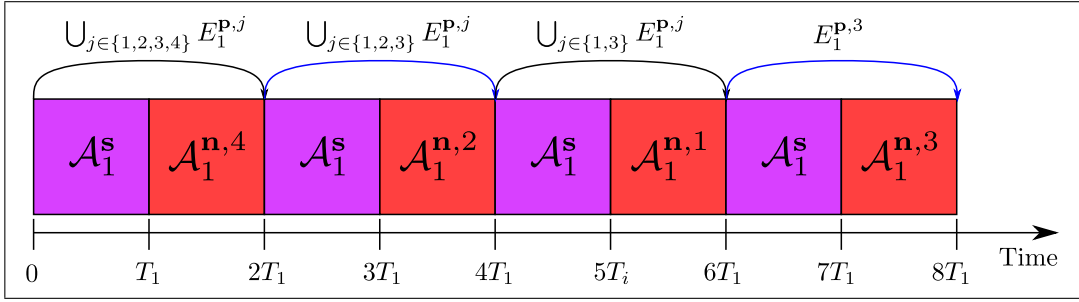
1 Function CREATEINSTANCE
2   Start with  $M \leftarrow \{1, 2, \dots, b_i\}$ .
3   for  $k$  from 1 until  $b_i$  do
4     // Start phase  $k$  at time  $2(k-1)T_{i-1}$ , by releasing background requests and
      calling  $\mathcal{A}_{i-1}^s$ .
5     For every  $j \in M$ , release a request on every element in  $E_{i-1}^{\mathbf{P}:j}$ , with a deadline
      that's  $2T_{i-1}$  time in the future.
6     Call  $\mathcal{A}_{i-1}^s$  and wait  $T_{i-1}$  time until its completion.
7     //  $T_{i-1}$  time after the start of phase  $k$ , call the second  $\mathcal{A}_{i-1}$  on a random
      negative copy of  $E_{i-1}$ .
8     Choose  $j_k \in M$  uniformly at random from  $M$ .
9     Call  $\mathcal{A}_{i-1}^{\mathbf{n}:j_k}$  and wait  $T_{i-1}$  time until its completion.
10    Set  $M \leftarrow M \setminus \{j_k\}$ .

```

---

The timeline of a possible instance created by  $\mathcal{A}_2$  is shown in Figure 3. This figure shows  $b_2 = 4$  phases, each of which contains two calls to  $\mathcal{A}_{i-1}$ . In this figure,  $\mathcal{A}_2$  randomly chose the permutation (4, 2, 1, 3) of the elements of  $[b_2]$ . The arrows above each phase show the release





■ **Figure 3** Request Timeline of  $\mathcal{A}_2$ .

times and deadlines of background requests – these requests are released at the beginning of each phase, and have deadlines at the end of the phase. Above the arrows are the sets of elements on which background requests are released.

### 3.2 Analysis

We now analyze the adversary described above, thus proving Lemma 3.1.

#### The Optimal Solution

The following lemma describes the optimal solution for instances generated by the adversary  $\mathcal{A}_i$ . Not only do they have a low cost relative to the algorithm, but they also buy every set exactly once, which is useful due to recursion.

► **Lemma 3.2.** *For every  $i$ , and for every instance generated by  $\mathcal{A}_i$ , there exists an offline solution that buys every set in  $S_i$  exactly once, and thus has cost  $m_i$ .*

**Proof.** The proof is by induction on  $i$ . For  $i = 0$ , the adversary  $\mathcal{A}_i$  releases a single request on a single element at time 0, with deadline at time 1. Thus, transmitting the single set in  $m$  at any time during the interval  $(0, 1]$  is a feasible solution.

Assume that the lemma holds for  $\mathcal{A}_{i-1}$  – i.e. there exists an offline solution SOL which buys every set in  $S_{i-1}$  exactly once.

Now, consider an instance generated by  $\mathcal{A}_i$ . This instance was generated in  $b_i$  phases, each associated with some index in  $[b_i]$  (the index chosen randomly by  $\mathcal{A}_i$  in this phase). These indices form a permutation on the elements of  $[b_i]$ , and we write them as a sequence  $(j_k)_{k=1}^{b_i}$ , such that  $j_k$  is the index chosen by  $\mathcal{A}_i$  in phase  $k$ .

We now describe an offline solution for the instance generated by  $\mathcal{A}_i$ . Upon phase  $k$ , consider the call to  $\mathcal{A}_{i-1}^s$  in the beginning of the phase. In the original universe of  $\mathcal{A}_{i-1}$ , i.e.  $E_{i-1}$  and  $S_{i-1}$ , the induction hypothesis implies that there exists an offline solution SOL for this instance which transmits every set of  $S_{i-1}$  exactly once. Thus, a solution for  $\mathcal{A}_{i-1}^s$  would be to transmit any positive copy of the set  $s$  whenever SOL transmits  $s$ . Our offline solution to  $\mathcal{A}_i$  will transmit only the positive copies of index  $j_k$ , i.e.  $s^{P,j_k}$  instead of  $s$ .

As for the call to  $\mathcal{A}_{i-1}^{n,j_k}$ , which is the second recursive call in phase  $k$ , we use a similar argument: we use the offline solution SOL for  $\mathcal{A}_{i-1}$  in the original universe of  $\mathcal{A}_{i-1}$ , and transmit  $s^{n,j_k}$  whenever SOL transmits  $j$ .

Observe that this offline solution for the instance generated by  $\mathcal{A}_i$  is feasible:

- The requests inside recursive calls to  $\mathcal{A}_i$  are satisfied from the induction hypothesis.
- The requests outside recursive calls, i.e. the background requests released in the beginning of the phase, are also satisfied: in each phase  $k$ , we transmit all sets in  $S_{i-1}^{P,j_k}$  (which satisfy all requests on  $E_{i-1}^{P,j_k}$ ) and  $S_{i-1}^{n,j_k}$  (which satisfy all requests on  $E_{i-1}^{P,j}$  for every  $j \neq j_k$ ).

Now note that all sets in  $S_i$  are bought exactly once: this is since in phase  $k$  the algorithm buys all sets from  $S_{i-1}^{\mathbf{p},j_k}$  and  $S_{i-1}^{\mathbf{n},j_k}$  exactly once (and no other sets). Since  $(j_k)_{k=1}^{b_i}$  are a permutation of  $[b_i]$ , this yields the desired claim.  $\blacktriangleleft$

## The Cost of the Algorithm

We now bound the expected cost of the algorithm. For every  $i$ , define  $c_i := \frac{i}{8}$ . The main result here is the following lemma.

► **Lemma 3.3.** *For every  $i$ , and for every deterministic online algorithm ALG against the adversary  $\mathcal{A}_i$ , it holds that the expected cost of the algorithm during the interval  $(0, T_i]$  is at least  $c_i \cdot m_i$  (where the expectation is over the random choices of  $\mathcal{A}_i$ ).*

We prove Lemma 3.3 by induction. For the base case of  $i = 0$ , note that this trivially holds for  $\mathcal{A}_i$ : the algorithm must transmit a set during  $(0, 1]$  at cost 1, which is more than  $\frac{1}{8}$ . For every  $i > 0$ , assuming that the lemma holds for  $i - 1$ , we prove the lemma for  $\mathcal{A}_i$ . We also henceforth fix ALG to be the online deterministic algorithm which runs against the adversary  $\mathcal{A}_i$ . Slightly abusing notation, we also use ALG to refer to the cost of the online algorithm during the interval  $(0, T_i]$ ; indeed, costs outside this interval are irrelevant to Lemma 3.3.

First, we show an important property of the universe of  $\mathcal{A}_i$ . For every universe with elements  $E$  and sets  $S$ , denote by  $\text{sc}(E, S)$  the cost of the optimal (classic, offline) set cover solution for this universe. For all (reasonable) universes, buying all sets is a feasible set cover solution; Proposition 3.4 shows that for the universe of  $\mathcal{A}_i$ , buying all sets is in fact the *only* solution.

► **Proposition 3.4.** *For every  $i$ , it holds that  $\text{sc}(E_i, S_i) = m_i$ .*

**Proof of Proposition 3.4.** Clearly, buying each set in  $S_i$  is a feasible solution of cost  $m_i$ . It now show that each set must be bought, which proves the proposition. To this end, we prove that for each set  $s \in S_i$  there exists an element  $e \in E_i$  such that  $e$  is in  $s$ , but in no other set in  $S_i$ . If this indeed holds for each  $s \in S_i$ , each set must be bought, completing the proof.

We prove this claim by induction on  $i$ . For the base case of  $i = 0$  this trivially holds. Now, for  $i > 0$ , assume that for each set  $\bar{s} \in S_{i-1}$  there exists an element in  $E_{i-1}$  which is in  $\bar{s}$  and in no other set in  $S_{i-1}$ .

Now, consider any set  $s \in S_i$ . This set is a copy of some set  $\bar{s} \in S_{i-1}$ , for which the induction hypothesis provides an element  $\bar{e} \in E_{i-1}$  which is in  $\bar{s}$  and not in any other set in  $S_{i-1}$ .

1. If  $s$  is a positive copy of  $\bar{s}$ , i.e.  $s = \bar{s}^{\mathbf{p},j}$  for some  $j$ , then observe the element  $e := \bar{e}^{\mathbf{a},j} \in E_i$ . It holds that  $e$  is in  $s$  but in no other set in  $S_i$ .
2. If  $s$  is a negative copy of  $\bar{s}$ , i.e.  $s = \bar{s}^{\mathbf{n},j}$  for some  $j$ , then observe the element  $e := \bar{e}^{\mathbf{n},j} \in E_i$ . It holds that  $e$  is in  $s$  but in no other set in  $S_i$ .

This completes the proof of the claim, and thus the proposition.  $\blacktriangleleft$

Note again that Proposition 3.4 refers to the offline cost of covering *the universe* of  $\mathcal{A}_i$ ; this is not the same as the cost of the optimal solution against  $\mathcal{A}_i$  (for example, ancillary copies in the universe are never requested by  $\mathcal{A}_i$ , and are only useful for future recursion).

We would now like to give a lower bound for the expected cost of the algorithm at each phase. For every phase  $k$ , let  $\text{ALG}_k$  be the cost of ALG during the phase  $k$ , i.e. during the time interval  $((2k - 2)T_{i-1}, 2kT_{i-1}]$ .

► **Lemma 3.5.** *For every phase  $k$ , it holds that:*

1. *If  $k \leq \frac{b_i}{2}$ , then  $\mathbb{E}[\text{ALG}_k] \geq (c_{i-1} + \frac{1}{4}) \cdot 2m_{i-1}$*
2. *If  $k > \frac{b_i}{2}$ , then  $\mathbb{E}[\text{ALG}_k] \geq c_{i-1} \cdot 2m_{i-1}$*

**Proof.** Fix any phase  $k$ , which starts at time  $\tau := 2(k-1) \cdot T_{i-1}$ . For this proof, fix the set of random choices made by  $\mathcal{A}_i$  until the start of phase  $k$ ; henceforth in the proof the expectations are thus only on random choices from phase  $k$  onwards. Since we bound the expected cost of the algorithm for every possible set of choices made up to phase  $k$  by  $\mathcal{A}_i$ , this lower bound also applies in expectation over those choices.

Let  $M$  denote the value of the variable of the same name in  $\mathcal{A}_i$  at the beginning of phase  $k$  (recall that  $M \subseteq [b_i]$  is a set of indices). Since we have fixed the random choices of  $\mathcal{A}_i$  before phase  $k$ , the set  $M$  is some fixed set.

We divide the transmissions made by the algorithm during the phase into three disjoint parts:

- Part  $P_1$ : transmissions of positive sets during  $(\tau, \tau + T_{i-1}]$  (the first half of the phase).
- Part  $P_2$ : transmissions of negative sets from  $S_{i-1}^{\mathbf{n}, j^k}$  during  $(\tau + T_{i-1}, \tau + 2T_{i-1}]$  (the second half of the phase).
- Part  $P_3$ : transmissions not in  $P_1, P_2$ .

We denote the number of transmissions in  $P_\ell$  (equivalently: the total cost of such transmissions) by  $C_\ell$ .

**Part  $P_1$ .** Observe that the positive sets are the only sets that can be used for  $\mathcal{A}_{i-1}^{\mathbf{s}}$  in the first part of the phase. Also note that intersecting all positive sets with  $E_{i-1}^{\mathbf{s}}$  yields a collection of sets which is identical to  $S_{i-1}$ ; that is, for every set  $s \in \bigcup_j S_{i-1}^{\mathbf{p}, j}$ , there exists a set  $s' \in S_{i-1}$  such that

$$s \cap E_{i-1}^{\mathbf{s}} = \{e^{\mathbf{s}} | e \in s'\}$$

(specifically, the set  $s'$  is such that  $s$  is a positive copy of  $s'$ )

Thus, covering the elements of  $E_{i-1}^{\mathbf{s}}$  with these sets is as hard as covering the elements of  $E_{i-1}$  with  $S_{i-1}$ . Now, recall the induction hypothesis made for Lemma 3.3, which implied that the expected cost of this part of the algorithm is at least  $c_{i-1} \cdot m_{i-1}$ .

**Part  $P_2$ .** Note that the sets of  $S_{i-1}^{\mathbf{n}, j^k}$  are the only sets that contain elements from  $E_{i-1}^{\mathbf{n}, j^k}$ , and are thus the only sets that can be used to serve  $\mathcal{A}_{i-1}^{\mathbf{n}, j^k}$ . Also note that intersecting the sets of  $S_{i-1}^{\mathbf{n}, j^k}$  with the elements  $E_{i-1}^{\mathbf{n}, j^k}$  yields a collection of sets which is identical to  $S_{i-1}$  (in a similar way to the argument for  $P_1$ ) We can thus again apply the induction hypothesis, and see that the expected cost of the algorithm in buying the sets of  $S_{i-1}^{\mathbf{n}, j^k}$  must be at least  $c_{i-1} \cdot m_{i-1}$ .

Combining Parts  $P_1$  and  $P_2$ , we have

$$\mathbb{E}[C_1] + \mathbb{E}[C_2] \geq c_{i-1} \cdot 2m_{i-1} \tag{1}$$

Equation (1) immediately implies the second claim of this lemma. It remains to prove the first claim.

Assume henceforth that  $k \leq \frac{b_i}{2}$ . If it holds that  $\mathbb{E}[C_1] \geq \frac{b_i}{4} \cdot m_{i-1}$ , then since  $\frac{b_i}{4} \cdot m_{i-1} = \frac{i}{2} \cdot m_{i-1} \geq (c_{i-1} + \frac{1}{2})m_{i-1}$ , we have  $\mathbb{E}[C_1] + \mathbb{E}[C_2] \geq (c_{i-1} + \frac{1}{4}) \cdot 2m_{i-1}$ , which completes the proof of the second claim.

We therefore assume henceforth that  $\mathbb{E}[C_1] < \frac{b_i}{4} \cdot m_{i-1}$ .

**Part  $P_3$ .** Consider the background requests of phase  $k$  which were released on  $E_{i-1}^{\mathbf{P},j}$ , for any index  $j \in M$ . As the restriction of  $S_i$  to  $E_{i-1}^{\mathbf{P},j}$  is identical to  $S_{i-1}$ , Proposition 3.4 implies  $\text{sc}(E_{i-1}^{\mathbf{P},j}, S_i) = m_{i-1}$ . Thus, the algorithm has to transmit at least  $m_{i-1}$  sets that contain elements from  $E_{i-1}^{\mathbf{P},j}$  during the phase.

Consider the sets transmitted in Part  $P_1$ . These sets are all positive sets. Each such positive set  $s^{\mathbf{P},j}$ , for some  $j$ , does not contain any positive elements outside  $E_{i-1}^{\mathbf{P},j}$ . Denote by  $C_1^j$  the number of sets from  $S_{i-1}^{\mathbf{P},j}$  transmitted in  $P_1$ , such that  $C_1 = \sum_j C_1^j$ . Then we know that for each  $j \in M$  we have that in  $P_2$  and  $P_3$  together, there must be at least  $m_{i-1} - C_1^j$  transmissions of sets containing elements from  $E_{i-1}^{\mathbf{P},j}$ .

Now, observe that choosing  $j = j_k$ , Part  $C_2$  transmits only sets from  $S_{i-1}^{\mathbf{n},j_k}$ , which do not contain elements from  $E_{i-1}^{\mathbf{P},j_k}$ . This thus yields a lower bound of  $C_3 \geq m_{i-1} - C_1^{j_k}$ .

Overall, we have that:

$$\begin{aligned} \mathbb{E}[C_3] &\geq \sum_{j \in M} \Pr(j_k = j) \cdot \mathbb{E}\left[m_{i-1} - C_1^j \mid j_k = j\right] = \frac{1}{|M|} \sum_{j \in M} \left(m_{i-1} - \mathbb{E}\left[C_1^j \mid j_k = j\right]\right) \\ &= \frac{1}{|M|} \sum_{j \in M} \left(m_{i-1} - \mathbb{E}\left[C_1^j\right]\right) = m_{i-1} - \frac{1}{|M|} \mathbb{E}[C_1] \geq m_{i-1} - \frac{2}{b_i} \mathbb{E}[C_1] \geq \frac{m_{i-1}}{2} \end{aligned}$$

The second equality is due to the fact that  $C_1^j$  is independent of the choice of  $j_k$  (indeed, the choice of  $j_k$  only affects the input from time  $\tau + T_{i-1}$ ). The second inequality is from the fact that  $k \leq \frac{b_i}{2}$ , which implies that  $|M| \geq \frac{b_i}{2}$ . The third inequality is from  $\mathbb{E}[C_1] < \frac{b_i}{4} \cdot m_{i-1}$ . Combining this with Equation (1), we obtain

$$\mathbb{E}[C_1] + \mathbb{E}[C_2] + \mathbb{E}[C_3] \geq 2c_{i-1} \cdot m_{i-1} + \frac{m_{i-1}}{2} = \left(c_{i-1} + \frac{1}{4}\right) \cdot 2m_{i-1} \quad \blacktriangleleft$$

**Proof of Lemma 3.3.** It holds that

$$\begin{aligned} \mathbb{E}[\text{ALG}] &= \sum_{k=1}^{b_i} \mathbb{E}[\text{ALG}_k] \geq \frac{b_i}{2} \left(c_{i-1} + \frac{1}{4}\right) \cdot 2m_{i-1} + \frac{b_i}{2} c_{i-1} \cdot 2m_{i-1} \\ &= \left(c_{i-1} + \frac{1}{8}\right) 2b_i \cdot m_{i-1} = c_i m_i \end{aligned}$$

where the first inequality is due to applying Lemma 3.5 to the phases (using the stronger claim for the earlier phases and the weaker claim for the later phases), and the final equality uses the fact that  $m_i = 2b_i \cdot m_{i-1}$ .  $\blacktriangleleft$

**Proof of Lemma 3.1.** The lemma results immediately from Lemmas 3.2 and 3.3.  $\blacktriangleleft$

### 3.3 Proofs of Theorems

We now use the construction above to prove the main theorems of this paper.

**Proof of Theorem 1.1.** Lemma 3.1 implies that any deterministic algorithm is  $\Omega(i)$ -competitive against  $\mathcal{A}_i$ , which uses a universe of  $\ell_i$  elements and  $m_i$  sets. Yao's principle now implies that for every randomized online algorithm, there exists an instance with  $\ell_i$  and  $m_i$  on which its competitive ratio is  $\Omega(i)$ .

For the set-based bound, note that  $m_i = 4^i \cdot i! \leq (4i)^i$ , which implies  $i \geq \frac{\log m_i}{4 \log i}$ . Now observe that  $m_i \geq 2^i$  and thus  $i \leq \log m_i$ . Together with the previous observation, we have that  $i = \Omega\left(\frac{\log m_i}{\log \log m_i}\right)$ , which yields the desired  $\Omega\left(\frac{\log m}{\log \log m}\right)$ -competitiveness lower bound.

For the element-based bound, note that  $2^i \leq \ell_i \leq (7i)^i$ . A similar argument thus yields that  $i = \Omega\left(\frac{\log \ell_i}{\log \log \ell_i}\right)$ , which gives us the  $\Omega\left(\frac{\log \ell}{\log \log \ell}\right)$ -competitiveness lower bound.  $\blacktriangleleft$

**Proofs of Theorems 1.2 and 1.3.** There exist folklore reductions from set cover to node-weighted Steiner tree and directed Steiner tree, which reduce a set cover instance with  $\ell$  elements and  $m$  sets to graphs with  $\ell + m + 1$  nodes. These reductions carry over to the deadline/delay setting (for a detailed description of these reductions, see e.g. [8]).

Now,  $\mathcal{A}_i$  as described for set cover yields a graph with  $\ell_i + m_i + 1$  nodes, which is at most  $3 \cdot (7i)^i$  nodes (and more than  $2^i$  nodes). Lemma 3.1, together with argument identical to the proof of Theorem 1.1, yield an  $\Omega\left(\frac{\log n}{\log \log n}\right)$  on the competitiveness of any randomized algorithm. ◀

## 4 Discussion and Open Problems

In this paper, we presented nearly-logarithmic lower bounds on competitiveness for some network design problems with deadlines (which therefore also apply to the delay cases). In [8], a framework is shown which solves every network design with deadlines problem using an approximation algorithm for the corresponding offline problem, losing a logarithmic factor in competitiveness; our results thus show that this logarithmic factor is nearly optimal.

However, the problems we consider in this paper might be tougher than other network design problems with deadlines. While our paper shows that logarithmic loss in approximation ratio is necessary for the general case, there exist many network design problems for which no superconstant lower bound on competitiveness exists. Examples of such problems with deadlines are (edge-weighted) Steiner tree and Steiner forest, facility location, and multicut. Resolving the competitive ratio for these problems remains an interesting open problem.

---

### References


- 1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006. doi:10.1145/1198513.1198522.
- 2 Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 1:1–1:20, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.
- 3 Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay – clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.
- 4 Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. In *Approximation and Online Algorithms – 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 21–35, 2018. doi:10.1007/978-3-030-04693-4\_2.
- 5 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 551–563, 2017. doi:10.1145/3055399.3055475.
- 6 Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. *CoRR*, abs/2011.02017, 2020. arXiv:2011.02017.
- 7 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

- 8 Yossi Azar and Noam Touitou. Beyond tree embeddings – a deterministic framework for network design with deadlines or delay. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.
- 9 Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems (extended abstract). In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing, STOC '97*, pages 344–353, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258618.
- 10 Marcin Bienkowski, Martin Böhm, Jaroslav Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.
- 11 Marcin Bienkowski, Jaroslav Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jiri Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.
- 12 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Approximation and Online Algorithms – 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 51–68, 2018. doi:10.1007/978-3-030-04693-4\_4.
- 13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Approximation and Online Algorithms – 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, pages 132–146, 2017. doi:10.1007/978-3-319-89441-6\_11.
- 14 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *Structural Information and Communication Complexity – 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, pages 237–248, 2018. doi:10.1007/978-3-030-01325-7\_22.
- 15 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64(4):584–605, 2012. doi:10.1007/s00453-011-9567-5.
- 16 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon.  $O(\text{depth})$ -competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.
- 17 Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms – ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 253–264, 2007. doi:10.1007/978-3-540-75520-3\_24.
- 18 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 952–961, 2008. URL: <http://dl.acm.org/citation.cfm?id=1347082.1347186>.
- 19 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *LATIN 2018: Theoretical Informatics – 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, pages 245–259, 2018. doi:10.1007/978-3-319-77404-6\_19.
- 20 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 389–398, 1998. doi:10.1145/276698.276792.

- 21 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344, 2016. doi:10.1145/2897518.2897557.
- 22 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. In *Algorithms and Complexity – 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 209–221, 2017. doi:10.1007/978-3-319-57586-5\_18.
- 23 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008. doi:10.1007/s00453-007-9049-y.
- 24 Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. *SIAM J. Comput.*, 41(6):1649–1672, 2012. doi:10.1137/09076725X.
- 25 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. doi:10.1145/3357713.3384277.
- 26 Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.
- 27 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgment and other stories about  $e/(e-1)$ . *Algorithmica*, 36(3):209–224, 2003.
- 28 Simon Korman. On the use of randomization in the online set cover problem. Master’s thesis, Weizmann Institute of Science, 2005.
- 29 J. Naor, D. Panigrahi, and M. Singh. Online node-weighted steiner tree and related problems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 210–219, 2011.



# Cryptographic Hardness Under Projections for Time-Bounded Kolmogorov Complexity

Eric Allender   

Rutgers University, Piscataway, NJ, USA

John Gouwar 

Northeastern University, Boston, USA

Shuichi Hirahara  

National Institute of Informatics, Japan

Caleb Robelle 

MIT, Boston, USA

---

## Abstract

A version of time-bounded Kolmogorov complexity, denoted  $KT$ , has received attention in the past several years, due to its close connection to circuit complexity and to the Minimum Circuit Size Problem  $MCSP$ . Essentially all results about the complexity of  $MCSP$  hold also for  $MKTP$  (the problem of computing the  $KT$  complexity of a string). Both  $MKTP$  and  $MCSP$  are hard for  $SZK$  (Statistical Zero Knowledge) under  $BPP$ -Turing reductions; neither is known to be  $NP$ -complete.

Recently, some hardness results for  $MKTP$  were proved that are not (yet) known to hold for  $MCSP$ . In particular,  $MKTP$  is hard for  $DET$  (a subclass of  $P$ ) under nonuniform  $\leq_m^{NC^0}$  reductions. In this paper, we improve this, to show that  $\overline{MKTP}$  is hard for the (apparently larger) class  $NISZK_L$  under not only  $\leq_m^{NC^0}$  reductions but even under projections. Also  $\overline{MKTP}$  is hard for  $NISZK$  under  $\leq_m^{P/poly}$  reductions. Here,  $NISZK$  is the class of problems with non-interactive zero-knowledge proofs, and  $NISZK_L$  is the non-interactive version of the class  $SZK_L$  that was studied by Dvir et al.

As an application, we provide several improved worst-case to average-case reductions to problems in  $NP$ , and we obtain a new lower bound on  $MKTP$  (which is currently not known to hold for  $MCSP$ ).

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Complexity classes; Theory of computation  $\rightarrow$  Problems, reductions and completeness; Theory of computation  $\rightarrow$  Circuit complexity

**Keywords and phrases** Kolmogorov Complexity, Interactive Proofs, Minimum Circuit Size Problem, Worst-case to Average-case Reductions

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.54

**Related Version** *Full Version:* <https://eccc.weizmann.ac.il/report/2021/010/>

**Funding** JG and CR were supported in part by the DIMACS 2020 REU program under NSF grants CCF-1852215 and CCF-1836666.

*Eric Allender:* Supported in part by NSF Grants CCF-1909216 and CCF-1909683.

**Acknowledgements** We thank Rahul Santhanam, Oded Goldreich, Salil Vadhan, Harsha Tirumala, Dieter van Melkebeek and Andrew Morgan for helpful discussions. We also thank the anonymous reviewers who provided helpful comments.

## 1 Introduction

The study of time-bounded Kolmogorov complexity is tightly connected to the study of circuit complexity. Indeed, the measure that we study most closely in this paper, denoted  $KT$ , was initially defined in order to capitalize on the framework of Kolmogorov complexity in investigations of the Minimum Circuit Size Problem ( $MCSP$ ) [4]. If  $f$  is a bit string of length  $2^k$  representing the truth-table of a  $k$ -ary Boolean function, then  $KT(f)$  is polynomially



© Eric Allender, John Gouwar, Shuichi Hirahara, and Caleb Robelle;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 54; pp. 54:1–54:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

related to the size of the smallest circuit computing  $f$ . Thus the problem of computing KT complexity (denoted MKTP) was initially viewed as a more-or-less equivalent encoding of MCSP, and it is still the case that all theorems that have been proved about the complexity of MCSP hold also for MKTP (such as those in [5, 9, 10, 17, 21–24, 30, 31, 33, 34]).

In recent years, however, a few hardness results were proved for MKTP that are not yet known to hold for MCSP [7, 8]. We believe that these results can be taken as an indication of what is likely to be true also for MCSP. The present work gives significantly improved hardness results for MKTP.

Reducibility and completeness are the most effective tools in the arsenal of complexity theory for giving evidence of intractability. However, it is not clear whether MCSP or MKTP is NP-complete; neither can be shown to be NP-complete – or even hard for ZPP – under the usual  $\leq_m^P$  reductions without first showing that  $\text{EXP} \neq \text{ZPP}$ , a long-standing open problem [17, 31].

The strongest hardness results that have been proved thus far for MCSP and MKTP are that both are hard for SZK under BPP-Turing reductions [5]. SZK is the class of problems that have Statistical Zero Knowledge Interactive Proofs, and contains many problems of interest to cryptographers. Indeed, if MCSP (or MKTP) is in P/poly, then there are no cryptographically-secure one-way functions [26].

Our main results involve improving the hardness results for MKTP, by reducing the number of queries from polynomially-many, to one. In the paragraphs that follow, we explain the sense in which we accomplish this goal. Along the way, we also obtain a new circuit lower bound for MKTP; it remains unknown whether this circuit lower bound also holds for MCSP.

SZK is not known to be contained in NP; until such a containment can be established, there is no hope of improving the BPP-Turing reduction of [5] to a  $\leq_m^P$  reduction. But we come close in this paper. NISZK is the “non-interactive” subclass of SZK; it contains intractable problems if and only if SZK does [18]. We show that  $\overline{\text{MKTP}}$  is hard for NISZK under  $\leq_m^{P/\text{poly}}$  reductions. (Thus, instead of asking many queries, as in [5], a single query suffices.<sup>1</sup>) Our proof also shows that MKTP is hard for NISZK under BPP reductions that ask only one query. Combined with [18], this shows that MKTP is hard for SZK under *non-adaptive* BPP reductions, yielding a modest improvement over [5]; this has implications regarding the study of worst-case to average-case reductions. (See Section 1.1.)

But  $\leq_m^{P/\text{poly}}$  reductions are still quite powerful. There is great interest currently in proving lower bounds for MCSP, MKTP, and related problems such as MKtP (the problem of computing a different kind of time-bounded Kolmogorov complexity, due to Levin [28]) on very limited classes of circuits and formulae, as part of the “hardness magnification” program. For instance, if modest lower bounds can be shown on the size required to compute MKtP on de Morgan formulae augmented with PARITY gates at the leaves, then EXP is not contained in non-uniform  $\text{NC}^1$  [32]. Also, there is great interest in finding lower bounds against a variety of other models, such as depth-three threshold gates, or circuits consisting of polynomial threshold gates [27]. If a lower bound is known against one of these limited classes of circuits for some problem  $A$  that is reducible to, say, MKTP or MKtP under  $\leq_m^{P/\text{poly}}$  reductions, it implies nothing about the complexity of MKTP or MKtP, since the circuitry involved in computing the reduction is much more powerful than the circuitry in the class of circuits for which the lower bound is known.

---

<sup>1</sup> Some readers may have mistakenly believed that we view our work as a step toward showing that MKTP (or MCSP) is hard for SZK under (uniform)  $\leq_m^P$  reductions. We do not. In fact, some of us doubt that hardness under uniform deterministic reductions holds.

Thus there is a great deal of interest in considering reductions that are much less powerful than  $\leq_m^{P/poly}$  reductions. For extremely weak (uniform) notions of reducibility (such as log-time reductions), it is known that MCSP and MKTP are *not* hard for any complexity class that contains the PARITY function [31]. However, this non-hardness result relies on uniformity; it was later shown that MKTP is hard for the complexity class DET under *nonuniform*  $\leq_m^{NC^0}$  reductions [8].

However, even  $\leq_m^{NC^0}$  reductions are too powerful a tool, when one is interested in lower bounds against the classes of circuits discussed above, since they do not seem to be closed under  $\leq_m^{NC^0}$  reductions. This motivates consideration of the most restrictive type of reduction that we will be considering: projections.

A projection is a reduction that is computed by a circuit consisting only of wires and NOT gates. Each output bit is either a constant, or is connected by a wire to a (possibly negated) input bit. All of the classes of circuits mentioned above (and – indeed – most conceivable classes of circuits) are closed under projections.

Prior to our work, the result of [8] showing that MKTP is hard for DET under  $\leq_m^{NC^0}$  reductions was improved, to show that MKTP is hard for DET even under projections [3]. Since DET is a subclass of P, this provides little ammunition when one is seeking to prove that MKTP is intractable. One of our main contributions is to show that  $\overline{\text{MKTP}}$  is hard for  $\text{NISZK}_L$  under projections. As a corollary, we obtain that MKTP cannot be computed by THRESHOLD $\circ$ MAJORITY circuits of size  $2^{n^{o(1)}}$ . This lower bound relies on the fact that MKTP is hard under projections.

The reader will not be familiar with  $\text{NISZK}_L$ ; this complexity class makes its first appearance in the literature here. It is the “non-interactive” counterpart to the complexity class  $\text{SZK}_L$  that was studied previously by Dvir et al. [15], and was shown there to contain several important natural problems of interest to cryptographers (such as Discrete Log and Decisional Diffie-Hellman).  $\text{NISZK}_L$  contains intractable problems if and only if  $\text{SZK}_L$  does (see Section 2). Thus, for the first time, we show that MKTP is hard under projections for a complexity class that is widely believed to contain intractable problems. Our hardness results carry over immediately to MKtP and to similar problems defined in terms of general Kolmogorov complexity; no hardness results under projections had been known previously for those problems. We present some complete problems for  $\text{NISZK}_L$  and establish some other basic facts about this class in Section 4.

## 1.1 Average-Case Complexity

Building on the techniques introduced in [20], we are able to establish new insights regarding the relationship between worst-case and average-case complexity. In Theorem 35, capitalizing on the fact that essentially every circuit complexity class  $\mathcal{C}$  is closed under projections, we show that if  $\text{NISZK}_L$  does not lie in  $\text{OR} \circ \mathcal{C}$ , then there are problems  $A$  in NP that cannot be solved *in the average case* by errorless heuristics in  $\mathcal{C}$ . For instance, if one were able to show that there is *any* problem  $\text{NISZK}_L$  (including, but not limited to, some of the candidate one-way functions believed to reside there) that cannot be solved *in the worst case* by depth-four  $\text{ACC}^0$  circuits, it would follow that there are problems in NP that are hard-on-average for depth-three  $\text{ACC}^0$  circuits. Such conclusions would *not* follow if our reductions to MKTP had merely been computable in  $\text{AC}^0$  or  $\text{NC}^0$ .

We are also able to shed more light on worst-case to average-case reductions, in the form that they were studied by Bogdanov and Trevisan [14]. Bogdanov and Trevisan showed that there were severe limits on the complexity of problems whose worst-case complexity could be reduced to the average-case complexity of problems in NP via *non-adaptive* reductions; all such problems lie in  $\text{NP/poly} \cap \text{coNP/poly}$ . But it was not known how large this class of

problems could be. Hirahara showed that every problem in SZK has an *adaptive* worst-case to average-case reduction to a problem in NP [20], but the upper bound of  $\text{NP/poly} \cap \text{coNP/poly}$  proved by Bogdanov and Trevisan does not apply for adaptive reductions. As a consequence of our Corollary 17, showing that MKTP is hard for SZK under nonadaptive BPP reductions, we are able to show (in Corollary 37) that the class identified by Bogdanov and Trevisan lies in the narrow range between SZK and  $\text{NP/poly} \cap \text{coNP/poly}$ .

► **Remark.** This is an illustration of the utility of studying MKTP, as an example of a theorem that does not explicitly mention MKTP or MCSP, but which was proved via the study of MKTP. No such argument based on MCSP is known. We believe that MKTP can in fact be viewed as a *particularly convenient* formulation of MCSP, since (a) KT complexity is closely related to circuit size, (b) essentially all theorems known to hold for MCSP also hold for MKTP, (c) some arguments that one might intend to formulate in terms of MCSP elude current approaches, but can instead be successfully carried through by use of MKTP. Furthermore, theorems proved for MKTP may serve as an indication of what is likely to be true for MCSP as well.

The rest of the paper is organized as follows: Our  $\leq_m^{\text{P/poly}}$ -hardness theorem for MKTP is proved in Section 3. Then, after establishing some basic facts about  $\text{NISZK}_L$  in Section 4, in Section 5 we show that  $\overline{\text{MKTP}}$  is hard for  $\text{NISZK}_L$  under projections. We present applications of our reductions and implications for average-case complexity in Section 6.

Due to space limitations, some proofs have been omitted from the version of this work that appears in the ISAAC proceedings. The interested reader is encouraged to consult [6] for complete details.

## 2 Preliminaries

### 2.1 Complexity Classes and Reducibilities

We assume familiarity with the complexity classes P, NP, L, BPP, and P/poly. We also make use of the circuit complexity classes  $\text{AC}^0$  and  $\text{NC}^0$ . For the purposes of this paper,  $\text{AC}^0$  can be understood as the set of problems for which there is a family of circuits  $\{C_n : n \in \mathbb{N}\}$  with unbounded-fan-in AND and OR gates (and NOT gates of fan-in 1) of polynomial size and constant depth.  $\text{NC}^0$  is defined similarly, but with AND and OR gates of bounded fan-in (and thus each output bit depends on only a constant number of bits of the input). We deal primarily with the “nonuniform” versions of these complexity classes (which means that the mapping  $n \mapsto C_n$  need not be computable).

*Branching programs* are a circuit-like model of computation that can be used to characterize logspace computation. A *branching program* is a directed acyclic graph with a single source and two sinks labeled 1 and 0, respectively. Each non-sink node in the graph is labeled with a variable in  $\{x_1, \dots, x_n\}$  and has two edges leading out of it: one labeled 1 and one labeled 0. A branching program computes a Boolean function  $f$  on input  $x = x_1 \dots x_n$  by first placing a pebble on the source node. At any time when the pebble is on a node  $v$  labeled  $x_i$ , the pebble is moved to the (unique) vertex  $u$  that is reached by the edge labeled 1 if  $x_i = 1$  (or by the edge labeled 0 if  $x_i = 0$ ). If the pebble eventually reaches the sink labeled  $b$ , then  $f(x) = b$ . Branching programs can also be used to compute functions  $f : \{0, 1\}^m \rightarrow \{0, 1\}^n$ , by concatenating  $n$  branching programs  $p_1, \dots, p_n$ , where  $p_i$  computes the function  $f_i(x) =$  the  $i$ -th bit of  $f(x)$ . For more information on the definitions, backgrounds, and nuances of these complexity classes, circuits, and branching programs, see the text by Vollmer [35].

A *promise problem*  $\Pi$  is a pair of disjoint sets  $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ . A *solution* to a promise problem is any set  $A$  such that  $\Pi_{\text{YES}} \subseteq A$  and  $\Pi_{\text{NO}} \subseteq \overline{A}$ . A *don't-care instance* of  $\Pi$  is any string that is not in  $\Pi_{\text{YES}} \cup \Pi_{\text{NO}}$ . A *language*  $A$  can be viewed as a promise problem that has no don't-care instances.

Given any class  $\mathcal{C}$  of functions, there is an associated notion of  $m$ -reducibility or *many-one reducibility*: For two languages  $A$  and  $B$ , we say that  $A \leq_m^{\mathcal{C}} B$  if there is a function  $f$  in  $\mathcal{C}$  such that  $x \in A$  iff  $f(x) \in B$ . This notion of reducibility extends naturally to promise problems, mapping yes-instances to yes-instances, and no-instances to no-instances. The most familiar notion of  $m$ -reducibility is Karp reducibility:  $\leq_m^P$ ; NP-completeness is most commonly defined in terms of Karp reducibility. However, in this paper, we will frequently be reducing problems that are not known to reside in NP to MKTP, which does lie in NP. Thus it is clear that a more powerful notion of reducibility is required. Some of our results are most conveniently stated in terms of  $\leq_m^{P/poly}$  reductions (i.e., reductions computed by nonuniform polynomial-size circuits). We also consider restrictions of  $\leq_m^{P/poly}$  reductions, computed by nonuniform  $AC^0$  and  $NC^0$  circuits:  $\leq_m^{AC^0}$  and  $\leq_m^{NC^0}$ . Finally we also consider *projections* ( $\leq_m^{proj}$ ), which are functions computed by  $NC^0$  circuits that have only NOT gates. That is, in a projection, each output bit is either a constant 0 or 1, or is connected by a wire to an input bit or its negation.

We will also make reference to various types of *Turing reducibility*, which are defined in terms of oracle Turing machines, or in terms of circuit families that are augmented with “oracle gates”. For instance, we say that  $A \leq_T^{BPP} B$  if there is a probabilistic polynomial time oracle Turing machine  $M$  with oracle  $B$  that accepts every  $x \in A$  with probability  $\frac{2}{3}$  and rejects every  $x \in \bar{A}$  with probability  $\frac{2}{3}$ . Note that the computation tree of such a BPP-Turing reduction can contain an exponential number of queries to different elements of  $B$ . Just as  $BPP \subseteq P/poly$ , it also holds that  $A \leq_T^{BPP} B$  implies  $A \leq_T^{P/poly} B$ . Thus, on any input  $x$ , the circuit computing the  $P/poly$ -Turing reduction queries only a polynomial number of elements of  $B$ . It was shown in [5] that every problem in SZK (that is, every problem with a statistical zero knowledge proof system) is  $\leq_T^{BPP}$ -reducible (and hence  $\leq_T^{P/poly}$ -reducible) to MCSP and to MKTP. The question of interest to us here is: Is it necessary to ask so many queries? What can we do if we ask only one query? What can be reduced to MKTP via a  $\leq_m^{P/poly}$  reduction?

The complexity class with which we are primarily concerned in this paper is the class of problems that have non-interactive statistical zero knowledge proof systems: NISZK. NISZK was originally defined and studied by Blum et al. [13]. The definition below (in terms of promise problems) is due to Goldreich et al. [18].

► **Definition 1.** A non-interactive statistical zero-knowledge proof system for a promise problem  $\Pi$  is defined by a triple of probabilistic machines  $P$ ,  $V$ , and  $S$ , where  $V$  and  $S$  are polynomial-time and  $P$  is computationally unbounded, and a polynomial  $r(n)$  (which will give the size of the random reference string  $\sigma$ ), such that:

1. (Completeness) For all  $x \in \Pi_{YES}$ , the probability that  $V(x, \sigma, P(x, \sigma))$  accepts is at least  $1 - 2^{-|x|}$ .
2. (Soundness) For all  $x \in \Pi_{NO}$ , the probability that  $V(x, \sigma, P(x, \sigma))$  accepts is at most  $2^{-|x|}$ .
3. (Zero Knowledge) For all  $x \in \Pi_{YES}$ , the statistical distance between the following two distributions bounded by  $1/\beta(|x|)$ 
  - a. Choose  $\sigma$  uniformly from  $\{0, 1\}^{r(|x|)}$ , sample  $p$  from  $P(x, \sigma)$ , and output  $(p, \sigma)$ .
  - b.  $S(x)$  (where the coins for  $S$  are chosen uniformly at random.)

where  $\beta(n)$  is superpolynomial, and the probabilities in Conditions 1 and 2 are taken over the random coins of  $V$  and  $P$ , and the choice of  $\sigma$  uniformly from  $\{0, 1\}^{r(n)}$ .

NISZK is the class of promise problems for which there is a non-interactive statistical zero knowledge proof system.

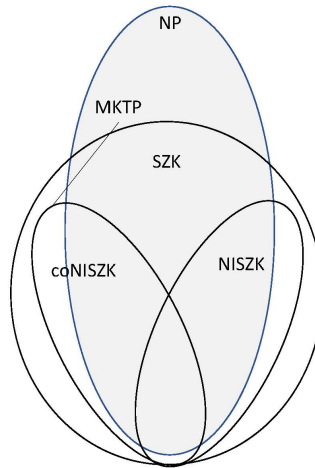
NISZK is not known to be closed under complementation; co-NISZK is defined as the class of promise problems  $\Pi = (\Pi_{YES}, \Pi_{NO})$  such that  $(\Pi_{NO}, \Pi_{YES})$  is in NISZK. It is known that  $SZK = NISZK$  iff  $NISZK = \text{co-NISZK}$ , and that every promise problem in SZK efficiently (and non-adaptively) Turing-reduces to a problem in NISZK [18]. Thus NISZK contains intractable problems if and only if SZK does.

A subclass of SZK, which we will denote by  $SZK_L$ , in which the verifier  $V$  and simulator  $S$  are restricted to being logspace machines, was defined and studied by Dvir et al. [15]. Among other things, they showed that many of the important natural problems in SZK lie in  $SZK_L$ , including Graph Isomorphism, Quadratic Residuosity, Discrete Log, and Decisional Diffie-Helman. The non-interactive version of  $SZK_L$ , which we denote by  $NISZK_L$ , has not been studied previously, but it figures prominently in our results.

► **Definition 2.** *The formal definition of  $NISZK_L$  is obtained by replacing each occurrence of “polynomial-time” in Definition 1 with “logspace”. (It is important to note that, in this model, the logspace-bounded verifier  $V$  and simulator  $S$  are allowed two-way access to the reference string  $\sigma$  and to their polynomially-long sequences of probabilistic coin flips.)*

The reduction presented in [18] carries over directly to the logspace setting, showing that  $NISZK_L$  contains intractable problems if and only if  $SZK_L$  does. In particular, we have:

► **Proposition 3.** *Every promise problem in  $SZK_L$  is non-adaptively  $AC^0$ -Turing-reducible to a problem in  $NISZK_L$ .*



■ **Figure 1** Diagram showing the classes NISZK, co-NISZK, and SZK. The shaded oval represents NP. Every problem in co-NISZK is  $\leq_m^{P/poly}$ -reducible to MKTP.

## 2.2 KT Complexity

The measure KT was defined in [4]. We provide a reproduction of that definition below.

► **Definition 4 (KT).** *Let  $U$  be a universal Turing machine. For each string  $x$ , define  $KT_U(x)$  to be*

$$\min\{|d| + T : (\forall \sigma \in \{0, 1, *\}) (\forall i \leq |x| + 1) U^d(i, \sigma) \text{ accepts in } T \text{ steps iff } x_i = \sigma\}$$

*We define  $x_i = *$  if  $i > |x|$ ; thus, for  $i = |x| + 1$  the machine accepts iff  $\sigma = *$ . The notation  $U^d$  indicates that the machine  $U$  has random access to the description  $d$ .*



To understand the motivation for this definition, see [4]. Briefly: KT is a version of time-bounded Kolmogorov complexity that (in contrast to other notions of resource-bounded Kolmogorov complexity that have been considered) is polynomially-related to circuit complexity. The minimum KT problem, henceforth MKTP, is defined below.

► **Definition 5 (MKTP).** *Suppose  $y \in \{0, 1\}^n$  and  $\theta \in \mathbb{N} \setminus \{0\}$ , then*

$$\text{MKTP} = \{(y, \theta) \mid \text{KT}(y) \leq \theta\}.$$

*In this paper when we view MKTP as a promise problem, yes-instances will be considered those that are in the language, and no-instances those that are not in the language.*

### 3 MKTP is Hard For NISZK

In this section, we prove our first hardness result for MKTP; MKTP is hard for co-NISZK under  $\leq_m^{\text{P/poly}}$  reductions. In order to prove hardness, it suffices to provide a reduction from the *entropy approximation* problem: EA, which is known to be complete for NISZK under  $\leq_m^{\text{P}}$  reductions [18].

► **Definition 6 (Promise-EA).** *Let a circuit  $C : \{0, 1\}^m \rightarrow \{0, 1\}^n$  represent a probability distribution  $X$  on  $\{0, 1\}^n$  induced by the uniform distribution on  $\{0, 1\}^m$ . We define Promise-EA to be the promise problem*

$$\begin{aligned} \text{EA}_{\text{YES}} &= \{(C, k) \mid H(X) > k + 1\} \\ \text{EA}_{\text{NO}} &= \{(C, k) \mid H(X) < k - 1\} \end{aligned}$$

where  $H(X)$  denotes the entropy of  $X$ .

We will make use of some machinery that was developed in [7], in order to relate the entropy of a distribution to the KT complexity of samples taken from the distribution. However, these tools are only useful when applied to distributions that are sufficiently “flat”. The next subsection provides the necessary tools to “flatten” a distribution.

#### 3.1 Flat Distributions

A distribution is considered *flat* if it is uniform on its support. Goldreich et al. [18] formalized a relaxed notion of flatness, termed  $\Delta$ -flatness, which relies on the concept of  $\Delta$ -typical elements. The definitions of both concepts follow:

► **Definition 7 ( $\Delta$ -typical elements).** *Suppose  $X$  is a distribution with element  $x$  in its support. We say that  $x$  is  $\Delta$ -typical if,*

$$2^{-\Delta} \cdot 2^{-H(X)} < \Pr[X = x] < 2^{\Delta} \cdot 2^{-H(X)}.$$

► **Definition 8 ( $\Delta$ -flatness).** *Suppose  $X$  is a distribution. We say that  $X$  is  $\Delta$ -flat if for every  $t > 0$  the probability that an element of the support,  $x$ , is  $t \cdot \Delta$ -typical is at least  $1 - 2^{-t^2+1}$ .*

► **Lemma 9 (Flattening Lemma, [18]).** *Suppose  $X$  is a distribution such that for all  $x$  in its support  $\Pr[X = x] \geq 2^{-m}$ . Then  $X^k$  is  $(\sqrt{k} \cdot m)$ -flat.*

Observe that if  $X$  is a distribution represented by a circuit  $C : \{0, 1\}^m \rightarrow \{0, 1\}^n$ , then the hypothesis of the Flattening Lemma holds for  $m$ . Note also that, for any distribution  $X$ ,  $H(X^k) = k \cdot H(X)$ . Thus the entropy of the distribution  $X^k$  grows linearly with respect to  $k$ , while the deviation from flatness diminishes much more rapidly with respect to  $k$ .



### 3.2 Encoding and Blocking

The *Encoding Lemma* is the primary tool that was developed in [7] to give short descriptions of samples from a given distribution. Below, we give a precise statement of the version of the Encoding Lemma that is stated informally as Remark 4.3 of [7]. (Although the statement there is informal, the proof of the Encoding Lemma that is given there does yield our Lemma 11.) First, we need to define  $\Lambda$ -encodings.

► **Definition 10** ( $\Lambda$ -encodings). *Let  $R : S \rightarrow T$  be a random variable that induces a distribution  $X$ . The  $\Lambda$ -heavy elements of  $T$  are those elements  $\lambda$  such that  $\Pr[X = \lambda] > 1/2^\Lambda$ . A  $\Lambda$ -encoding of  $R$  is given by a mapping  $D : [N] \rightarrow S$  such that for every  $\Lambda$ -heavy element  $\lambda$ , there exists  $i \in [N]$  such that  $R(D(i)) = \lambda$ . We refer to  $\lceil \log(N) \rceil$  as the length of the encoding. The function  $D$  is also called the decoder for the encoding.*

► **Lemma 11** (Encoding Lemma). *[7, Lemma 4.1] Consider an ensemble  $\{R_x\}$  of random variables that sample distributions on strings of some length  $\text{poly}_1(|x|)$ , where there are circuits  $C_x$  of size  $\text{poly}_2(|x|)$  representing each  $R_x$ . Then there is a polynomial  $\text{poly}_3$  such that, for every integer  $\Lambda$ , each  $R_x$  has a  $\Lambda$ -encoding of length  $\Lambda + \log(\Lambda) + O(1)$  that is decodable by circuits of size  $\text{poly}_3(|x|)$ .*

By itself, the Encoding Lemma says nothing about KT complexity. The other important ingredient in the toolbox developed in [7] is the *Blocking Lemma*, which refers to the process of chopping a string into blocks. Let  $y$  be a string of length  $tn$ , which we think of as being the concatenation of  $t$  samples  $y_i$  of a distribution  $X$  on strings of length  $n$ . Thus  $y = y_1 \dots y_t$ . Let  $r = \lceil t/b \rceil$ . Equivalently, we consider  $y$  to be equal to  $z_1 \dots z_r$  where each  $z_i$  is a string of length  $bn$  sampled according to  $X^b$ . (In the case when  $|y|$  is not a multiple of  $b$ ,  $z_r$  is shorter; this does not affect the analysis. We call the strings  $z_i$  the *blocks* of  $y$ .)

► **Lemma 12** (Blocking Lemma). *[7, Lemma 3.3] Let  $\{T_x\}$  be an ensemble of sets of strings such that all strings in  $T_x$  have the same length  $\text{poly}(|x|)$ . Suppose that for each  $x \in \{0, 1\}^*$  and for each  $b \in \mathbb{N}$  there is an integer  $\Lambda_b$  and a random variable  $R_{x,b}$  whose image contains  $(T_x)^b$ , and such that  $R_{x,b}$  is computable by a circuit of size  $\text{poly}(|x|, b)$  and has a  $\Lambda_b$ -encoding of length  $s'(x, b)$  decodable by a circuit of size  $\text{poly}(|x|, b)$ . Then there are constants  $c_1$  and  $c_2$  so that, for every constant  $\alpha > 0$ , every  $t \in \mathbb{N}$ , every sufficiently large  $x$ , and every  $\lceil t^\alpha \rceil$ -suitable  $y \in (T_x)^t$ ,*

$$\text{KT}(y) \leq t^{1-\alpha} \cdot s'(x, \lceil t^\alpha \rceil) + t^{\alpha c_1} \cdot |x|^{c_2}.$$

Here, we say that  $y \in (T_x)^t$  is  $b$ -suitable if each block of  $y$  (of length  $bn$ ) is  $\Lambda_b$ -heavy.

With the Encoding and Blocking Lemmas in hand, we can now show how to give upper and lower bounds on the KT complexity of concatenated samples from a distribution. The following lemma gives the upper bound.

► **Lemma 13.** *Suppose  $X$  is a distribution sampled by a circuit  $C_x : \{0, 1\}^m \rightarrow \{0, 1\}^n$  of size polynomial in  $|x|$ . For every polynomial  $w = w(|x|)$  with  $|x| \leq w$ , there exist constants  $c_0, c_2$ , and  $\alpha_0$  such that for every sufficiently large polynomial  $t$  and for all large  $x$ , if  $y$  is the concatenation of  $t$  samples from  $X$ , then with probability at least  $(1 - 1/2^{2^{|x|}})$ ,*

$$\text{KT}(y) \leq tH(X) + wm(t^{1-\alpha_0/2}) + t^{1-\alpha_0}|x|^{c_0+c_2}$$

We now turn to a lower bound on  $\text{KT}(y)$ .

► **Lemma 14.** Let  $\text{poly}(|x|)$  denote some fixed polynomial in  $|x|$ , and let  $\alpha_0$  be such that  $0 < \alpha_0 < 1/2$ . For all large  $x$ , if  $X$  is a distribution sampled by a circuit  $C_x : \{0, 1\}^m \rightarrow \{0, 1\}^n$  of polynomial size, then it holds that for every  $w$  and every  $t > w^4$ , if  $y$  is sampled from  $X^t$ , then with probability at least  $1 - 2^{-w^2}$ ,

$$\text{KT}(y) \geq tH(X) - wm\sqrt{t} - t^{1-\alpha_0}\text{poly}(|x|)$$

### 3.3 Reducing co-NISZK to MKTP

► **Theorem 15.** MKTP is hard for co-NISZK under P/poly many-one reductions.

**Proof.** We prove the claim by reduction from the NISZK-complete problem EA. Let  $x = (C_x, k)$  be an arbitrary instance of Promise-EA, where  $C_x : \{0, 1\}^m \rightarrow \{0, 1\}^n$  is a circuit that represents distribution  $X$ . Let  $w = 2|x|$ , and let  $\alpha_0, c_0$ , and  $c_2$  be the constants from Lemma 13. Let  $\lambda = wm t^{1-\alpha_0/2}$ . Pick the polynomial  $t$  so that  $t(|x|) > 2(\lambda + t^{1-\alpha_0}|x|^{c_0+c_2})$  and  $w^4 < t$  (and note that all large polynomials have this property). Construct  $y$  as  $t$  samples from  $X$ . Let  $\theta = tk + \lambda + t^{1-\alpha_0}|x|^{c_0+c_2}$ . We claim that, with probability at least  $1 - \frac{1}{2^{2|x|}}$ , if  $(X, k) \in \text{EA}_{YES}$ , then  $(y, \theta) \in \text{MKTP}_{NO}$  and if  $(X, k) \in \text{EA}_{NO}$ , then  $(y, \theta) \in \text{MKTP}_{YES}$ .

If  $(X, k) \in \text{EA}_{NO}$ , then  $H(X) < k$ . Then by Lemma 13, we have that, with high probability,

$$\begin{aligned} \text{KT}(y) &\leq tH(X) + \lambda + t^{1-\alpha_0}|x|^{c_0+c_2} \\ &< tk + \lambda + t^{1-\alpha_0}|x|^{c_0+c_2} \\ &= \theta \end{aligned}$$

thus  $\text{KT}(y) \leq \theta$ , and thus  $(y, \theta) \in \text{MKTP}_{YES}$ .

If  $(X, k) \in \text{EA}_{YES}$ , then  $H(X) > k + 1$ . Then by Lemma 14, with probability at least  $1 - 2^{-w^2} > 1 - 2^{2|x|}$ , we have that

$$\begin{aligned} \text{KT}(y) &\geq tH(X) - wm\sqrt{t} - t^{1-\alpha_0}|x|^{c_0+c_2}, \\ &> tH(X) - \lambda - t^{1-\alpha_0}|x|^{c_0+c_2} && \text{(since } \alpha_0 < 1/2\text{)} \\ &> t(k+1) - \lambda - t^{1-\alpha_0}|x|^{c_0+c_2} \\ &> tk + \lambda + t^{1-\alpha_0}|x|^{c_0+c_2} && \text{(since } t > 2(\lambda + t^{1-\alpha_0}|x|^{c_0+c_2})\text{)} \\ &= \theta \end{aligned}$$

thus  $\text{KT}(y) > \theta$ , and thus  $(y, \theta) \in \text{MKTP}_{NO}$ .

We have shown that there is a polynomial-time-computable function  $f$ , such that, if  $x \in \text{EA}_{YES}$ , then with high probability (for random  $r$ )  $f(x, r) = (y, \theta)$  is in  $\text{MKTP}_{NO}$ , and if  $x \in \text{EA}_{NO}$ , then with high probability  $f(x, r) = (y, \theta)$  is in  $\text{MKTP}_{YES}$ . By a standard counting argument (similar to the proof that  $\text{BPP} \subseteq \text{P/poly}$ ), since the probability of success for either bound is greater than  $(1 - 1/2^{2^n})$ , we can fix a sequence of random bits to hardwire in to this reduction and obtain a family of circuits computing a  $\leq_m^{\text{P/poly}}$  reduction from any problem in NISZK to  $\overline{\text{MKTP}}$ . ◀

► **Corollary 16.** MKTP is hard for NISZK under BPP reductions that make at most one query along any path of the BPP machine.

**Proof.** This follows from the proof of Theorem 15. Namely, on input  $x = (C_x, k)$ , construct the string  $y$  consisting of  $t$  random samples from  $C_x$  and query the oracle on  $(y, \theta)$ . On Yes-instances,  $y$  will have KT complexity greater than  $\theta$  (with high probability), and on No-instances,  $y$  will have KT complexity less than  $\theta$  (with high probability). ◀

► **Corollary 17.** *MKTP is hard for SZK under non-adaptive BPP-Turing reductions.*

**Proof.** Recall from [18] that SZK reduces to Promise-EA via non-adaptive (deterministic) reductions. The result is now immediate, from Corollary 16. ◀

## 4 A Complete Problem for $\text{NISZK}_L$

Having established a hardness result for MKTP under  $\leq_m^{\text{P/poly}}$  reductions, we now establish an analogous hardness result under the much more restrictive  $\leq_m^{\text{proj}}$  reductions. For this, we first need to present a complete problem for  $\text{NISZK}_L$ .

Recall that the NISZK-complete problem EA deals with the question of approximating the entropy of a distribution represented by a circuit. In order to talk about  $\text{NISZK}_L$ , we shall need to consider probability distributions that are represented using restricted class of circuits. In particular, we shall focus on a problem that we denote  $\text{EA}_{\text{NC}^0}$ .

► **Definition 18** (Promise- $\text{EA}_{\text{NC}^0}$ ). *Promise- $\text{EA}_{\text{NC}^0}$  is the promise problem obtained from Promise-EA, by considering only instances  $(C, k)$  such that  $C$  is a circuit of fan-in two gates, where no output gate depends on more than four input gates.*

It is not surprising that  $\text{EA}_{\text{NC}^0}$  is complete for  $\text{NISZK}_L$ . The completeness proof that we present owes much to the proof presented by Dvir et al. [15] (showing that an  $\text{NC}^0$ -variant of the SZK-complete problem ENTROPYDIFFERENCE is complete for  $\text{SZK}_L$ ) and to the proof presented by Goldreich et al. [18] showing that EA is complete for NISZK. We will need to make use of various detailed aspects of the constructions presented in this prior work, and thus we will present the full details here.

First, we show membership in  $\text{NISZK}_L$ .

### 4.1 Membership in $\text{NISZK}_L$

► **Theorem 19.** *Promise- $\text{EA}_{\text{NC}^0} \in \text{NISZK}_L$*

The following corollary is a direct analog to [18, Proposition 1].

► **Corollary 20.** *If  $\Pi$  is any promise problem that is  $\leq_m^L$  reducible to  $\text{EA}_{\text{NC}^0}$ , then  $\Pi \in \text{NISZK}_L$ .*

We close this section by presenting an example of a well-studied natural problem in  $\text{NISZK}_L$ . (A graph is said to be *rigid* if it has no nontrivial automorphism.)

► **Corollary 21.** *The Non-Isomorphism Problem for Rigid Graphs lies in  $\text{NISZK}_L$*

**Proof.** First note that the proof of Theorem 19 carries over to show that a problem that we may call  $\text{EA}_{\text{BP}}$  (defined just as  $\text{EA}_{\text{NC}^0}$  but where the distribution is represented as a branching program instead of as an  $\text{NC}^0$  circuit) also lies in  $\text{NISZK}_L$ . Now observe that the reduction given in Section 3.1 of [7] shows how to take as input two rigid graphs on  $n$  vertices  $(G_0, G_1)$  and build a branching program that takes as input a bitstring  $w$  of length  $t$  and  $t$  permutations  $\pi_1, \dots, \pi_t$  and output the sequence of  $t$  permuted graphs  $\pi_i(G_{w_i})$ . It is observed in [7] that this distribution has entropy  $t(1 + \log n!)$  if the graphs are non-isomorphic, and has entropy at most  $t \log n!$  otherwise. ◀

## 4.2 Hardness for NISZK<sub>L</sub>

In order to re-use the tools developed in [18], we will follow the structure of the proof given there, showing that EA is hard for NISZK. Namely, we introduce the problem SDU (STATISTICAL DISTANCE FROM UNIFORM) and its NC<sup>0</sup> variant, and prove hardness for SDU<sub>NC<sup>0</sup></sub>.

► **Definition 22** (SDU and SDU<sub>NC<sup>0</sup></sub>). *Consider Boolean circuits  $C_X : \{0, 1\}^m \rightarrow \{0, 1\}^n$  representing distributions  $X$ . The promise problem*

$$\text{SDU} = (\text{SDU}_{YES}, \text{SDU}_{NO})$$

is given by

$$\begin{aligned} \text{SDU}_{YES} &\stackrel{\text{def}}{=} \{C_X : \Delta(X, U_n) < 1/n\} \\ \text{SDU}_{NO} &\stackrel{\text{def}}{=} \{C_X : \Delta(X, U_n) > 1 - 1/n\} \end{aligned}$$

where  $\Delta(X, Y) = \sum_{\alpha} |\Pr[X = \alpha] - \Pr[Y = \alpha]|/2$ .

SDU<sub>NC<sup>0</sup></sub> is the analogous problem, where the distributions  $X$  are represented by NC<sup>0</sup> circuits where no output bit depends on more than four input bits.

It is shown in [18, Lemma 4.1] that  $C_X$  is in SDU if and only if  $(C_X, n - 3)$  is in EA. This yields the following corollary:

► **Corollary 23.**  $\text{SDU}_{\text{NC}^0} \leq_{\text{m}}^{\text{proj}} \text{EA}_{\text{NC}^0}$ .

**Proof.** This is trivial if we assume an encoding of SDU<sub>NC<sup>0</sup></sub> instances, such that the NC<sup>0</sup> circuits  $C_X : \{0, 1\}^m \mapsto \{0, 1\}^n$  are encoded by strings of length given by the standard pairing function  $\frac{m^2 + 3m + 2mn + n + n^2}{2}$ , so that the length of an instance of SDU<sub>NC<sup>0</sup></sub> completely determines  $n$ . (An NC<sup>0</sup> circuit with  $m$  inputs and  $n$  outputs has a description of size  $O(n \log m)$ , and thus it is easy to devise a padded encoding of this much larger length.) Thus, in the projection circuit computing the reduction  $C_X \mapsto (C_X, n - 3)$ , the output bits encoding  $n - 3$  are hardwired to constants, and the input bits encoding  $C_X$  are copied directly to the output. ◀

► **Theorem 24.** *Promise-EA<sub>NC<sup>0</sup></sub> and Promise-SDU<sub>NC<sup>0</sup></sub> are hard for NISZK<sub>L</sub> under  $\leq_{\text{m}}^{\text{proj}}$  reductions.*

**Proof.** By Corollary 23, it suffices to show hardness for SDU<sub>NC<sup>0</sup></sub>. In order to establish hardness, we need to develop the machinery of *perfect randomized encodings*, which were developed by Applebaum et al. [12] and then were applied in the setting of SZK<sub>L</sub> by Dvir et al. [15]. Due to space limitations, we refer the reader to [6] for the discussion of perfect randomized encodings.

### 4.2.1 SDU<sub>NC<sup>0</sup></sub> is Complete for NISZK<sub>L</sub>

We now have all of the tools required to complete the proof of Theorem 24.

Let  $\Pi$  be an arbitrary promise problem in NISZK<sub>L</sub> with proof system  $(P, V)$  and simulator  $S$  and let  $x$  be an instance of  $\Pi$ . Recall that the job of the simulator  $S$  is to take a string  $x$  and some uniformly-generated random bits as input, and produce as output a transcript of the form  $(\sigma, p)$ , such that the probability that any transcript  $(\sigma, p)$  is output by  $S$  is very close to the probability that, on input  $x$  with shared randomness  $\sigma$ , the prover  $P$  sends message  $p$  to the verifier  $V$ . Let  $M_x(s)$  denote a routine that simulates  $S(x)$  with randomness  $s$  to obtain a transcript  $(\sigma, p)$ ; if  $V(x, \sigma, p)$  accepts, then  $M_x(s)$  outputs  $\sigma$ , otherwise it outputs  $0^{|\sigma|}$ . (We can assume without loss of generality that  $|\sigma| = |x|^k$ .) It is shown in [18, Lemma 4.2] that the map  $x \mapsto M_x$  is a reduction of  $\Pi$  to SDU:

## 54:12 Cryptographic Hardness Under Projections for Kolmogorov Complexity

▷ **Claim 25.** If  $x \in \prod_{YES}$ , then  $\Delta(M_x, U_{|x|^k}) < 1/|x|^k$ , and  $x \in \prod_{NO}$  implies  $\Delta(M_x, U_{|x|^k}) > 1 - 1/|x|^k$ .

The proof of the preceding claim in [18, Lemma 4.2] actually shows a stronger result. It shows that, if the statistical difference between the output distribution of the simulator and the distribution of true transcripts is at most  $1/e(n)$ , then the statistical difference of  $M_x$  and the uniform distribution is at most  $1/e(n) + 2^{-n}$  on inputs of length  $n$ . Thus, using Definition 1 (which is equivalent to the definition of NISZK given in [18]), the simulator produces a distribution that differs from the uniform distribution by only  $1/n^{\omega(1)}$ . Thus we have the following claim:

▷ **Claim 26.** Let  $c \in \mathbb{N}$ . Then for all large  $x$ , If  $x \in \prod_{YES}$ , then  $\Delta(M_x, U_{|x|^k}) < 1/|x|^{kc}$ , and  $x \in \prod_{NO}$  implies  $\Delta(M_x, U_{|x|^k}) > 1 - 1/|x|^{kc}$ .

Furthermore, it is also shown in [18, Lemma 3.1] that EA has a NISZK protocol in which the completeness error, soundness error, and simulator deviation are all at most  $2^{-m}$  on inputs of length  $m$ . Furthermore, that proof carries over to show that  $\text{EA}_{\text{BP}} \in \text{NISZK}_L$  with these same parameters. Thus we obtain the following fact, which we will use later in Section 6.

▷ **Claim 27.** Let  $c \in \mathbb{N}$ . Then for all large  $x$ , If  $x$  is a Yes-instance of  $\text{EA}_{\text{BP}}$ , then  $\Delta(M_x, U_{|x|^k}) < 1/2^{|x|^{c-1}}$ , and if  $x$  is a No-instance of  $\text{EA}_{\text{BP}}$ , then  $\Delta(M_x, U_{|x|^k}) > 1 - 1/2^{|x|^{c-1}}$ .

Since  $S$  runs in logspace, each bit of  $M_x(s)$  can be simulated with a branching program  $Q_x$ . Furthermore, it is straightforward to see that there is an  $\text{AC}^0$ -computable function that takes  $x$  as input and produces an encoding of  $Q_x$  as output, and it can even be seen that this function can be a *projection*. (To see this, note that in the standard construction of a Turing machine from a logspace-bounded Turing machine  $S$  (with input  $(x, s)$ ) each node of the branching program has a name that encodes a configuration of the machine, a time step, and the position of the input head. This branching program can be constructed in  $\text{AC}^0$ , given only the *length* of  $x$ . In order to construct  $Q_x$ , it suffices merely to hardwire in the transitions from any node that is “scanning” some bit position  $x_i$ . That is, if the transition out of node  $v$  goes to node  $v_0$  if  $x_i = 0$  and to node  $v_1$  if  $x_i = 1$ , then in the adjacency matrix for  $Q_x$ , entry  $(v, v_1) = x_i$  and entry  $(v, v_0)$  is  $\neg x_i$ . This is clearly a projection.)

Now apply the projection of [6, Lemma 37] to (each output bit of) the branching program  $Q_x$  of size  $\ell$ , to obtain an  $\text{NC}^0$  circuit  $C_x$  computing a perfect randomized encoding with blowup  $b = 2^{|x|^k \binom{\ell}{2} - 1}$ . ( $C_x$  has  $\log b + |x|^k$  output bits.)

Now consider  $|H(C_x) - H(U_{\log b + |x|^k})|$ . By [6, Lemma 28] this is equal to  $|H(Q_x) + \log b - H(U_{\log b + |x|^k})|$ . Since  $H(Q_x) = H(M_x)$  and  $H(U_{\log b + |x|^k}) = \log b + H(U_{|x|^k})$ , we have that  $|H(C_x) - H(U_{\log b + |x|^k})| = |H(M_x) - H(U_{|x|^k})|$ . The proof of Theorem 24 is now complete, by appeal to Claim 26. ◀

## 5 Hardness of MKTP under Projections

► **Theorem 28.** MKTP is hard for  $\text{co-NISZK}_L$  under nonuniform  $\leq_m^{\text{AC}^0}$  reductions.

An immediate corollary (making use of the “Gap Theorem” of [1]) is that MKTP is hard for  $\text{co-NISZK}_L$  under  $\leq_m^{\text{NC}^0}$  reductions. Below, we improve this, showing hardness under projections.

► **Corollary 29.** MKTP is hard for  $\text{co-NISZK}_L$  under nonuniform  $\leq_m^{\text{NC}^0}$  reductions.

► **Corollary 30.** MKTP is hard for  $\text{co-NISZK}_L$  under nonuniform  $\leq_m^{\text{proj}}$  reductions.

**Proof.** We now need to claim that the  $\text{AC}^0$ -computable reduction of Theorem 28 can be replaced by a projection. Note that, since  $\text{SDU}_{\text{NC}^0}$  is complete for  $\text{NISZK}_L$  under projections, and since the reduction from  $\text{SDU}_{\text{NC}^0}$  to  $\text{EA}_{\text{NC}^0}$  given in Corollary 23 always uses the same entropy bound  $n - 3$ , we have that it suffices to consider  $\text{EA}_{\text{NC}^0}$  instances  $x = (C_x, k)$  where the bound  $k$  depends only on the length of  $x$ . Thus the bound  $\theta$  produced by our  $\text{AC}^0$  reduction also depends only on the length of  $x$ , and hence can be hardwired in.

We now need only consider the string  $y$ . The  $\leq_m^{\text{AC}^0}$  reduction presented in the proof of Theorem 28 takes as input  $C_x$  and  $r$  and produces the bits of  $y$  by feeding bits of  $r$  into  $C_x$ . Let us recall where the  $\text{NC}^0$  circuitry producing the  $i$ -th bit of  $y$  comes from.

[6, Lemma35] shows how to take an arbitrary branching program and encode the problem of whether the program accepts as a question about the entropy of a distribution represented as a matrix of degree-three polynomials. Each term in this matrix is of the form  $\sum_{j,k} R_1(i,k) L(k,j) R_2(j,m)$ , where the matrices  $R_1$  and  $R_2$  are the same for all inputs of the same length. Thus we need only concern ourselves with the matrix  $L$ .

In Section 4.2.1, it is observed that, given an instance  $x$  of a promise problem in  $\text{NISZK}_L$ , the branching program  $Q_x$  that is used, in order to build the matrix  $L$ , can be constructed from  $x$  by means of a projection. The “input” to this branching program  $Q_x$  is a sequence of random bits (part of the random sequence  $r$  that is hardwired in, in order to create the nonuniform  $\text{AC}^0$  reduction in the proof of Theorem 28). Thus, the only entries of the matrix  $L$  that depend on  $x$  are entries of the form  $(u, v)$  where  $u$  and  $v$  are configurations of a logspace machine, where the machine is scanning  $x_i$  in configuration  $u$ , and it is possible to move to configuration  $v$ . [6, Lemma 37] then shows how to construct  $\text{NC}^0$  circuitry for each term in the degree-three polynomial constructed from  $Q_x$  in the proof of [6, Lemma 35]. The important thing to notice here is that each output bit in the  $\text{NC}^0$  circuit depends on at most one term of one of the degree-three polynomials, and hence it depends on at most one entry of the matrix  $L$  – which means that it depends on at most one bit of the string  $x$ .

Thus, consider any bit  $y_i$  of the string  $y$  produced by the nonuniform  $\text{AC}^0$  reduction from Theorem 28. Either  $y_i$  does not depend on any bit of  $x$ , or it depends on exactly one bit  $x_j$  of  $x$ . In the latter case, either  $y_i = x_j$  or  $y_i = \neg x_j$ . This defines the projection, as required. ◀

The following corollary was pointed out to us by Rahul Santhanam.

► **Corollary 31.** MKTP does not have  $\text{THRESHOLD} \circ \text{MAJORITY}$  circuits of size  $2^{n^{o(1)}}$ .

**Proof.** This is immediate from the lower bound on the Inner Product mod 2 function that is presented in [16]. (See also [11] for a slightly stronger lower bound.) ◀

It should be noted that it remains unknown whether MCSP has  $\text{THRESHOLD} \circ \text{MAJORITY}$  circuits of polynomial size.

## 6 An Application: Average-Case Complexity

The efficient reductions that we have presented have some immediate applications regarding worst-case to average-case reductions. First, we recall the definition of errorless heuristics:

► **Definition 32.** Let  $A$  be any language. An errorless heuristic for  $A$  is an algorithm (or oracle)  $H$  such that, for every  $x$ ,  $H(x) \in \{\text{YES}, \text{NO}, ?\}$ , and

- $H(x) = \text{YES}$  implies  $x \in A$ .
- $H(x) = \text{NO}$  implies  $x \notin A$ .



## 54:14 Cryptographic Hardness Under Projections for Kolmogorov Complexity

► **Definition 33.** A language  $A$  has no average-case errorless heuristics in  $\mathcal{C}$  if, for every polynomial  $p$ , and every errorless heuristic  $H \in \mathcal{C}$  for  $A$ , there exist infinitely many  $n$  such where  $\Pr_{x \in U_n}[H(x) = ?] > 1 - 1/p(n)$ .

In order to state our first theorem relating to average-case complexity, we need the following circuit-based definition:

► **Definition 34.** Let  $\mathcal{C}$  be any complexity class. (Usually, we will think of  $\mathcal{C}$  being a class defined in terms of circuits, and the definition is thus phrased in terms of circuits, although it can be adapted for other complexity classes as well.) The class  $\text{OR} \circ \mathcal{C}$  is the class of problems that can be solved by a family of circuits whose output gate is an unbounded fan-in OR gate, connected to the outputs of circuits in the class  $\mathcal{C}$ .

If problems in  $\text{NISZK}_L$  are hard in the worst case, then there are problems in NP that are hard on average:

► **Theorem 35.** Let  $\mathcal{C}$  be any complexity class that is closed under  $\leq_m^{\text{proj}}$  reductions. If  $\text{NISZK}_L \not\subseteq \text{OR} \circ \mathcal{C}$ , then there is a set  $A$  in NP that has no average-case errorless heuristics in  $\mathcal{C}$ .

The following definition is implicit in the work of Bogdanov and Trevisan [14].

► **Definition 36.** A worst-case to errorless average-case reduction from a promise problem  $\Pi$  to a language  $A$  is given by a polynomial  $p$  and BPP machine  $M$ , such that  $A$  is accepted by  $M^H$  for every oracle errorless heuristic  $H$  for  $A$  such that  $\Pr_{x \in U_n}[H(x) = ?] < 1 - 1/p(n)$ .

► **Corollary 37.** There is a problem  $A \in \text{NP}$  such that there is a non-adaptive worst-case to errorless average-case reduction from every problem in SZK to  $A$ .

► **Remark.** It is implicitly shown by Hirahara [20] that Corollary 37 holds under *adaptive* reductions. The significance of the improvement from adaptive and non-adaptive reductions lies in the fact that Bogdanov and Trevisan showed that the problems in NP for which there is a non-adaptive worst-case to errorless average-case reduction to a problem in NP lie in  $\text{NP/poly} \cap \text{coNP/poly}$  [14, Remark (iii) in Section 4]. Thus SZK may be close to the largest class of problems for which non-adaptive worst-case to errorless average-case reductions to problems in NP exist.

The worst-case to average-case reductions of Definition 36, must work for *every* errorless heuristic that has a sufficiently small probability of producing “?” as output. If we consider a less-restrictive notion (allowing a different reduction for different errorless heuristics) then in some cases we can lower the complexity of the reduction from BPP to  $\text{AC}^0$ .

► **Definition 38.** Let  $\mathcal{D}$  be a complexity class, and let  $\mathcal{R}$  be a class of reducibilities. We say that errorless heuristics for language  $A$  are average-case hard for  $\mathcal{D}$  under  $\mathcal{R}$  reductions if, for every polynomial  $p$  and every errorless heuristic  $H$  for  $A$  where  $\Pr_{x \in U_{|x|}}[H(x) = ?] < 1 - 1/p(|x|)$ , and for every language  $B \in \mathcal{D}$ , there is a reduction  $r \in \mathcal{R}$  reducing  $B$  to  $H$ .

► **Corollary 39.** There is a language  $A \in \text{NP}$ , such that errorless heuristics for  $A$  are average-case hard for  $\text{SZK}_L$  under non-adaptive  $\text{AC}^0$ -Turing reductions.

► **Corollary 40.** Let  $\mathcal{C}$  be any class that is closed under non-adaptive  $\text{AC}^0$ -Turing reductions. If  $\text{SZK}_L \not\subseteq \mathcal{C}$ , then there is a problem in NP that has no average-case errorless heuristic in  $\mathcal{C}$ .

**Proof.** If  $\text{SZK}_L \not\subseteq \mathcal{C}$ , then by Proposition 3,  $\text{NISZK}_L$  is also not contained in  $\mathcal{C}$ . The result is now immediate from Theorem 35. ◀



► Remark. Building on earlier work of Goldwasser et al. [19], average-case hardness results for some subclasses of  $P$  based on reductions computable by constant-depth threshold circuits were presented in [2]. (Although certain aspects of the reductions presented in [2, 19] are computable in  $AC^0$ , in order to obtain deterministic worst-case algorithms, MAJORITY gates are required in those constructions.) We are not aware of any prior work that provides average-case hardness results based on reductions computable in  $AC^0$ , particularly for classes that are believed to contain problems whose complexity is suitable for cryptographic applications.

## 7 Conclusion and Open Problems

By focusing on non-uniform versions of  $\leq_m^P$  reductions, we have shed additional light on how MKTP relates to subclasses of SZK. Some researchers are of the opinion that MCSP (and MKTP) are likely complete for NP under some type of reducibility, and some recent progress seems to support this [25]. For those who share this opinion, a plausible first step would be to show that MKTP is hard not only for co-NISZK, but also for NISZK, under  $\leq_m^{P/poly}$  reductions. (Work by Lovett and Zhang points out obstacles to providing such a reduction via “black box” techniques [29].) And of course, it will be very interesting to see if our hardness results for MKTP hold also for MCSP.

---

### References

- 1 Manindra Agrawal, Eric Allender, and Steven Rudich. Reductions in circuit complexity: An isomorphism theorem and a gap theorem. *Journal of Computer and System Sciences*, 57(2):127–143, 1998.
- 2 Eric Allender, V Arvind, Rahul Santhanam, and Fengming Wang. Uniform derandomization from pathetic lower bounds. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1971):3512–3535, 2012. doi:10.1098/rsta.2011.0318.
- 3 Eric Allender, Azucena Garvia Bosshard, and Amulya Musipatla. A note on hardness under projections for graph isomorphism and time-bounded Kolmogorov complexity. Technical Report TR20-158, Electronic Colloquium on Computational Complexity (ECCC), 2020.
- 4 Eric Allender, Harry Buhrman, Michal Koucký, Dieter Van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006. doi:10.1007/978-3-662-03927-4.
- 5 Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. *Information and Computation*, 256:2–8, 2017. Special issue for MFCS ’14. doi:10.1016/j.ic.2017.04.004.
- 6 Eric Allender, John Gouwar, Shuichi Hirahara, and Caleb Robelle. Cryptographic hardness under projections for time-bounded Kolmogorov complexity. Technical Report TR21-010, Electronic Colloquium on Computational Complexity (ECCC), 2021.
- 7 Eric Allender, Joshua A Grochow, Dieter Van Melkebeek, Christopher Moore, and Andrew Morgan. Minimum circuit size, graph isomorphism, and related problems. *SIAM Journal on Computing*, 47(4):1339–1372, 2018. doi:10.1137/17M1157970.
- 8 Eric Allender and Shuichi Hirahara. New insights on the (non-) hardness of circuit minimization and related problems. *ACM Transactions on Computation Theory*, 11(4):1–27, 2019. doi:10.1145/3349616.
- 9 Eric Allender, Dhiraj Holden, and Valentine Kabanets. The minimum oracle circuit size problem. *Computational Complexity*, 26(2):469–496, 2017. doi:10.1007/s00037-016-0124-0.
- 10 Eric Allender, Rahul Ilango, and Neekon Vafa. The non-hardness of approximating circuit size. *Theory of Computing Systems*, 65(3):559–578, 2021. doi:10.1007/s00224-020-10004-x.
- 11 Kazuyuki Amano. On the size of depth-two threshold circuits for the inner product mod 2 function. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications – 14th International Conference (LATA)*, volume 12038 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2020. doi:10.1007/978-3-030-40608-0\_16.

- 12 Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in  $NC^0$ . *SIAM Journal on Computing*, 36(4):845–888, 2006. doi:10.1137/S0097539705446950.
- 13 Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, 1991. doi:10.1137/0220068.
- 14 Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. *SIAM J. Comput.*, 36(4):1119–1159, 2006. doi:10.1137/S0097539705446974.
- 15 Zeev Dvir, Dan Gutfreund, Guy N Rothblum, and Salil P Vadhan. On approximating the entropy of polynomial mappings. In *Second Symposium on Innovations in Computer Science*, 2011.
- 16 Jürgen Forster, Matthias Krause, Satyanarayana V. Lokam, Rustam Mubarakzjanov, Niels Schmitt, and Hans Ulrich Simon. Relations between communication complexity, linear arrangements, and computational complexity. In *Proc. 21st Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2245 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2001. doi:10.1007/3-540-45294-X\_15.
- 17 Bin Fu. Hardness of sparse sets and minimal circuit size problem. In *Proc. Computing and Combinatorics – 26th International Conference (COCOON)*, volume 12273 of *Lecture Notes in Computer Science*, pages 484–495. Springer, 2020. doi:10.1007/978-3-030-58150-3\_39.
- 18 Oded Goldreich, Amit Sahai, and Salil Vadhan. Can statistical zero knowledge be made non-interactive? or On the relationship of SZK and NISZK. In *Annual International Cryptology Conference*, pages 467–484. Springer, 1999. doi:10.1007/3-540-48405-1\_30.
- 19 Shafi Goldwasser, Dan Gutfreund, Alexander Healy, Tali Kaufman, and Guy N. Rothblum. A (de)constructive approach to program checking. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 143–152. ACM, 2008. doi:10.1145/1374376.1374399.
- 20 Shuichi Hirahara. Non-black-box worst-case to average-case reductions within NP. In *59th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 247–258. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00032.
- 21 Shuichi Hirahara. Non-disjoint promise problems from meta-computational view of pseudorandom generator constructions. In *35th Computational Complexity Conference (CCC)*, volume 169 of *LIPICs*, pages 20:1–20:47. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.20.
- 22 Shuichi Hirahara and Rahul Santhanam. On the average-case complexity of MCSP and its variants. In *32nd Conference on Computational Complexity (CCC)*, volume 79 of *LIPICs*, pages 7:1–7:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CCC.2017.7.
- 23 Shuichi Hirahara and Osamu Watanabe. Limits of minimum circuit size problem as oracle. In *31st Conference on Computational Complexity (CCC)*, volume 50 of *LIPICs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CCC.2016.18.
- 24 John M. Hitchcock and Aduri Pavan. On the NP-completeness of the minimum circuit size problem. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 45 of *LIPICs*, pages 236–245. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.FSTTCS.2015.236.
- 25 Rahul Ilango, Bruno Loff, and Igor Carboni Oliveira. NP-hardness of circuit minimization for multi-output functions. In *35th Computational Complexity Conference (CCC)*, volume 169 of *LIPICs*, pages 22:1–22:36. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.22.
- 26 Valentine Kabanets and Jin-yi Cai. Circuit minimization problem. In *Proceedings of the Thirty-Second Symposium on Theory of Computing (STOC)*, pages 73–79, 2000. doi:10.1145/335305.335314.
- 27 Valentine Kabanets, Daniel M. Kane, and Zhenjian Lu. A polynomial restriction lemma with applications. In *Proceedings of the 49th Annual Symposium on Theory of Computing (STOC)*, pages 615–628. ACM, 2017. doi:10.1145/3055399.3055470.

- 28 Leonid A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984. doi:10.1016/S0019-9958(84)80060-1.
- 29 Shachar Lovett and Jiapeng Zhang. On the impossibility of entropy reversal, and its application to zero-knowledge proofs. In *Theory of Cryptography – 15th International Conference (TCC)*, volume 10677 of *Lecture Notes in Computer Science*, pages 31–55. Springer, 2017. doi:10.1007/978-3-319-70500-2\_2.
- 30 Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. Weak lower bounds on resource-bounded compression imply strong separations of complexity classes. In *Proceedings of the 51st Annual Symposium on Theory of Computing (STOC)*, pages 1215–1225, 2019. doi:10.1145/3313276.3316396.
- 31 Cody Murray and Ryan Williams. On the (non) NP-hardness of computing circuit complexity. *Theory of Computing*, 13(4):1–22, 2017. doi:10.4086/toc.2017.v013a004.
- 32 Igor Carboni Oliveira, Ján Pich, and Rahul Santhanam. Hardness magnification near state-of-the-art lower bounds. In *34th Computational Complexity Conference (CCC)*, volume 137 of *LIPICs*, pages 27:1–27:29. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CCC.2019.27.
- 33 Michael Rudow. Discrete logarithm and minimum circuit size. *Information Processing Letters*, 128:1–4, 2017. doi:10.1016/j.ip1.2017.07.005.
- 34 Michael Saks and Rahul Santhanam. Circuit lower bounds from NP-hardness of MCSP under Turing reductions. In *35th Computational Complexity Conference (CCC)*, volume 169 of *LIPICs*, pages 26:1–26:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CCC.2020.26.
- 35 Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 1999.



# Identity Testing Under Label Mismatch

Clément L. Canonne ✉ 

The University of Sydney, Australia

Karl Wimmer ✉

Duquesne University, Pittsburgh, PA, USA

---

## Abstract

Testing whether the observed data conforms to a purported model (probability distribution) is a basic and fundamental statistical task, and one that is by now well understood. However, the standard formulation, *identity testing*, fails to capture many settings of interest; in this work, we focus on one such natural setting, *identity testing under promise of permutation*. In this setting, the unknown distribution is assumed to be equal to the purported one, up to a relabeling (permutation) of the model: however, due to a systematic error in the reporting of the data, this relabeling may not be the identity. The goal is then to test identity under this assumption: equivalently, whether this systematic labeling error led to a data distribution statistically far from the reference model.

**2012 ACM Subject Classification** Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Hypothesis testing and confidence interval computation

**Keywords and phrases** distribution testing, property testing, permutations, lower bounds

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.55

**Related Version** *Full Version*: <https://arxiv.org/abs/2105.01856>

## 1 Introduction

Imagine you painstakingly gathered observations, data point after data point, and managed to form an accurate estimate of the data distribution; unfortunately, you did not record the labels correctly, and due to a systematic error the data labels have been permuted in an unknown and arbitrary way. You did make your best educated guess to fix this though, and are confident the data, once carefully relabeled, *should* reflect the reality. Can you check this, without having to go through the whole process of obtaining an entirely new dataset?

In this paper, we are concerned with a variant of identity testing which captures the above scenario, where one is promised that the unknown distribution is equal to the reference distribution  $\mathbf{q}$  *up to a permutation of the domain*. Formally, the algorithm has access to i.i.d. samples from a probability distribution  $\mathbf{p}$  over a finite domain  $[n] := \{1, 2, \dots, n\}$  such that  $\mathbf{p} \circ \pi = \mathbf{q}$  for some (unknown)  $\pi \in \mathcal{S}_n$ , and, on input  $0 \leq \varepsilon' < \varepsilon \leq 1$ , must output **yes** or **no** such that

- if  $d_{TV}(\mathbf{p}, \mathbf{q}) \leq \varepsilon'$ , then the algorithm outputs **yes** with probability at least  $2/3$ ;
- if  $d_{TV}(\mathbf{p}, \mathbf{q}) > \varepsilon$ , then the algorithm outputs **no** with probability at least  $2/3$ .

When  $\varepsilon' = 0$ , the task is termed *identity testing (under promise of permutation)*; otherwise, it is *tolerant identity testing*. It is worth noting that this permutation promise fundamentally changes the problem, and makes it incomparable to the standard identity testing problem. As an illustrative example, it is known that *uniformity testing*, where the reference distribution  $\mathbf{q}$  is uniform over  $[n]$ , is the “hardest” case of identity testing, with sample complexity  $\Theta(\sqrt{n})$  and  $\Theta(n/\log n)$  for the testing and tolerant testing versions, respectively [18, 22, 23, 13]. However, it is easy to see that under the permutation promise, uniformity testing is a trivial problem which can be solved with *zero* samples: any permutation of the uniform distribution is itself the uniform distribution.



© Clément L. Canonne and Karl Wimmer;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 55; pp. 55:1–55:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our results further demonstrate this stark difference, showing how the difficulty of testing and tolerant testing differ under this promise. In particular, we show an exponential gap between the sample complexities of non-tolerant and tolerant identity testing under this promise: to the best of our knowledge, this constitutes the first example of such a gap between the tolerant and non-tolerant version of a problem in distribution testing.

## 1.1 Our results

Our results show that, quite surprisingly, the promise of equality up to permutation of the domain fundamentally changes the sample complexity landscape, and is both qualitatively and quantitatively different from what one could expect from the known bounds on identity and tolerant identity testing without this promise.

Our first set of results indeed establishes that, in contrast to the  $\Theta(\sqrt{n})$  sample complexity of “regular” identity testing, identity testing under promise of permutation has sample complexity merely *polylogarithmic* in the domain size:

► **Theorem 1** (Theorems 5 and 8, (Informal)). *Identity testing under promise of permutation has sample complexity  $\Theta(\log^2 n)$ , where  $n$  is the domain size.*

Given the fact that (regular) tolerant identity testing has sample complexity nearly quadratically higher than (regular) identity testing, one could conjecture that the sample complexity tolerant testing under our promise remains polylogarithmic. Our next set of results shows that this is far from being the case: instead, allowing for some noise tolerance makes the promise of equality up to permutation essentially useless, as the sample complexity blows up *exponentially*, growing from polylogarithmic to nearly linear in the domain size:

► **Theorem 2** (Theorems 8 and 9, (Informal)). *Tolerant identity testing under promise of permutation has sample complexity  $\Theta(n^{1-o(1)})$ , where  $n$  is the domain size.*

We also show that relaxing the tolerance allowed from additive (as in the usual tolerant testing setting) to multiplicative in the distance parameter does not really help, as the sample complexity still remains polynomial:

► **Theorem 3** (Theorem 17, (Informal)). *Multiplicative-factor tolerant identity testing under promise of permutation, where one needs to distinguish between  $\varepsilon$ -close and  $C\varepsilon$ -far, has sample complexity  $\Omega(\sqrt{n})$  for any constant factor  $C > 1$ , where  $n$  is the domain size.*

We emphasize once more that those results, and in particular the lower bounds, do not follow from the known results on standard identity testing, as the promise of equality up to permutation, by strengthening the promise, drastically changes the problem. In particular, the case where the reference  $\mathbf{q}$  is uniform, while known to be the hardest case for identity and tolerant identity testing, is actually a trivially easy case under our promise (as any distribution promised to be a permutation of the uniform distribution is, of course, the uniform distribution itself.)

## 1.2 Previous work

Distribution testing has a long history in Statistics, that one can trace back to the work of Pearson [12]. More recently, from the computer science perspective, Goldreich, Goldwasser, and Ron initiated the field of property testing [14]; of which distribution testing emerged through the seminal work of Batu, Fortnow, Rubinfeld, Smith, and White [2]. We refer the reader to the survey [5] for a review of the area of distribution testing.

Among the problems tackled in this field, *identity testing* (also known as goodness-of-fit or one-sample testing), in which the goal is to decide whether an unknown probability distribution  $\mathbf{p}$  is equal to a purported model  $\mathbf{q}$ , has received significant attention. It is known that for identity testing with any reference distribution  $\mathbf{q}$  over a domain of size  $n$ ,  $\Theta(\sqrt{n})$  samples are necessary and sufficient [18, 7, 1, 23]; moreover, the exact asymptotic dependence on the distance parameter and the probability of error of the test [15, 9], as well as some good understanding of the dependence on the reference distribution  $\mathbf{q}$  itself [23, 4], are now understood. Further, we also have tight bounds for the harder problem where one seeks to allow for some noise in the data (i.e., perform *tolerant* identity testing, where the algorithm has to accept distributions sufficient close to the reference  $\mathbf{q}$ ):  $\Theta(n/\log n)$  samples, a nearly linear dependence on the domain size, are known to be necessary and sufficient [20, 21, 22, 16].

However, how the identity testing problem changes under natural constraints on the input data, or under some variations of the formulation, remains largely unexplored. Among the works concerned with such problems, [3, 8] consider identity testing under monotonicity or  $k$ -modality constraints; and [10] focuses on a broad class of shape constraints on the density. Finally, [6] focuses on a variant of identity testing, “identity up to binning,” where two distributions are considered equal if some binning of the domain can make them coincide. To the best of our knowledge, the question considered in the present work, albeit arguably quite natural, has not been previously considered in the Statistics or distribution testing literature.

**Organization.** We provide in Section 3.1 our algorithm for testing identity under promise of permutation, before complementing it in Section 3.2 by our matching lower bound. Section 4 is then concerned with the upper and lower bounds for the tolerant version of the problem; the bulk of which lies in proving the two lower bounds.

## 2 Preliminaries

Let  $\mathcal{S}_n$  denote the set of permutations of  $[n] := \{1, 2, \dots, n\}$ . We identify a probability distribution  $\mathbf{p}$  over  $[n]$  with its probability mass function (pmf), that is, a function  $\mathbf{p}: [n] \rightarrow [0, 1]$  such that  $\sum_{i=1}^n \mathbf{p}(i) = 1$ . For a subset  $S \subseteq [n]$ , we then write  $\mathbf{p}(S) = \sum_{i \in S} \mathbf{p}(i)$  for the probability mass assigned to  $S$  by  $\mathbf{p}$ . Given two probability distributions  $\mathbf{p}, \mathbf{q}$  over  $[n]$ , their total variation distance is

$$d_{\text{TV}}(\mathbf{p}, \mathbf{q}) = \sup_{S \subseteq [n]} (\mathbf{p}(S) - \mathbf{q}(S)) = \frac{1}{2} \|\mathbf{p} - \mathbf{q}\|_1 \quad (1)$$

where  $\|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^n |\mathbf{p}(i) - \mathbf{q}(i)|$  is the  $\ell_1$  distance between the two pmfs. In what follows, given a probability distribution  $\mathbf{q}$  over  $[n]$ , we define

$$\Pi_n(\mathbf{q}) := \{ \mathbf{q} \circ \pi : \pi \in \mathcal{S}_n \}, \quad (2)$$

the set of distributions equal to  $\mathbf{q}$  up to permutation of the domain.

Finally, we will rely on the so-called DKW inequality, which roughly states that  $O(1/\varepsilon^2)$  samples from any univariate distribution suffice to learn it to Kolmogorov distance  $\varepsilon$  with high probability: this is a result due to Dvoretzky, Kiefer, and Wolfowitz from 1956 [11] (with the optimal constant due to Massart, in 1990 [17]).

► **Theorem 4 (DKW Inequality).** *Let  $\hat{\mathbf{p}}$  denote the empirical distribution on  $m$  i.i.d. samples from an arbitrary distribution  $\mathbf{p}$  on  $\mathbb{R}$ . Then, for every  $\varepsilon > 0$ ,*

$$\Pr[d_{\text{K}}(\hat{\mathbf{p}}, \mathbf{p}) > \varepsilon] \leq 2e^{-2m\varepsilon^2},$$

where, for two univariate distributions  $\mathbf{p}, \mathbf{q}$ ,  $d_{\text{K}}(\mathbf{p}, \mathbf{q}) = \sup_{x \in \mathbb{R}} |\mathbf{p}((-\infty, x]) - \mathbf{q}((-\infty, x])|$  denotes the Kolmogorov distance between  $\mathbf{p}$  and  $\mathbf{q}$ .



### 3 Testing

In this section, we establish our matching upper and lower bounds for testing under promise of permutation, Theorems 8 and 9.

#### 3.1 Upper bound

We begin by proving our  $O(\log^2 n)$  upper bound for identity testing under promise of permutation.

► **Theorem 5.** *There exists an algorithm (Algorithm 1) which, for any reference distribution  $\mathbf{q}$  over  $[n]$  and any  $0 < \varepsilon \leq 1$ , given  $O\left(\frac{\log^2 n}{\varepsilon^4}\right)$  samples from an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least  $2/3$  between (i)  $\mathbf{p} = \mathbf{q}$  and (ii)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon$ .*

**Proof.** We first partition the domain into  $L := O(\log(n/\varepsilon)/\varepsilon)$  buckets  $B_1, \dots, B_L$ , where

$$B_\ell := \left\{ i \in [n] : \frac{1}{(1+\varepsilon/4)^\ell} < \mathbf{q}(i) \leq \frac{1}{(1+\varepsilon/4)^{\ell-1}} \right\}, \quad 1 \leq \ell \leq L-1 \quad (3)$$

and  $B_L := \left\{ i \in [n] : \mathbf{q}(i) \leq \frac{1}{(1+\varepsilon/4)^{L-1}} \right\}$ . Note that since  $\mathbf{q}$  is known, we can exactly compute the partition  $B_1, \dots, B_L$ , and in particular those  $L$  sets can be efficiently obtained.

■ **Algorithm 1** Algorithm for identity testing under promise of permutation.

**Require:** Reference distribution  $\mathbf{q}$ , distance parameter  $\varepsilon \in (0, 1]$ , sample access to  $\mathbf{p} \in \Pi_n(\mathbf{q})$

- 1: Set  $L \leftarrow 1 + \left\lceil \frac{\log(4n/\varepsilon)}{\log(1+\varepsilon/4)} \right\rceil = O\left(\frac{\log(n/\varepsilon)}{\varepsilon}\right)$ ,  $\delta \leftarrow \frac{\varepsilon}{4(L-1)}$
- 2: Compute the bucketing  $B_1, \dots, B_L$ , as in (3)
- 3: Using  $O(1/\delta^2)$  samples from  $\mathbf{p}$ , use the empirical estimator to learn the distribution

$$\hat{\mathbf{p}} := (\mathbf{p}(B_1), \dots, \mathbf{p}(B_L))$$

over  $[L]$  to Kolmogorov distance  $\frac{\delta}{3}$ , with probability of error  $1/10$ . Let  $\hat{\mathbf{p}}$  be the output.

- 4: **if**  $\hat{\mathbf{p}}(B_L) > \frac{3\varepsilon}{8}$  or there exists  $\ell^*$  such that  $|\hat{\mathbf{p}}(\{\ell^*, \dots, L-1\}) - \mathbf{q}(\bigcup_{\ell=\ell^*}^{L-1} B_\ell)| > \frac{\delta}{3}$  **then**
- 5:     **return no**
- 6: **else**
- 7:     **return yes**
- 8: **end if**

For our choice of  $L$ ,  $\frac{1}{(1+\varepsilon/4)^{L-1}} \leq \frac{\varepsilon}{4n}$ , so the last bucket  $B_L$  has small probability mass under the reference distribution:  $\mathbf{q}(B_L) \leq \frac{\varepsilon}{4}$ . Now, distinguishing with high constant probability between  $\mathbf{p}(B_L) \leq \frac{\varepsilon}{4}$  and  $\mathbf{p}(B_L) \geq \frac{\varepsilon}{2}$  can be done with  $O(1/\varepsilon)$  samples, so we can detect a discrepancy in  $B_L$  with high probability if there is one (we will argue this part formally at the end of the proof). Consequently, we hereafter assume that  $\mathbf{p}(B_L) < \frac{\varepsilon}{2}$ .

If  $\mathbf{p} = \mathbf{q}$ , clearly  $\mathbf{p}(B_L) \leq \frac{\varepsilon}{4}$  (so the first check above passes) and  $\mathbf{q}(B_\ell) = \mathbf{p}(B_\ell)$  for all  $1 \leq \ell \leq L-1$ . However, if  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon$ , then  $\sum_{\ell=1}^{L-1} \sum_{i \in B_\ell} |\mathbf{p}(i) - \mathbf{q}(i)| > 2\varepsilon - \frac{3\varepsilon}{4} = \frac{5\varepsilon}{4}$ . Moreover, letting  $\pi \in \mathcal{S}_n$  be the permutation such that  $\mathbf{p} = \mathbf{q} \circ \pi$ , consider the set  $S \subseteq [n]$  of elements which  $\pi$  maps to an element from the same bucket:

$$S := \{ i \in [n] \setminus B_L : \exists \ell \in [L-1], i \in B_\ell, \pi(i) \in B_\ell \}.$$

For each such element  $i$ , by definition of the bucketing,  $|\mathbf{p}(i) - \mathbf{q}(i)| = |\mathbf{q}(i) - \mathbf{q}(\pi(i))| \leq \frac{\varepsilon}{4} \mathbf{q}(i)$ . It follows that the elements from  $S$  amount for a total  $\ell_1$  distance of at most  $\frac{\varepsilon}{4}$ , and therefore a constant fraction of the distance between  $\mathbf{p}$  and  $\mathbf{q}$  comes from the set  $T := [n] \setminus (B_L \cup S)$  of elements that  $\pi$  “moves to a different bucket:”

$$\frac{5}{4} \varepsilon < \sum_{i \in S} |\mathbf{p}(i) - \mathbf{q}(i)| + \sum_{i \in T} |\mathbf{p}(i) - \mathbf{q}(i)| \leq \frac{\varepsilon}{4} \mathbf{q}(S) + \sum_{i \in T} |\mathbf{p}(i) - \mathbf{q}(i)| \leq \frac{\varepsilon}{4} + \sum_{i \in T} |\mathbf{p}(i) - \mathbf{q}(i)|$$

that is,  $\sum_{i \in T} |\mathbf{p}(i) - \mathbf{q}(i)| > \varepsilon$ .

Partition the set  $T$  by setting  $T_\ell := T \cap B_\ell$ , for  $\ell \in [L-1]$ . Rewriting the above inequality, we obtained that

$$\sum_{i \in T} |\mathbf{p}(i) - \mathbf{q}(i)| = \sum_{\ell=1}^{L-1} \sum_{i \in T_\ell} |\mathbf{p}(i) - \mathbf{q}(i)| > \varepsilon. \quad (4)$$

We will use this to prove the following result.

▷ **Claim 6.** Suppose that  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon$ . Then there exists some  $\ell^* \in [L-1]$  such that  $\left| \mathbf{p}(\bigcup_{\ell=\ell^*}^{L-1} B_\ell) - \mathbf{q}(\bigcup_{\ell=\ell^*}^{L-1} B_\ell) \right| > \delta$ , where  $\delta = \frac{\varepsilon}{4(L-1)}$ .

*Proof.* We note that since  $\mathbf{p}(B_\ell) = \mathbf{p}(S_\ell) + \mathbf{p}(T_\ell)$  and that  $\mathbf{p}(S_\ell) = \mathbf{q}(S_\ell)$  (by definition of  $S_\ell \subseteq S$ )<sup>1</sup> for every  $\ell$ , it suffices to prove the statement for  $T_\ell$ , that is, that there exists  $\ell^*$  such that

$$\left| \mathbf{p}\left(\bigcup_{\ell=\ell^*}^{L-1} T_\ell\right) - \mathbf{q}\left(\bigcup_{\ell=\ell^*}^{L-1} T_\ell\right) \right| > \delta.$$

The key property we will use is that, for every  $\ell < \ell'$ , we have  $\mathbf{q}(i) \geq \mathbf{q}(j)$  for every  $i \in B_\ell, j \in B_{\ell'}$ . This property, which follows from the definition of bucketings, guarantees that if  $\pi$  maps an element  $i \in T_{\ell'}$  to element  $\pi(i) \in T_\ell$ , then  $\mathbf{p}(i) \geq \mathbf{q}(i)$ .

Let  $U, V \subseteq [L-1]$  be the buckets whose probability mass under  $\mathbf{p}$  is greater than or equal to (resp., less than or equal to) the probability mass under  $\mathbf{q}$ , i.e.,

$$U := \{ \ell \in [L-1] : \mathbf{p}(T_\ell) \geq \mathbf{q}(T_\ell) \}, \quad V := \{ \ell \in [L-1] : \mathbf{p}(T_\ell) \leq \mathbf{q}(T_\ell) \}$$

This lets us rewrite (4) as

$$\varepsilon < \sum_{\ell \in U} \sum_{i \in T_\ell} |\mathbf{p}(i) - \mathbf{q}(i)| + \sum_{\ell \in V} \sum_{i \in T_\ell} |\mathbf{p}(i) - \mathbf{q}(i)|$$

and so at least one of the two terms in the RHS must exceed  $\frac{\varepsilon}{2}$ . Without loss of generality, suppose  $\sum_{\ell \in U} \sum_{i \in T_\ell} |\mathbf{p}(i) - \mathbf{q}(i)| > \frac{\varepsilon}{2}$ . This implies there exists  $\ell^* \in U$  such that  $\sum_{i \in T_{\ell^*}} |\mathbf{p}(i) - \mathbf{q}(i)| > \frac{\varepsilon}{2(L-1)}$ ; we will focus on this  $\ell^*$ .

Partition  $T_{\ell^*}$  further into  $T_{\ell^*}^+$  and  $T_{\ell^*}^-$ , where  $T_{\ell^*}^+$  (resp.  $T_{\ell^*}^-$ ) is the set of elements  $i \in T_{\ell^*}$  such that  $\pi(i)$  belongs to a bucket  $B_\ell$  with  $\ell < \ell^*$  (resp.,  $\ell > \ell^*$ ). Note that, for any  $i \in T_{\ell^*}^+$ , we then have  $\mathbf{p}(i) = \mathbf{q}(\pi(i)) \geq \mathbf{q}(i)$ , and conversely for  $i \in T_{\ell^*}^-$ : so that we can rewrite the above as

$$\frac{\varepsilon}{2(L-1)} < (\mathbf{p}(T_{\ell^*}^+) - \mathbf{q}(T_{\ell^*}^+)) + (\mathbf{q}(T_{\ell^*}^-) - \mathbf{p}(T_{\ell^*}^-))$$

<sup>1</sup> Indeed, we have  $\mathbf{p}(S_\ell) = \sum_{i \in S_\ell} \mathbf{p}(i) = \sum_{i \in S_\ell} \mathbf{q}(\pi(i)) = \mathbf{q}(\pi^{-1}(S_\ell))$ , and  $\pi(S_\ell) = S_\ell$  by definition.

## 55:6 Identity Testing Under Label Mismatch

Now, since  $\mathbf{p}(T_{\ell^*}) \geq \mathbf{q}(T_{\ell^*})$  (as  $\ell^* \in U$ ), we have  $\mathbf{p}(T_{\ell^*}^+) - \mathbf{q}(T_{\ell^*}^+) \geq \mathbf{q}(T_{\ell^*}^-) - \mathbf{p}(T_{\ell^*}^-)$  and therefore  $\mathbf{p}(T_{\ell^*}^+) - \mathbf{q}(T_{\ell^*}^+) > \frac{\varepsilon}{4(L-1)}$ . This implies the claim: indeed, we then have

$$\mathbf{p}(\cup_{\ell=\ell^*}^{L-1} T_\ell) > \mathbf{q}(\cup_{\ell=\ell^*}^{L-1} T_\ell) + \frac{\varepsilon}{4(L-1)}$$

since  $T_{\ell^*}$  “receives” a difference of at least  $\frac{\varepsilon}{4(L-1)}$  probability mass from lower-index buckets, and besides this the total probability mass of the suffix of buckets  $\cup_{\ell=\ell^*}^{L-1} T_\ell$  cannot decrease by any internal swap of elements. ◀

With the above claim in hand, we can conclude the analysis. Indeed, as the sets  $B_1, \dots, B_L$  are known, one can estimate the induced probability distribution  $\bar{\mathbf{p}} := (\mathbf{p}(B_1), \dots, \mathbf{p}(B_L))$  to Kolmogorov distance  $\frac{\delta}{3}$  (with probability at least 9/10) using  $O(1/\delta^2) = O(L^2/\varepsilon^2) = O(\log^2(n/\varepsilon)/\varepsilon^4)$  samples (this follows from Theorem 4). Let  $\hat{\mathbf{p}}$  be the resulting distribution over  $[L]$ . Whenever this step is successful (i.e., with probability at least 9/10, the following holds.

- If  $\mathbf{p} = \mathbf{q}$ , then  $|\hat{\mathbf{p}}(L) - \mathbf{q}(B_L)| \leq \frac{\delta}{3}$ , so  $\hat{\mathbf{p}}(B_L) \leq \frac{\varepsilon}{4} + \frac{\delta}{3} \leq \frac{3}{8}\varepsilon$ ; and  $|\hat{\mathbf{p}}(\{\ell^*, \dots, L-1\}) - \mathbf{q}(\cup_{\ell=\ell^*}^{L-1} B_\ell)| \leq \frac{\delta}{3}$  for all  $\ell$ . Thus, the test accepts.
- If  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon$ , then either
  - $\mathbf{p}(B_L) > \frac{\varepsilon}{2}$ , in which case  $\hat{\mathbf{p}}(L) \geq \frac{\varepsilon}{2} - \frac{\delta}{3} > \frac{3}{8}\varepsilon$  and the test rejects; or
  - $\mathbf{p}(B_L) \leq \frac{\varepsilon}{2}$ , in which case by Claim 6 there exists some  $\ell^* \in [L-1]$  such that  $|\mathbf{p}(\cup_{\ell=\ell^*}^{L-1} B_\ell) - \mathbf{q}(\cup_{\ell=\ell^*}^{L-1} B_\ell)| > \delta$ . Then,

$$\left| \hat{\mathbf{p}}(\{\ell^*, \dots, L\}) - \mathbf{q}\left(\bigcup_{\ell=\ell^*}^{L-1} B_\ell\right) \right| \geq \left| \mathbf{p}\left(\bigcup_{\ell=\ell^*}^{L-1} B_\ell\right) - \mathbf{q}\left(\bigcup_{\ell=\ell^*}^{L-1} B_\ell\right) \right| - \frac{\delta}{3} > \frac{2}{3}\delta$$

and the test rejects.

This concludes the proof of correctness of the algorithm. The claimed sample complexity readily follows from our choice of  $\delta = \Theta(L/\varepsilon)$  and the  $O(1/\delta^2)$  sample complexity of learning an arbitrary real-valued distribution to Kolmogorov distance  $\delta$ . ◀

► **Remark 7 (On the tolerance of the tester).** We note that the above analysis establishes a slightly stronger statement; namely, that the testing algorithm allows for some small tolerance, accepting distributions that are  $O(\varepsilon/\log n)$ -close to  $\mathbf{q}$ , and rejecting those that are  $\varepsilon$ -far. As we will see later, this  $\Omega(\log n)$  factor in the amount of tolerance is essentially optimal, as by Theorem 17 reducing it to  $o(\log n)$  would require sample complexity  $n^{1/2-o(1)}$ .

### 3.2 Lower bound

In this section, we show that the  $O(\log^2 n)$  upper bound from the previous section is tight, by proving a matching lower bound on the sample complexity of identity testing under promise of permutation.

► **Theorem 8.** *Any algorithm which, given a reference distribution  $\mathbf{q}$  over  $[n]$ ,  $0 < \varepsilon \leq 1$  such that  $\varepsilon = \tilde{\Omega}(1/n^{1/4})$ , and sample access to an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least 2/3 between (i)  $\mathbf{p} = \mathbf{q}$  and (ii)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon$ , must have sample complexity  $\Omega\left(\frac{\log^2 n}{\varepsilon^2}\right)$ .*

**Proof.** We first describe a construction with constant distance  $\varepsilon = 1/9$ , leading to an  $\Omega(\log^2 n)$  lower bound; before explaining how to obtain the claimed  $\Omega\left(\frac{1}{\varepsilon^2} \log^2 n\right)$  lower bound from it. Our lower bound will rely on a reference distribution  $\mathbf{q}$  piecewise-constant

on  $L = \Theta(\log n)$  buckets, where bucket  $\ell$  has a number of elements proportional to  $2^\ell$ . The first and last buckets (that is, the smallest and largest) will each have total probability mass  $1/3$  under  $\mathbf{q}$ , and be uniform. The remaining “middle”  $L - 2$  buckets all have  $1/(3(L - 2))$  total probability mass, and are uniform as well. We then build a family of perturbations  $\{\mathbf{p}_\pi = \mathbf{q} \circ \pi\}_\pi \subseteq \Pi_n(\mathbf{q})$ , such that under each perturbation  $\mathbf{p}_\pi$  the middle buckets keep the exact same total probability mass  $1/(3(L - 2))$ , by “cascading” mass from one bucket to the next. Details follow.

Set  $L := \Theta(\log n)$  to be the largest integer such that  $L2^L \leq \sqrt{n}$ , and assume for convenience that  $\lceil \sqrt{n} \rceil$  is a multiple of 3. The  $\ell$ th bucket  $B_\ell$ , for  $0 \leq \ell \leq L - 2$ , has size

$$|B_\ell| = \lceil \sqrt{n} \rceil \cdot 2^\ell$$

and  $|B_{L-1}| = 2(L - 2)|B_{L-2}|$ , so that  $\frac{n}{8} \leq \sum_{\ell=0}^{L-1} |B_\ell| = \lceil \sqrt{n} \rceil \cdot 2^{L-1}(L - 1) \leq n$ . (We hereafter focus on the first part of the domain, and will ignore the last  $n - \sum_{\ell=0}^{L-1} |B_\ell|$  elements.) Note that each bucket contains at least  $\sqrt{n}$  elements by construction, and has a size which is a multiple of 3. The reference distribution  $\mathbf{q}$  is then uniform inside each bucket, where

- $\mathbf{q}(B_0) = \mathbf{q}(B_{L-1}) = \frac{1}{3}$ , and
- $\mathbf{q}(B_\ell) = \frac{1}{3(L-2)}$  for all  $0 < \ell < L - 1$ .

In particular, our choice of  $|B_{L-1}|$  ensures that each element of the last bucket, under  $\mathbf{q}$ , will have probability mass

$$\frac{1}{3|B_{L-1}|} = \frac{1}{2} \cdot \frac{1}{3(L-2)|B_{L-2}|}$$

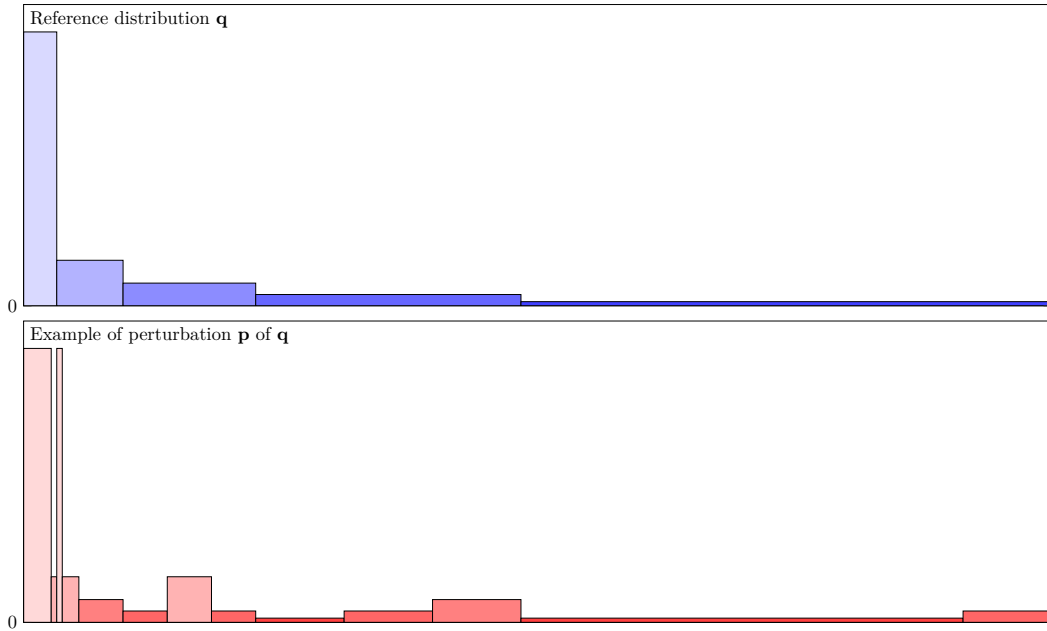
that is, half the probability mass of elements of the  $(L - 2)$ th bucket.

Each perturbation will then have the same distribution over buckets:

- each of the  $L - 2$  middle buckets  $B_\ell$  is (independently) partitioned uniformly at random into 3 sets  $S_{\ell,1}, S_{\ell,2}, S_{\ell,3}$  of equal size. The permutation then swaps  $S_{\ell,2} \cup S_{\ell,3}$  and  $S_{\ell+1,1}$ , for  $1 \leq \ell \leq L - 3$  (note that indeed  $|S_{\ell,2} \cup S_{\ell,3}| = |S_{\ell+1,1}|$ , but  $\mathbf{q}(S_{\ell,2} \cup S_{\ell,3}) = 2\mathbf{q}(S_{\ell+1,1}) = \frac{2}{9(L-2)}$ ).
- a uniformly random subset  $S_0 \subseteq B_0$  of size  $\frac{|B_0|}{3(2L-5)} = O(|S_{1,1}|/L)$  is selected, and the permutation swaps it with a uniformly random subset  $T_1 \subseteq S_{1,1}$  of equal size. By choice of the size, we had  $\mathbf{q}(S_0) = \frac{2}{9(2L-5)}$  and  $\mathbf{q}(T_1) = \frac{1}{9(2L-5)(L-2)}$ , so that  $\mathbf{q}(S_0) - \mathbf{q}(T_1) = \frac{1}{9(L-2)}$ .
- similarly, the subset  $S_{L-2,2} \cup S_{L-2,3}$  of size  $\frac{2}{3}|B_{L-2}| = \frac{|B_{L-1}|}{3(L-2)}$  is swapped with a uniformly random subset  $T_{L-1} \subseteq B_{L-1}$  of equal size. By choice of the size, we had  $\mathbf{q}(S_{L-2,2} \cup S_{L-2,3}) = \frac{2}{9(L-2)}$  and  $\mathbf{q}(T_{L-1}) = \frac{1}{9(L-2)}$ , so that again  $\mathbf{q}(S_{L-2,2} \cup S_{L-2,3}) - \mathbf{q}(T_{L-1}) = \frac{1}{9(L-2)}$ .

As a result, we get that for each such perturbation  $\mathbf{p} = \mathbf{q} \circ \pi$ ,  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) \geq \frac{1}{9}$ . The construction is illustrated in Figure 1.

By a birthday paradox-type argument, no element will be sampled twice unless the number of samples is at least  $\Omega(1/\sqrt{\sum_{i=1}^n \mathbf{p}(i)^2}) = \Omega(1/\sqrt{n \max_{i \in [n]} \mathbf{p}(i)}) = \Omega(n^{1/4})$ , which is far beyond the polylogarithmic regime we are working in. By construction, under each  $\mathbf{p}$ , all  $L - 2$  middle buckets have exactly the same probability mass  $\frac{1}{3(L-2)}$ , and elements inside are perturbed randomly, either having probability (compared to  $\mathbf{q}$ ) multiplied by 2 with probability  $1/3$  or divided by 2 with probability  $1/3$ . Because of the uniformly random choice of the 3-way partition inside each bucket and the fact that each of all those inner



■ **Figure 1** Reference distribution  $\mathbf{q}$  and example of perturbation  $\mathbf{p}$ , for  $L = 5$ . Note that the total probability mass of each bucket of  $\mathbf{q}$  is preserved under  $\mathbf{p}$ , except for the first and last one whose mass decreases and increases by  $\Theta(1/L)$ , respectively.

partitions are chosen independently across buckets, the information from those  $L - 2$  buckets does not provide any advantage in distinguishing them from  $\mathbf{p}$  unless the same element is hit twice.<sup>2</sup>

This addresses the case of the middle  $L - 2$  buckets. Turning to the remaining two, the probability mass of both end buckets, under any perturbation  $\mathbf{p}$ , deviates from what it is under  $\mathbf{q}$  by an additive  $\delta := \frac{1}{9(L-2)}$ . Since those buckets each have total probability mass  $1/3$  under  $\mathbf{p}$  and  $1/3 \pm \delta$  under each  $\mathbf{q}$  and we do not see any collisions with high probability, detecting this requires  $\Omega(1/\delta^2) = \Omega(\log^2 n)$  samples, giving the lower bound for constant  $\varepsilon = 1/9$ .

To obtain the inverse quadratic dependence on the distance parameter, one can then simply repeat the above argument for any  $0 < \varepsilon < 1/9$  by replacing our reference distribution  $\mathbf{q}$  and all the perturbations  $\mathbf{p}_\pi = \mathbf{q} \circ \pi$  by the mixtures

$$\mathbf{q}_\varepsilon := (1 - 9\varepsilon)\mathbf{u} + 9\varepsilon\mathbf{q}, \quad \mathbf{p}_{\varepsilon,\pi} := (1 - 9\varepsilon)\mathbf{u} + 9\varepsilon\mathbf{p}_\pi = \mathbf{q}_\varepsilon \circ \pi$$

the last equality crucially using the fact that the uniform distribution  $\mathbf{u}$  (over the domain) is invariant by permutation. Note that every such  $\mathbf{p}_{\varepsilon,\pi}$  then does belong to  $\Pi_n(\mathbf{q}_\varepsilon)$ , and is at total variation distance exactly  $\varepsilon$  from  $\mathbf{q}_\varepsilon$ . Moreover, we can repeat the previous argument *mutatis mutandis*: (i) the middle buckets provide no information whatsoever unless an element is seen twice, which requires  $\Omega(n^{1/4}/\varepsilon)$  samples (the extra  $1/\varepsilon$  due to our mixture with weight  $9\varepsilon$ ); while the two outer buckets have a discrepancy only  $\delta := \frac{\varepsilon}{L-2}$ , which to be detected requires at least  $\Omega(1/\delta^2) = \Omega((\log^2 n)/\varepsilon^2)$  samples overall. The minimum of these two quantities gives the claimed lower bound, as long as  $n^{1/4}/\varepsilon = \Omega((\log^2 n)/\varepsilon^2)$ , that is,  $\varepsilon = \Omega((\log^2 n)/n^{1/4})$ . ◀

<sup>2</sup> That is, conditioned on seeing each element of those  $L - 2$  buckets at most once, the conditional distribution over those  $L - 2$  buckets under (i)  $\mathbf{q}$  and (ii) the uniform mixture of all perturbations  $\mathbf{p}$  are indistinguishable.

## 4 Tolerant testing

We now turn to the task of *tolerant* testing. As mentioned in the introduction, tolerant testing is well known to be harder than standard (non-tolerant) testing, with a nearly quadratic gap for the standard identity testing problem ( $\sqrt{n}$  vs.  $\frac{n}{\log n}$  sample complexity). Surprisingly, we are able to show that under the promise of permutation, the task does not suffer a merely polynomial blowup – the sample complexity of tolerant identity testing becomes *exponentially* harder than that of standard testing, jumping from  $\log^2 n$  to  $n^{1-o(1)}$ .

The first component, an  $O(n/\log n)$  upper bound for tolerant testing under promise of permutation (Theorem 9), is straightforward, and simply follows from the corresponding upper bound absent this promise. A much more challenging task is in establishing the lower bound. We actually provide two lower bounds: the first, an  $\Omega(n^{1-o(1)})$  lower bound (Theorem 10), applies for the usual setting of tolerant testing with an additive gap  $\delta$  between  $\varepsilon'$  and  $\varepsilon$ . The second (Theorem 17) is an  $\Omega\left(\sqrt{n/2^{O(C)}}\right)$  sample complexity lower bound for any  $C$ -factor approximation of the distance, that is to distinguish between  $\varepsilon$ -close and  $C\varepsilon$ -far.

### 4.1 Upper bound

The claimed upper bound readily follows from the analogous upper bound on tolerant testing *without* the promise of permutation, due to Valiant and Valiant [22, Theorem 4] (see, also, [16]). Indeed, any such estimator can be used for our problem, ignoring the additional promise of identity up to permutation.

► **Theorem 9.** *There exists an algorithm which, for any reference distribution  $\mathbf{q}$  over  $[n]$  and any  $0 \leq \varepsilon, \delta \leq 1$  such that  $\delta = \Omega(1/\sqrt{\log n})$ , and given  $O\left(\frac{n}{\delta^2 \log n}\right)$  samples from an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least  $2/3$  between (i)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) \leq \varepsilon$  and (ii)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon + \delta$ .*

We note that the requirement  $\delta = \Omega(1/\sqrt{\log n})$  has been relaxed in [16].

### 4.2 Lower bound

In this section, we prove the theorem below, our lower bound on the sample complexity of tolerant testing under promise of permutation. Before doing so, we emphasize that the known  $\Omega\left(\frac{n}{\delta^2 \log n}\right)$  sample complexity lower bound for tolerant testing *absent* this promise does not apply to our setting, as the promise of permutation makes the testing problem easier. In particular, the hard instances used to prove the aforementioned  $\Omega\left(\frac{n}{\delta^2 \log n}\right)$  lower bound do not satisfy this promise.<sup>3</sup>

► **Theorem 10.** *Any algorithm which, given a reference distribution  $\mathbf{q}$  over  $[n]$ ,  $0 < \varepsilon, \delta \leq 1$ , and sample access to an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least  $2/3$  between (i)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) \leq \varepsilon$  and (ii)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) > \varepsilon + \delta$ , must have sample complexity  $\Omega(\delta^2 n^{1-O(1/\log(1/\delta))})$ .*

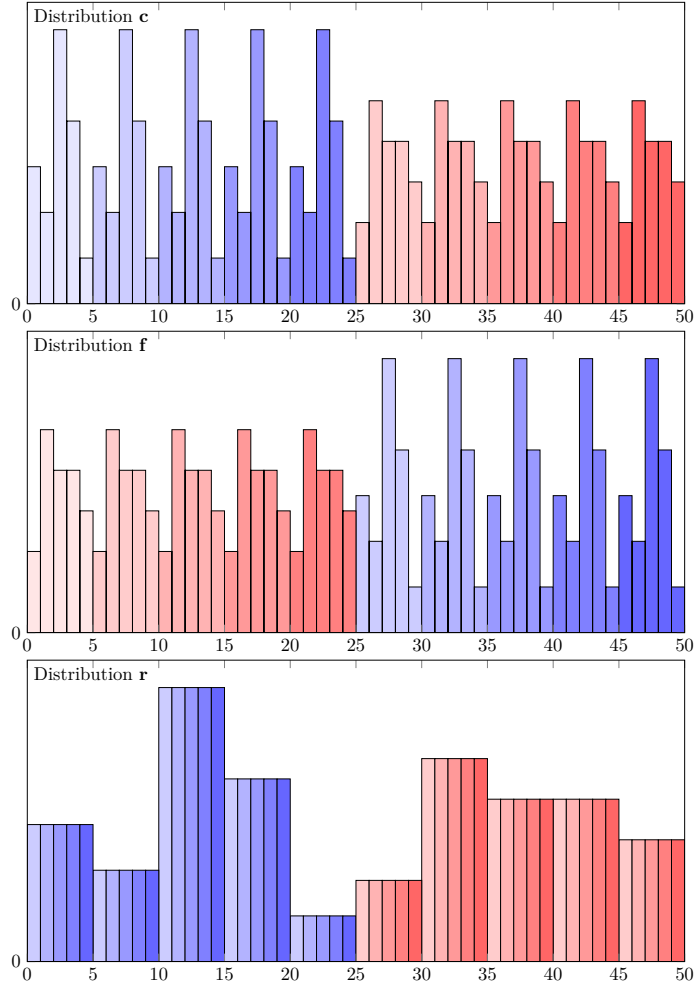
<sup>3</sup> One can also note that the lower bound for “standard” tolerant testing is obtained by choosing the reference distribution to be uniform over  $[n]$ . Under promise of permutation, this particular instance of the problem is trivial, as any permutation of the uniform distribution is still the uniform distribution.

## 55:10 Identity Testing Under Label Mismatch

**Proof.** In what follows, we assume that  $\delta = \Omega(1/\sqrt{n})$ , as otherwise there is nothing to prove. Let  $k \geq 1$  be an integer to be chosen during the course of the analysis (we will set  $k = \Theta(1/\delta)$ ), and write  $n = 2mk^2$  for some integer  $m \geq 1$  (this can be done without loss of generality, as our assumption on  $\delta$  ensures that  $n \geq 2mk^2$ ). For  $1 \leq \ell \leq 2k$ , we define the integer interval  $I_{k,\ell} := [k] + (\ell - 1)k$ , so that  $[2k^2] = \bigcup_{\ell=1}^{2k} I_{k,\ell}$ .

Given two distributions  $\mathbf{p}, \mathbf{q}$  over  $[k]$ , we define families of distributions  $\mathcal{C}_{\mathbf{p},\mathbf{q}}$  and  $\mathcal{F}_{\mathbf{p},\mathbf{q}}$  over  $[n]$  as follows: first, we consider the distributions  $\mathbf{c}, \mathbf{f}$ , each over  $[2k^2]$ , obtained by “repeating and alternating”  $\mathbf{p}$  and  $\mathbf{q}$  as follows:

- For  $1 \leq \ell \leq k$  and  $j \in I_{k,\ell}$ ,  $\mathbf{c}(j) = \frac{1}{2k}\mathbf{p}(j)$ .
- For  $1 \leq \ell \leq k$  and  $j \in I_{k,k+\ell}$ ,  $\mathbf{c}(j) = \frac{1}{2k}\mathbf{q}(j)$ .



■ **Figure 2** An example of  $\mathbf{c}$  (top),  $\mathbf{f}$  (middle), and  $\mathbf{r}$  (bottom) over  $[2k^2]$ , for  $k = 5$ ; here, we took  $\mathbf{p} = \frac{1}{16}(3, 2, 6, 4, 1)$  and  $\mathbf{q} = \frac{1}{18}(2, 5, 4, 4, 3)$ .

We obtain  $\mathbf{f}$  over  $[2k^2]$  in a similar fashion, but swapping  $I_{k,\ell}$  and  $I_{k,k+\ell}$ :

- For  $1 \leq \ell \leq k$  and  $j \in I_{k,\ell}$ ,  $\mathbf{f}(j) = \frac{1}{2k}\mathbf{q}(j)$ .
- For  $1 \leq \ell \leq k$  and  $j \in I_{k,k+\ell}$ ,  $\mathbf{f}(j) = \frac{1}{2k}\mathbf{p}(j)$ .

Further, we define our “reference” distribution  $\mathbf{r}$  over  $[2k^2]$  as

- For  $1 \leq \ell \leq k$  and  $j \in I_{k,\ell}$ ,  $\mathbf{r}(j) = \frac{1}{2k}\mathbf{p}(\ell)$ .
- For  $k + 1 \leq \ell \leq 2k$  and  $j \in I_{k,\ell}$ ,  $\mathbf{r}(j) = \frac{1}{2k}\mathbf{q}(\ell)$ .



We also define the reference distribution  $\mathbf{r}_{\mathbf{p},\mathbf{q}}^*$  over  $[n] = [2k^2m]$  by concatenating  $m$  copies of  $\mathbf{r}$  and normalizing the result; that is,

$$\mathbf{r}_{\mathbf{p},\mathbf{q}}^* := \frac{1}{2m}(\mathbf{r} \sqcup \mathbf{r} \sqcup \cdots \sqcup \mathbf{r}),$$

where  $\sqcup$  denotes the vector concatenation. Note that both  $\mathbf{c}$  and  $\mathbf{f}$  are permutations of  $\mathbf{r}$ , and that  $\|\mathbf{r}\|_1 = \|\mathbf{c}\|_1 = \|\mathbf{f}\|_1 = 1$ . Next, we bound the gap between  $d_{\text{TV}}(\mathbf{f}, \mathbf{r})$  and  $d_{\text{TV}}(\mathbf{c}, \mathbf{r})$ , relating it to the distance between  $\mathbf{p}$  and  $\mathbf{q}$ .

▷ **Claim 11.**  $d_{\text{TV}}(\mathbf{f}, \mathbf{r}) \geq d_{\text{TV}}(\mathbf{c}, \mathbf{r}) + \frac{1}{k}d_{\text{TV}}(\mathbf{p}, \mathbf{q})$

*Proof.* We will analyze the contributions to  $d_{\text{TV}}(\mathbf{c}, \mathbf{r})$  and  $d_{\text{TV}}(\mathbf{c}, \mathbf{f})$  on  $I_{k,\ell}$  and  $I_{k,k+\ell}$  for  $1 \leq \ell \leq k$ . Without loss of generality, we can assume that  $\mathbf{p}, \mathbf{q}$  are non-decreasing. Then, from our definition of  $\mathbf{c}, \mathbf{r}$ , and  $\mathbf{f}$ , we have

$$\begin{aligned} d_{\text{TV}}(\mathbf{f}, \mathbf{r}) &= \frac{1}{4k} \sum_{i=1}^k \sum_{j=1}^k (|\mathbf{p}(i) - \mathbf{q}(j)| + |\mathbf{q}(i) - \mathbf{p}(j)|) = \frac{1}{2k} \left( \sum_{i=1}^k \sum_{j=1}^k |\mathbf{p}(i) - \mathbf{q}(j)| \right) \\ &= \frac{1}{2k} \left( \sum_{i=1}^k |\mathbf{p}(i) - \mathbf{q}(i)| + \sum_{i=1}^k \sum_{j=1}^{i-1} (|\mathbf{p}(i) - \mathbf{q}(j)| + |\mathbf{p}(j) - \mathbf{q}(i)|) \right) \\ d_{\text{TV}}(\mathbf{c}, \mathbf{r}) &= \frac{1}{4k} \sum_{i=1}^k \sum_{j=1}^k (|\mathbf{p}(i) - \mathbf{p}(j)| + |\mathbf{q}(i) - \mathbf{q}(j)|) \\ &= \frac{1}{2k} \sum_{i=1}^k \sum_{j=1}^{i-1} ((\mathbf{p}(i) - \mathbf{p}(j)) + (\mathbf{q}(i) - \mathbf{q}(j))) \end{aligned}$$

where for the last equality we used the assumption that  $\mathbf{p}, \mathbf{q}$  were non-decreasing to write

$$\sum_{i=1}^k \sum_{j=1}^k |\mathbf{p}(i) - \mathbf{p}(j)| = \sum_{i=1}^k \sum_{j=1}^{i-1} (\mathbf{p}(i) - \mathbf{p}(j)) + \sum_{i=1}^k \sum_{j=i+1}^k (\mathbf{p}(j) - \mathbf{p}(i)) = 2 \sum_{i=1}^k \sum_{j=1}^{i-1} (\mathbf{p}(i) - \mathbf{p}(j)),$$

The conclusion then follows from recalling that  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) = \frac{1}{2} \sum_{i=1}^k |\mathbf{p}(i) - \mathbf{q}(i)|$ , and observing that  $(\mathbf{p}(i) - \mathbf{p}(j)) + (\mathbf{q}(i) - \mathbf{q}(j)) = (\mathbf{p}(i) - \mathbf{q}(j)) + (\mathbf{q}(i) - \mathbf{p}(j)) \leq |\mathbf{p}(i) - \mathbf{q}(j)| + |\mathbf{p}(j) - \mathbf{q}(i)|$ . ◁

To define  $\mathcal{C}_{\mathbf{p},\mathbf{q}}$  and  $\mathcal{F}_{\mathbf{p},\mathbf{q}}$ , we will need one further piece of notation. We denote by  $\mathcal{B}_k \subseteq \mathcal{S}_{2k^2}$  the set of all permutations of  $[2k^2]$  “respecting the buckets,” that is,

$$\mathcal{B}_k := \{ \pi \in \mathcal{S}_{2k^2} : \pi(I_{k,\ell}) = I_{k,\ell} \forall \ell \in [2k] \}$$

We then let

$$\mathcal{C}_{\mathbf{p},\mathbf{q}} = \left\{ \frac{1}{2mk} (\mathbf{c} \circ \pi_1 \sqcup \mathbf{c} \circ \pi_2 \sqcup \cdots \sqcup \mathbf{c} \circ \pi_m) : \pi_1, \dots, \pi_m \in \mathcal{B}_k \right\}$$

and

$$\mathcal{F}_{\mathbf{p},\mathbf{q}} = \left\{ \frac{1}{2mk} (\mathbf{f} \circ \pi_1 \sqcup \mathbf{f} \circ \pi_2 \sqcup \cdots \sqcup \mathbf{f} \circ \pi_m) : \pi_1, \dots, \pi_m \in \mathcal{B}_k \right\}$$

where as before  $\sqcup$  denotes the vector concatenation; that is, we stitch together  $m$  blocks, each consisting on a permuted version of either  $\mathbf{c}$  or  $\mathbf{f}$ . Note that since  $n = m \cdot 2k^2$  and each  $\mathbf{c}$  (resp.  $\mathbf{f}$ ) is a  $(2k^2)$ -dimensional vector,  $\mathcal{C}_{\mathbf{p},\mathbf{q}}$  and  $\mathcal{F}_{\mathbf{p},\mathbf{q}}$  are indeed families of probability distributions over  $[n]$ , and  $\mathcal{C}_{\mathbf{p},\mathbf{q}}, \mathcal{F}_{\mathbf{p},\mathbf{q}} \subseteq \Pi_n(\mathbf{r}_{\mathbf{p},\mathbf{q}}^*)$ .

The construction above allows us to convert any two distributions  $\mathbf{p}, \mathbf{q}$  with sufficiently many matching moments to families of distributions (whose elements are all permutations of a single reference one) hard to distinguish:

## 55:12 Identity Testing Under Label Mismatch

▷ **Claim 12.** There exists some absolute constant  $c > 0$  such that, if  $\mathbf{p}, \mathbf{q}$  have matching first  $r$ -way moments, it is impossible to distinguish a uniformly random element of  $\mathcal{C}_{\mathbf{p}, \mathbf{q}}$  from a uniformly random element of  $\mathcal{F}_{\mathbf{p}, \mathbf{q}}$  given fewer than  $cm^{1-\frac{1}{r+1}}$  samples.

*Proof.* By assumption on  $\mathbf{p}, \mathbf{q}$  and our construction of  $\mathbf{c}, \mathbf{f}$  from them, for every of the  $m$  contiguous blocks of  $2k^2$  elements, the  $r$ -way moments of the corresponding conditional distributions exactly match. Given that a uniformly element drawn of  $\mathbf{p}'$  from  $\mathcal{C}_{\mathbf{p}, \mathbf{q}}$  and  $\mathbf{q}'$  from  $\mathcal{F}_{\mathbf{p}, \mathbf{q}}$  corresponds to independent permutations inside each block, any block in which fewer than  $r + 1$  samples falls brings exactly zero information about whether it comes from  $\mathbf{p}'$  or  $\mathbf{q}'$  (specifically, one could simulate the distribution of those  $s < r + 1$  samples without getting any sample from the real distribution). Since each of these  $m$  blocks has total probability  $1/m$  under both  $\mathbf{p}'$  and  $\mathbf{q}'$ , by a generalized birthday paradox (see, e.g., [19]), with probability at least  $9/10$  no block will receive more than  $r$  samples unless the total number of samples is at least  $cm^{1-\frac{1}{r+1}}$ , for some absolute constant  $c > 0$ . ◁

It remains to specify *which* pair of distributions with “sufficiently many matching moments” we will use. While we could argue directly about the existence of such a pair of distributions with desirable properties, it is simpler to leverage a construction due to Valiant and Valiant [22], which exhibits the desired properties.

▷ **Claim 13.** There exists some  $\varepsilon_0 > 0$  such that the following holds. For every sufficiently large  $r$ , there exists a pair of distributions (without loss of generality, non-decreasing)  $\mathbf{p}_{VV}, \mathbf{q}_{VV}$  over  $k = O(r2^r)$  elements with matching first  $r$ -way moments, but  $d_{TV}(\mathbf{p}_{VV}, \mathbf{q}_{VV}) \geq \varepsilon_0$ .

*Proof.* This follows from the lower bound construction of [22]. ◁

We will rely on this pair of distributions  $\mathbf{p}_{VV}, \mathbf{q}_{VV}$ , and hereafter write  $\mathcal{C}, \mathcal{F}$ , and  $\mathbf{r}^*$  for  $\mathcal{C}_{\mathbf{p}_{VV}, \mathbf{q}_{VV}}, \mathcal{F}_{\mathbf{p}_{VV}, \mathbf{q}_{VV}}$ , and  $\mathbf{r}_{\mathbf{p}_{VV}, \mathbf{q}_{VV}}^*$ , respectively.

▷ **Claim 14.** For every  $\mathbf{p}' \in \mathcal{C}$  and  $\mathbf{q}' \in \mathcal{F}$ , we have  $d_{TV}(\mathbf{q}', \mathbf{r}^*) > d_{TV}(\mathbf{p}', \mathbf{r}^*) + \frac{\varepsilon_0}{k}$ .

*Proof.* Due to the definition of  $\mathcal{C}, \mathcal{F}$ , and  $\mathbf{r}^*$  as  $m$ -fold concatenations, and since  $\mathbf{r}$  is invariant by permutations from  $\mathcal{B}_k$ , it is sufficient to prove the claim for  $\mathbf{p}_{VV}, \mathbf{q}_{VV}$ , and  $\mathbf{r}$  (over  $[2k^2]$ ). The claimed bound then immediately follows from Claim 11. ◁

To finish the argument, it only remains to combine the various claims. We choose  $k \geq \frac{\varepsilon_0}{\delta}$  and  $m = n/(2k^2) \geq 1$  (since  $\delta = \Omega(1/\sqrt{n})$ ). By Claim 13, we can then set  $r := \Omega(\log k)$  and obtain, from Claim 12, a sample complexity lower bound of

$$\Omega\left(m^{1-\frac{1}{r+1}}\right) = \Omega\left(\delta^2 n^{1-O\left(\frac{1}{\log(1/\delta)}\right)}\right)$$

as desired. ◀

The theorem immediately implies the following two corollaries.

► **Corollary 15.** For every  $c > 0$ , there exists some  $\delta > 0$  such that the following holds. Any algorithm which, given a reference distribution  $\mathbf{q}$  over  $[n]$ ,  $\varepsilon \in (0, 1)$ , and sample access to an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least  $2/3$  between (i)  $d_{TV}(\mathbf{p}, \mathbf{q}) \leq \varepsilon$  and (ii)  $d_{TV}(\mathbf{p}, \mathbf{q}) > \varepsilon + \delta$ , must have sample complexity  $\Omega(n^{1-c})$ .

► **Corollary 16.** Any algorithm which, given a reference distribution  $\mathbf{q}$  over  $[n]$ ,  $\varepsilon \in (0, 1)$ , and sample access to an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least  $2/3$  between (i)  $d_{TV}(\mathbf{p}, \mathbf{q}) \leq \varepsilon$  and (ii)  $d_{TV}(\mathbf{p}, \mathbf{q}) > \varepsilon + 1/2\sqrt{\log n}$ , must have sample complexity  $\frac{n}{2^{O(\sqrt{\log n})}}$ .

## Tolerant testing $C$ -approximation

We now turn to our second tolerant testing lower bound, which applies to algorithms providing a  $C$ -factor approximation of the distance to the reference distribution.

► **Theorem 17.** *Any algorithm which, given a reference distribution  $\mathbf{q}$  over  $[n]$ ,  $C \geq 2$ , and sample access to an unknown distribution  $\mathbf{p} \in \Pi_n(\mathbf{q})$ , distinguishes with probability at least  $2/3$  between (i)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) \leq \frac{1}{4^{C-1}}$  and (ii)  $d_{\text{TV}}(\mathbf{p}, \mathbf{q}) \geq \frac{C}{4^{C-1}}$ , must have sample complexity  $\Omega(\sqrt{\frac{n}{4^C}})$ .*

► **Remark 18.** As discussed in Remark 7, Theorem 17 is essentially optimal, as it matches (up to polylogarithmic factors in the sample complexity) the upper bound from Theorem 5 when  $C = \Theta(\log n)$ .

**Proof of Theorem 17.** We will prove the theorem via a sequence of lemmas. We will assume that  $C \geq 2$  is an integer, and we define  $m = 2^C - 1$ . Our proof will proceed similarly to the proof of Theorem 10. We will begin by working over  $[m(2^{C+1} + 2^{C-1} - 3)]$ . Throughout this section, we partition  $[m(2^{C+1} + 2^{C-1} - 3)]$  into  $C + 1$  buckets, which we will denote  $B_0, B_1, \dots, B_C$ , such that each  $B_i$  is a set of consecutive integers,  $|B_C| = m2^{C-1}$ ,  $|B_0| = m$ , and  $|B_i| = m2^{i+1}$  for  $1 \leq i \leq C - 1$ . For convenience, we define  $s := m(4C - 1)2^{C-1}$ .

We define a distribution  $\mathbf{r}$  in the following way:

- For each  $j \in B_0$ ,  $\mathbf{r}(j) = \frac{2^C}{s}$ .
- For each  $1 \leq i \leq C - 1$  and  $j \in B_i$ ,  $\mathbf{r}(j) = \frac{2^{C-i}}{s}$ .
- For each  $j \in B_C$ ,  $\mathbf{r}(j) = \frac{1}{s}$ .

We define two distributions  $\mathbf{p}$  and  $\mathbf{q}$  such that  $\mathbf{p}$  and  $\mathbf{q}$  are hard to distinguish with few samples, such that  $d_{\text{TV}}(\mathbf{r}, \mathbf{p})$  and  $d_{\text{TV}}(\mathbf{r}, \mathbf{q})$  are far apart. We define  $\mathbf{q}$  in the following way:

- For each  $j \in B_0$ ,  $\mathbf{q}(j) = \frac{2^{C-1}}{s}$ .
- For each  $1 \leq i \leq C - 1$ ,
  - For  $j$  in the first  $m2^i$  elements of  $B_i$ ,  $\mathbf{q}(j) = \frac{2^{C-i-1}}{s}$ .
  - For  $j$  in the next  $m2^{i-1}$  elements of  $B_i$ ,  $\mathbf{q}(j) = \frac{2^{C-i}}{s}$ .
  - For  $j$  in the last  $m2^{i-1}$  elements of  $B_i$ ,  $\mathbf{q}(j) = \frac{2^{C-i+1}}{s}$ .
- For each  $j \in B_C$ ,  $\mathbf{q}(j) = \frac{2}{s}$ .

We define  $\mathbf{p}$  as follows:

- For each  $j \in B_0$ ,
  - If  $j$  is in the first  $2^{C-1}$  elements of  $B_0$ , then  $\mathbf{p}(j) = \frac{1}{s}$ .
  - If  $j$  is in the last  $m - 2^{C-1} = 2^{C-1} - 1$  elements of  $B_0$ , then  $\mathbf{p}(j) = \frac{2^C}{s}$ .
- For each  $1 \leq i \leq C - 1$  and  $j \in B_i$ ,  $\mathbf{p}(j) = \mathbf{r}(j) = \frac{2^{C-i}}{s}$ .
- For each  $j \in B_C$ ,
  - If  $j$  is in the first  $(m - 1)2^{C-1}$  elements of  $B_C$ , then  $\mathbf{p}(j) = \frac{1}{s}$ .
  - If  $j$  is in the last  $2^{C-1}$  elements of  $B_j$ , then  $\mathbf{p}(j) = \frac{2^C}{s}$ .

► **Lemma 19.** *For  $0 \leq i \leq C$ ,  $\sum_{j \in B_i} \mathbf{p}(j) = \sum_{j \in B_i} \mathbf{q}(j)$ .*

**Proof.** The proof is simply direct calculation. Observe that in bucket  $C$ ,

$$\begin{aligned} s \sum_{j \in B_C} \mathbf{q}(j) &= m2^{C-1} \cdot 2 = (m - 1)2^{C-1} + (m + 1)2^{C-1} \\ &= (m - 1)2^{C-1} \cdot 1 + 2^C \cdot 2^{C-1} = s \sum_{j \in B_C} \mathbf{p}(j). \end{aligned}$$

## 55:14 Identity Testing Under Label Mismatch

In bucket 0, we have

$$\begin{aligned} s \sum_{j \in B_0} \mathbf{q}(j) &= m \cdot 2^{C-1} = (m-1)2^{C-1} + 2^{C-1} = (2^C - 2)2^{C-1} + 2^{C-1} \\ &= (2^{C-1} - 1) \cdot 2^C + 2^{C-1} \cdot 1 = s \sum_{j \in B_0} \mathbf{p}(j). \end{aligned}$$

For  $1 \leq i \leq C-1$ , we have

$$\begin{aligned} s \sum_{j \in B_i} \mathbf{p}(j) &= m2^{i+1} \cdot 2^{C-i} \\ &= m(2^i + 2(2^{i-1}) + 2^{i+1})2^{C-1-i} \\ &= m2^i \cdot 2^{C-i-1} + m2^{i-1} \cdot 2^{C-i} + m2^{i-1} \cdot 2^{C-i+1} \\ &= s \sum_{j \in B_i} \mathbf{q}(j). \end{aligned}$$

The claim follows by dividing the equalities by  $s$ . ◀

► **Lemma 20.**  $d_{\text{TV}}(\mathbf{r}, \mathbf{q}) = \frac{C}{4C-1}$

**Proof.** By direct calculation,

$$\begin{aligned} 2s d_{\text{TV}}(\mathbf{r}, \mathbf{q}) &= s \sum_{j=1}^s |\mathbf{r}(j) - \mathbf{q}(j)| \\ &= m2^{C-1}(2-1) + m(2^C - 2^{C-1}) \\ &\quad + \frac{1}{2} \sum_{i=1}^{C-1} (m2^i(2^{C-i} - 2^{C-i-1}) + m2^{i-1}(2^{C-i+1} - 2^{C-i})) \\ &= m2^C + \sum_{i=1}^{C-1} (2^{i-1}m2^{C-i} + m2^{C-1-i}2^i) \\ &= m2^C + \sum_{i=1}^{C-1} (m2^{C-1} + m2^{C-1}) \\ &= Cm2^C. \end{aligned}$$

Dividing both sides by  $2s$  yields the lemma. ◀

► **Lemma 21.** For every  $0 \leq i \leq C$ ,  $\mathbf{p}(B_i) \leq \frac{2}{C+1}$  (and similarly for  $\mathbf{q}(B_i)$ ).

**Proof.** We apply Lemma 19 and directly calculate. For bucket  $C$ , we get

$$\mathbf{p}(B_C) = \mathbf{q}(B_C) = \frac{2}{s} \cdot m2^C - 1 = \frac{2}{4C-1}.$$

For bucket 0, we get

$$\mathbf{p}(B_0) = \mathbf{q}(B_0) = \frac{2^{C-1}}{s} \cdot m = \frac{1}{4C-1}.$$

For  $1 \leq i \leq C-1$ , we get

$$\mathbf{q}(B_i) = \mathbf{p}(B_i) = \frac{2^i}{s} \cdot m(2^{C+1-i}) = \frac{4}{4C-1}.$$

The claim follows by observing that  $\frac{4}{4C-1} \leq \frac{2}{C+1}$  when  $C \geq \frac{3}{2}$ . ◀

► **Lemma 22.**  $d_{\text{TV}}(\mathbf{r}, \mathbf{p}) = \frac{1}{4^C - 1}$

**Proof.** By direct calculation,

$$2s d_{\text{TV}}(\mathbf{r}, \mathbf{p}) = 2^{C-1} \cdot (2^C - 1) + 2^{C-1} \cdot (2^C - 1) = 2^C (2^C - 1) = m 2^C.$$

Dividing both sides by  $2s$  yields the lemma. ◀

Let  $w = m(2^{C+1} + 2^{C-1} - 3)$ . We assume that  $n$  is a multiple of  $w$ , and define  $t := \frac{n}{w}$ . To define  $\mathcal{C}$  and  $\mathcal{F}$  over  $[n]$ , we will need one further piece of notation. We denote by  $\mathcal{B}'_w \subseteq \mathcal{S}_w$  the set of all permutations of  $[w]$  “respecting the buckets,” that is, for every  $0 \leq i \leq C$ ,

$$\mathcal{B}'_w = \{\pi \in \mathcal{S}_w : \pi(B_i) = B_i \forall i \in \{0, 1, \dots, C\}\}$$

We then let  $\mathbf{r}^* := \frac{1}{t}(\mathbf{r} \sqcup \mathbf{r} \sqcup \dots \sqcup \mathbf{r})$  as well as

$$\mathcal{C} = \left\{ \frac{1}{t}(\mathbf{c} \circ \pi_1 \sqcup \mathbf{c} \circ \pi_2 \sqcup \dots \sqcup \mathbf{c} \circ \pi_t) : \pi_1, \dots, \pi_t \in \mathcal{B}'_w \right\}$$

$$\mathcal{F} = \left\{ \frac{1}{t}(\mathbf{f} \circ \pi_1 \sqcup \mathbf{f} \circ \pi_2 \sqcup \dots \sqcup \mathbf{f} \circ \pi_t) : \pi_1, \dots, \pi_t \in \mathcal{B}'_w \right\}$$

where as before  $\sqcup$  denotes vector concatenation. Since  $d_{\text{TV}}(\mathbf{r}, \mathbf{c} \circ \pi) = d_{\text{TV}}(\mathbf{r}, \mathbf{c})$  and  $d_{\text{TV}}(\mathbf{r}, \mathbf{f} \circ \pi) = d_{\text{TV}}(\mathbf{r}, \mathbf{f})$  for all  $\pi \in \mathcal{B}'_w$ , we have that  $d_{\text{TV}}(\mathbf{r}^*, \mathbf{p}) = \frac{1}{4^C - 1}$  for every distribution  $\mathbf{p} \in \mathcal{C}$ , and  $d_{\text{TV}}(\mathbf{r}^*, \mathbf{q}) = \frac{1}{4^C - 1}$  for every distribution  $\mathbf{q} \in \mathcal{F}$ . Further, repeating the same partitioning of each interval of  $s$  elements of  $[n]$  into buckets  $B_0, B_1, \dots, B_C$ , we have  $t(C+1)$  buckets, such that distinguishing a distribution in  $\mathcal{C}$  from a distribution in  $\mathcal{F}$  requires seeing at 2 samples in at least one of these buckets. Since the probability mass on each of the buckets is at most  $\frac{2}{t(C+1)}$  by Lemma 21, at least  $\Omega(\sqrt{t(C+1)}) = \Omega(\sqrt{n(C+1)/w})$  queries to distinguish in  $\mathcal{C}$  from a distribution in  $\mathcal{F}$ , completing the proof of Theorem 17. ◀

---

## References

- 1 Jayadev Acharya, Constantinos Daskalakis, and Gautam Kamath. Optimal testing for properties of distributions. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3591–3599, 2015. URL: <https://proceedings.neurips.cc/paper/2015/hash/1f36c15d6a3d18d52e8d493bc8187cb9-Abstract.html>.
- 2 Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing that distributions are close. In *41st Annual Symposium on Foundations of Computer Science (Redondo Beach, CA, 2000)*, pages 259–269. IEEE Comput. Soc. Press, Los Alamitos, CA, 2000. doi:10.1109/SFCS.2000.892113.
- 3 Tuğkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *Symposium on Theory of Computing Conference, STOC'04*, pages 381–390, New York, NY, USA, 2004. ACM. doi:10.1145/1007352.1007414.
- 4 Eric Blais, Clément L. Canonne, and Tom Gur. Distribution testing lower bounds via reductions from communication complexity. *ACM Trans. Comput. Theory*, 11(2):Art. 6, 37, 2019. doi:10.1145/3305270.
- 5 Clément L. Canonne. *A Survey on Distribution Testing: Your Data is Big. But is it Blue?* Number 9 in Graduate Surveys. Theory of Computing Library, 2020. doi:10.4086/toc.gs.2020.009.

- 6 Clément L. Canonne and Karl Wimmer. Testing data binnings. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference, volume 176 of *LIPICs*, pages 24:1–24:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.APPROX/RANDOM.2020.24.
- 7 Siu-on Chan, Ilias Diakonikolas, Paul Valiant, and Gregory Valiant. Optimal algorithms for testing closeness of discrete distributions. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1193–1203. SIAM, 2014. doi:10.1137/1.9781611973402.88.
- 8 Constantinos Daskalakis, Ilias Diakonikolas, Rocco A. Servedio, Gregory Valiant, and Paul Valiant. Testing  $k$ -modal distributions: Optimal algorithms via reductions. In *Proceedings of SODA*, pages 1833–1852. Society for Industrial and Applied Mathematics (SIAM), 2013. URL: <http://dl.acm.org/citation.cfm?id=2627817.2627948>.
- 9 Ilias Diakonikolas, Themis Gouleakis, John Peebles, and Eric Price. Sample-optimal identity testing with high probability. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPICs. Leibniz Int. Proc. Inform.*, pages Art. No. 41, 14. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.
- 10 Ilias Diakonikolas, Daniel M. Kane, and Vladimir Nikishkin. Testing identity of structured distributions. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1841–1854. SIAM, Philadelphia, PA, 2015. doi:10.1137/1.9781611973730.123.
- 11 Aryeh Dvoretzky, Jack Kiefer, and Jacob Wolfowitz. Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. *Ann. Math. Statist.*, 27:642–669, 1956. doi:10.1214/aoms/1177728174.
- 12 Karl Pearson F.R.S. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900. doi:10.1080/14786440009463897.
- 13 Oded Goldreich. The uniform distribution is complete with respect to testing identity to a fixed distribution. In *Computational Complexity and Property Testing*, volume 12050 of *Lecture Notes in Computer Science*, pages 152–172. Springer, 2020.
- 14 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, July 1998.
- 15 Dayu Huang and Sean Meyn. Generalized error exponents for small sample universal hypothesis testing. *IEEE Trans. Inform. Theory*, 59(12):8157–8181, 2013. doi:10.1109/TIT.2013.2283266.
- 16 Jiantao Jiao, Yanjun Han, and Tsachy Weissman. Minimax estimation of the  $L_1$  distance. *IEEE Trans. Inf. Theory*, 64(10):6672–6706, 2018.
- 17 Pascal Massart. The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality. *Ann. Probab.*, 18(3):1269–1283, 1990.
- 18 Liam Paninski. A coincidence-based test for uniformity given very sparsely sampled discrete data. *IEEE Trans. Inform. Theory*, 54(10):4750–4755, 2008. doi:10.1109/TIT.2008.928987.
- 19 Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. Birthday paradox for multi-collisions. In *Information security and cryptology – ICISC 2006*, volume 4296 of *Lecture Notes in Comput. Sci.*, pages 29–40. Springer, Berlin, 2006. doi:10.1007/11927587\_5.
- 20 Gregory Valiant and Paul Valiant. A CLT and tight lower bounds for estimating entropy. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:179, 2010. URL: <http://eccc.hpi-web.de/report/2010/179>.
- 21 Gregory Valiant and Paul Valiant. Estimating the unseen: A sublinear-sample canonical estimator of distributions. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:180, 2010. URL: <http://eccc.hpi-web.de/report/2010/180>.

- 22 Gregory Valiant and Paul Valiant. The power of linear estimators. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science – FOCS 2011*, pages 403–412. IEEE Computer Soc., Los Alamitos, CA, 2011. doi:10.1109/FOCS.2011.81.
- 23 Gregory Valiant and Paul Valiant. An automatic inequality prover and instance optimal identity testing. *SIAM J. Comput.*, 46(1):429–455, 2017. doi:10.1137/151002526.





# Unique-Neighbor-Like Expansion and Group-Independent Cystolic Expansion

Tali Kaufman ✉

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

David Mass ✉

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

---

## Abstract

---

In recent years, high dimensional expanders have been found to have a variety of applications in theoretical computer science, such as efficient CSPs approximations, improved sampling and list-decoding algorithms, and more. Within that, an important high dimensional expansion notion is *cosystolic expansion*, which has found applications in the construction of efficiently decodable quantum codes and in proving lower bounds for CSPs.

Cosystolic expansion is considered with systems of equations over a group where the variables and equations correspond to faces of the complex. Previous works that studied cosystolic expansion were tailored to the specific group  $\mathbb{F}_2$ . In particular, Kaufman, Kazhdan and Lubotzky (FOCS 2014), and Evra and Kaufman (STOC 2016) in their breakthrough works, who solved a famous open question of Gromov, have studied a notion which we term “parity” expansion for small sets. They showed that small sets of  $k$ -faces have proportionally many  $(k + 1)$ -faces that contain *an odd number* of  $k$ -faces from the set. Parity expansion for small sets could then be used to imply cosystolic expansion only over  $\mathbb{F}_2$ .

In this work we introduce a stronger *unique-neighbor-like* expansion for small sets. We show that small sets of  $k$ -faces have proportionally many  $(k + 1)$ -faces that contain *exactly one*  $k$ -face from the set. This notion is fundamentally stronger than parity expansion and cannot be implied by previous works.

We then show, utilizing the new unique-neighbor-like expansion notion introduced in this work, that cosystolic expansion can be made *group-independent*, i.e., unique-neighbor-like expansion for small sets implies cosystolic expansion *over any group*.

**2012 ACM Subject Classification** Theory of computation → Computational complexity and cryptography

**Keywords and phrases** High dimensional expanders, Unique-neighbor-like expansion, Cosystolic expansion

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.56

**Funding** *Tali Kaufman*: Supported by ERC and BSF.

*David Mass*: Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

## 1 Introduction

### High dimensional expanders

High dimensional expanders are the high dimensional analog of expander graphs. A  $d$ -dimensional simplicial complex is a hypergraph with hyperedges of size at most  $d + 1$  which is downwards closed, i.e., if  $\sigma$  is an hyperedge and  $\tau \subset \sigma$  then  $\tau$  is also an hyperedge. A hyperedge of size  $k + 1$  is called a  $k$ -face of the complex.

In recent years, high dimensional expanders have found a variety of applications in theoretical computer science, such as efficient CSPs approximations [2], improved sampling algorithms [5, 4, 3, 8, 7, 6, 16], improved list-decoding algorithms [11, 1], sparse agreement tests [12, 9, 19] and more.



© Tali Kaufman and David Mass;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 56; pp. 56:1–56:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An especially important high dimensional expansion notion is *cosystolic expansion*. It has been shown to be a key ingredient in the construction of efficiently decodable quantum LDPC codes with a large distance [15], and recently it has been used in the construction of explicit 3XOR instances that are hard for the Sum-of-Squares hierarchy [10].

### Cosystolic expansion as an expanding system of equations

A simplicial complex can be viewed as forming a *system of equations* over some group  $G$ . Consider a  $d$ -dimensional simplicial complex and some dimension  $k < d$ . The variables of the system are the  $k$ -faces of the complex, and the equations are defined by the  $(k+1)$ -faces; each  $(k+1)$ -face  $\sigma$  corresponds to the equation  $\sum_{i=0}^{k+1} \tau_i = 0$ , where  $\tau_i$  are the  $k$ -faces contained in  $\sigma$  and the sum is performed over the group (e.g., addition modulo 2 when the group is  $\mathbb{F}_2$ ).

For any assignment of values to the variables which does not satisfy all the equations, there are two measures of interest. One measure is the fraction of unsatisfied equations (out of all the equations), and the second measure is the fraction of variables (out of all the variables) that their value needs to be changed in order to satisfy all the equations. The second measure is also called the *distance* of the assignment from a satisfying assignment.

A system of equations is said to be *expanding* if for any assignment of values to the variables it holds that either all the equations are satisfied or the fraction of unsatisfied equations is proportional to the distance of the assignment from a satisfying assignment. A  $d$ -dimensional simplicial complex is said to be a *cosystolic expander* over a group  $G$  if for all  $k < d$ , the system of equations formed by its  $k$ -faces is expanding.

As a simple example, consider a 1-dimensional simplicial complex (i.e., a graph) and the field  $\mathbb{F}_2$ . The variables of the system are the vertices of the graph, and the equations are  $v_i + v_j = 0 \pmod{2}$  for each edge  $\{v_i, v_j\}$ . In this case, if the given graph is an expander graph (i.e., each subset of vertices has proportionally many outgoing edges), then the system of equations is expanding. This is true since each assignment of values over  $\mathbb{F}_2$  to the vertices can be identified with a subset of vertices, and the unsatisfied equations are exactly the outgoing edges of this set.

### Parity expansion for small sets

Kaufman, Kazhdan and Lubotzky [18], and Evra and Kaufman [14] in their breakthrough works proved the existence of cosystolic expanders of every dimension, solving a famous open question of Gromov [17]. In their works they have studied a notion which we term “parity” expansion for small sets: They have shown that certain high dimensional expansion properties imply that small sets of  $k$ -faces have proportionally many  $(k+1)$ -faces that contain *an odd number* of  $k$ -faces from the given set. Then they utilized this property in order to imply cosystolic expansion over the group  $\mathbb{F}_2$ .

### $\delta_1$ -expansion for small sets

In this work we study a fundamentally stronger “unique-neighbor-like” expansion in simplicial complexes, which we call  $\delta_1$ -*expansion*. Let  $X$  be a  $d$ -dimensional simplicial complex and  $A$  a set of  $k$ -faces in  $X$ . We define  $\delta_1(A)$  to be the set of  $(k+1)$ -faces which contain *exactly one*  $k$ -face from  $A$ . We say that  $A$  is  $\delta_1$ -*expanding* if the fraction of  $(k+1)$ -faces in  $\delta_1(A)$  (out of all the  $(k+1)$ -faces) is proportional to the fraction of  $k$ -faces in  $A$  (out of all the  $k$ -faces). Our main result is that certain high dimensional expansion properties imply that small sets are  $\delta_1$ -expanding.

### $\delta_1$ -expansion and group-independent cosystolic expansion

The strength of our  $\delta_1$ -expansion can be demonstrated by its relation to cosystolic expansion. As explained above, cosystolic expansion is considered with a system of equations over a group. Hence, when proving cosystolic expansion, one has to take the group into account. For instance, previous works could obtain cosystolic expansion only over  $\mathbb{F}_2$ , because only over  $\mathbb{F}_2$  there is an equivalence between an unsatisfied equation and an equation that contains an odd number of non-zero variables.

The  $\delta_1$ -expansion property that we study in this work has the interesting property that it can make cosystolic expansion to be *group-independent*, i.e., it implies cosystolic expansion over any group. The key point is that an equation with exactly one non-zero variable must be unsatisfied *regardless of the group*. Thus, even though cosystolic expansion is defined over a group,  $\delta_1$ -expansion implies it over any group.

We expect that this stronger  $\delta_1$ -expansion notion may have further implications to quantum codes and CSPs lower bounds.

### On the novelty of our work

We would like to provide a general outline of the differences between our work and previous works [18, 14].<sup>1</sup>

One fundamental difference is the object we analyze. The major part of previous works is dedicated to the analysis of the expansion of *arbitrary small sets*. In our work, the main analysis is focused on the expansion of “structured” *small sets* (given by the coboundary of a small set). We use a similar machinery as in previous works, but we leverage the extra structure of the small sets in order to obtain the stronger  $\delta_1$ -expansion.

We note that it is not trivial how to utilize this extra structure of small sets in order to obtain  $\delta_1$ -expansion. One cannot just plug it in the proof of previous works and obtain  $\delta_1$ -expansion. It requires a completely different proof strategy, which we describe next.

Briefly, the proof strategy of [18] and [14] is as follows. Given a small set  $A$  of  $k$ -faces, they define a notion of “fat” faces, where an  $\ell$ -face,  $\ell \leq k$ , is considered fat if a large fraction of the  $k$ -faces that contain it belongs to  $A$ . It is trivial that in dimension  $\ell = k$ , the set  $A$  sits only on fat  $k$ -faces, since every  $k$ -face that belongs to  $A$  is fat (because the only  $k$ -face that contains a  $k$ -face is itself). It is also trivial that in dimension  $\ell = -1$ ,  $A$  sits only on thin  $(-1)$ -faces, since the only  $(-1)$ -face is the empty set which is contained in all of the  $k$ -faces of the complex, and  $A$  is a small set of  $k$ -faces. Therefore, there must exist a dimension  $\ell \leq k$  for which a transition from mostly fat faces to mostly thin faces occurs, i.e., in dimension  $\ell$ ,  $A$  sits mostly on fat  $\ell$ -faces, and in dimension  $\ell - 1$ ,  $A$  sits mostly on thin  $(\ell - 1)$ -faces. Their argument is then that the fat  $\ell$ -faces contribute to the parity expansion of  $A$ , whereas the thin  $(\ell - 1)$ -faces account for a negligible error term.

The proof strategy in our work is essentially the opposite. A fat face, which contributes to the parity expansion in the works of [18] and [14], does not have a large  $\delta_1$ , and hence it is impossible to obtain  $\delta_1$ -expansion from the fat faces. Our main idea is to gain the  $\delta_1$ -expansion *from the thin faces*. We observe that if a set sits mostly on thin faces of *one dimension below* then it has a large  $\delta_1$ . Thus, it is crucial for us to show that a small set  $A$

---

<sup>1</sup> Informal note: We are aware that the following explanation might seem not entirely clear at this stage of the paper for an unfamiliar reader. Nevertheless, since we have been repeatedly asked for the differences between our work and previous works, it is important for us to point that out as early as possible. We hope that the main ideas are still clear, even if not all the details are.

of  $k$ -faces actually sits mostly on thin faces of one dimension below. Using the terminology of previous paragraph, we have to show that the transition from mostly fat faces to mostly thin faces happens in dimension  $k$  itself.

This is where the “structure” comes into play. By considering small sets that are obtained as a coboundary of another set, we know that their own coboundary is 0. Without getting too much into the details, it allows us to bound the fraction of fat faces of dimension  $\ell$  by the fraction of fat faces of dimension  $\ell - 1$ , for every  $0 \leq \ell \leq k - 1$ . Thus, since there are no fat faces in dimension  $-1$  (because  $A$  is small), we conclude that there are no fat faces at any dimension! Therefore,  $A$  sits mostly on thin  $(k - 1)$ -faces and hence has a large  $\delta_1$ .

### 1.1 Some basic definitions

#### Coboundary and cosystolic expansion

For the sake of introduction we formally define coboundary and cosystolic expansions only over the field  $\mathbb{F}_2$ . The general definitions will be given in section 2.

Recall that a  $d$ -dimensional simplicial complex  $X$  is a downwards closed  $(d+1)$ -hypergraph. A  $k$ -face of  $X$  is a hyperedge of size  $k + 1$ , and the set of  $k$ -faces of  $X$  is denoted by  $X(k)$ . An assignment of values from  $\mathbb{F}_2$  to the  $k$ -faces,  $k \leq d$ , is called a  $k$ -cochain, and the space of all  $k$ -cochains over  $\mathbb{F}_2$  is denoted by  $C^k(X; \mathbb{F}_2)$ .

Any assignment to the  $k$ -faces  $f \in C^k(X; \mathbb{F}_2)$  induces an assignment to the  $(k + 1)$ -faces by the coboundary operator  $\delta$ . For any  $(k + 1)$ -face  $\sigma = \{v_0, \dots, v_{k+1}\}$ ,  $\delta(f)(\sigma)$  is defined by

$$\delta(f)(\sigma) = \sum_{i=0}^{k+1} f(\sigma \setminus \{v_i\}) \pmod{2}.$$

We can view the complex as inducing a system of equations, where the equations are determined by the coboundary operator; i.e., each  $(k + 1)$ -face  $\sigma \in X(k + 1)$  defines the equation  $\delta(f)(\sigma) = 0$ . The assignments that satisfy all the equations are called the  $k$ -cocycles and denoted by

$$Z^k(X; \mathbb{F}_2) = \{f \in C^k(X; \mathbb{F}_2) \mid \delta(f) = \mathbf{0}\}.$$

One can check that  $\delta(\delta(f)) = \mathbf{0}$  always holds; i.e., every assignment that is obtained as a coboundary of one dimension below satisfies all the equations. These assignments, that are the coboundary of an assignment of one dimension below, are called the  $k$ -coboundaries and denoted by

$$B^k(X; \mathbb{F}_2) = \{\delta(f) \mid f \in C^{k-1}(X; \mathbb{F}_2)\}.$$

Note that  $B^k(X; \mathbb{F}_2) \subseteq Z^k(X; \mathbb{F}_2) \subseteq C^k(X; \mathbb{F}_2)$ .

For a  $d$ -dimensional simplicial complex  $X$ , let  $P_d : X(d) \rightarrow \mathbb{R}_{\geq 0}$  be a probability distribution over the  $d$ -faces of the complex. For simplicity, we will assume in this work that  $P_d$  is the uniform distribution. This probability distribution over the  $d$ -faces induces a probability distribution  $P_k$  for every dimension  $k < d$  by selecting a  $d$ -face  $\sigma_d$  according to  $P_d$  and then selecting a  $k$ -face  $\sigma_k \subset \sigma_d$  uniformly at random.

The weight of any  $k$ -cochain  $f \in C^k(X; \mathbb{F}_2)$  is defined by

$$\|f\| = \Pr_{\sigma \sim P_k} [f(\sigma) \neq 0],$$

i.e., the (weighted) fraction of non-zero elements in  $f$ . The distance between two  $k$ -cochains  $f, g \in C^k(X; \mathbb{F}_2)$  is defined as  $\text{dist}(f, g) = \|f - g\|$ .

We can now introduce the notions of coboundary and cosystolic expansion. As mentioned, a complex is said to be a cosystolic expander if for any assignment that does not satisfy all the equations it holds that the fraction of unsatisfied equations is proportional to the distance of the assignment from a satisfying assignment. Formally:

► **Definition 1.1** (Cosystolic expansion). *A  $d$ -dimensional simplicial complex  $X$  is said to be an  $(\varepsilon, \mu)$ -cosystolic expander over  $\mathbb{F}_2$ , if for every  $k < d$ :*

1. *For any  $f \in C^k(X; \mathbb{F}_2) \setminus Z^k(X; \mathbb{F}_2)$  it holds that*

$$\frac{\|\delta(f)\|}{\text{dist}(f, Z^k(X; \mathbb{F}_2))} \geq \varepsilon,$$

where  $\text{dist}(f, Z^k(X; \mathbb{F}_2)) = \min\{\text{dist}(f, g) \mid g \in Z^k(X; \mathbb{F}_2)\}$ .

2. *For any  $f \in Z^k(X; \mathbb{F}_2) \setminus B^k(X; \mathbb{F}_2)$  it holds that  $\|f\| \geq \mu$ .*

The second condition in the definition ensures that the complex cannot be split into many small pieces, i.e., any satisfying assignment that is not obtained as a coboundary must be large.

Coboundary expansion has been introduced by Linial and Meshulam [20] and independently by Gromov [17]. It is a similar but stronger notion than cosystolic expansion. The main difference is that the only satisfying assignments in a coboundary expander are coboundaries (unlike cosystolic expansion, where there could be satisfying assignments which are not coboundaries as long as they are large). Formally:

► **Definition 1.2** (Coboundary expansion). *A  $d$ -dimensional simplicial complex  $X$  is said to be an  $\varepsilon$ -coboundary expander over  $\mathbb{F}_2$  if for every  $k < d$  and  $f \in C^k(X; \mathbb{F}_2) \setminus B^k(X; \mathbb{F}_2)$  it holds that*

$$\frac{\|\delta(f)\|}{\text{dist}(f, B^k(X; \mathbb{F}_2))} \geq \varepsilon,$$

where  $\text{dist}(f, B^k(X; \mathbb{F}_2)) = \min\{\text{dist}(f, g) \mid g \in B^k(X; \mathbb{F}_2)\}$ .

### Local spectral expansion

Another notion of high dimensional expansion, called *local spectral expansion* is concerned with the spectral properties of local parts of the complex.

For every face  $\sigma \in X$ , its local view, also called its *link*, is a  $(d - |\sigma| - 1)$ -dimensional simplicial complex defined by  $X_\sigma = \{\tau \setminus \sigma \mid \sigma \subseteq \tau \in X\}$ . The probability distribution over the top faces of  $X_\sigma$  is induced from the probability distribution of  $X$ , where for any top face  $\tau \in X_\sigma(d - |\sigma| - 1)$ , its probability is the probability to choose  $\sigma \cup \tau$  in  $X$  conditioned on choosing  $\sigma$ . Since we assume in this work that the probability distribution over the top faces of  $X$  is the uniform distribution, it follows that the probability distribution over the top faces of  $X_\sigma$  is the uniform distribution.

We can now introduce the notion of a local spectral expander.

► **Definition 1.3** (Local spectral expansion). *A  $d$ -dimensional simplicial complex  $X$  is called a  $\lambda$ -local spectral expander if for every  $k \leq d - 2$  and  $\sigma \in X(k)$ , the underlying graph<sup>2</sup> of  $X_\sigma$  is a  $\lambda$ -spectral expander.*

<sup>2</sup> The graph whose vertices are  $X_\sigma(0)$  and its edges are  $X_\sigma(1)$ .

## 1.2 Summary of main results

Our main result is a “unique-neighbor-like” expansion for non-local small sets, which we call  $\delta_1$ -expansion. We start with the definition of the  $\delta_1$  of a set.

► **Definition 1.4** ( $\delta_1$ ). *Let  $X$  be a  $d$ -dimensional simplicial complex. For any set of  $k$ -faces  $A \subseteq X(k)$ ,  $0 \leq k \leq d-1$ , we define  $\delta_1(A) \subseteq X(k+1)$  to be the set of  $(k+1)$ -faces that contain exactly one  $k$ -face from  $A$ .*

Towards proving that small sets have a large  $\delta_1$  we introduce an intermediate notion of *non-local sets*. Roughly speaking, we say that a set of  $k$ -faces is non-local if its “local view” in almost all of the  $(k-1)$ -faces resemble the global picture.

In order to define this notion of non-local sets, we first define the *localization* of a set to a link of a face. For any set  $A \subseteq X(k)$  and an  $\ell$ -face  $\sigma \in X(\ell)$ ,  $\ell < k$ , the localization of  $A$  to the link of  $\sigma$  is a set of  $(k-\ell-1)$ -faces in the link of  $\sigma$  defined by

$$A_\sigma = \{\tau \in X_\sigma(k-\ell-1) \mid \sigma \cup \tau \in A\}.$$

We also add a useful definition of a mutual weight of two sets. For  $\ell < k$  and two sets  $A \subseteq X(k), B \subseteq X(\ell)$  we define their mutual weight by

$$\|(A, B)\| = \Pr_{\sigma_k \sim P_k, \sigma_\ell \subset \sigma_k} [\sigma_k \in A \wedge \sigma_\ell \in B],$$

where  $\sigma_k$  is chosen according to the distribution  $P_k$  and  $\sigma_\ell$  is an  $\ell$ -face chosen uniformly from  $\sigma_k$  (i.e.,  $\sigma_\ell$  is chosen according to  $P_\ell$  conditioned on  $\sigma_k$  being chosen). This notion captures how much the sets are related. For instance, if  $\|(A, B)\| \approx \|A\|$ , it means that  $A$  contains mostly faces from  $B$ .

We can now define non-local sets.

► **Definition 1.5** (Non-local sets). *Let  $X$  be a  $d$ -dimensional simplicial complex and  $0 < \eta, \varepsilon < 1$ . For any set of  $k$ -faces  $A \subseteq X(k)$ ,  $0 \leq k \leq d-1$ , we define the following set of  $(k-1)$ -faces:*

$$S_{k-1} = \{\sigma \in X(k-1) \mid \|A_\sigma\| \leq \eta\}.$$

*We say that  $A$  is  $(\eta, \varepsilon)$ -non-local if  $\|(A, S_{k-1})\| \geq (1-\varepsilon)\|A\|$ .*

As a simple example of a “local” set, consider a set of edges  $A$  composed of all the edges touching a single vertex. In this case,  $\|(A, S_0)\| = (1/2)\|A\|$ . It can be easily checked that in this example, all triangles contain either 0 or 2 edges. As can be seen from this example, local sets are not necessarily  $\delta_1$ -expanding.

The first theorem we show is that non-local sets are  $\delta_1$ -expanding.

► **Theorem 1.6** (Non-local sets are  $\delta_1$ -expanding - informal). *Let  $X$  be a  $d$ -dimensional local spectral expander. For any  $A \subseteq X(k)$ ,  $1 \leq k \leq d-1$ , if  $A$  is non-local then  $\|\delta_1(A)\| \geq \Omega(\|A\|)$ .*

We consider now a bounded degree local spectral expander whose links are coboundary expanders, where a complex is said to be  $q$ -bounded degree if every vertex is contained in at most  $q$  top faces. We show that every set of unsatisfied equations can be treated as if it is non-local. Specifically, we consider sets of the form  $\text{supp}(\delta(f))$  for a  $k$ -cochain  $f \in C^k(X; G)$ , over some group  $G$ . We show a procedure that is given a  $k$ -cochain  $f$  such that  $\|\delta(f)\|$  is small, and returns a  $k$ -cochain  $f'$  which is close to  $f$  such that  $\delta(f')$  is non-local.



► **Theorem 1.7** (Correction algorithm – informal). *Let  $X$  be a  $d$ -dimensional bounded degree local spectral expander with coboundary expanding links over a group  $G$ . For any  $f \in C^k(X; G)$ ,  $1 \leq k \leq d - 2$ , if  $\|\delta(f)\|$  is sufficiently small, then  $f$  is close to a  $k$ -cochain  $f' \in C^k(X; G)$  such that  $\delta(f')$  is small and non-local. Furthermore, there is an efficient algorithm that is given  $f$  and finds  $f'$ .*

We conclude by a similar reduction as in [18] in order to obtain cosystolic expansion over any group. [18] and [14] could obtain cosystolic expansion only over  $\mathbb{F}_2$  because their expansion for small sets only guaranteed that they touch many faces of one dimension above an odd number of times. Since we show here  $\delta_1$ -expansion for such sets, we obtain cosystolic expansion which does not depend on the group.

► **Theorem 1.8** (Cosystolic expansion over any group – informal). *Let  $X$  be a  $d$ -dimensional bounded degree local spectral expander with coboundary expanding links over a group  $G$ . Then the  $(d - 1)$ -skeleton<sup>3</sup> of  $X$  is a cosystolic expander over  $G$ .*

A concrete example of simplicial complexes for which our theorems apply to are the famous Ramanujan complexes [25, 24], which are the high dimensional analog of the celebrated LPS Ramanujan graphs [23]. These complexes are local spectral expanders [14] and their links, called spherical buildings, are coboundary expanders [22]. We note that [22] proved that spherical buildings are coboundary expanders only over  $\mathbb{F}_2$ , but their proof can be easily generalized to any abelian group by considering localizations with orientations of  $k$ -cochains. As for non-abelian groups, [13] proved that spherical buildings are coboundary expanders over non-abelian groups as well. For more on Ramanujan complexes, we refer the reader to [21].

► **Corollary 1.9** (Ramanujan complexes are cosystolic expanders over any group). *Let  $X$  be a  $d$ -dimensional Ramanujan complex. If  $X$  is sufficiently thick<sup>4</sup>, then the  $(d - 1)$ -skeleton of  $X$  is a cosystolic expander over any group  $G$ .*

### 1.3 Organization

In section 2 we provide some required preliminaries. In section 3 we prove the  $\delta_1$ -expansion and cosystolic expansion results over abelian groups. In section 4 we provide the definitions for cochains over non-abelian groups and we repeat the same process as in section 3, but this time for non-abelian groups. The general strategy is the same for abelian and non-abelian groups, but the details are different, hence we split them into different sections.

## 2 Preliminaries

### Coboundary and cosystolic expansion over abelian groups

Let  $X$  be a  $d$ -dimensional simplicial complex and  $G$  an abelian group<sup>5</sup>. We first consider an ordered version of the complex and denote it by  $\vec{X}$ , where

$$\vec{X} = \{(v_0, \dots, v_k) \mid k \leq d, \{v_0, \dots, v_k\} \in X\},$$

i.e.,  $\vec{X}$  contains all possible orderings of every face in  $X$ .

<sup>3</sup> The complex which contains the faces of  $X$  up to dimension  $d - 1$ .

<sup>4</sup> The explanation of the “thickness” of a Ramanujan complex is out of scope of this paper. It is only important for us that a Ramanujan complex can be made arbitrarily thick in order to satisfy the required criteria.

<sup>5</sup> For simplicity we deal here only with abelian groups. We discuss non-abelian groups in section 4.

A  $k$ -cochain over  $G$ ,  $k \leq d$ , is an antisymmetric function  $f : \vec{X}(k) \rightarrow G$ , where  $f$  is said to be antisymmetric if for any permutation  $\pi \in \text{Sym}(k+1)$ ,

$$f((v_{\pi(0)}, v_{\pi(1)}, \dots, v_{\pi(k)})) = \text{sgn}(\pi) f((v_0, v_1, \dots, v_k)).$$

The space of all  $k$ -cochains over  $G$  is denoted by  $C^k(X; G)$ .

Any  $k$ -cochain is an assignment to the  $k$ -faces and it induces a  $(k+1)$ -cochain, i.e., an assignment to the  $(k+1)$ -faces, by the coboundary operator  $\delta$ . For any ordered  $(k+1)$ -face  $\sigma = (v_0, \dots, v_{k+1})$ ,  $\delta(f)(\sigma)$  is defined by

$$\delta(f)(\sigma) = \sum_{i=0}^{k+1} (-1)^i f(v_0, \dots, v_{i-1}, v_{i+1}, \dots, v_{k+1}),$$

where the sum is performed over the group. It is not hard to check that for every  $k$  and  $f \in C^k(X; G)$ ,  $\delta(f)$  is antisymmetric, i.e., a  $(k+1)$ -cochain.

We can view the complex as inducing a system of equations, where the equations are determined by the coboundary operator; i.e., each  $(k+1)$ -face  $\sigma \in X(k+1)$  defines the equation  $\delta(f)(\sigma) = 0$  (note that the ordering of the face does not matter for the satisfaction of the equation). The assignments that satisfy all the equations are called the  $k$ -cocycles and denoted by

$$Z^k(X; G) = \{f \in C^k(X; G) \mid \delta(f) = \mathbf{0}\}.$$

One can check that  $\delta(\delta(f)) = \mathbf{0}$  always holds; i.e., every assignment that is obtained as a coboundary of one dimension below satisfies all the equations. These assignments, that are the coboundary of an assignment of one dimension below, are called the  $k$ -coboundaries and denoted by

$$B^k(X; G) = \{\delta(f) \mid f \in C^{k-1}(X; G)\}.$$

Note that  $B^k(X; G) \subseteq Z^k(X; G) \subseteq C^k(X; G)$ .

Recall that the weight of a  $k$ -cochain  $f \in C^k(X; G)$  is defined by

$$\|f\| = \Pr_{\sigma \sim P_k} [f(\sigma) \neq 0],$$

i.e., the (weighted) fraction of non-zero elements in  $f$ . Since the weight of a cochain is dependent only on its non-zero elements, it is often convenient to consider the set  $\text{supp}(f)$  (i.e., the set of non-zero elements in  $f$ ) and define equivalently

$$\|f\| = \|\text{supp}(f)\| = \Pr_{\sigma \sim P_k} [\sigma \in \text{supp}(f)].$$

For simplicity, we might abuse the notation and write  $\sigma \in f$  where we mean that  $\sigma \in \text{supp}(f)$ .

Now, the formal definitions of coboundary and cosystolic expansion over general abelian groups are identical to definitions 1.1 and 1.2 given in the introduction, with the replacement of  $\mathbb{F}_2$  by an abelian group  $G$ .

### Links and localization

Recall that the link of a  $k$ -face  $\sigma \in X(k)$  is a  $(d-k-1)$ -dimensional complex defined by  $X_\sigma = \{\tau \setminus \sigma \mid \sigma \subseteq \tau \in X\}$ , where the probability distribution over faces of  $X_\sigma$  is induced from the probability distribution over faces of  $X$ . Since we assume in this work that  $P_d$  is

the uniform distribution over the  $d$ -faces of  $X$ , it follows that the probability distribution over the top faces of  $X_\sigma$  is the uniform distribution. In the rest of the paper, we will omit the explicit probability distribution when it is clear from the context.

Recall also that cochains over abelian groups are defined on ordered faces of the complex. For convenience sake, we fix an arbitrary ordering of the faces so that for any face  $\sigma \in X$  there is a unique corresponding ordered face  $\vec{\sigma} \in \vec{X}$ .

For two disjoint ordered faces  $\vec{\sigma} = (v_0, \dots, v_k)$  and  $\vec{\tau} = (u_0, \dots, u_\ell)$  we denote their concatenation by  $\vec{\sigma\tau} = (v_0, \dots, v_k, u_0, \dots, u_\ell)$ . For any  $k$ -face  $\sigma \in X(k)$  and a  $(k + \ell + 1)$ -cochain  $f \in C^{k+\ell+1}(X; G)$ , the *localization* of  $f$  to the link of  $\sigma$  is an  $\ell$ -cochain in the link of  $\sigma$ ,  $f_\sigma \in C^\ell(X_\sigma; G)$  defined as follows. For any ordered  $\ell$ -face  $\vec{\tau} \in \vec{X}_\sigma(\ell)$ ,  $f_\sigma(\vec{\tau}) = f(\vec{\sigma\tau})$ , where  $\vec{\sigma\tau}$  is the concatenation of  $\vec{\sigma}$  (i.e., the unique corresponding ordered face of  $\sigma$ ) and  $\vec{\tau}$ .

By the law of total probability, the weight of any  $k$ -cochain  $f \in C^k(X; G)$  can be decomposed as a sum of its weight in the links of the  $\ell$ -faces of the complex:

► **Lemma 2.1.** *Let  $X$  be a  $d$ -dimensional simplicial complex and  $G$  an abelian group. For every  $f \in C^k(X; G)$ ,  $k \leq d$  and  $\ell < k$ ,*

$$\|f\| = \sum_{\tau \in X(\ell)} \|(f, \tau)\|,$$

where  $\|(f, \tau)\|$  is the mutual weight of  $\text{supp}(f) \subseteq X(k)$  and  $\{\tau\} \subseteq X(\ell)$ .

**Proof.** It follows immediately from the definitions:

$$\|f\| = \Pr_{\sigma \sim P_k} [\sigma \in \text{supp}(f)] = \sum_{\tau \in X(\ell)} \Pr_{\sigma \sim P_k, \tau' \subset \sigma} [\sigma \in \text{supp}(f) \wedge \tau' = \tau] = \sum_{\tau \in X(\ell)} \|(f, \tau)\|,$$

where the second equality follows from the law of total probability. ◀

### Minimal and locally minimal cochains

One of the technical notions we use in this work is the notion of a minimal cochain. We say that a  $k$ -cochain  $f \in C^k(X; G)$  is *minimal* if its weight cannot be reduced by adding a coboundary to it, i.e., for every  $g \in B^k(X; G)$  it holds that  $\|f\| \leq \|f - g\|$ . Recall that the distance of  $f$  from the coboundaries is defined by  $\text{dist}(f, B^k(X; G)) = \min\{\text{dist}(f, g) \mid g \in B^k(X; G)\}$ . Since  $\mathbf{0} \in B^k(X; G)$ , it follows that for every  $f \in C^k(X; G)$ ,  $\|f\| \geq \text{dist}(f, B^k(X; G))$ . Hence,  $f$  is minimal if and only if  $\|f\| = \text{dist}(f, B^k(X; G))$ .

We also define the notion of a locally minimal cochain, where we say that  $f \in C^k(X; G)$  is *locally minimal* if for every vertex  $v$ , the localization of  $f$  to the link of  $v$  is minimal in the link, i.e.,  $f_v$  is minimal in  $X_v$  for every  $v \in X(0)$ .

### Cheeger inequality for graphs

A 1-dimensional simplicial complex  $X$  is just a graph. In this case the known Cheeger inequality gives the following (see e.g. [18] for a proof):

► **Lemma 2.2.** *Let  $X$  be a 1-dimensional simplicial complex which is a  $\lambda$ -spectral expander graph. For any set of vertices  $A \subseteq X(0)$  it holds that*

1.  $\|E(A, \bar{A})\| \geq 2(1 - \lambda) \|A\| \|\bar{A}\|,$
2.  $\|E(A)\| \leq (\|A\| + \lambda) \|A\|,$

where  $E(A, \bar{A})$  is the set of edges with one endpoint in  $A$  and one endpoint in  $\bar{A}$ , and  $E(A)$  is the set of edges with both endpoints in  $A$ .

### 3 Result for abelian groups

#### 3.1 Non-local sets are $\delta_1$ -expanding

In this section we show our results for abelian groups.

Our first theorem is that non-local sets in a local spectral expander have  $\delta_1$  that is proportional to their size. We prove Theorem 1.6 which we restate here in a formal way.

► **Theorem 3.1** (Non-local sets are  $\delta_1$ -expanding). *Let  $X$  be a  $d$ -dimensional  $\lambda$ -local spectral expander and  $0 < \eta, \varepsilon < 1$ . For any  $A \subseteq X(k)$ ,  $0 \leq k \leq d-1$ , such that  $A$  is  $(\eta, \varepsilon)$ -non-local it holds that*

$$\|\delta_1(A)\| \geq \left(1 - \binom{k+2}{k} (\lambda + \eta + 2\varepsilon)\right) \|A\|.$$

**Proof.** Recall that we denote by  $S_{k-1}$  the set of  $(k-1)$ -faces  $\sigma \in X(k-1)$  satisfying  $\|A_\sigma\| \leq \eta$ . Let us define the following sets of  $(k+1)$ -faces:

- $\Gamma(A) = \{\tau \in X(k+1) \mid \exists \sigma \in A \text{ s.t. } \sigma \subset \tau\}$ .
- $\Gamma(A, \overline{S_{k-1}}) = \{\tau \in X(k+1) \mid \exists \sigma \in A, \sigma' \in \overline{S_{k-1}} \text{ s.t. } \sigma' \subset \sigma \subset \tau\}$ .
- $\Upsilon = \{\tau \in X(k+1) \mid \exists \sigma, \sigma' \in A \text{ s.t. } \sigma, \sigma' \subset \tau, \sigma \cap \sigma' \in S_{k-1}\}$ .

In words:  $\Gamma(A)$  is the set of all  $(k+1)$ -faces that contain a  $k$ -face from  $A$ ,  $\Gamma(A, \overline{S_{k-1}})$  is the set of all  $(k+1)$ -faces that contain a  $k$ -face from  $A$  which contains a  $(k-1)$ -face from  $\overline{S_{k-1}} = X(k-1) \setminus S_{k-1}$ , and  $\Upsilon$  is the set of all  $(k+1)$ -faces that contain two  $k$ -faces from  $A$  such that their intersection is a  $(k-1)$ -face from  $S_{k-1}$ .

Note that for every  $\tau \in \Gamma(A) \setminus \Gamma(A, \overline{S_{k-1}})$  one of the following cases must hold: Either  $\tau$  contains exactly one  $k$ -face from  $A$ , i.e.,  $\tau \in \delta_1(A)$ , or  $\tau$  contains at least two  $k$ -faces from  $A$  such that their intersection belongs to  $S_{k-1}$ , i.e.,  $\tau \in \Upsilon$ . It follows that

$$\|\delta_1(A)\| \geq \|\Gamma(A) \setminus (\Gamma(A, \overline{S_{k-1}}) \cup \Upsilon)\| \geq \|\Gamma(A)\| - \|\Gamma(A, \overline{S_{k-1}})\| - \|\Upsilon\|. \quad (1)$$

Let us bound each of the above terms separately. First, by simple laws of probability

$$\begin{aligned} \|\Gamma(A)\| &= \Pr[\sigma_{k+1} \in \Gamma(A)] \\ &\geq \Pr[\sigma_{k+1} \in \Gamma(A) \wedge \sigma_k \in A] \\ &= \Pr[\sigma_k \in A] \cdot \Pr[\sigma_{k+1} \in \Gamma(A) \mid \sigma_k \in A] \\ &= \Pr[\sigma_k \in A] = \|A\|. \end{aligned} \quad (2)$$

Second, again by laws of probability

$$\begin{aligned} \|\Gamma(A, \overline{S_{k-1}})\| &= \Pr[\sigma_{k+1} \in \Gamma(A, \overline{S_{k-1}})] \\ &= \frac{\Pr[\sigma_{k+1} \in \Gamma(A, \overline{S_{k-1}}) \wedge \sigma_k \in A \wedge \sigma_{k-1} \notin S_{k-1}]}{\Pr[\sigma_k \in A \wedge \sigma_{k-1} \notin S_{k-1} \mid \sigma_{k+1} \in \Gamma(A, \overline{S_{k-1}})]} \\ &\leq (k+2)(k+1) \Pr[\sigma_k \in A \wedge \sigma_{k-1} \notin S_{k-1}] \\ &= (k+2)(k+1) \|(A, \overline{S_{k-1}})\| \\ &\leq (k+2)(k+1)\varepsilon \|A\|, \end{aligned} \quad (3)$$

where the first inequality holds since the probability that  $\sigma_k \in A$  and  $\sigma_{k-1} \notin S_{k-1}$  given that  $\sigma_{k+1} \in \Gamma(A, \overline{S_{k-1}})$  is at least  $1/((k+2)(k+1))$ , and the second inequality follows since  $A$  is an  $(\eta, \varepsilon)$ -non-local set.

Lastly, consider a  $(k + 1)$ -face  $\tau \in \Upsilon$ . By definition,  $\tau$  contains two  $k$ -faces  $\sigma, \sigma' \in A$  such that  $\sigma \cap \sigma' \in S_{k-1}$ . Let us denote  $\tilde{\tau} = \sigma \cap \sigma'$ . Note that  $\tau$  is seen in the link of  $\tilde{\tau}$  as an edge between two vertices in  $A_{\tilde{\tau}}$ , i.e.,  $\tau \setminus \tilde{\tau} \in E(A_{\tilde{\tau}})$ . Thus,

$$\begin{aligned}
 \|\Upsilon\| &= \sum_{\tau \in \Upsilon} \Pr[\sigma_{k+1} = \tau] \\
 &= \sum_{\tau \in \Upsilon} \frac{\Pr[\sigma_{k+1} = \tau \wedge \sigma_{k-1} = \tilde{\tau}]}{\Pr[\sigma_{k-1} = \tilde{\tau} \mid \sigma_{k+1} = \tau]} \\
 &\leq \sum_{\tau \in \Upsilon} \binom{k+2}{k} \Pr[\sigma_{k+1} = \tau \mid \sigma_{k-1} = \tilde{\tau}] \cdot \Pr[\sigma_{k-1} = \tilde{\tau}] \\
 &\leq \binom{k+2}{k} \sum_{\sigma \in S_{k-1}} \|E(A_\sigma)\| \cdot \Pr[\sigma_{k-1} = \sigma] \\
 &\leq \binom{k+2}{k} \sum_{\sigma \in S_{k-1}} (\|A_\sigma\| + \lambda) \|A_\sigma\| \cdot \Pr[\sigma_{k-1} = \sigma] \\
 &\leq \binom{k+2}{k} (\eta + \lambda) \sum_{\sigma \in S_{k-1}} \|(A_\sigma, \sigma)\| \leq \binom{k+2}{k} (\eta + \lambda) \|A\|,
 \end{aligned} \tag{4}$$

where the third inequality follows since  $X$  is a  $\lambda$ -local spectral expander, and the fourth inequality follows since  $\sigma \in S_{k-1}$ .

Substituting (2), (3) and (4) in (1) finishes the proof.  $\blacktriangleleft$

An immediate corollary of Theorem 3.1 is that any non-local cocycle must be zero.

**► Corollary 3.2 (Non-local cocycles vanish).** *For any  $d \in \mathbb{N}$ , an abelian group  $G$ , and  $0 < \lambda, \eta, \varepsilon < 1$  such that  $\lambda + \eta + 2\varepsilon \leq 2/(d + 1)^2$  the following holds: Let  $X$  be a  $d$ -dimensional  $\lambda$ -local spectral expander. For any  $f \in Z^k(X; G)$ ,  $0 \leq k \leq d - 1$ , if  $f$  is  $(\eta, \varepsilon)$ -non-local then  $f = \mathbf{0}$ .*

**Proof.** Since  $f$  is  $(\eta, \varepsilon)$ -non-local, then by Theorem 3.1 it holds that

$$\|\delta_1(f)\| \geq \frac{\|f\|}{d + 1}.$$

On the other hand,  $f \in Z^k(X; G)$  and hence  $\|\delta_1(f)\| \leq \|\delta(f)\| = 0$ . It follows that  $f = \mathbf{0}$  as required.  $\blacktriangleleft$

### 3.2 The correction procedure

Our aim now is to show a correction procedure for small coboundaries. We show an algorithm that gets a cochain  $f$  such that  $\|\delta(f)\|$  is small and returns a cochain  $f'$  by making a few changes to  $f$  such that  $\delta(f')$  is non-local.

We start by showing that any small and locally minimal cocycle is non-local. We note that the following proposition is the main technical contribution of our work.

**► Proposition 3.3 (Small and locally minimal cocycles are non-local).** *For any  $d \in \mathbb{N}$ , an abelian group  $G$ , and  $0 < \beta, \varepsilon < 1$  there exist  $0 < \lambda, \eta \leq \varepsilon$  such that the following holds: Let  $X$  be a  $d$ -dimensional  $\lambda$ -local spectral expander with  $\beta$ -coboundary expanding links. For any  $f \in Z^k(X; G)$ ,  $1 \leq k \leq d - 1$ , if  $\|f\| \leq \eta^{2^{k+1}-1}$  and locally minimal then  $f$  is  $(\eta, \varepsilon)$ -non-local.*

## 56:12 Unique-Neighbor-Like Expansion and Group-Independent Cosystolic Expansion

In order to prove Proposition 3.3 we need a few more definitions and lemmas. The proofs of the lemmas can be found in the full version of the paper. Let  $f \in C^k(X; G)$ ,  $0 \leq k \leq d-1$ . Recall that  $S_{k-1}$  is the set of  $(k-1)$ -faces  $\sigma$  satisfying  $\|f_\sigma\| \leq \eta$ . For any  $-1 \leq i \leq k-2$ , we define the following set of  $i$ -faces:

$$S_i = \{\sigma \in X(i) \mid (\overline{S_{i+1}})_\sigma \leq \eta^{2^{k-i-1}}\}.$$

We first show that if  $\|f\|$  is sufficiently small then  $\|S_{-1}\| = 1$ , i.e., the empty set belongs to  $S_{-1}$ . We will use the following lemma:

► **Lemma 3.4.** *Let  $X$  be a  $d$ -dimensional simplicial complex,  $G$  an abelian group and  $0 < \eta < 1$ . For any  $f \in C^k(X; G)$ ,  $0 \leq k \leq d$ , if  $\|f\| \leq \eta^{2^{k+1}-1}$  then  $\|S_{-1}\| = 1$ .*

Let  $\Upsilon \subseteq X(k+1)$  be the set of  $(k+1)$ -faces which contain two  $i$ -faces  $\sigma, \sigma' \in \overline{S_i}$  such that  $\sigma \cap \sigma' \in S_{i-1}$ . We show that  $\|\Upsilon\|$  is a negligible fraction of  $\|f\|$ .

► **Lemma 3.5.** *Let  $X$  be a  $d$ -dimensional  $\lambda$ -local spectral expander,  $G$  an abelian group and  $0 < \eta < 1$  such that  $\lambda \leq \eta^{2^{d-1}}$ . For any  $f \in C^k(X; G)$ ,  $0 \leq k \leq d-1$ , it holds that*

$$\|\Upsilon\| \leq \eta \binom{k+2}{2} 2^{k+2} \|f\|.$$

For any  $\sigma \in X(i)$ , denote by  $f \downarrow \sigma$  the set of  $k$ -faces  $\tau \in f$  which have a sequence of containments of faces from  $\overline{S_j}$ ,  $i < j < k$ , down to  $\sigma$ . Formally,

$$f \downarrow \sigma = \{\tau \in f \mid \exists \tau_{k-1} \in \overline{S_{k-1}}, \dots, \tau_{i+1} \in \overline{S_{i+1}} \text{ s.t. } \tau \supset \tau_{k-1} \supset \dots \supset \tau_{i+1} \supset \sigma\}.$$

We show that for any cocycle  $f \in Z^k(X; G)$  and  $0 \leq i \leq k-1$ , the fraction of  $f$  that sits on  $i$ -faces from  $\overline{S_i}$  is approximately the fraction of  $f$  that sits on  $(i-1)$ -faces from  $\overline{S_{i-1}}$ .

► **Lemma 3.6.** *Let  $X$  be a  $d$ -dimensional simplicial complex such that its links are  $\beta$ -coboundary expanders over an abelian group  $G$ . For any locally minimal  $f \in Z^k(X; G)$ ,  $1 \leq k \leq d-1$ , and  $0 \leq i \leq k-1$  it holds that*

$$\sum_{\sigma \in \overline{S_i}} \|(f \downarrow \sigma, \sigma)\| \leq \frac{1}{\beta} \left( (k+1-i)(i+1) \sum_{\sigma' \in \overline{S_{i-1}}} \|(f \downarrow \sigma', \sigma')\| + \|\Upsilon\| \right).$$

We can now prove Proposition 3.3.

**Proof of Proposition 3.3.** Let

$$\eta = \frac{\beta^{d-1} \varepsilon}{2^d ((d+1)!)^2} \quad \text{and} \quad \lambda = \eta^{2^{d-1}}.$$

Applying Lemma 3.6 on dimension  $i = k-1, k-2, \dots, 0$  step after step yields

$$\begin{aligned} \|(f, \overline{S_{k-1}})\| &= \sum_{\sigma \in \overline{S_{k-1}}} \|(f \downarrow \sigma, \sigma)\| \\ &\leq \frac{1}{\beta} \|\Upsilon\| + \frac{1}{\beta} \cdot 2 \cdot k \sum_{\sigma \in \overline{S_{k-2}}} \|(f \downarrow \sigma, \sigma)\| \\ &\leq \frac{1}{\beta} \|\Upsilon\| + \frac{1}{\beta^2} \cdot 2 \cdot k \|\Upsilon\| + \frac{1}{\beta^2} \cdot 2 \cdot k \cdot 3 \cdot (k-1) \sum_{\sigma \in \overline{S_{k-3}}} \|(f \downarrow \sigma, \sigma)\| \\ &\leq \left( \frac{1}{\beta} + \frac{1}{\beta^2} \cdot 2 \cdot k + \dots + \frac{1}{\beta^k} (k!)^2 \right) \|\Upsilon\| + \frac{1}{\beta^k} (k!)^2 (k+1) \sum_{\sigma \in \overline{S_{-1}}} \|(f \downarrow \sigma, \sigma)\| \\ &= \left( \frac{1}{\beta} + \frac{1}{\beta^2} \cdot 2 \cdot k + \dots + \frac{1}{\beta^k} (k!)^2 \right) \|\Upsilon\| \leq k \beta^{-k} (k!)^2 \|\Upsilon\| \end{aligned} \tag{5}$$

Substituting Lemma 3.5 in (5) completes the proof. ◀

Now, the idea of the correction algorithm is to make  $\delta(f)$  locally minimal by correcting  $f$  in a few local parts. The algorithm runs in iterations, where at every iteration it does the following one step of correction.

► **Lemma 3.7** (One step of correction). *Let  $X$  be a  $d$ -dimensional simplicial complex and  $G$  an abelian group. For any  $f \in C^k(X; G)$ ,  $1 \leq k \leq d - 1$ , if  $f$  is not locally minimal then there exists a vertex  $v \in X(0)$  and a  $(k - 1)$ -cochain  $g \in C^{k-1}(X; G)$  such that  $\|g\| \leq k \|v\|$  and  $\|f - \delta(g)\| < \|f\|$ .*

**Proof.** Since  $f$  is not locally minimal, there exists a vertex  $v \in X(0)$  such that  $f_v$  is not minimal in  $X_v$ . By definition there exists a  $(k - 2)$ -cochain  $h \in C^{k-2}(X_v; G)$  in the link of  $v$  such that  $\|f_v - \delta(h)\| < \|f_v\|$ . Define  $g \in C^{k-1}(X; G)$  by

$$g(\sigma) = \begin{cases} h(\tau) & \sigma = v\tau, \\ 0 & \text{otherwise.} \end{cases}$$

Note that  $g_v = h$ , therefore  $\|f - \delta(g)\| < \|f\|$ . Furthermore, since  $g(\sigma) = 0$  for every  $\sigma$  which does not contain  $v$  it follows that

$$\|g\| = \Pr[\sigma_{k-1} \in g] = \frac{\Pr[\sigma_{k-1} \in g \wedge \sigma_0 = v]}{\Pr[\sigma_0 = v \mid \sigma_{k-1} \in g]} \leq k \|v\|. \quad \blacktriangleleft$$

We use Lemma 3.7 iteratively to prove Theorem 1.7 which we restate here in a formal way.

► **Theorem 3.8** (Correction algorithm). *For any  $d, q \in \mathbb{N}$ , an abelian group  $G$ , and  $0 < \beta, \varepsilon < 1$  there exist constants  $0 < \lambda, \eta \leq \varepsilon$  such that the following holds: Let  $X$  be a  $d$ -dimensional  $q$ -bounded degree  $\lambda$ -local spectral expander with  $\beta$ -coboundary expanding links. For any  $f \in C^k(X; G)$ ,  $1 \leq k \leq d - 2$ , if  $\|\delta(f)\| \leq \eta^{2^{k+2}-1}$  then there exists  $f' \in C^k(X; G)$  such that  $\text{dist}(f, f') \leq q \binom{d}{k+1} \|\delta(f)\|$ ,  $\|\delta(f')\| \leq \|\delta(f)\|$ , and  $\delta(f')$  is  $(\eta, \varepsilon)$ -non-local.*

**Proof.** Let  $\lambda$  and  $\eta$  be as in Proposition 3.3. Apply Lemma 3.7 for  $\delta(f)$  step by step until no more corrections are possible. Since at every step the weight decreases, this process terminates after some  $r \geq 0$  steps. Denote by  $v_1, v_2, \dots, v_r$  the vertices and by  $g_1, g_2, \dots, g_r$  the  $k$ -cochains given by applying Lemma 3.7 for  $r$  steps, where at step  $i$  we apply it for  $\delta(f - g_1 - \dots - g_{i-1})$ .

Let  $f' = f - g_1 - g_2 - \dots - g_r$ . Since the norm of  $\delta(f)$  decreases at every step of correction, it follows that  $\|\delta(f')\| \leq \|\delta(f)\| \leq \eta^{2^{k+2}-1}$ . Furthermore, since no more corrections are possible, it must be that  $\delta(f')$  is locally minimal. Thus, by Proposition 3.3,  $\delta(f')$  is  $(\eta, \varepsilon)$ -non-local.

It is left to show that  $\|f - f'\|$  is proportional to  $\|\delta(f)\|$ . By definition, for any  $\sigma \in X(k+1)$  it holds that  $\|\sigma\| \geq \left(|X(d)| \binom{d+1}{k+2}\right)^{-1}$ , hence  $r \leq |X(d)| \binom{d+1}{k+2} \|\delta(f)\|$ . Thus,

$$\begin{aligned} \text{dist}(f, f') &= \|g_1 + g_2 + \dots + g_r\| \leq \sum_{i=1}^r (k+1) \|v_i\| \\ &\leq |X(d)| \binom{d+1}{k+2} (k+1) \frac{q}{|X(d)|(d+1)} \|\delta(f)\| \leq \binom{d}{k+1} q \|\delta(f)\| \end{aligned}$$

which finishes the proof. ◀



### 3.3 Cosystolic expansion

We use a similar reduction as in [18] in order to show that  $\delta_1$ -expansion of small sets implies cosystolic expansion over any abelian group. Recall that a complex is a cosystolic expander if the following two properties hold: (1) The systems of equations are expanding, i.e., any assignment that does not satisfy all the equations has a large fraction of unsatisfied equations (proportional to the distance from a satisfying assignment). (2) Every cocycle which is not a coboundary is large. We use the following lemmas which their proofs can be found in the full version of the paper.

► **Lemma 3.9** (The systems of equations are expanding). *For any  $d, q \in \mathbb{N}$ , an abelian group  $G$ , and  $0 < \beta < 1$  there exist  $0 < \lambda, \eta < 1$  such that the following holds: Let  $X$  be a  $d$ -dimensional  $q$ -bounded degree  $\lambda$ -local spectral expander with  $\beta$ -coboundary expanding links over  $G$ . For any  $f \in C^k(X; G) \setminus Z^k(X; G)$ ,  $1 \leq k \leq d - 2$ , it holds that*

$$\|\delta(f)\| \geq \min \left\{ \eta^{2^{k+2}-1}, \frac{1}{q \binom{d}{k+1}} \right\} \cdot \text{dist}(f, Z^k(X; G)).$$

► **Lemma 3.10** (Every cocycle which is not a coboundary is large). *For any  $d \in \mathbb{N}$ , an abelian group  $G$ , and  $0 < \beta < 1$ , there exists  $0 < \lambda, \eta < 1$  such that the following holds: Let  $X$  be a  $d$ -dimensional  $\lambda$ -local spectral expander with  $\beta$ -coboundary expanding links over  $G$ . For any  $f \in Z^k(X; G) \setminus B^k(X; G)$ ,  $0 \leq k \leq d - 1$ , it holds that  $\|f\| \geq \eta^{2^d - 1}$ .*

Theorem 1.8, which we restate here in a formal way, follows immediately from the above two lemmas.

► **Theorem 3.11** (Cosystolic expansion over any abelian group). *For any  $d, q \in \mathbb{N}$ , an abelian group  $G$ , and  $0 < \beta < 1$  there exist  $0 < \lambda, \eta < 1$  such that the following holds: Let  $X$  be a  $d$ -dimensional  $q$ -bounded degree  $\lambda$ -local spectral expander with  $\beta$ -coboundary expanding links over  $G$ . Then the  $(d - 1)$ -skeleton of  $X$  is an  $(\varepsilon, \mu)$ -cosystolic expander over  $G$ , where*

$$\varepsilon = \min \left\{ \eta^{2^d - 1}, \frac{1}{qd^{d/2}} \right\} \quad \text{and} \quad \mu = \eta^{2^d - 1}.$$

**Proof.** Immediate from Lemmas 3.9 and 3.10. ◀

## 4 Result for non-abelian groups

### 4.1 Non-abelian groups

When the group is non-abelian, the coboundary operator is defined only in dimensions 0 and 1, and its definition is more delicate. Let  $G$  be a group with a multiplicative operation. The coboundary of a 0-cochain  $f \in C^0(X; G)$  is a 1-cochain  $\delta(f)$  defined by

$$\delta(f)(u, v) = f(u)f(v)^{-1}.$$

The coboundary of a 1-cochain  $g \in C^1(X; G)$  is a 2-cochain  $\delta(g)$  defined by

$$\delta(g)(u, v, w) = g(u, v)g(v, w)g(w, u).$$

One can check that for  $f \in C^i(X; G)$ ,  $i \in \{0, 1\}$ ,  $\delta(f)$  is an antisymmetric functions, i.e.,  $\delta(f)$  is an  $(i + 1)$ -cochain.

The distance between two cochains  $f, g \in C^i(X; G)$  is defined by  $\text{dist}(f, g) = \|gf^{-1}\|$ , where  $gf^{-1}(\sigma) = g(\sigma)f(\sigma)^{-1}$  for every  $\sigma \in \vec{X}(i)$ .

Similar to the abelian case, we say that  $f \in C^i(X; G)$  is a cocycle if  $\delta(f) = \mathbf{1}$ .<sup>6</sup> The distance of a cochain  $f \in C^i(X; G)$  from the  $i$ -cocycles is defined by

$$\text{dist}(f, Z^i(X; G)) = \min\{\text{dist}(f, g) \mid g \in Z^i(X; G)\}.$$

In order to measure the distance of a 1-cochain from the 1-coboundaries, we define an action of  $C^0(X; G)$  on  $C^1(X; G)$ , where for  $f \in C^0(X; G)$  and  $g \in C^1(X; G)$ ,  $f.g(u, v)$  is defined by

$$f.g(u, v) = f(u)g(u, v)f(v)^{-1}.$$

Now, the distance of  $g$  from the 1-coboundaries is defined by

$$\text{dist}(g, B^1(X; G)) = \min\{\text{dist}(g, f.g) \mid f \in C^0(X; G)\}.$$

## 4.2 Weakly-non-local sets and cosystolic expansion

In the case of a non-abelian group, we cannot get the same non-local property as in abelian groups, rather we get a slightly weaker notion which we call *weakly-non-local*. Roughly speaking, a set of  $k$ -faces in a given complex is weakly-non-local if its  $k$ -faces are evenly distributed on their  $(k - 2)$ -subfaces.

► **Definition 4.1** (Weakly-non-local sets). *Let  $X$  be a  $d$ -dimensional simplicial complex and  $0 < \eta, \varepsilon, \alpha < 1$ . For any set of  $k$ -faces  $A \subseteq X(k)$ ,  $1 \leq k \leq d - 1$ , we define the following set of  $(k - 2)$ -faces:*

$$S_{k-2} = \{\sigma \in X(k - 2) \mid \|A_\sigma\| \leq \eta\}.$$

*We say that  $A$  is  $(\eta, \varepsilon, \alpha)$ -weakly-non-local if  $\|S_{k-2}\| \geq 1 - \varepsilon \|A\|$  and for every  $\tau \in X(k - 1)$  it holds that  $\|A_\tau\| \leq 1 - \alpha$ .*

We show that this weakly-non-local property also implies that the set is  $\delta_1$ -expanding.

► **Theorem 4.2** (Weakly-non-local sets are  $\delta_1$ -expanding). *Let  $X$  be a  $d$ -dimensional  $\lambda$ -local spectral expander and  $0 < \eta, \varepsilon, \alpha < 1$ . There exists a constant  $c = c(d, \lambda, \eta, \varepsilon, \alpha)$  such that for any  $A \subseteq X(k)$ ,  $1 \leq k \leq d - 1$ , if  $A$  is  $(\eta, \varepsilon, \alpha)$ -weakly-non-local then*

$$\|\delta_1(A)\| \geq c \|A\|.$$

*In particular, if  $\varepsilon \leq \alpha/3d^3$ ,  $\lambda \leq \varepsilon^2$  and  $\eta \leq \varepsilon^3$  then  $\|\delta_1(A)\| \geq \alpha \|A\|$ .*

The rest of the steps are similar to the abelian case (with modifications for non-abelian groups) and can be found in the full version of the paper. We just state here the final theorem for cosystolic expansion over non-abelian groups.

► **Theorem 4.3** (Cosystolic expansion over any group). *For any group  $G$ ,  $q \in \mathbb{N}$  and  $0 < \beta < 1$  there exist  $0 < \lambda, \eta < 1$  such that the following holds: Let  $X$  be a 3-dimensional  $q$ -bounded degree  $\lambda$ -local spectral expander with  $\beta$ -coboundary expanding links over  $G$ . Then the 2-skeleton of  $X$  is an  $(\varepsilon, \mu)$ -cosystolic expander over  $G$ , where*

$$\varepsilon = \min \left\{ \frac{\beta\eta}{2}, \frac{1}{2q} \right\} \quad \text{and} \quad \mu = \frac{\beta\eta}{2}.$$

<sup>6</sup> It is common to denote the identity element of a multiplicative group by 1 and not by 0 as in an additive group.

## References

- 1 V. L. Alev, F. G. Jeronimo, D. Quintana, S. Srivastava, and M. Tulsiani. List decoding of direct sum codes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1412–1425. SIAM, 2020.
- 2 V. L. Alev, F. G. Jeronimo, and M. Tulsiani. Approximating constraint satisfaction problems on high-dimensional expanders. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–201, 2019.
- 3 V. L. Alev and L. C. Lau. Improved analysis of higher order random walks and applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1198–1211, 2020.
- 4 N. Anari, K. Liu, and S. O. Gharan. Spectral independence in high-dimensional expanders and applications to the hardcore model. *arXiv preprint*, 2020. [arXiv:2001.00303](https://arxiv.org/abs/2001.00303).
- 5 N. Anari, K. Liu, S. O. Gharan, and C. Vinzant. Log-concave polynomials II: high-dimensional walks and an FPRAS for counting bases of a matroid. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1–12, 2019.
- 6 Z. Chen, A. Galanis, D. Štefankovič, and E. Vigoda. Rapid mixing for colorings via spectral independence. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1548–1557. SIAM, 2021.
- 7 Z. Chen, K. Liu, and E. Vigoda. Optimal mixing of glauber dynamics: Entropy factorization via high-dimensional expansion. *arXiv preprint*, 2020. [arXiv:2011.02075](https://arxiv.org/abs/2011.02075).
- 8 Z. Chen, K. Liu, and E. Vigoda. Rapid mixing of glauber dynamics up to uniqueness via contraction. *arXiv preprint*, 2020. [arXiv:2004.09083](https://arxiv.org/abs/2004.09083).
- 9 Y. Dikstein and I. Dinur. Agreement testing theorems on layered set systems. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1495–1524. IEEE, 2019.
- 10 I. Dinur, Y. Filmus, P. Harsha, and M. Tulsiani. Explicit SoS lower bounds from high-dimensional expanders. *arXiv preprint*, 2020. [arXiv:2009.05218](https://arxiv.org/abs/2009.05218).
- 11 I. Dinur, P. Harsha, T. Kaufman, I. L. Navon, and A. Ta Shma. List decoding with double samplers. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2134–2153. SIAM, 2019.
- 12 I. Dinur and T. Kaufman. High dimensional expanders imply agreement expanders. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 974–985, 2017.
- 13 Irit Dinur and Roy Meshulam. Near coverings and cosystolic expansion—an example of topological property testing. *arXiv preprint*, 2019. [arXiv:1909.08507](https://arxiv.org/abs/1909.08507).
- 14 S. Evra and T. Kaufman. Bounded degree cosystolic expanders of every dimension. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 36–48, 2016.
- 15 S. Evra, T. Kaufman, and G. Zemor. Decodable quantum LDPC codes beyond the  $\sqrt{n}$  distance barrier using high dimensional expanders. In *61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020. to appear.
- 16 W. Feng, H. Guo, Y. Yin, and C. Zhang. Rapid mixing from spectral independence beyond the boolean domain. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1558–1577. SIAM, 2021.
- 17 M. Gromov. Singularities, Expanders and Topology of Maps. Part 2: from Combinatorics to Topology Via Algebraic Isoperimetry. *Geometric And Functional Analysis*, 20(2):416–526, 2010.
- 18 T. Kaufman, D. Kazhdan, and A. Lubotzky. Ramanujan Complexes and Bounded Degree Topological Expanders. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 484–493, 2014.

- 19 T. Kaufman and D. Mass. Local-To-Global Agreement Expansion via the Variance Method. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 74:1–74:14, 2020.
- 20 N. Linial and R. Meshulam. Homological connectivity of random 2-complexes. *Combinatorica*, 26(4):475–487, 2006.
- 21 A. Lubotzky. Ramanujan complexes and high dimensional expanders. *Japanese Journal of Mathematics*, 9(2):137–169, 2014.
- 22 A. Lubotzky, R. Meshulam, and S. Mozes. Expansion of building-like complexes. *Groups, Geometry, and Dynamics*, 10(1):155–175, 2016.
- 23 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
- 24 A. Lubotzky, B. Samuels, and U. Vishne. Explicit constructions of ramanujan complexes of type  $\tilde{A}_d$ . *European Journal of Combinatorics*, 26(6):965–993, 2005.
- 25 A. Lubotzky, B. Samuels, and U. Vishne. Ramanujan complexes of type  $\tilde{A}_d$ . *Israel Journal of Mathematics*, 149(1):267–299, 2005.



# Group Evacuation on a Line by Agents with Different Communication Abilities

**Jurek Czyzowicz**

Département d'informatique,  
Université du Québec en Outaouais,  
Gatineau, Canada

**Ryan Killick**

School of Computer Science, Carleton University,  
Ottawa, Canada

**Evangelos Kranakis**

School of Computer Science, Carleton University,  
Ottawa, Canada

**Danny Krizanc**

Department of Mathematics & Computer Science,  
Wesleyan University, Middletown, CT, USA

**Lata Narayanan**

Department of Computer Science and Software  
Engineering, Concordia University,  
Montreal, Canada

**Jaroslav Opatrny**

Department of Computer Science and Software  
Engineering, Concordia University,  
Montreal, Canada

**Denis Pankratov**

Department of Computer Science and Software  
Engineering, Concordia University,  
Montreal, Canada

**Sunil Shende**

Department of Computer Science,  
Rutgers University, Piscataway, NJ, USA

---

## Abstract

---

We consider evacuation of a group of  $n \geq 2$  autonomous mobile agents (or robots) from an unknown exit on an infinite line. The agents are initially placed at the origin of the line and can move with any speed up to the maximum speed 1 in any direction they wish and they all can communicate when they are co-located. However, the agents have different wireless communication abilities: while some are fully wireless and can send and receive messages at any distance, a subset of the agents are *senders*, they can only transmit messages wirelessly, and the rest are *receivers*, they can only receive messages wirelessly. The agents start at the same time and their communication abilities are known to each other from the start. Starting at the origin of the line, the goal of the agents is to collectively find a target/exit at an unknown location on the line while minimizing the *evacuation time*, defined as the time when the last agent reaches the target.

We investigate the impact of such a mixed communication model on evacuation time on an infinite line for a group of cooperating agents. In particular, we provide evacuation algorithms and analyze the resulting competitive ratio ( $CR$ ) of the evacuation time for such a group of agents. If the group has two agents of two different types, we give an *optimal* evacuation algorithm with competitive ratio  $CR = 3 + 2\sqrt{2}$ . If there is a single sender or fully wireless agent, and multiple receivers we prove that  $CR \in [2 + \sqrt{5}, 5]$ , and if there are multiple senders and a single receiver or fully wireless agent, we show that  $CR \in [3, 5.681319]$ . Any group consisting of only senders or only receivers requires competitive ratio 9, and any other combination of agents has competitive ratio 3.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Mixed discrete-continuous optimization; Theory of computation  $\rightarrow$  Models of computation; Theory of computation  $\rightarrow$  Theory and algorithms for application domains

**Keywords and phrases** Agent, Communication, Evacuation, Mobile, Receiver, Search, Sender

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.57

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.12676>

**Funding** Research supported by NSERC, Canada.



© Jurek Czyzowicz, Ryan Killick, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Denis Pankratov, and Sunil Shende;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiro Sadakane; Article No. 57; pp. 57:1–57:24



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

Search by a group of cooperating autonomous mobile robots for a target in a given domain is a fundamental topic in the theoretical computer science. In the search problem one is interested in finding a target at an unknown location as soon as possible. In the related *evacuation problem* one is interested in optimizing the time it takes the last robot in the group to find the target, often called the *exit*. There has been a lot of interest in trying to understand the impact of communication between agents on the search and evacuation time in the distributed computing area. The design of optimal robot trajectories leading to tight bounds depends not only on the fault-tolerant characteristics of the agents but also on the communication model employed (see [14, 16]). In previous works, agents are assumed to either have full wireless communication abilities, i.e., they can both transmit and receive messages across any distance [10], or limited distance [4], or they have no wireless communication abilities, and can only communicate when they are *face-to-face (F2F)*, i.e., co-located. In terms of communication abilities, the agents are identical.

The present work considers evacuation on an infinite line by a group  $G$  of cooperating robots (initially located at the origin) whose wireless communication abilities are different, which compel them to employ a *mixed* communication model. At a *rudimentary* level they can always communicate reliably using F2F. However, some agents in  $G$  are *senders* that can transmit messages wirelessly at any distance but only receive F2F, yet others are *receivers* in that they can receive messages wirelessly from any distance but can transmit only F2F, and the remaining are fully wireless, and can both send and receive messages wirelessly. This situation might occur because it is cheaper to build agents with limited wireless capabilities, or because the sender or receiver module failed in receiver or sender robots, respectively. Further, we assume the capabilities of the robots are known to each other in advance and remain the same for the duration of an evacuation algorithm. Robots can move at any speed up to maximum 1. We give upper and lower bounds on the competitive ratio of evacuation algorithms, depending on the number of senders and receivers among the agents.

If there are at least two fully wireless agents in the group, then the optimal competitive ratio is 3, see [10]. By pairing up a sender and a receiver we can simulate a fully wireless agent. Consequently, if there is one fully wireless agent, one sender agent, and one receiver agent, the competitive ratio is 3. Consider now the case when there is one fully wireless agent, and one or more senders. Since the sender agents cannot receive wireless transmissions, the sending capabilities of the fully wireless agent are useless, and it is equivalent to a receiver agent. Similarly if there is one fully wireless agent, and one or more receivers, the receiving module in the fully wireless agent is useless, and it is equivalent to a sender agent.

Thus we no longer consider fully wireless agents, and only consider sender and receiver agents. When all of the agents are senders, or all of them are receivers, the only possible mode of communication between agents is F2F; in this case, it has previously been demonstrated that the optimal competitive ratio of evacuation for  $n$  F2F agents is 9. If there are at least two sender agents and two receiver agents, by pairing up sender and receiver agents, we obtain a competitive ratio of 3. It follows that the only interesting cases to consider are when there is exactly one sender and one receiver; one sender and several receivers; one receiver and several senders. These are the cases investigated in detail in this paper.

### 1.1 Model and preliminaries

We consider the problem of evacuation by  $n \geq 2$  mobile agents beginning at the origin of the infinite line. All agents are assumed to have maximum speed 1 and can move in either the positive direction (referred to as moving to the right) or the negative direction (referred



to as moving to the left). The agents may change their speed and the direction of motion instantaneously and arbitrarily often. Moreover, the robots can choose any speed as long as it does not exceed the maximum speed 1.

All agents have the ability to communicate F2F, however the wireless communication abilities of the agents are limited and are not all the same. Indeed the group of  $n$  agents consists of a subset of agents that can only send wireless messages, called *senders*, and a subset that can only receive wireless messages, called *receivers*. We represent by  $n_s \geq 0$  and  $n_r = n - n_s$  the number of senders and receivers respectively.

The cost of an algorithm for the evacuation problem on a given instance of the problem is the time the *last* agent reaches the target, called the *evacuation time*. We denote by  $E(\mathcal{A}, x)$  the evacuation time of algorithm  $\mathcal{A}$  when the target is at location  $x$ . Note that an offline algorithm in which agents know the position of the target can reach it in time  $|x|$ . The goal is to minimize the *competitive ratio*, denoted by CR, defined as the supremum, over all possible target locations, of the normalized cost  $E(\mathcal{A}, x)/|x|$ , i.e.,  $\text{CR}(\mathcal{A}) := \sup_{|x|>1} \frac{E(\mathcal{A}, x)}{|x|}$ .

An evacuation algorithm can be primarily viewed as a set of trajectories, one for each agent. The trajectory of an agent specifies where the agent should be located at any given time. More specifically, the trajectory of an agent is a continuous mapping from the non-negative reals (i.e. time) to the reals (i.e., position on the line). In general, we will represent the trajectory of an agent using the notation  $X = X(t)$  with the interpretation that the agent with trajectory  $X$  will be located at position  $X(t)$  at time  $t$ . Due to our assumption that the agents have maximum unit speed, an agent trajectory  $X$  must satisfy  $|X(t') - X(t)| \leq t' - t$ ,  $\forall t' \geq t \geq 0$ . Agents are assumed to begin their search at the origin and so we must also have  $X(0) = 0$ . Taken together, these equations imply that  $|X(t)| \leq t$ ,  $\forall t \geq 0$ .

We assume that the agents are labelled so that we may assign a specific trajectory to a specific agent. Each agent is assumed to know the trajectory of all other agents. All agents follow their assigned trajectories until they either find the target or are otherwise notified of the target's location. What an agent does in the event that it finds the target depends on the communication ability of the finder and of the other agents. For example, if the finder is a sender and all other agents are receivers, then the sender can immediately notify the other agents who can then proceed to move at full speed to the target. On the other hand, if the finder is a receiver, then it must move to notify another agent(s) of the target's location. In any case, the cost of the algorithm will depend both on the trajectories assigned to the agents to search for the target, as well as the subsequent phase of informing all agents, and the agents travelling to the target.

## 1.2 Related work

Search by a single agent on the infinite line was initiated independently by Beck [6, 7, 8] and Bellman [9]. These seminal papers proved the competitive ratio 9 for search on an infinite line and also gave the impetus for additional studies, including those by Gal [20] which proposes a minimax solution for a game in which player I chooses a real number and player II seeks it by choosing a trajectory represented by a positive function, Friestedt [18], Friestedt and Heath [19], and Baeza-Yates et.al.[2, 3] where search by agents in domains other than the line were proposed, e.g. the ‘‘Lost Cow’’ problem in the plane or at the origin of  $w$  concurrent rays. Additional information on search games and rendezvous can also be found in the book [1].

Group search on an infinite line has been researched in several papers and under various models. Evacuation by multiple cooperating robots was proposed in [10], for the case where the robots can communicate only F2F. More recently, search on the line was considered for

two robots which have distinct speeds in [5] and in [17] when turning costs are taken into account. In addition, in two papers [11, 12] the authors are concerned with minimizing the energy consumed during the search.

There are several types of robot communication models in the literature. The most restricted type of robot communication is F2F in which robots may exchange messages only when they are co-located. At the other extreme is wireless in which robots may communicate regardless of how far apart they are [13]. A model where the wireless communication range is limited has been explored for the equilateral triangle domain in [4].

Within these communication models researchers have considered search with crash [16] and Byzantine [14] faults. The former are innocent faults caused by robot sensor malfunctioning causing the robots an inability to communicate and/or perform their tasks. The latter, however, are malicious faults (intentionally or otherwise) in that the robots may lie and communicate maliciously the wrong information. Lower bounds for search in the crash fault model are proved in [22] and for Byzantine faults in [23]. The competitive ratio for search and evacuation in the near majority case (of  $n = 2f + 1$  robots with  $f$  faulty) is a notoriously hard problem and additional results can be found in the recent paper [15]. Additional information and results can also be found in the recent PhD thesis [21].

In our paper we investigate evacuation time by agents with different wireless communication abilities; as stated earlier, some of them can only transmit wirelessly but not receive, and the others can only receive messages sent wirelessly but not transmit. To our knowledge, in all previous works, the communication abilities of agents were identical; group search or evacuation by agents of different communication abilities has not been studied before.

### 1.3 Results

As mentioned above, we need to consider three cases: when there is exactly one sender and one receiver; one sender and several receivers; one receiver and several senders. In Section 2 we give evacuation algorithms and analyze upper bounds on the competitive ratios for each of these three cases. When we have one receiver and one sender agent, i.e.,  $n_s = n_r = 1$ , we give an evacuation algorithm whose competitive ratio is at most  $3 + 2\sqrt{2}$ . In case when we have one sender and several receivers, i.e.,  $n_s = 1$  and  $n_r > 1$  our evacuation algorithm has competitive ratio at most 5, and when  $n_s > 1$  and  $n_r = 1$  we specify an algorithm with competitive ratio at most  $\approx 5.681319$ . These results can be found in Theorems 1, 2, and 3 respectively.

In Section 3 we consider lower bounds on the competitive ratio of evacuation algorithms for only the cases with only one sender agent, i.e.,  $n_s = 1$ . In particular, we prove a lower bound matching our upper bound for the case of  $n_s = n_r = 1$ , which proves the optimality of our algorithm. For the case of  $n_s = 1$  and  $n_r > 1$  we demonstrate that the evacuation cannot be completed with a competitive ratio less than  $2 + \sqrt{5}$ . We conclude the paper with a discussion of open problems in Section 4.

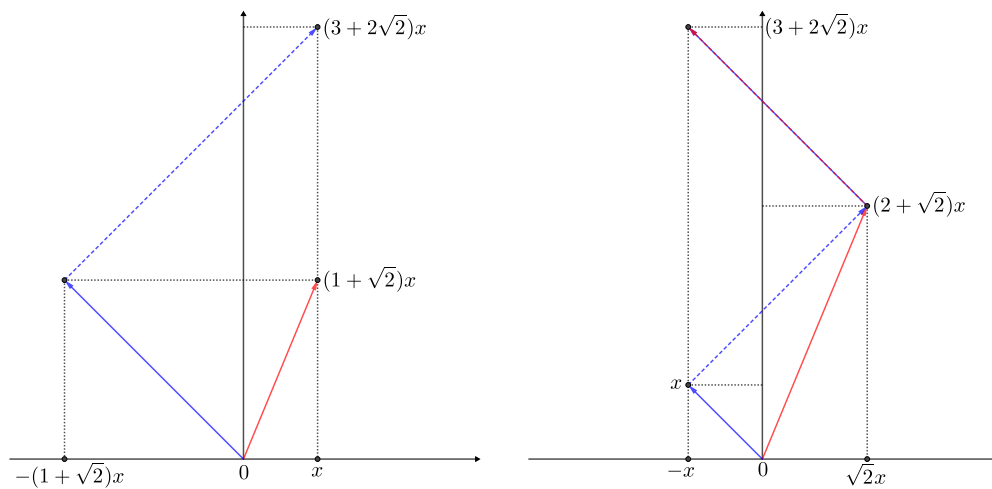
## 2 Evacuation Algorithms and their Competitive Ratios

In this section we give evacuation algorithms for our communication model and investigate their competitive ratios. We consider separately the cases, first the single sender and single receiver, second the single sender and multiple receivers, and lastly the multiple senders and single receiver.

### 2.1 One sender, one receiver

► **Theorem 1.** *When  $n_s = n_r = 1$  there exists an evacuation algorithm with competitive ratio  $3 + 2\sqrt{2}$ .*

**Proof.** The proof is constructive and based on the following algorithm: the receiver moves to the left at unit speed and the sender moves to the right with speed  $\sqrt{2} - 1$ . If the sender finds the target first then it notifies the receiver (wirelessly) and the receiver moves at unit speed to the target. If the receiver finds the target first then it moves at unit speed to the right until it reaches the sender at which time both agents will move at full speed back to the target. We illustrate this algorithm in Figure 1 using a space-time diagram which plots an agent’s position on the  $x$ -axis, and uses the  $y$ -axis to indicate the flow of time.



■ **Figure 1** The trajectories of the agents when the target is at location  $+x$  (left) and  $-x$  (right). The sender is colored red and the receiver is blue. A dashed line indicates when an agent deviates from its assigned search trajectory. Significant times and positions are indicated.

Suppose that the target is at location  $x > 1$ . The sender will find this target first and will do so at the time  $\frac{x}{\sqrt{2}-1} = (1 + \sqrt{2})x$ . The sender immediately notifies the receiver which is at location  $-(1 + \sqrt{2})x$ . Moving at unit speed, the receiver will travel distance  $x + (1 + \sqrt{2})x = (2 + \sqrt{2})x$  to reach the target and will arrive at time  $(1 + \sqrt{2})x + (2 + \sqrt{2})x = (3 + 2\sqrt{2})x$ . The competitive ratio when  $x > 1$  is thus  $3 + 2\sqrt{2}$ .

Suppose now that the target is at location  $-x < -1$ . The receiver will find the target first and will do so at the time  $x$ . The receiver must move to notify the sender who is located at  $(\sqrt{2} - 1)x$  at time  $x$ . Hence, the distance between the agents is  $x + (\sqrt{2} - 1)x = \sqrt{2}x$  and the receiver will need to cross this distance with a relative speed of  $1 - (\sqrt{2} - 1) = 2 - \sqrt{2}$ . The receiver will thus take time  $\frac{\sqrt{2}x}{2-\sqrt{2}} = \frac{x}{\sqrt{2}-1} = (1 + \sqrt{2})x$ , and both agents will take an additional time  $(1 + \sqrt{2})x$  to reach the target. The time to evacuate is thus  $x + 2(1 + \sqrt{2})x = (3 + 2\sqrt{2})x$ , and, evidently, the competitive ratio in this case is also  $3 + 2\sqrt{2}$ . ◀

Notice that in the algorithm of Theorem 1 it is essential that the sender moves initially at speed less than 1.

### 2.2 One sender, multiple receivers

► **Theorem 2.** When  $n_s = 1$  and  $n_r > 1$  there exists an evacuation algorithm with competitive ratio 5.

**Proof sketch.** The proof is constructive and based on the following algorithm: one receiver moves to the left at unit speed and one receiver moves to the right at unit speed. When one of the receivers finds the target it immediately moves to notify the sender (and all other agents) at the origin. The sender immediately notifies the remaining receiver, and all agents proceed to the target at unit speed. The complete analysis of this algorithm appears in Appendix A. ◀

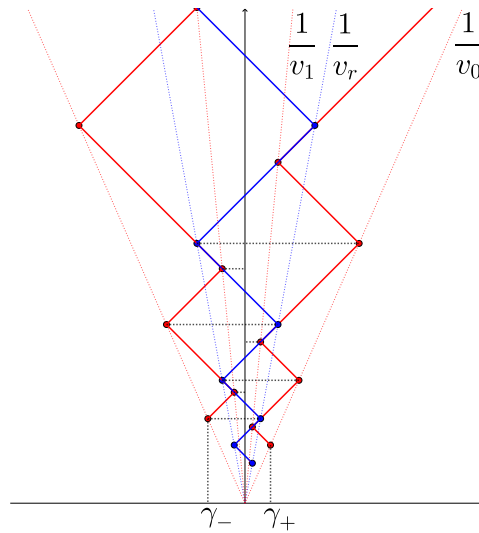
### 2.3 Multiple senders, one receiver

► **Theorem 3.** When  $n_s > 2$  and  $n_r = 1$  there exists an algorithm  $\mathcal{A}$  with competitive ratio  $CR(\mathcal{A}) < 5.681319$ . More exactly, the competitive ratio is upper bounded by

$$CR(\mathcal{A}) \leq 1 + \frac{1 + v_r}{1 - v_r} \left( \frac{1 + 4v_r - v_r^2}{v_r(3 - v_r)} \right) \tag{1}$$

with  $v_r$  chosen to be the root of the equation  $v_r^4 - 16v_r^3 + 26v_r^2 + 8v_r - 3 = 0$  satisfying  $0 \leq v_r < 1$ .

The proof of this result is much more involved than the previous two cases. When there are more than two senders, all but two of the senders will remain at the origin until they are notified of the target (at which time they move to the announced location). Thus, in the rest of this section we will present our algorithm for the specific case of two senders and one receiver, i.e.,  $n_s = 2$  and  $n_r = 1$ .



■ **Figure 2** Trajectories of the agents for the evacuation algorithm  $EVAC_{RAYS}(v_r)$ . The sender trajectories are red, and the receiver trajectory is blue.

**High level idea.** The robots jointly maintain an interval around the origin of positions that have already been explored by at least one robot. They wish to expand this interval at a fast pace while maintaining the ability to notify all robots quickly in case an exit is found

by at least one robot. The idea behind our algorithm is to make one sender responsible for extending the right end of the searched interval, and another sender responsible for extending the left end of the searched interval. The receiver zig-zags around the origin (with the lengths of zigs and zags increasing in rounds), so that if one of the senders finds an exit, the receiver is “close” to the other sender and can quickly notify it via F2F. In order for this idea to work, the senders cannot simply move away from the origin at full speed, but instead they perform zig-zags of their own (however, unlike the receiver their zig-zags are drifting away from the origin). One can think of a sender as first extending the searched region for a while and then coming back partway towards the origin to get notified by the receiver about what’s happening on the other side of the origin. This strategy is illustrated in Figure 2. When the exit is found by one of the senders, the receiver goes to intercept the other sender and they both move towards the exit.

An interesting feature of the algorithm is that the zig-zag trajectory of the receiver non trivially overlaps with the zig-zag trajectories of the senders, i.e., it does not simply touch them. For example, take a particular time when the receiver meets the right sender for the first time during one zig-zag round. Then the receiver and the right sender travel to the right together for some time. During this time the left sender extends the searched region on the left. If an exit is found by the left sender at this point, this is good – both the right sender and receiver will learn about it instantaneously and will start moving towards the exit. However, the right sender and receiver cannot keep travelling together for very long, since the trajectory needs to have certain symmetries, lest the left sender gets too far. Thus, at some point the receiver and the right sender part ways with the receiver moving towards the left sender and the right sender continuing to the right. At precisely this point, the left sender stops extending the search interval and starts to move towards the receiver (this situation is indicated by dashed lines in Figure 2). Intuitively, this is a good timing for the left sender to switch direction, because otherwise if it finds an exit soon after the receiver and right sender part ways then the receiver would not be able to catch up with the right sender for quite a while (until the right sender’s next “zag”).

Formalizing and analyzing this algorithm takes a lot of work and careful calculations, which are deferred to the appendix. Here, we just state some of the main ingredients. We begin by introducing a class of search trajectories that are parameterized by a four-tuple  $[\eta, v_0, v_1, \gamma]$  where:  $\eta = \pm 1$ , and  $v_0, v_1$ , and  $\gamma$  are real numbers satisfying  $0 \leq v_0 \leq 1$ ,  $-1 \leq v_1 < v_0$ , and  $0 < \gamma \leq 1$ .

■ **Algorithm 1** RAYS( $\eta, v_0, v_1, \gamma$ ).

---

**Begin:** Move to location  $\eta\gamma$  and wait until time  $\frac{\gamma}{v_0}$ .

1: **repeat**

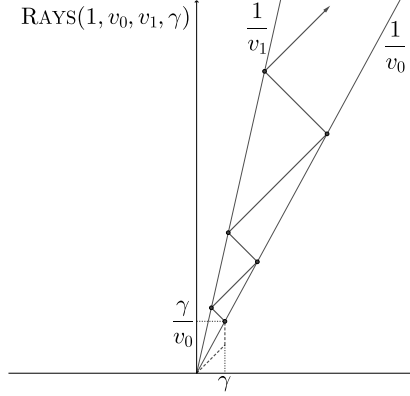
2:     Move in direction  $-\eta$  at unit speed until my position  $x$  at time  $t$  satisfies  $\frac{x}{t} = \eta v_1$ ;

3:     Move in direction  $\eta$  at unit speed until my position  $x$  at time  $t$  satisfies  $\frac{x}{t} = \eta v_0$ ;

**:End**

---

An example of this type of search trajectory is illustrated in Figure 3. As one can observe, after an initial setup phase, the trajectory RAYS( $\eta, v_0, v_1, \gamma$ ) will bounce back and forth between two space-time rays with slopes  $\frac{\eta}{v_0}$  and  $\frac{\eta}{v_1}$ . The parameter  $\gamma$  dictates the “beginning” position on the ray with slope  $\frac{\eta}{v_0}$ , and  $\eta$  is a symmetry parameter in the sense that trajectories RAYS( $\pm 1, v_0, v_1, \gamma$ ) are reflections of each other about the time-axis. Although the above specification of the trajectory RAYS( $\eta, v_0, v_1, \gamma$ ) is simple to understand, it will be more convenient to express these trajectories in terms of their *turning-points* – space-time points at



■ **Figure 3** Example of the trajectory  $\text{RAYS}(1, v_0, v_1, \gamma)$ .

which an agent changes its travel direction, and between which an agent moves at constant unit speed. One can observe from Figure 3 that the turning-points of  $\text{RAYS}(\eta, v_0, v_1, \gamma)$  are precisely the points where the trajectory bounces off the rays with slopes  $\frac{\eta}{v_0}$  and  $\frac{\eta}{v_1}$ . The next lemma provides expressions for the turning-points of the trajectory  $\text{RAYS}(\eta, v_0, v_1, \gamma)$ .

► **Lemma 4.** *The turning-points  $(D_j, T_j)$ ,  $j = 0, 1, \dots$ , of the trajectory  $\text{RAYS}(\eta, v_0, v_1, \gamma)$  are given by*

$$D_j = \eta\gamma \left[ \frac{(1-v_1)(1+v_0)}{(1+v_1)(1-v_0)} \right]^{\lfloor \frac{j}{2} \rfloor} \begin{cases} 1, & \text{even } j \\ \frac{v_1(1+v_0)}{v_0(1+v_1)}, & \text{odd } j \end{cases}, \quad T_j = \frac{D_j}{\eta} \begin{cases} \frac{1}{v_0}, & \text{even } j \\ \frac{1}{v_1}, & \text{odd } j. \end{cases}$$

We will describe our evacuation algorithm in terms of the trajectories  $\text{RAYS}(1, v_0, v_1, \gamma)$ . To this end, we represent by  $X_{\pm}(t)$  the trajectories of the senders and we refer to the sender with trajectory  $X_+$  (resp.  $X_-$ ) as the *right-sender* (resp. *left-sender*). We use  $X_r(t)$  to represent the trajectory of the receiver. The turning-points of the trajectories  $X_{\pm}$  will be represented by  $(D_j^{\pm}, T_j^{\pm})$ , and the turning-points of the trajectory  $X_r$  will be represented by  $(D_j^r, T_j^r)$ . With this notation our evacuation algorithm can be expressed as in Algorithm 2. We refer to this algorithm by  $\text{EVAC}_{\text{RAYS}}(v_r)$ .

■ **Algorithm 2**  $\text{EVAC}_{\text{RAYS}}(v_r)$ ,  $0 \leq v_r < 1$ .

1:

$$X_{\pm}(t) = \text{RAYS}(\pm 1, v_0, v_1, \gamma_{\pm}), \quad X_r(t) = \text{RAYS}(1, v_r, -v_r, 1). \quad (2)$$

$$v_0 = \frac{v_r(3-v_r)}{1+v_r}, \quad v_1 = \frac{v_r(1-v_r)}{1+3v_r}, \quad \gamma_+ = \frac{3-v_r}{1-v_r}, \quad \gamma_- = \frac{3-v_r}{1-v_r} \frac{1+v_r}{1-v_r} \quad (3)$$

Figure 2 illustrates the trajectories of the agents for the algorithm  $\text{EVAC}_{\text{RAYS}}(v_r)$ . The choices of  $v_0, v_1, \gamma_{\pm}$  in (3) ensure that the trajectories enjoy a number of important properties, some of which are evident in Figure 2. One immediately obvious property is the fact that the right/left-sender spends all of its time to the right/left of the origin (and hence the naming convention). Some other properties that are evident in Figure 2 are given in Observation 5.

► **Observation 5.** For all  $k = 0, 1, 2, \dots$  the following properties hold for the algorithm  $EVAC_{\text{RAYS}}(v_r)$ :

1. the receiver reaches its turning point  $2k+1$  (resp.  $2k+2$ ) at the same time the right-sender (resp. left-sender) reaches its turning-point  $2k$ .
2. the receiver and right-sender (resp. left-sender) are co-located at all times in the interval  $[T_{2k+1}^+, T_{2k+2}^r]$  (resp.  $[T_{2k+1}^-, T_{2k+3}^r]$ ),

In order to establish these properties, we carefully calculate the turning points of all agents in terms of the parameter  $v_r$ . The following lemmas summarize the calculations, which are carried out in Appendix A due to space limit. Equipped with these formulas, Observation 5 follows, which is also demonstrated in the appendix.

► **Lemma 6.** The turning-points of the receiver are

$$D_j^r = (-1)^j \left( \frac{1+v_r}{1-v_r} \right)^j, \quad T_j^r = \frac{1}{v_r} \left( \frac{1+v_r}{1-v_r} \right)^j.$$

► **Lemma 7.** The turning-points of the right-sender are

$$D_j^+ = v_r T_j^r \begin{cases} \frac{3-v_r}{1-v_r}, & \text{even } j \\ \frac{1-v_r}{1+v_r}, & \text{odd } j \end{cases}, \quad T_j^+ = T_j^r \begin{cases} \frac{1+v_r}{1-v_r}, & \text{even } j \\ \frac{1+3v_r}{1+v_r}, & \text{odd } j \end{cases}$$

► **Lemma 8.** The turning-points of the left-sender are

$$D_j^- = -v_r T_{j+1}^r \begin{cases} \frac{3-v_r}{1-v_r}, & \text{even } j \\ \frac{1-v_r}{1+v_r}, & \text{odd } j \end{cases}, \quad T_j^- = T_{j+1}^r \begin{cases} \frac{1+v_r}{1-v_r}, & \text{even } j \\ \frac{1+3v_r}{1+v_r}, & \text{odd } j \end{cases}.$$

The next theorem provides an expression for the competitive ratio of  $EVAC_{\text{RAYS}}(v_r)$  as a function of  $v_r$  (see Appendix A for the proof).

► **Theorem 9.** The competitive ratio of algorithm  $EVAC_{\text{RAYS}}(v_r)$  satisfies

$$CR \leq 1 + \frac{1+v_r}{1-v_r} \left( \frac{1+4v_r-v_r^2}{v_r(3-v_r)} \right).$$

Now that we have an expression for a bound on the competitive ratio we can finally prove Theorem 3.

**Proof (Theorem 3).** We need to optimize the competitive ratio with respect to  $v_r$  and so we need to compute the derivative of the right hand side of (7). This is most easily done with the aid of a computer. We find that

$$\frac{d}{dv_r} \left[ 1 + \frac{1+v_r}{1-v_r} \left( \frac{1+4v_r-v_r^2}{v_r(3-v_r)} \right) \right] = \frac{v_r^4 - 16v_r^3 + 26v_r^2 + 8v_r - 3}{v_r^2(3-v_r)^2(1-v_r)^2}$$

and so the optimum choice of  $v_r$  is a root of the quartic equation  $v_r^4 - 16v_r^3 + 26v_r^2 + 8v_r - 3 = 0$  satisfying  $0 \leq v_r < 1$ . Numerically solving this equation for  $v_r$  yields  $v_r \approx 0.228652$ . For this choice of  $v_r$  one can confirm that  $CR < 5.681319$ . ◀



### 3 Lower bounds

In this section we investigate lower bounds on the competitive ratio of evacuation in our communication model. Our goal is the proof of the following theorem:

► **Theorem 10.** *Let  $\mathcal{A}$  be an evacuation algorithm for one sender and  $n_r \geq 1$  receivers.*

$$CR(\mathcal{A}) \geq \begin{cases} 3 + 2\sqrt{2}, & n_r = 1 \\ 2 + \sqrt{5}, & n_r > 1 \end{cases}.$$

We will need to introduce a number of concepts and definitions. The first definition concerns the knowledge that is available to an agent at a given time.

► **Definition 11.** *An agent is said to know of a location  $x$  at time  $t$  if it has direct or indirect knowledge of  $x$  at time  $t$ . An agent with direct knowledge of  $x$  at time  $t$  has visited location  $x$  at a time  $t' \leq t$ . An agent has indirect knowledge of  $x$  at time  $t$  if it can be notified of  $x$  at a time  $t' \leq t$ .*

The direct knowledge of an agent depends only on its own trajectory, whereas an agent's indirect knowledge depends on both its own and the other agents' trajectories. We define the direct knowledge set  $K_X^D(t)$  as the set of all locations that the agent with trajectory  $X$  has direct knowledge of at time  $t$ . We similarly define the indirect knowledge set  $K_X^I(t; \mathcal{A})$ . The (total) knowledge set is the set  $K_X(t; \mathcal{A}) = K_X^D(t) \cup K_X^I(t; \mathcal{A})$ . We make the following simple observation which results from the unit speed assumption.

► **Observation 12.**  $K_X^D(t) \subseteq [X(t) - t, X(t) + t]$ .

We can use the knowledge set of an agent to lower bound the competitive ratio.

► **Lemma 13.** *For any evacuation algorithm  $\mathcal{A}$ , any  $X \in \mathcal{A}$ , and any time  $t > 0$  we have*

$$CR(\mathcal{A}) \geq \sup_{x \notin K_X(t; \mathcal{A})} \frac{|X(t) - x| + t}{|x|}.$$

Thus, we can derive a lower bound on the competitive ratio by bounding the size of an agent's knowledge set.

We define the functional  $\mu(X)$  which maps a search trajectory to a non-negative real number:

$$\mu(X) =: \limsup_{t \rightarrow \infty} \frac{|X(t)|}{t}. \quad (4)$$

The quantity  $\mu(X)$  can be thought of as an upper bound on the average rate at which the direct knowledge of an agent with trajectory  $X$  grows. We naturally extend the definition of  $\mu$  to take as input an evacuation algorithm:

$$\mu(\mathcal{A}) := \max_{X \in \mathcal{A}} \mu(X). \quad (5)$$

In Appendix B we establish several properties of  $\mu(X)$  and  $K_X$  (and relationships between them) for trajectories  $X$  in evacuation algorithms. These are used in the proofs of the following theorems, from which Theorem 10 follows.

► **Theorem 14.** *Let  $\mathcal{A}$  be an evacuation for one sender and one receiver. Then  $CR(\mathcal{A}) \geq 3 + 2\sqrt{2}$ .*

**Proof.** Let  $\mathcal{A} = \{S, R\}$  with  $S$  and  $R$  the trajectories of the sender and receiver respectively. We must have  $\mu(S) < \mu(\mathcal{A}) = \mu(R)$  since otherwise the competitive ratio is at least 9. Then, Lemma 26 states that

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{(1 + \mu(R))(1 + \mu(S))}{\mu(R)(1 - \mu(S))}$$

and from Lemma 29 we have

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{(1 + \mu(S))(1 + \mu(R))}{\mu(S)(1 + \mu(S))} = \frac{1 + \mu(R)}{\mu(S)}$$

where we have used the fact that  $\mathcal{A}' = \{S\}$  when there is only one receiver. We therefore have

$$\text{CR}(\mathcal{A}) \geq 1 + \max \left\{ \frac{(1 + \mu(R))(1 + \mu(S))}{\mu(R)(1 - \mu(S))}, \frac{1 + \mu(R)}{\mu(S)} \right\}.$$

The first term in the max decreases with  $\mu(R)$  and the second term increases with  $\mu(R)$  and so our best-lower bound is achieved when increasing  $\mu(R)$  is such that the two terms are equal. We find that we need

$$\mu(R) = \frac{\mu(S)(1 + \mu(S))}{1 - \mu(S)}.$$

We then find that

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{1 + \frac{\mu(S)(1 + \mu(S))}{1 - \mu(S)}}{\mu(S)} = 1 + \frac{1}{\mu(S)} + \frac{1 + \mu(S)}{1 - \mu(S)}.$$

Let  $g(u) = \frac{1}{u} + \frac{1+u}{1-u}$  and observe that

$$\begin{aligned} \frac{dg(u)}{du} &= -\frac{1}{u^2} + \frac{1}{1-u} + \frac{1+u}{(1-u)^2} \\ &= -\frac{1}{u^2} + \frac{2}{(1-u)^2} = \frac{(1-u)^2 - 2u^2}{u^2(1-u)^2} = \frac{(1-u-\sqrt{2}u)(1-u+\sqrt{2}u)}{u^2(1-u)^2} \\ &= \frac{[1 - (1 + \sqrt{2})u][1 - (1 - \sqrt{2})u]}{u^2(1-u)^2}. \end{aligned}$$

From this last expression it is clear that  $g(u)$  is minimized when  $u = \frac{1}{1+\sqrt{2}} = \sqrt{2} - 1$ . The minimum is

$$g(\sqrt{2} - 1) = \sqrt{2} + 1 + \frac{\sqrt{2}}{2 - \sqrt{2}} = 2 + 2\sqrt{2}$$

and we can conclude that

$$\text{CR}(\mathcal{A}) \geq 3 + 2\sqrt{2}. \quad \blacktriangleleft$$

► **Corollary 15.** *The evacuation algorithm for one sender and one receiver given in the proof of Theorem 1 is optimal.*

► **Theorem 16.** *Let  $\mathcal{A}$  be an evacuation for one sender and  $n_r > 1$  receivers. Then  $\text{CR}(\mathcal{A}) \geq 2 + \sqrt{5}$ .*

## 57:12 Group Evacuation on a Line by Agents with Different Communication Abilities

**Proof.** Let  $\mathcal{A} = \{S, R, R', \dots\}$  with  $S$  the trajectory of the sender,  $R$  the trajectory of the receiver with largest value of  $\mu(R)$ , and  $R'$  the trajectory of the receiver with second largest  $\mu(R')$ . We must have  $\mu(S) < \mu(\mathcal{A}) = \mu(R)$  since otherwise the competitive ratio is at least 9. Then, Lemma 26 states that

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{(1 + \mu(R))(1 + \mu(S))}{\mu(R)(1 - \mu(S))}$$

and from Lemma 29 we have

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{(1 + \mu(\mathcal{A}'))(1 + \mu(R))}{\mu(\mathcal{A}')(1 + \mu(S))}.$$

If  $\mu(\mathcal{A}') = \mu(S)$  then it follows from Theorem 14 that we will have  $\text{CR}(\mathcal{A}) \geq 3 + 2\sqrt{2}$ . Thus, we assume that  $\mu(S) < \mu(\mathcal{A}')$ . Then  $\mu(R') = \mu(\mathcal{A}')$  and we have

$$\text{CR}(\mathcal{A}) \geq 1 + \max \left\{ \frac{(1 + \mu(R))(1 + \mu(S))}{\mu(R)(1 - \mu(S))}, \frac{(1 + \mu(R'))(1 + \mu(R))}{\mu(R')(1 + \mu(S))} \right\}.$$

The second term in the max increases with decreasing  $\mu(R')$  and the first term does not depend on  $\mu(R')$ . Thus, we set  $\mu(R')$  as large as possible, i.e., we take  $\mu(R') = \mu(R)$ . Then

$$\text{CR}(\mathcal{A}) \geq 1 + \max \left\{ \frac{(1 + \mu(R))(1 + \mu(S))}{\mu(R)(1 - \mu(S))}, \frac{(1 + \mu(R))^2}{\mu(R)(1 + \mu(S))} \right\}.$$

Now both terms in the max increase with decreasing  $\mu(R)$  and so we take  $\mu(R)$  as large as possible, i.e.,  $\mu(R) = 1$ . Then

$$\text{CR}(\mathcal{A}) \geq 1 + 2 \max \left\{ \frac{1 + \mu(S)}{1 - \mu(S)}, \frac{2}{1 + \mu(S)} \right\}.$$

The first term in the max increases with  $\mu(S)$  and the second term decreases with  $\mu(S)$  and so our best-lower bound is achieved when  $\mu(S)$  is such that the two terms are equal. We find that we need  $(1 + \mu(S))^2 = 2(1 - \mu(S))$  or

$$\mu(S)^2 + 4\mu(S) - 1 = 0.$$

The only non-negative solution to this quadratic equation is

$$\mu(S) = \frac{-4 + \sqrt{20}}{2} = \sqrt{5} - 2$$

and we can conclude that

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{4}{\sqrt{5} - 1} = 1 + \frac{4(\sqrt{5} + 1)}{4} = 2 + \sqrt{5}$$

as required. ◀

Our upper bound for the case that  $n_s = 1$  and  $n_r > 1$  was  $5 > 2 + \sqrt{5}$  and so either the lower bound is not tight and/or the upper bound must come down. If one refers to the proof of Theorem 16 then one can observe that our best lower bound was achieved when  $\mu_s = \sqrt{5} - 2$  and  $\mu_1 = \mu_2 = 1$ . However, one can easily confirm that any algorithm with  $\mu_1 = \mu_2 = 1$  has a competitive ratio of at least 5 and so it is evident that, at least, the lower bound is not tight. Thus, in order to make progress on this problem, we believe a different lower bounding technique will be required.

## 4 Conclusions

We have introduced a novel communication model that puts an interesting twist on the classic linear group search problem. We provide upper bounds on the evacuation for the three interesting combinations of agents – one sender and one receiver, one sender and multiple receivers, and multiple senders and one receiver. We demonstrate that our algorithm for the case of one sender and one receiver is optimal by providing a lower bound matching our upper bound. For the case of one sender and two receivers we provide a non-trivial lower bound of  $2 + \sqrt{5}$  which compares to our upper bound of 5. We do not provide any non-trivial lower bounds for the case of multiple senders and one receiver and it is believed that this is the most difficult case to do so (indeed, the upper bound for this case was considerably more complex than the other two cases).

The most immediate open problems concern the lower bounds for the cases of multiple senders and multiple receivers. For the multiple receiver case we provided arguments demonstrating that the lower bound presented here cannot be tight and so in order to close the gap between the lower and upper bounds a different lower bounding technique will be required. Of course, it can also be the case that the upper bound must come down as well (although this does not seem likely). We did not attempt to provide a lower bound for the case of multiple senders (there is a trivial lower bound of 3 which can be derived by considering the first time any agent reaches location  $\pm x$ ).

Our upper bounds on the evacuation seem to hint at the fact that it is better to “listen” than it is to “speak” since our upper bound for the case of multiple receivers is 5 and for multiple senders it is  $\approx 5.681319$  (and we do not believe that these can be improved). Closing the gap between the upper and lower bounds would be interesting even just from the standpoint of answering the question of whether or not it is better to “listen” than it is to “speak”.

---

## References

- 1 S. Alpern and S. Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.
- 2 R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.
- 3 R. Baeza-Yates and R. Schott. Parallel searching in the plane. *Computational Geometry*, 5(3):143–154, 1995.
- 4 I. Bagheri, L. Narayanan, and J. Opatrny. Evacuation of equilateral triangles by mobile agents of limited communication range. In *ALGOSENSORS 2019*, pages 3–22, 2019.
- 5 E. Bampas, J. Czyzowicz, L. Gąsieniec, D. Ilcinkas, R. Klasing, T. Kociumaka, and D. Pająk. Linear search by a pair of distinct-speed robots. *Algorithmica*, 81(1):317–342, 2019.
- 6 A. Beck. On the linear search problem. *Israel J. of Mathematics*, 2(4):221–228, 1964.
- 7 A. Beck. More on the linear search problem. *Israel J. of Mathematics*, 3(2):61–70, 1965.
- 8 A. Beck and D. Newman. Yet more on the linear search problem. *Israel J. of Mathematics*, 8(4):419–429, 1970.
- 9 R. Bellman. An optimal search. *SIAM Review*, 5(3):274–274, 1963.
- 10 M. Chrobak, L. Gąsieniec, Gorry T., and R. Martin. Group search on the line. In *SOFSEM 2015*, pages 164–176, Czech Republic, 2015. Springer.
- 11 J. Czyzowicz, K. Georgiou, R. Killick, E. Kranakis, D. Krizanc, M. Lafond, L. Narayanan, J. Opatrny, and S. Shende. Energy consumption of group search on a line. In *ICALP 2019*, pages 137:1–137:15, Patras, Greece, 2019. LIPIcs.

- 12 J. Czyzowicz, K. Georgiou, R. Killick, E. Kranakis, D. Krizanc, M. Lafond, L. Narayanan, J. Opatrny, and S. Shende. Time-energy tradeoffs for evacuation by two robots in the wireless model. *Theoretical Computer Science*, 852:61–72, 2021.
- 13 J. Czyzowicz, K. Georgiou, and E. Kranakis. Group search and evacuation. In *Special Issue of Moving and Computing, La Maddalena, Italy*, NY, 5-9 June, 2017. Springer.
- 14 J. Czyzowicz, K. Georgiou, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, and S. Shende. Search on a line by byzantine robots. In *ISAAC 2016*, pages 27:1–27:12, Toronto, Canada, 2016. LIPIcs.
- 15 J. Czyzowicz, R. Killick, E. Kranakis, and G. Stachowiak. Search and evacuation with a near majority of faulty agents. In *1st SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, USA, July 19–21, 2021. SIAM.
- 16 J. Czyzowicz, E. Kranakis, D. Krizanc, L. Narayanan, and Opatrny J. Search on a line with faulty robots. In *PODC 2016*, pages 405–414, Chicago, Illinois, 2016. ACM.
- 17 E.D. Demaine, S.P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361(2):342–355, 2006.
- 18 B. Fristedt. Hide and seek in a subset of the real line. *International Journal of Game Theory*, 6(3):135–165, 1977.
- 19 B. Fristedt and D. Heath. Searching for a particle on the real line. *Advances in Applied Probability*, 6(1):79–102, 1974. doi:10.2307/1426208.
- 20 S. Gal. A general search game. *Israel Journal of Mathematics*, 12(1):32–45, 1972.
- 21 R. Killick. *Search and Rendezvous by Mobile Robots in Continuous Domains*. PhD thesis, School of Computer Science, Carleton University, June 2021.
- 22 A. Kupavskii and E. Welzl. Lower bounds for searching robots, some faulty. In *PODC 2018*, pages 447–453, Egham, United Kingdom, 2018. ACM.
- 23 X. Sun, Y. Sun, and J. Zhang. Better upper bounds for searching on a line with byzantine robots. In *Complexity and Approximation*, pages 151–171. Springer, 2020.

## A Proofs for Section 2 (Evacuation Algorithms and their Competitive Ratios)

► **Theorem 17** (Theorem 2 restated). *When  $n_s = 1$  and  $n_r > 1$  there exists an evacuation algorithm with competitive ratio 5.*

**Proof.** The proof is constructive and based on the following algorithm: one receiver moves to the left at unit speed and one receiver moves to the right at unit speed. When one of the receivers finds the target it immediately moves to notify the sender (and all other agents) at the origin. The sender immediately notifies the remaining receiver, and all agents proceed to the target at unit speed. An illustration of this algorithm is provided in Figure 4 for the case that the target is at location  $-x < -1$ . The situation is symmetric when  $x > 1$ .

Suppose that the target is at location  $-x < -1$ . The left receiver will find this target first and will do so at the time  $x$ . It then immediately moves to the origin to notify the sender, arriving at time  $2x$ . The sender notifies the right receiver who is at location  $2x$ . The right receiver then moves at unit speed to the target arriving at time  $5x$ . The competitive ratio is thus 5. The case when  $x > 1$  is totally symmetric and also yields a competitive ratio of 5. ◀

► **Lemma 18** (Lemma 4 restated). *The turning-points  $(D_j, T_j)$ ,  $j = 0, 1, \dots$ , of the trajectory  $\text{RAYS}(\eta, v_0, v_1, \gamma)$  are given by*

$$D_j = \eta\gamma \left[ \frac{(1-v_1)(1+v_0)}{(1+v_1)(1-v_0)} \right]^{\lfloor \frac{j}{2} \rfloor} \begin{cases} 1, & \text{even } j \\ \frac{v_1(1+v_0)}{v_0(1+v_1)}, & \text{odd } j \end{cases}, \quad T_j = \frac{D_j}{\eta} \begin{cases} \frac{1}{v_0}, & \text{even } j \\ \frac{1}{v_1}, & \text{odd } j. \end{cases}$$



57:16 Group Evacuation on a Line by Agents with Different Communication Abilities

Unrolling this recursion then gives

$$D_j = \left[ \frac{(1-v_1)(1+v_0)}{(1+v_1)(1-v_0)} \right]^{\lfloor \frac{j}{2} \rfloor} \begin{cases} D_0, & \text{even } j \\ D_1, & \text{odd } j \end{cases} = \gamma \left[ \frac{(1-v_1)(1+v_0)}{(1+v_1)(1-v_0)} \right]^{\lfloor \frac{j}{2} \rfloor} \begin{cases} 1, & \text{even } j \\ \frac{v_1(1+v_0)}{v_0(1+v_1)}, & \text{odd } j \end{cases}$$

where we have used the fact that  $D_0 = \gamma$ , and our expression for  $D_j$  when  $j$  is odd. ◀

► **Lemma 19** (Lemma 6 restated). *The turning-points of the receiver are*

$$D_j^r = (-1)^j \left( \frac{1+v_r}{1-v_r} \right)^j, \quad T_j^r = \frac{1}{v_r} \left( \frac{1+v_r}{1-v_r} \right)^j.$$

**Proof.** With  $X_r(t) = \text{RAYS}(1, -v_r, v_r, 1)$  it follows from Lemma 4 that for even  $j$  we have

$$D_j^r = \left[ \frac{(1+v_r)(1+v_r)}{(1-v_r)(1-v_r)} \right]^{\lfloor \frac{j}{2} \rfloor} = \left( \frac{1+v_r}{1-v_r} \right)^{2\lfloor \frac{j}{2} \rfloor} = \left( \frac{1+v_r}{1-v_r} \right)^j.$$

and for odd  $j$  we similarly find that  $D_j^r = -\left( \frac{1+v_r}{1-v_r} \right)^j$ . Thus, for  $j$  even or odd we have

$D_j^r = (-1)^j \left( \frac{1+v_r}{1-v_r} \right)^j$ . The times  $T_j^r$  are

$$T_j^r = D_j^r \begin{cases} \frac{1}{v_r}, & \text{even } j \\ -\frac{1}{v_r}, & \text{odd } j. \end{cases} = (-1)^j \frac{D_j^r}{v_r} = \frac{1}{v_r} \left( \frac{1+v_r}{1-v_r} \right)^j.$$

◀

We note the following identities concerning the turning-points  $(D_j^r, T_j^r)$  which we will use these identities often and without reference.

$$T_j^r = \frac{1+v_r}{1-v_r} T_{j-1}^r = \frac{1-v_r}{1+v_r} T_{j+1}^r, \quad D_j^r = -\frac{1+v_r}{1-v_r} D_{j-1}^r = -\frac{1-v_r}{1+v_r} D_{j+1}^r.$$

► **Lemma 20** (Lemma 7 restated). *The turning-points of the right-sender are*

$$D_j^+ = v_r T_j^r \begin{cases} \frac{3-v_r}{1-v_r}, & \text{even } j \\ \frac{1-v_r}{1+v_r}, & \text{odd } j \end{cases}, \quad T_j^+ = T_j^r \begin{cases} \frac{1+v_r}{1-v_r}, & \text{even } j \\ \frac{1+3v_r}{1+v_r}, & \text{odd } j \end{cases}$$

**Proof.** With  $X_+(t) = \text{RAYS}(1, v_0, v_1, \gamma_+)$  it follows from Lemma 4 that

$$D_j^+ = \gamma_+ \left[ \frac{(1-v_1)(1+v_0)}{(1+v_1)(1-v_0)} \right]^{\lfloor \frac{j}{2} \rfloor} \begin{cases} 1, & \text{even } j \\ \frac{v_1(1+v_0)}{v_0(1+v_1)}, & \text{odd } j \end{cases}, \quad T_j^+ = D_j^+ \begin{cases} \frac{1}{v_0}, & \text{even } j \\ \frac{1}{v_1}, & \text{odd } j. \end{cases}$$

With  $v_0$  and  $v_1$  given by (3) we have

$$\begin{aligned} \frac{(1-v_1)(1+v_0)}{(1+v_1)(1-v_0)} &= \frac{\left(1 - \frac{v_r(1-v_r)}{1+3v_r}\right) \left(1 + \frac{v_r(3-v_r)}{1+v_r}\right)}{\left(1 + \frac{v_r(1-v_r)}{1+3v_r}\right) \left(1 - \frac{v_r(3-v_r)}{1+v_r}\right)} \\ &= \frac{(1+2v_r+v_r^2)(1+3v_r+v_r(1-v_r))}{(1+3v_r+v_r(1-v_r))(1-2v_r+v_r^2)} = \left( \frac{1+v_r}{1-v_r} \right)^2 \end{aligned}$$



We also observe that

$$\begin{aligned} \frac{v_1(1+v_0)}{v_0(1+v_1)} &= \frac{\frac{v_r(1-v_r)}{1+3v_r} \left(1 + \frac{v_r(3-v_r)}{1+v_r}\right)}{\frac{v_r(3-v_r)}{1+v_r} \left(1 + \frac{v_r(1-v_r)}{1+3v_r}\right)} \\ &= \frac{v_r(1-v_r)(1+v_r+v_r(3-v_r))}{v_r(3-v_r)(1+3v_r+v_r(1-v_r))} = \frac{1-v_r}{3-v_r} = \frac{1}{\gamma_+}. \end{aligned}$$

Substituting these last two results into our expression for  $D_j^+$  then yields

$$D_j^+ = \left(\frac{1+v_r}{1-v_r}\right)^{2\lfloor \frac{j}{2} \rfloor} \begin{cases} \gamma_+, & \text{even } j \\ 1, & \text{odd } j \end{cases} = \begin{cases} \gamma_+ \left(\frac{1+v_r}{1-v_r}\right)^j, & \text{even } j \\ \left(\frac{1+v_r}{1-v_r}\right)^{j-1}, & \text{odd } j \end{cases} = v_r T_j^r \begin{cases} \frac{3-v_r}{1-v_r}, & \text{even } j \\ \frac{1-v_r}{1+v_r}, & \text{odd } j \end{cases}$$

as required.

For  $T_j^+$  we have

$$T_j^+ = D_j^+ \begin{cases} \frac{1}{v_0}, & \text{even } j \\ \frac{1}{v_1}, & \text{odd } j. \end{cases} = v_r \begin{cases} \frac{\gamma_+ T_j^r}{v_0}, & \text{even } j \\ \frac{1}{v_1} T_{j-1}^r, & \text{odd } j. \end{cases}$$

We observe that

$$\frac{\gamma_+}{v_0} = \frac{\frac{3-v_r}{1-v_r}}{\frac{v_r(3-v_r)}{1+v_r}} = \frac{1}{v_r} \left(\frac{1+v_r}{1-v_r}\right)$$

and thus

$$T_j^+ = v_r \begin{cases} \frac{1}{v_r} \left(\frac{1+v_r}{1-v_r}\right) T_j^r, & \text{even } j \\ \frac{1+3v_r}{v_r(1-v_r)} T_{j-1}^r, & \text{odd } j \end{cases} = \begin{cases} T_{j+1}^r, & \text{even } j \\ \frac{1+3v_r}{1-v_r} T_{j-1}^r, & \text{odd } j \end{cases} = T_j^r \begin{cases} \frac{1+v_r}{1-v_r}, & \text{even } j \\ \frac{1+3v_r}{1+v_r}, & \text{odd } j \end{cases}.$$

This completes the proof. ◀

► **Lemma 21** (Lemma 8). *The turning-points of the left-sender are*

$$D_j^- = -v_r T_{j+1}^r \begin{cases} \frac{3-v_r}{1-v_r}, & \text{even } j \\ \frac{1-v_r}{1+v_r}, & \text{odd } j \end{cases}, \quad T_j^- = T_{j+1}^r \begin{cases} \frac{1+v_r}{1-v_r}, & \text{even } j \\ \frac{1+3v_r}{1+v_r}, & \text{odd } j \end{cases}.$$

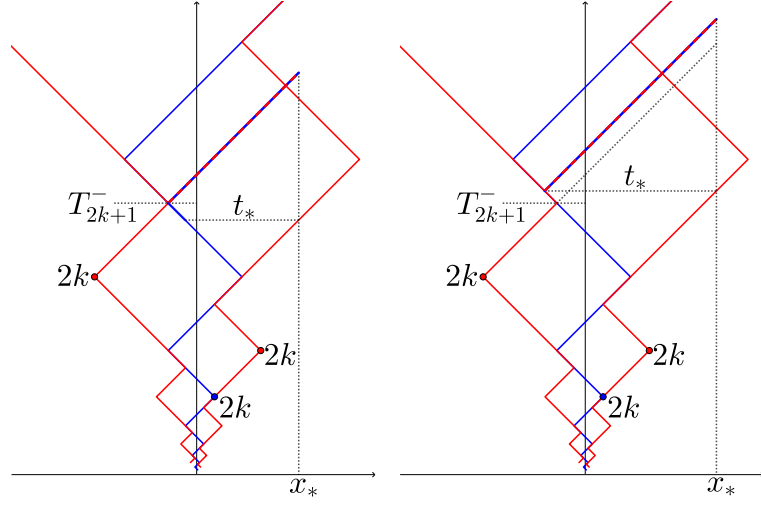
**Proof.** The proof is essentially identical to the proof of Lemma 7. ◀

► **Observation 22** (Observation 5 restated). *For all  $k = 0, 1, 2, \dots$  the following properties hold for the algorithm  $EVAC_{RAVS}(v_r)$ :*

1. *the receiver reaches its turning point  $2k+1$  (resp.  $2k+2$ ) at the same time the right-sender (resp. left-sender) reaches its turning-point  $2k$ .*
2. *the receiver and right-sender (resp. left-sender) are co-located at all times in the interval  $[T_{2k+1}^+, T_{2k+2}^r]$  (resp.  $[T_{2k+1}^-, T_{2k+3}^r]$ ),*

**Proof.** (Observation 5) We will prove the properties for the right-sender only. Those for the left-sender follow in a nearly identical manner.

The first statement we want to prove is: “the receiver reaches its turning point  $2k+1$  at the same time the right-sender reaches its turning-point  $2k$ ”. The receiver reaches its turning-point  $2k+1$  at time  $T_{2k+1}^r$ . The right-sender reaches its turning-point  $2k$  at time  $T_{2k}^+$  and by Lemma 7 we have  $T_{2k}^+ = \frac{1+v_r}{1-v_r} T_{2k}^r = T_{2k+1}^r$ , which proves the statement.



■ **Figure 5** Setup for the proof of Theorem 9. The turning-point  $2k$  of each agent is indicated. On the left the target is found at time  $t_* \leq T_{2k+1}^-$  and on the right the target is found at time  $t_* > T_{2k+1}^-$ .

The second statement we want to prove is: “the receiver and right-sender are co-located at all times in the interval  $[T_{2k+1}^+, T_{2k+2}^+]$ ”. During the interval  $[T_{2k+1}^+, T_{2k+2}^+]$  the right-sender will be moving to the right along the space-time line

$$t = x - D_{2k+1}^+ + T_{2k+1}^+ = x - v_r \frac{1 - v_r}{1 + v_r} T_{2k+1}^r + \frac{1 + 3v_r}{1 + v_r} T_{2k+1}^r = x + \frac{1 + 2v_r + v_r^2}{1 + v_r} T_{2k+1}^r$$

and finally

$$t = x + (1 + v_r) T_{2k+1}^r. \quad (6)$$

During the interval  $[T_{2k+1}^r, T_{2k+2}^r]$  the receiver will be moving to the right along the space-time line

$$t = x - D_{2k+2}^r + T_{2k+2}^r = x - v_r T_R(k+2) + T_R^{2k+2} = x + (1 - v_r) T_{2k+2}^r = x + (1 + v_r) T_{2k+1}^r.$$

We can thus conclude that the right-sender and receiver will be travelling along the same space-time line and will be co-located during the interval  $[T_{2k+1}^+, T_{2k+2}^+] \cap [T_{2k+1}^r, T_{2k+2}^r] = [T_{2k+1}^+, T_{2k+2}^r]$ . ◀

▶ **Theorem 23** (Theorem 9 restated). *The competitive ratio of algorithm  $EVAC_{\text{RAYS}}(v_r)$  satisfies*

$$CR \leq 1 + \frac{1 + v_r}{1 - v_r} \left( \frac{1 + 4v_r - v_r^2}{v_r(3 - v_r)} \right).$$

**Proof.** Due to the symmetry between the right/left-senders, we may assume without loss of generality that the target is found by the right-sender. Moreover, the sequence of intervals  $(D_{2k}^+, D_{2k+2}^+]$ ,  $k = 0, 1, 2, \dots$ , collectively covers the entire line extending from  $D_0^+$  to  $+\infty$  and so we may assume without loss of generality that the target is at location  $x_* \in (D_{2k}^+, D_{2k+2}^+]$ , for some fixed value of  $k \geq 0$ .

The right-sender will reach  $x_*$  while travelling to the right between its turning points  $2k + 1$  and  $2k + 2$ , and, we demonstrated in the proof of Observation 5 that while doing so this sender will be moving along the space-time line with equation (6). Thus, the time  $t_*$  at which the right-sender reaches the target is

$$t_* = x_* + (1 + v_r)T_{2k+1}^r.$$

After reaching the target the right-sender will wirelessly notify the receiver and the receiver will move to notify the left-sender. There are two cases to consider, each of which is illustrated in Figure 5. In the first case – left side of Figure 5 – the target is found at location  $x_*$  such that  $t_* \leq T_{2k+1}^-$ . We know from Observation 5 that the receiver will be co-located with the left-receiver at all times within the interval  $[T_{2k+1}^-, T_{2k+3}^r]$ , and before time  $T_{2k+1}^-$  the receiver and left-sender will be moving towards each other, each at unit speed. Thus, the earliest time that the left-sender could be notified of the target is at the time  $T_{2k+1}^-$ . Evidently, the evacuation time for this case is

$$\begin{aligned} E &= T_{2k+1}^- + |x_* - D_{2k+1}^-| = x_* + \frac{1 + 3v_r}{1 + v_r}T_{2k+2}^r + v_r \frac{1 - v_r}{1 + v_r}T_{2k+2}^r \\ &= x_* + \frac{1 + 3v_r + v_r(1 - v_r)}{1 + v_r}T_{2k+2}^r. \end{aligned}$$

and the competitive ratio is

$$\text{CR} = \frac{E}{x_*} = 1 + \frac{\frac{1+3v_r+v_r(1-v_r)}{1+v_r}T_{2k+2}^r}{x_*}.$$

The competitive ratio increases with decreasing  $x_*$ , and with  $x_* > D_{2k}^+ = v_r \frac{3-v_r}{1-v_r}T_{2k}^r$  we get

$$\begin{aligned} \text{CR} &\leq 1 + \frac{1 + 3v_r + v_r(1 - v_r)}{1 + v_r} \frac{T_{2k+2}^r}{v_r \frac{3-v_r}{1-v_r}T_{2k}^r} \\ &= 1 + \frac{1 + 3v_r + v_r(1 - v_r)}{1 + v_r} \cdot \frac{1 - v_r}{v_r(3 - v_r)} \cdot \frac{(1 + v_r)^2}{(1 - v_r)^2} \\ &= 1 + \frac{1 + v_r}{1 - v_r} \left( \frac{1 + 3v_r + v_r(1 - v_r)}{v_r(3 - v_r)} \right) = 1 + \frac{1 + v_r}{1 - v_r} \left( \frac{1 + 4v_r - v_r^2}{v_r(3 - v_r)} \right). \end{aligned}$$

and finally

$$\text{CR} \leq 1 + \frac{1 + v_r}{1 - v_r} \left( \frac{1 + 4v_r - v_r^2}{v_r(3 - v_r)} \right). \tag{7}$$

The second case – the right side of Figure 5 – occurs when the target is found at a time  $t_* \in (T_{2k+1}^-, T_{2k+2}^+) = (T_{2k+1}^-, T_{2k+3}^r]$ . The left-sender and receiver are co-located during the time interval  $(T_{2k+1}^-, T_{2k+3}^r]$ , and so the left-sender will be notified of the target at time  $t_*$ . By referring to Figure 5 one can observe that the evacuation time for this case will be  $2(t_* - T_{2k+1}^-)$  more than the evacuation time of the previous case, i.e.,

$$E = x_* + \frac{1 + 3v_r + v_r(1 - v_r)}{1 + v_r}T_{2k+2}^r + 2(t_* - T_{2k+1}^-).$$

Since  $t_* = x_* + (1 + v_r)T_{2k+1}^r = x_* + (1 - v_r)T_{2k+2}^r$  and  $T_{2k+1}^- = \frac{1+3v_r}{1+v_r}T_{2k+2}^r$  we get

$$\begin{aligned} E &= 3x_* + \frac{1 + 3v_r + v_r(1 - v_r)}{1 + v_r}T_{2k+2}^r + 2 \left( (1 - v_r) - \frac{1 + 3v_r}{1 + v_r} \right) T_{2k+2}^r \\ &= 3x_* + \frac{1 + 3v_r + v_r(1 - v_r) + 2(1 + v_r)(1 - v_r) - 2(1 + 3v_r)}{1 + v_r}T_{2k+2}^r \\ &= 3x_* + \frac{-(1 + 3v_r) + (2 + 3v_r)(1 - v_r)}{1 + v_r}T_{2k+2}^r \\ &= 3x_* + \frac{1 - v_r(2 + 3v_r)}{1 + v_r}T_{2k+2}^r \end{aligned}$$

and the competitive ratio is

$$\text{CR} = 3 + \frac{1 - v_r(2 + 3v_r)}{1 + v_r} \frac{T_{2k+2}^r}{x_*}$$

When  $v_r(2 + 3v_r) \geq 1$  the competitive ratio is  $\leq 3$ . When  $v_r(2 + 3v_r) < 1$  the competitive ratio is  $> 3$  and increases with decreasing  $x_*$ , or, equivalently, with decreasing  $t_*$ . Thus, we should take  $t_*$  arbitrarily close to  $T_{2k+1}^-$ . However,  $t_* = T_{2k+1}^-$  gave the best-case evacuation time for the case that  $t_* \leq T_{2k+1}^-$ . We can thus conclude that a worst-case competitive ratio can be achieved when  $t_* \leq T_{2k+1}^-$  and the competitive ratio of the algorithm is upper bounded by (7).  $\blacktriangleleft$

## B Proofs for Section 3 (Lower bounds)

► **Lemma 24.** *Let  $X$  be a search trajectory. If  $\mu(X) < 1$  then for all  $0 < \epsilon < 1 - \mu(X)$  there exists a time  $T > 0$  such that*

$$K_X^D(t) \subseteq \left[ -\frac{(\mu(X) + \epsilon)(t - X(t))}{1 + \mu(X) + \epsilon}, \frac{(\mu(X) + \epsilon)(t + X(t))}{1 + \mu(X) + \epsilon} \right], \quad \forall t > T.$$

*In the case of  $\mu(X) = 1$  the parameter  $\epsilon$  can be taken to be 0 in the above expression.*

**Proof.** By the definition of  $\mu(X)$ , it follows that for all  $\epsilon > 0$  there exists a time  $T' > 0$  such that

$$-(\mu(X) + \epsilon)t \leq X(t) \leq (\mu(X) + \epsilon)t, \quad \forall t > T'. \quad (8)$$

Moreover, when  $\mu(X) = 1$  the  $\epsilon$  can be taken to be 0, since  $|X(t)| \leq t$ .

In order to have direct knowledge of location  $x$  at time  $t > T'$  there must exist a time  $t' \leq t$  such that  $X(t') = x$ . The unit speed of the agents implies that  $|X(t) - x| \leq t - t'$  or

$$t' - t + X(t) \leq x \leq t + X(t) - t'. \quad (9)$$

Assume that  $t' > T'$ . Then we can combine (8) and (9) to get

$$\max\{-(\mu(X) + \epsilon)t', t' - t + X(t)\} \leq x \leq \min\{(\mu(X) + \epsilon)t', t + X(t) - t'\}. \quad (10)$$

On the left, the first term in the max decreases with  $t'$  and the second term increases with  $t'$ . Thus, the best lower bound is achieved when the two terms are equal. This will occur when

$$t' = \frac{t - X(t)}{1 + \mu(X) + \epsilon}.$$

For this value of  $t'$  we get

$$x \geq -\frac{(\mu(X) + \epsilon)(t - X(t))}{1 + \mu(X) + \epsilon}.$$

At time  $t$  we have  $X(t) \leq (\mu(X) + \epsilon)t$  and thus

$$t' = \frac{t - X(t)}{1 + \mu(X) + \epsilon} \geq \frac{t - (\mu(X) + \epsilon)t}{1 + \mu(X) + \epsilon} = \frac{1 - \mu(X) - \epsilon}{1 + \mu(X) + \epsilon}t.$$

Hence, we will have  $t' > T'$  for all

$$t > T = \frac{1 + \mu(X) + \epsilon}{1 - \mu(X) - \epsilon}T'. \quad (11)$$

When  $\mu(X) = 1$  the above expression is vacuously true, since  $T'$  can be taken to be 0.

In a similar manner, we get from the right side of (10) that

$$x \leq \frac{(\mu(X) + \epsilon)(t + X(t))}{1 + \mu(X) + \epsilon}.$$

for all  $t$  satisfying (11). This completes the proof.  $\blacktriangleleft$

In a similar manner we can bound the total knowledge available to an agent that can only receive messages face-to-face.

► **Lemma 25.** *Let  $\mathcal{A}$  be an evacuation algorithm and let  $X_{f2f} \in \mathcal{A}$  represent the trajectory of an agent that can only receive messages face-to-face. If  $\mu(X_{f2f}) < 1$  then for all  $0 < \epsilon < 1 - \mu(X_{f2f})$  there exists a time  $T > 0$  such that*

$$K_{X_{f2f}}(t; \mathcal{A}) \subseteq \left[ -\frac{(\mu(\mathcal{A}) + \epsilon)(t - X_{f2f}(t))}{1 + \mu(\mathcal{A}) + \epsilon}, \frac{(\mu(\mathcal{A}) + \epsilon)(t + X_{f2f}(t))}{1 + \mu(\mathcal{A}) + \epsilon} \right], \quad \forall t > T.$$

In the case of  $\mu(X_{f2f}) = 1$  the parameter  $\epsilon$  can be taken to be 0 in the above expression.

**Proof.** When  $\mu(X) < 1$  it follows from the definitions of  $\mathcal{A}$  and  $\mu(X)$  that there exists a time  $T' > 0$  such that for any  $X \in \mathcal{A}$  we have

$$-(\mu(\mathcal{A}) + \epsilon)t \leq X(t) \leq (\mu(\mathcal{A}) + \epsilon)t, \quad \forall t > T', \quad \forall X \in \mathcal{A}. \quad (12)$$

In order to have direct knowledge of location  $x$  at time  $t > T'$  there must exist a time  $t' \leq t$  such that  $X_{f2f}(t') = x$ . The unit speed of the agents implies that  $|X_{f2f}(t) - x| \leq t - t'$  or that (9) must be satisfied by the trajectory  $X_{f2f}$  at time  $t$ .

In order to have indirect knowledge of location  $x$  at time  $t$  there must exist another agent that visits  $x$  at time  $t' \leq t$  and can reach location  $X_{f2f}(t)$  by time  $t$ . Indeed, this agent must be able to catch the agent with trajectory  $X_{f2f}$  at or before time  $t$ , and the agent with trajectory  $X_{f2f}$  will be at location  $X_{f2f}(t)$  at time  $t$ . Thus, in order to have indirect knowledge of  $x$ , the unit speed condition implies again that  $|X_{f2f}(t) - x| \leq t - t'$  or that (9) is satisfied. To complete the proof we follow the same steps of the proof of Lemma 24 except with (12) used in place of (8).  $\blacktriangleleft$

We will now focus on the case that there is only a single sender involved in the evacuation. The sender can only be communicated with face-to-face and so Lemma 25 applies in this case. We can use it to get the following result.

► **Lemma 26.** *Let  $\mathcal{A}$  be an evacuation algorithm with one sender and let  $S \in \mathcal{A}$  represent the trajectory of this sender. If  $\mu(S) = 1$  then  $CR(\mathcal{A})$  is unbounded. If  $\mu(S) < 1$  then we have*

$$CR(\mathcal{A}) \geq 1 + \frac{(1 + \mu(\mathcal{A}))(1 + \mu(S))}{\mu(\mathcal{A})(1 - \mu(S))}.$$

**Proof.** If  $\mu(S) = 1$  then the previous lemma tells us that  $K_S(t; \mathcal{A}) = [-\frac{t-S(t)}{2}, \frac{t+S(t)}{2}]$ . Suppose without loss of generality that  $S(t) = t$ . Then  $K_S(t) = [0, t]$  and by Lemma 13 we have

$$CR(\mathcal{A}) \geq \sup_{x \notin K_S(t; \mathcal{A})} \frac{|X(t) - x| + t}{|x|} = \sup_{\epsilon > 1} \frac{2t + \epsilon}{\epsilon} > 2t + 1.$$

Since the above holds for infinitely many arbitrary large  $t$  values, we conclude that  $CR(\mathcal{A})$  is unbounded.

Suppose now that  $\mu(S) < 1$ . Then for all  $0 < \epsilon < 1 - \mu(S)$  there exists a time  $T$  such that

$$K_S(t; \mathcal{A}) \subseteq \left[ -\frac{(\mu(\mathcal{A}) + \epsilon)(t - S(t))}{1 + \mu(\mathcal{A}) + \epsilon}, \frac{(\mu(\mathcal{A}) + \epsilon)(t + S(t))}{1 + \mu(\mathcal{A}) + \epsilon} \right], \forall t > T.$$

Moreover, from the definition of  $\mu(X)$  it follows that for any  $\Delta > 0$  there exists a time  $\tau$  such that  $|S(\tau)| = \mu(S)\tau \pm o(\tau)$ . Take  $\Delta > T$  and assume without loss of generality that  $S(\tau) = \mu(S)\tau \pm o(\tau)$ . Then

$$K_S(\tau; \mathcal{A}) \subseteq \left[ -\frac{(\mu(\mathcal{A}) + \epsilon)(1 - \mu(S))\tau}{1 + \mu(\mathcal{A}) + \epsilon}, \frac{(\mu(\mathcal{A}) + \epsilon)(1 + \mu(S))\tau}{1 + \mu(\mathcal{A}) + \epsilon} \right]$$

and

$$\begin{aligned} CR(\mathcal{A}) &\geq \sup_{x \notin K_S(\tau; \mathcal{A})} \frac{|S(\tau) - x| + \tau}{|x|} \\ &= 1 + \sup_{\epsilon' > 0} \frac{(1 + \mu(S))\tau}{\frac{(\mu(\mathcal{A}) + \epsilon)(1 - \mu(S))\tau}{1 + \mu(\mathcal{A}) + \epsilon} + \epsilon'} = 1 + \frac{(1 + \mu(\mathcal{A}) + \epsilon)(1 + \mu(S))}{(\mu(\mathcal{A}) + \epsilon)(1 - \mu(S))} \\ &> 1 + \frac{(1 + \mu(\mathcal{A}))(1 + \mu(S))}{\mu(\mathcal{A})(1 - \mu(S))} \left[ \frac{1}{1 + \frac{\epsilon}{\mu(\mathcal{A})}} \right]. \end{aligned}$$

The term in square brackets approaches 1 from below as  $\epsilon \rightarrow 0$  and thus for any fixed  $\delta > 0$  we can choose  $\epsilon > 0$  small enough that

$$CR(\mathcal{A}) > 1 - \delta + \frac{(1 + \mu(\mathcal{A}))(1 + \mu(S))}{\mu(\mathcal{A})(1 - \mu(S))}.$$

◀

► **Corollary 27.** *Let  $\mathcal{A}$  be an evacuation algorithm with one sender and let  $S \in \mathcal{A}$  represent the trajectory of this sender. If  $\mu(S) = \mu(\mathcal{A})$  we have  $CR(\mathcal{A}) \geq 9$ .*

**Proof.** With  $\mu(S) = \mu(\mathcal{A})$  we have from Lemma 26 that

$$CR(\mathcal{A}) \geq 1 + \frac{(1 + \mu(S))^2}{\mu(S)(1 - \mu(S))}$$

for all  $\delta > 0$ . Let  $g(u) = \frac{(1+u)^2}{u(1-u)}$  and observe that

$$\begin{aligned} \frac{dg(u)}{du} &= \frac{2(1+u)}{u(1-u)} - \frac{(1+u)^2(1-2u)}{u^2(1-u)^2} = (1+u) \left[ \frac{2u(1-u) - (1+u)(1-2u)}{u^2(1-u)^2} \right] \\ &= (1+u) \left[ \frac{2u - 2u^2 - 1 + 2u - u + 2u^2}{u^2(1-u)^2} \right] = \frac{(1+u)(3u-1)}{u^2(1-u)^2}. \end{aligned}$$

From this last expression it is clear that  $u = 1/3$  is the only non-negative minimizer. When  $u = 1/3$  we find  $g(1/3) = \frac{(1+\frac{1}{3})^2}{\frac{1}{3}(1-\frac{1}{3})} = 8$  and thus we can conclude that  $\text{CR}(\mathcal{A}) > 9 - \delta$  for arbitrary  $\delta > 0$  as required. ◀

In the next lemma we consider the knowledge set of the receivers.

► **Lemma 28.** *Let  $\mathcal{A}$  be an evacuation algorithm with one sender and at least one receiver. Let  $S$  be the trajectory of the sender and suppose that  $\mu(S) < \mu(\mathcal{A})$ . Let  $R$  be the trajectory of a receiver with  $\mu(R) = \mu(\mathcal{A})$ . If  $\mu(R) < 1$  then for all  $0 < \epsilon < 1 - \mu(R)$  there exists a time  $T > 0$  such that*

$$K_R(t; \mathcal{A}) = K_S(t; \mathcal{A}) \cup \left[ -\frac{(\mu(R) + \epsilon)(t - R(t))}{1 + \mu(R) + \epsilon}, \frac{(\mu(R) + \epsilon)(t + R(t))}{1 + \mu(R) + \epsilon} \right], \quad \forall t > T.$$

In the case of  $\mu(R) = 1$  the parameter  $\epsilon$  can be taken to be 0 in the above expression.

**Proof.** The receivers can receive wireless messages from the sender and so at any time they know what the sender knows. If we exclude knowledge from the sender, a receiver can only possess direct knowledge, or receive knowledge indirectly from a different receiver. However, receivers can't send messages and so communication between receivers is face-to-face. Thus, to complete the proof, we only need to invoke Lemma 25. ◀

► **Lemma 29.** *Let  $\mathcal{A}$  be an evacuation algorithm with one sender and at least one receiver. Let  $S \in \mathcal{A}$  represent the trajectory of this sender; let  $R \in \mathcal{A}$  represent the trajectory of the receiver with the largest value of  $\mu(R)$ ; and define  $\mathcal{A}' = \mathcal{A} \setminus \{R\}$ . Then, we have*

$$\text{CR}(\mathcal{A}) \geq 1 + \frac{(1 + \mu(\mathcal{A}'))(1 + \mu(R))}{\mu(\mathcal{A}')(1 + \mu(S))}.$$

**Proof.** We make use of Lemma 28. If  $\mu(R) = 1$  then

$$K_R(t; \mathcal{A}) = K_S(t; \mathcal{A}) \cup \left[ -\frac{t - R(t)}{2}, \frac{t + R(t)}{2} \right].$$

If  $\mu(R) < 1$  then for all  $0 < \epsilon < 1 - \mu(R)$  there exists a time  $T_R > 0$  such that

$$K_R(t; \mathcal{A}) = K_S(t; \mathcal{A}) \cup \left[ -\frac{(\mu(R) + \epsilon)(t - R(t))}{1 + \mu(R) + \epsilon}, \frac{(\mu(R) + \epsilon)(t + R(t))}{1 + \mu(R) + \epsilon} \right], \quad \forall t > T_R.$$

Moreover, for any  $\Delta > 0$  there exists a time  $\tau > \Delta$  such that  $|R(t)| = \mu(R)\tau$ . Assume without loss of generality that  $R(\tau) = \mu(R)\tau$ . Then  $K_R(\tau; \mathcal{A}) = K_S(\tau; \mathcal{A}) \cup [0, \tau]$  if  $\mu(R) = 1$  and for  $\mu(R) < 1$  we can take any  $\Delta > T_R$  to get

$$K_R(\tau; \mathcal{A}) = K_S(\tau; \mathcal{A}) \cup \left[ -\frac{(\mu(R) + \epsilon)(1 - \mu(R))\tau}{1 + \mu(R) + \epsilon}, \frac{(\mu(R) + \epsilon)(1 + \mu(R))\tau}{1 + \mu(R) + \epsilon} \right].$$



In light of the proof of Lemma 26 and its corollary, it is clear that unless the sender can increase the lower bound of the receiver's knowledge, we will find that  $\text{CR}(\mathcal{A})$  is unbounded when  $\mu(R) = 1$ , and when  $\mu(R) < 1$  we will get  $\text{CR}(\mathcal{A}) > 9 - \delta$  for all  $\delta > 0$ . Thus, we assume that the sender can increase the lower bound of the receiver's knowledge.

Consider the knowledge set  $K_S(t; \mathcal{A})$  of the sender. Since this must extend the knowledge of the receiver with trajectory  $R$  we can exclude this receiver from the computation of  $K_S(t; \mathcal{A})$ . Thus, if we take  $\mathcal{A}' = \mathcal{A} \setminus \{R\}$  we can invoke Lemma 25 with respect to  $\mathcal{A}'$  to conclude that for all  $0 < \epsilon < 1 - \mu(S)$  there exists a time  $T_S > 0$  such that

$$K_S(t; \mathcal{A}') \subseteq \left[ -\frac{(\mu(\mathcal{A}') + \epsilon)(t - S(t))}{1 + \mu(\mathcal{A}') + \epsilon}, \frac{(\mu(\mathcal{A}') + \epsilon)(t + S(t))}{1 + \mu(\mathcal{A}') + \epsilon} \right], \quad \forall t > T_S.$$

We can take  $\Delta > \max\{T_R, T_S\}$  so that  $\tau > T_S$  and as a result

$$K_S(\tau; \mathcal{A}) \subseteq \left[ -\frac{(\mu(\mathcal{A}') + \epsilon)(\tau - S(\tau))}{1 + \mu(\mathcal{A}') + \epsilon}, \frac{(\mu(\mathcal{A}') + \epsilon)(\tau + S(\tau))}{1 + \mu(\mathcal{A}') + \epsilon} \right].$$

By definition of  $\mu(S)$ , at any time  $t > T_s$  we have  $|S(t)| \leq (\mu(S) + \epsilon)t$  and thus

$$K_S(\tau; \mathcal{A}) \subseteq \left[ -\frac{(\mu(\mathcal{A}') + \epsilon)(1 + \mu(S) + \epsilon)\tau}{1 + \mu(\mathcal{A}') + \epsilon}, \frac{(\mu(\mathcal{A}') + \epsilon)(1 + \mu(S) + \epsilon)\tau}{1 + \mu(\mathcal{A}') + \epsilon} \right].$$

By Lemma 13 we then have

$$\begin{aligned} \text{CR}(\mathcal{A}) &\geq \sup_{x \notin K_R(\tau; \mathcal{A})} \frac{|R(\tau) - x| + \tau}{|x|} \\ &= 1 + \sup_{\epsilon' > 0} \frac{(1 + \mu(R))\tau}{\frac{(\mu(\mathcal{A}') + \epsilon)(1 + \mu(S))\tau}{1 + \mu(\mathcal{A}') + \epsilon} + \epsilon'} = 1 + \frac{(1 + \mu(\mathcal{A}') + \epsilon)(1 + \mu(R))}{(\mu(\mathcal{A}') + \epsilon)(1 + \mu(S) + \epsilon)} \\ &> 1 + \frac{(1 + \mu(\mathcal{A}'))(1 + \mu(R))}{\mu(\mathcal{A}')(1 + \mu(S))} \left[ \frac{1}{1 + \frac{\epsilon}{\mu(\mathcal{A}')}} \right] \left[ \frac{1}{1 + \frac{\epsilon}{\mu(S)}} \right]. \end{aligned}$$

It is clear that  $\mu(S) > 0$  (and, thus, also  $\mu(\mathcal{A}') > 0$ ) since otherwise the sender would not extend the receiver's knowledge. Then both of the terms in square brackets approach 1 from below as  $\epsilon \rightarrow 0$  and so we can choose  $\epsilon > 0$  small enough that for any fixed  $\delta > 0$  we have

$$\text{CR}(\mathcal{A}) > 1 - \delta + \frac{(1 + \mu(\mathcal{A}'))(1 + \mu(R))}{\mu(\mathcal{A}')(1 + \mu(S))}$$

as required. ◀

# Lower Bounds for Induced Cycle Detection in Distributed Computing

François Le Gall ✉

Graduate School of Mathematics, Nagoya University, Japan

Masayuki Miyamoto ✉

Graduate School of Mathematics, Nagoya University, Japan

---

## Abstract

The distributed subgraph detection asks, for a fixed graph  $H$ , whether the  $n$ -node input graph contains  $H$  as a subgraph or not. In the standard CONGEST model of distributed computing, the complexity of clique/cycle detection and listing has received a lot of attention recently.

In this paper we consider the induced variant of subgraph detection, where the goal is to decide whether the  $n$ -node input graph contains  $H$  as an *induced* subgraph or not. We first show a  $\tilde{\Omega}(n)$  lower bound for detecting the existence of an induced  $k$ -cycle for any  $k \geq 4$  in the CONGEST model. This lower bound is tight for  $k = 4$ , and shows that the induced variant of  $k$ -cycle detection is much harder than the non-induced version. This lower bound is proved via a reduction from two-party communication complexity. We complement this result by showing that for  $5 \leq k \leq 7$ , this  $\tilde{\Omega}(n)$  lower bound cannot be improved via the two-party communication framework.

We then show how to prove stronger lower bounds for larger values of  $k$ . More precisely, we show that detecting an induced  $k$ -cycle for any  $k \geq 8$  requires  $\tilde{\Omega}(n^{2-\Theta(1/k)})$  rounds in the CONGEST model, nearly matching the known upper bound  $\tilde{O}(n^{2-\Theta(1/k)})$  of the general  $k$ -node subgraph detection (which also applies to the induced version) by Eden, Fiat, Fischer, Kuhn, and Oshman [DISC 2019].

Finally, we investigate the case where  $H$  is the diamond (the diamond is obtained by adding an edge to a 4-cycle, or equivalently removing an edge from a 4-clique), and show non-trivial upper and lower bounds on the complexity of the induced version of diamond detecting and listing.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Distributed computing models; Theory of computation  $\rightarrow$  Lower bounds and information complexity

**Keywords and phrases** Distributed computing, Lower bounds, Subgraph detection

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.58

**Funding** This work was partially supported by JSPS KAKENHI grants Nos. JP19H04066, JP20H05966, JP20H00579, JP20H04139, JP21H04879 and by the MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) grants No. JPMXS0118067394 and JPMXS0120319794.

**Acknowledgements** The authors are grateful to Keren Censor-Hillel, Orr Fischer, Pierre Fraigniaud, Dean Leitersdorf and Rotem Oshman for helpful discussions and comments, and Shin-ichi Minato for his support.

## 1 Introduction

**Background.** The subgraph detection problem asks us to decide if the  $n$ -node input graph contains a copy of some fixed subgraph  $H$  or not. This problem has received a lot of attention in the past 40 years, and has recently been investigated in the setting of distributed computing as well. There are actually two versions for this problem. The first version simply requires to decide if the input network contains  $H$ . The second version cares about induced  $H$ , and asks to decide if the input network contains a *vertex-induced* copy of  $H$ . We refer to Figure 1 for an illustration of the difference between the two versions. In this paper we call the former version “non-induced  $H$  detection” and the latter version “induced  $H$  detection”.



© François Le Gall and Masayuki Miyamoto;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiro Sadakane; Article No. 58; pp. 58:1–58:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** This graph contains a 4-cycle as a subgraph but not an induced 4-cycle.

When considering subgraph detection in the (synchronous) distributed setting, the communication network is identified with the input graph, i.e., we ask whether the  $n$ -node communication network contains  $H$  as a subgraph (induced or non-induced, depending on the version considered). The complexity is characterized by the number of rounds of (synchronous) communication needed to solve the problem. For networks with unbounded bandwidth (the so-called LOCAL model in distributed computing), both versions of the subgraph detection problem are essentially trivial: for any  $O(1)$ -node subgraph  $H$ , the problem can be solved in  $O(1)$  rounds by a naive approach. For networks with bounded bandwidth (the so-called CONGEST model in distributed computing, in which the size of each message is restricted to  $O(\log n)$  bits), on the other hand, the same approach may take many more rounds due to possible congestion in the network (see the next paragraph for the definition of the CONGEST model). This is one of the reasons why the subgraph detection problem is interesting in the distributed setting. In the last few years, there has been significant progress in understanding the complexity of the non-induced subgraph detection in the CONGEST model.

**The CONGEST model.** In this paper we use the standard CONGEST model, as used in prior works [2, 3, 5, 8, 9, 11, 14, 16, 22]. In the CONGEST model, a distributed network of  $n$  computers is represented as a simple undirected graph  $G = (V, E)$  of  $n$  nodes, where each node corresponds to a computational device, and each edge corresponds to a communication link. Each node  $v \in V$  initially has a  $\Theta(\log n)$ -bit unique identifier  $\text{ID}(v)$ , and knows the list of IDs of its neighbors and the parameter  $n = |V|$ . The communication proceeds in synchronous rounds. In each round, each  $v \in V$  can perform unlimited local computation, and can send an  $O(\log n)$ -bit distinct message to each of its neighbors.

When considering subgraph detection in the CONGEST model, the communication network is identified with the input graph, i.e., we ask whether the  $n$ -node communication network contains  $H$  as a subgraph. If the network contains  $H$  as a subgraph, at least one node outputs 1 (Yes), otherwise all nodes output 0 (No). We assume that each node knows the graph  $H$  to be detected. The complexity is characterized by the number of rounds of communication needed to solve the problem.

**Non-induced subgraph detection in the distributed setting.** Typical examples of the non-induced subgraph detection in the CONGEST model that have been studied intensively are cliques and cycles. For cliques, the first sublinear-round algorithm of  $k$ -clique detection in the CONGEST model is due to Izumi and Le Gall [22], for  $k = 3$  (i.e., triangle detection), which runs in  $\tilde{O}(n^{2/3})$  rounds. Later, the complexity was brought down to  $\tilde{O}(\sqrt{n})$  by Chang et al. [8], and then further to  $\tilde{O}(n^{1/3})$  by Chang and Saranurak [9]. These upper bounds also hold for 3-clique listing,<sup>1</sup> and the  $\tilde{O}(n^{1/3})$  upper bound is tight up to polylogarithmic factors due to the lower bounds by Pandurangan, Robinson and Scquizzato [28] and Izumi and Le

<sup>1</sup> The listing version of the problem asks to list all instances of  $H$  in the graph.

■ **Table 1** Prior results for non-induced subgraph detecting and listing in the distributed setting. Here  $n$  denotes the number of nodes in the network.

Problem	Time bound	Paper	Model	
Triangle detection	$\tilde{O}(n^{1/3})$	[9]	CONGEST	
	$\tilde{O}(n^{1/4})$	[21]	QUANTUM CONGEST	
	$O(n^{0.159})$	[6]	CONGESTED CLIQUE	
Triangle listing	$\tilde{\Theta}(n^{1/3})$	[22, 9, 28]	CONGEST	
$k$ -clique detection, $k \geq 4$	$\Omega(n^{1/2}/\log n)$	[11]	CONGEST	
	$\tilde{O}(n^{1-2/k})$	[2]	CONGEST	
$k$ -clique listing, $k \geq 4$	$\tilde{\Theta}(n^{1-2/k})$	[16, 2]	CONGEST	
2k-cycle detection	$k \geq 2$	$\Omega(n^{1/2}/\log n)$	[13, 24]	CONGEST
	$k = 7, 9, 11, \dots$	$\tilde{O}(n^{1-2/(k^2-k+2)})$	[14]	CONGEST
	$k = 6, 8, 10, \dots$	$\tilde{O}(n^{1-2/(k^2-2k+4)})$	[14]	CONGEST
	$2 \leq k \leq 5$	$\tilde{O}(n^{1-1/k})$	[4, 13]	CONGEST
	any constant $k$	$O(1)$	[4]	CONGESTED CLIQUE
$(2k + 1)$ -cycle detection, $k \geq 2$	$\tilde{\Theta}(n)$	[13, 24]	CONGEST	
Detecting some $\Theta(k)$ -node subgraph $H$	$\Omega(n^{2-1/k}/\log n)$	[16]	CONGEST	
Detecting a $k$ -node tree	$O(k^k)$	[17, 24]	CONGEST	

Gall [22]. For  $k$ -cliques with  $k \geq 4$ , the first sublinear algorithm of  $k$ -clique listing is due to Eden et al. [14]. They showed that one can list all  $k$ -cliques in  $\tilde{O}(n^{5/6})$  rounds for  $k = 4$  and  $\tilde{O}(n^{21/22})$  rounds for  $k = 5$ . These results were improved to  $\tilde{O}(n^{k/(k+2)})$  rounds for all  $k \geq 4$  by Censor-Hillel, Le Gall, and Leitersdorf [5], and very recently,  $\tilde{O}(n^{1-2/k})$  rounds for all  $k \geq 4$  by Censor-Hillel et al. [2]. The latter bound is tight up to polylogarithmic factors due to the lower bounds by [16].

For  $k$ -cycles with  $k \geq 4$ , it is known that non-induced  $k$ -cycle ( $C_k$ ) detection requires  $\Omega(ex(n, C_k)/n)$  rounds by Drucker et al. [13], where  $ex(n, C_k)$  is the Turán number of  $k$ -cycle (Turán number  $ex(n, C_k)$  is the maximum number of edges in an  $n$ -node graph which does not have a  $k$ -cycle as a subgraph). This implies the  $\tilde{\Omega}(n)$  lower bounds for odd  $k$  and  $\tilde{\Omega}(\sqrt{n})$  lower bound for  $k = 4$ . Korhonen and Rybicki [24] showed  $\tilde{O}(n)$ -round algorithms of non-induced  $k$ -cycle detection for any odd constant  $k$ . They also showed the  $\tilde{\Omega}(\sqrt{n})$  lower bounds for even  $k \geq 6$ . For  $k = 4$ , an optimal algorithm for non-induced 4-cycle detection is known due to drucker et al. [13]. For even  $k \geq 6$ , Fischer et al. [16] showed an  $\tilde{O}(n^{1-\frac{1}{k(k-1)}})$ -round algorithm, and this was improved to  $\tilde{O}(n^{1-2/\Theta(k^2)})$  by Eden et al. [14]. Recently, Censor-Hillel et al. [4] showed that for  $3 \leq k \leq 5$ , non-induced  $C_{2k}$  detection can be solved in  $\tilde{O}(n^{1-1/k})$  rounds.

We refer to Table 1 for the summary of all these results.

**Induced subgraph detection in the distributed setting.** All of the above results for cycles are only for the non-induced distributed subgraph detection problem. While for the case of cliques, non-induced detection and induced subgraph detection are the same problem, this is not the case for cycles (see again Figure 1 for an illustration). For instance, if we want to know if the input graph contains a chordless cycle, we need to consider the induced version. In the centralized (i.e., non-distributed) setting, the induced version of subgraph detection has thus also been extensively studied [10, 15, 20, 25, 27, 30], leading to several algorithms that significantly differ from the algorithms for the non-induced version of the problem.

Despite its importance, the induced version has almost not been studied at all in the distributed setting. The only known results on the complexity of the induced subgraph detection problem in the CONGEST model are generic bounds describing how large the round

■ **Table 2** Our results on the round complexity of induced-subgraph detecting, and the corresponding known results. Here  $n$  denotes the number of nodes in the network.

Problem	Time Bound	Reference	Model
induced $k$ -node subgraph detection	$\tilde{O}(n^{2-2/(3k+1)})$	[14]	CONGEST
induced $k$ -cycle detection	$\Omega(n/\log n)$ , for $k \geq 4$	Theorem 1	CONGEST
	$\Omega(n^{2-1/\lceil k/8 \rceil} / \log n)$ , for $k \geq 8$	Theorem 2	CONGEST
induced diamond listing	$\tilde{O}(n^{5/6})$	Theorem 7	CONGEST
	$\Omega(\sqrt{n}/\log n)$	Theorem 5	CONGEST

complexity can be with respect to the number of nodes in the subgraph: Fischer et al. [16] constructed a family of graphs  $H$  with  $\Theta(k)$  nodes such that (induced and non-induced)  $H$  detection requires  $\Omega(n^{2-1/k}/\log n)$  rounds. Later, Eden et al. [14] showed that the  $n^{1/k}$  term cannot be removed, and also showed that for any  $k$ -node subgraph  $H$ , induced  $H$  detection can be solved in  $\tilde{O}(n^{2-\frac{2}{3k-2}}) = \tilde{O}(n^{2-\Theta(1/k)})$  rounds. These results, however, actually hold for the non-induced version as well. Therefore, to our knowledge, it is still open whether there exists a graph  $H$  such that the round complexities of non-induced  $H$  detection and induced  $H$  detection are different in the CONGEST model.

**Our results.** In this paper we answer this question. More precisely, we seek to improve our understanding of the round complexity of the distributed induced subgraph detection in the CONGEST model by showing lower bounds for constant-length cycles. We refer to Table 2 for the summary of our results.

We first show that for any  $k \geq 4$ , detecting an induced  $k$ -cycle requires a near-linear amount of rounds; previously, no lower bound for induced cycle detection was known.

► **Theorem 1.** *For any  $k \geq 4$ , deciding if a graph contains an induced  $k$ -cycle requires  $\Omega(n/\log n)$  rounds in the CONGEST model.*

For  $k = 4$ , the trivial solution of induced  $k$ -cycle detection is to have each node send its entire neighborhood to all its neighbors, which can be done in  $O(n)$  rounds. Therefore, our bound in Theorem 1 is tight up to logarithmic factor. Since, as already mentioned, the non-induced version of 4-cycle detection has complexity  $\tilde{\Theta}(\sqrt{n})$ , Theorem 1 proves that the induced version is significantly harder in the CONGEST model.

We then show stronger lower bounds for induced  $C_k$ -detection for larger values of  $k$ .

► **Theorem 2.** *For any constant  $k = 8\ell + m$  where  $\ell \geq 1$  and  $m \in \{0, 1, \dots, 7\}$ , deciding if a graph contains an induced  $k$ -cycle requires  $\Omega(n^{2-1/\ell}/\log n)$  rounds in the CONGEST model, even when the diameter of the network is 3.*

These bounds are asymptotically tight with respect to  $k$ , since for any  $k$ -node subgraph  $H$ , induced  $H$  detection can be solved in  $\tilde{O}(n^{2-\Theta(1/k)})$  rounds by the algorithm of [14]. We can summarize this as follows.

► **Corollary 3.** *The round complexity of induced  $k$ -cycle detection in the CONGEST model is  $\tilde{\Theta}(n^{2-\Theta(1/k)})$ .*

For small  $k$ , there still exist gaps between our lower bounds and known upper bounds. For instance, we do not know if induced 5-cycle detection can be solved in  $\tilde{O}(n)$  rounds. This leads to the following question: can we show any improved lower bounds in the case of  $k \geq 5$ ? We complement our results by showing that reductions from two-party communication

complexity, which is the technique we used to show our lower bounds (as well as most of the other lower bounds in the literature), do not have the ability to derive better lower bounds for the case of  $k \leq 7$ .

► **Theorem 4 (Informal statement).** *For  $k = 5, 6, 7$  and any  $\varepsilon > 0$ , reductions from two-party communication complexity cannot give an  $\tilde{\Omega}(n^{1+\varepsilon})$  lower bound for induced  $C_k$  detection in the CONGEST model.*

Theorem 4 is shown via an argument similar to the arguments in [11, 14]. These papers showed that reductions from two-party communication complexity (more precisely, the *family of lower bound graphs* technique) cannot show  $\tilde{\Omega}(n^{1/2+\varepsilon})$  lower bounds of 4-clique detection and (non-induced) 6-cycle detection. As mentioned above, to date, all known lower bounds on the subgraph detection in the CONGEST model used reductions from two-party communication complexity. Therefore, Theorem 4 shows that we need a fundamentally different approach to improve these lower bounds.

The graph that is obtained by removing one edge from a 4-clique is called a *diamond*. Diamonds are interesting since it is in some sense intermediate between 4-cycle and 4-clique. We show a lower bound of induced diamond listing in the CONGEST model.

► **Theorem 5.** *Listing all induced diamonds requires  $\Omega(\sqrt{n}/\log n)$  rounds in the CONGEST model.*

We also prove the same result as in Theorem 4 for induced diamond listing.

► **Theorem 6 (Informal statement).** *For any  $\varepsilon > 0$ , reductions from two-party communication complexity cannot give an  $\tilde{\Omega}(n^{1/2+\varepsilon})$  lower bound for induced diamond listing in the CONGEST model.*

Finally, we show that induced diamond listing can be done in sublinear rounds.

► **Theorem 7.** *There exists an algorithm that solves induced diamond listing in  $\tilde{O}(n^{5/6})$  rounds in the CONGEST model.*

Due to space constraints, the proofs of Theorem 6 and Theorem 7 are omitted from the main body of this paper – they can be found in Appendix C and Appendix D.

**Other related works.** Several works investigate the complexity of subgraph detection in other models of distributed computing. In the powerful CONGESTED CLIQUE model, which allows global communication, the induced subgraph detection (and even listing) can be solved in sublinear rounds [12]: for any  $k$ -node subgraph  $H$ , induced  $H$  detection and listing can be solved in  $O(n^{1-2/k})$  rounds. This algorithm was used as a subroutine to construct sublinear-round CONGEST algorithms for clique detection and listing [2, 5, 8, 9, 21, 22]. In the CONGESTED CLIQUE model, for any constant  $k \geq 3$ ,  $k$ -cycle can be detected in  $O(2^{O(k)}n^{0.158})$  rounds by an algebraic algorithm which uses the matrix multiplication [6] and for  $k \geq 2$ ,  $2k$ -cycle can be detected in  $O(1)$  rounds [4]. In the QUANTUM CONGEST model, in which each node represents a quantum computer and each edge represents a quantum channel, Izumi, Le Gall and Magniez [21] showed that triangle detection can be solved in  $\tilde{O}(n^{1/4})$  rounds by using quantum distributed search [26] which is the distributed implementation of Grover's quantum search. For any  $\varepsilon > 0$ , showing a lower bound of  $\Omega(n^\varepsilon)$  on directed triangle detection implies strong circuit complexity lower bounds [4]. These bounds are included in Table 1.

Other relevant works include constant-round detection of constant-sized trees in the CONGEST model [17, 24], and investigations of the distributed subgraph detection problem in the framework of *property testing* [3, 18, 19].

**Recent independent work.** Lower bounds for induced cycle detection similar to some of the bounds in our paper have been concurrently (and independently) obtained very recently by Korhonen and Nikabadi [23]. They showed the following results using graph constructions different from ours (but still using reductions from two-party communication complexity):

- $\Omega(n/\log n)$  lower bound for induced  $2k$ -cycle detection for  $k \geq 3$ ,
- $\Omega(n/\log n)$  lower bound for a multicolored variant of  $k$ -cycle detection for  $k \geq 4$ .

## 2 Preliminaries

To prove lower bounds, we use reductions from two-party communication complexity problems. This is the common technique to show lower bounds in the CONGEST model [16, 7, 11, 13, 1]. Here we give the precise definition of the *family of lower bound graphs*, which is the standard notion to show these lower bounds (see, e.g., [7]).

► **Definition 8** (Family of Lower Bound Graphs). *Given an integer  $K$ , a boolean function  $f : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{0, 1\}$  and a graph predicate  $P$ , a set of graphs  $\{G_{x,y} = (V, E_{x,y}) \mid x, y \in \{0, 1\}^K\}$  is called a family of lower bound graphs with respect to  $f$  and  $P$  if the following hold:*

1. *The set of vertices  $V$  is the same for all the graphs in the family, and has a fixed partition  $V = V_A \cup V_B$ . The set of edges of the cut  $E_{cut} = E(V_A, V_B)$  is the same for all graphs in the family.*
2. *Given  $x, y \in \{0, 1\}^K$ ,  $E(V_A, V_A)$  only depends on  $x$ .*
3. *Given  $x, y \in \{0, 1\}^K$ ,  $E(V_B, V_B)$  only depends on  $y$ .*
4.  *$G_{x,y}$  satisfies  $P$  if and only if  $f(x, y) = 1$ .*

► **Theorem 9** ([7]). *Fix a boolean function  $f : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{0, 1\}$  and a graph predicate  $P$ . If there exists a family of lower bound graphs  $\{G_{x,y}\}$  with respect to  $f$  and  $P$ , then any randomized algorithm for deciding  $P$  in the CONGEST model takes  $\Omega(CC^R(f)/|E_{cut}| \log n)$ , where  $CC^R(f)$  is the randomized communication complexity of  $f$ .*

Throughout this paper, we use the set-disjointness function  $DISJ_K : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{0, 1\}$ . For two bit strings  $x, y \in \{0, 1\}^K$ ,  $DISJ_K(x, y)$  is equal to 0 if and only if there exists some index  $i \in [K]$  such that  $x_i = y_i = 1$ . It is well known that  $CC^R(DISJ_K) = \Omega(K)$  [29].

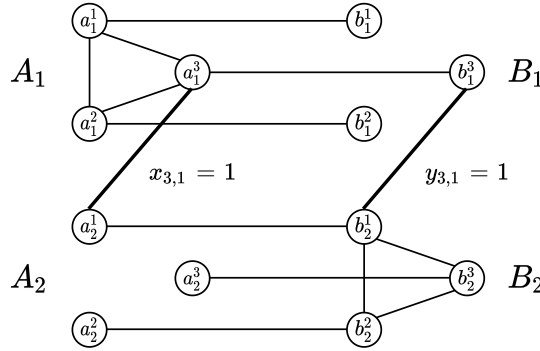
## 3 Lower Bounds for $k$ -cycles, $k \geq 4$

In this section we prove Theorem 1. To prove Theorem 1, we describe families of lower bound graphs with respect to the set-disjointness function of the two-party communication complexity, and the predicate  $P$  that says the graph does not contain an induced  $k$ -cycle. We start by describing the fixed graph construction for the case of  $k = 4$ , and then define the corresponding family of lower bound graphs.

**The fixed graph construction.** Create a graph  $G$  as follows: The vertex set is  $A_1 \cup A_2 \cup B_1 \cup B_2$  such that  $A_1$  and  $B_2$  are  $n$ -vertex cliques and  $A_2$  and  $B_1$  are a set of  $n$  vertices with no edges inside of them. Denote the vertices in  $G$  as  $A_i = \{a_i^1, \dots, a_i^n\}$ ,  $B_i = \{b_i^1, \dots, b_i^n\}$  for  $i \in \{1, 2\}$ . We add edges  $(a_i^j, b_i^j)$ ,  $(a_i^j, b_j^i)$  for all  $i \in [n]$ .

**Creating  $G_{x,y}$ .** For two input bit strings  $x, y \in \{0, 1\}^{n^2}$  and  $i, j \in [n]$ , we denote the  $(i + (j - 1)n)$ -th bit of  $x$  and  $y$  as  $x_{ij}$  and  $y_{ij}$ . Edges corresponding to inputs are added as follows (see Figure 2 for the illustration):





■ **Figure 2** An illustration of  $G_{x,y}$  for the case of  $n = 3$ . The graph contains a copy of induced  $C_4$  if and only if it holds that  $x_{ij} = y_{ij} = 1$  for some index  $i, j \in [n]$ . This illustration shows the case of  $x_{3,1} = y_{3,1} = 1$ .

- We add an edge between  $a_1^i$  and  $a_2^j$  if and only if  $x_{ij} = 1$ .
- We add an edge between  $b_1^i$  and  $b_2^j$  if and only if  $y_{ij} = 1$ .

This concludes the description of  $G_{x,y}$ . Next, we prove that the family  $\{G_{x,y} | x, y \in \{0, 1\}^{n^2}\}$  is a family of lower bound graphs with respect to set-disjointness and the predicate that says the graph does not contain an induced 4-cycle.

▷ **Claim 10.**  $G_{x,y}$  contains an induced  $C_4$  if and only if there exists a pair of index  $i, j \in [n]$  such that  $x_{ij} = y_{ij} = 1$ .

*Proof.* Let  $U = \{v_1, v_2, v_3, v_4\}$  be a subset of  $V$ . It is clear that if it holds  $|U \cap S| = 4$  for some  $S \in \{A_1, A_2, B_1, B_2\}$ , then  $U$  does not induce  $C_4$ . We analyse  $U$  as follows:

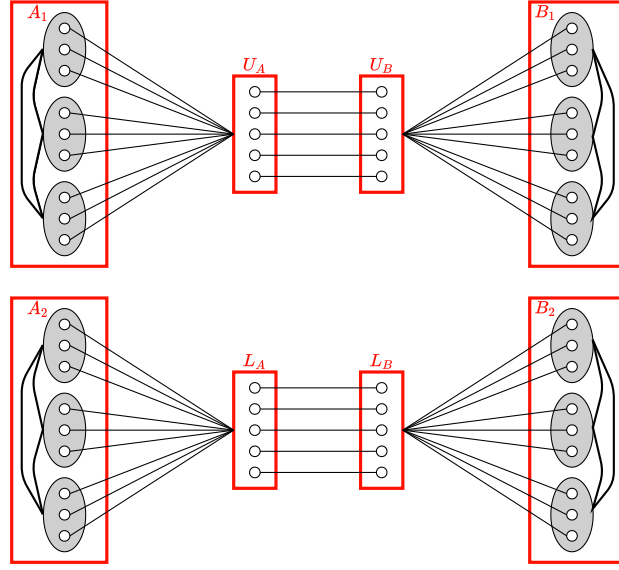
- If it holds  $|U \cap A_1| = 3$  or  $|U \cap B_2| = 3$ ,  $U$  induces a triangle.
- If it holds  $|U \cap A_2| = 3$  or  $|U \cap B_1| = 3$ ,  $U$  induces at most three edges.
- If it holds  $|U \cap A_1| = 2$  or  $|U \cap B_2| = 2$ , and  $U$  induces a 4-cycle, the other two vertices of  $U$  are both in  $A_2$  or both in  $B_1$ . However, It is impossible since there is no edge between any two vertices in  $A_2$  and any two vertices in  $B_1$ .
- If it holds  $|U \cap A_2| = 2$  or  $|U \cap B_1| = 2$ , the two vertices does not share neighbors. Hence,  $U$  does not induce a 4-cycle.

Now all we need is to verify whether four vertices  $a_1^i, a_2^j, b_1^k,$  and  $b_2^\ell$  induce a 4-cycle or not. If  $(a_1^i, a_2^j, b_1^k, b_2^\ell)$  induces a 4-cycle, we can say that  $i = k$  since  $a_1^i$  has to be connected to  $a_2^j$  and  $b_1^k$  (similarly we can say  $j = \ell$ ). It is straightforward to show that  $(a_1^i, a_2^j, b_1^i, b_2^j)$  induces a 4-cycle if and only if  $x_{ij} = y_{ij} = 1$ . ◀

**Proof of Theorem 1.** Divide the vertices of the graph  $G_{x,y}$  into  $V_A = A_1 \cup A_2$  and  $V_B = B_1 \cup B_2$ . The size of the cut is  $|E_{cut}| = |E(V_A, V_B)| = 2n$ . Claim 10 shows that the family of the graphs  $\{G_{x,y} | x, y \in \{0, 1\}^{n^2}\}$  is a family of lower bound graphs for  $f = \text{DISJ}_{n^2}$  and a predicate that says the graph include an induced  $C_4$ . Hence, using Theorem 9 and  $CC^R(\text{DISJ}_{n^2}) = \Theta(n^2)$ , any randomized algorithms for induced 4-cycle detection in the CONGEST model requires  $\Omega(n/\log n)$ .

To extend this result to  $k$ -cycles for  $k \geq 5$ , we modify the graph  $G_{x,y}$  as follows:

- For any  $i \in [n]$ , replace the edge  $(a_1^i, b_1^i)$  to a path with  $\lceil \frac{k-4}{2} \rceil + 2$  vertices.
- For any  $i \in [n]$ , replace the edge  $(a_2^i, b_2^i)$  to a path with  $\lfloor \frac{k-4}{2} \rfloor + 2$  vertices. ◀



■ **Figure 3** An illustration of the fixed part of  $G_{x,y}$ . Some edges are bundled for clarity. Observe that  $A_1^i \subseteq A_1$  and  $A_2^j \subseteq A_2$  are connected by additional edges iff  $x_{i,j} = 1$ . Also,  $B_1^i \subseteq B_1$  and  $B_2^j \subseteq B_2$  are connected by additional edges iff  $y_{i,j} = 1$ .

#### 4 Lower Bounds for Larger Cycles

In this section we prove Theorem 2, i.e., we show subquadratic, but superlinear lower bounds for induced cycles  $C_{k \geq 8}$ , which gives nearly tight bounds for induced cycles  $C_{k \geq 8}$  with respect to  $k$ . The main difficulty to obtain the bounds of Theorem 2 is to reduce the size of the cut edges of graphs while retaining the ability to simulate the set-disjointness function of size  $\Omega(n^2)$ . We overcome this difficulty by considering induced cycles that go around  $G_{x,y}$  more than once instead of cycles that go around  $G_{x,y}$  exactly once (we pay for this by an increased size of a cycle). This enables us to reduce the size of cut edges.

##### 4.1 The fixed graph construction

We refer to Figure 3 for an illustration of the construction.

**Vertices.** We define the sets of vertices as follows:

- $A_k = A_k^1 \cup \dots \cup A_k^n$ , where  $A_k^i = \{a_k^{i,j} \mid 0 \leq j \leq \ell - 1\}$  for  $i \in [n]$  and  $k \in \{1, 2\}$ .
- $B_k = B_k^1 \cup \dots \cup B_k^n$ , where  $B_k^i = \{b_k^{i,j} \mid 0 \leq j \leq \ell - 1\}$  for  $i \in [n]$  and  $k \in \{1, 2\}$ .
- $U_A = \{u_A^i \mid 0 \leq i \leq \ell n^{1/\ell}\}$ ,  $L_A = \{l_A^i \mid 0 \leq i \leq \ell n^{1/\ell}\}$ .
- $U_B = \{u_B^i \mid 0 \leq i \leq \ell n^{1/\ell}\}$ ,  $L_B = \{l_B^i \mid 0 \leq i \leq \ell n^{1/\ell}\}$ .

Each  $S \in \{A_1, A_2, B_1, B_2\}$  contains  $\ell n$  vertices, and they are divided into  $n$  subsets of size  $\ell$ . Each  $C \in \{U_A, U_B, L_A, L_B\}$  contains  $\ell n^{1/\ell}$  vertices. The number of vertices  $V = A_1 \cup A_2 \cup B_1 \cup B_2 \cup U_A \cup L_A \cup U_B \cup L_B$  is  $\Theta(\ell n + \ell n^{1/\ell}) = \Theta(n)$ .

**Edges.** First, we add  $2\ell n^{1/\ell}$  edges  $\{(u_A^i, u_B^i), (l_A^i, l_B^i) \mid i \in [\ell n^{1/\ell}]\}$ . Then, we consider a map from  $[n]$  to  $[\ell n^{1/\ell}]^\ell$ , where  $[\ell n^{1/\ell}]^\ell$  is  $\ell$  times direct product of the set  $[\ell n^{1/\ell}]$ . Since

$$\binom{\ell n^{1/\ell}}{\ell} = \frac{\ell n^{1/\ell}}{\ell} \cdot \frac{\ell n^{1/\ell} - 1}{\ell - 1} \cdots \frac{\ell n^{1/\ell} - \ell + 1}{1} \geq \left(\frac{\ell n^{1/\ell}}{\ell}\right)^\ell = n$$

holds, there exists an injection  $\sigma : [n] \rightarrow [\ell n^{1/\ell}]^\ell$ . We arbitrarily choose one of these injections. For  $i \in [n]$ , we denote  $\sigma(i) = \{k_1, \dots, k_\ell\} \in [\ell n^{1/\ell}]^\ell$ . For all  $i \in [n], j \in [\ell]$ , we add the edge sets  $\{(a_1^{i,j}, u_A^{k_j}) \mid i \in [n], j \in [\ell]\}$ ,  $\{(a_2^{i,j}, l_A^{k_j}) \mid i \in [n], j \in [\ell]\}$ ,  $\{(b_1^{i,j}, u_B^{k_j}) \mid i \in [n], j \in [\ell]\}$ , and  $\{(b_2^{i,j}, l_B^{k_j}) \mid i \in [n], j \in [\ell]\}$ . Now we can determine exactly  $\ell$  vertices of  $U_A$  that are adjacent to vertices of  $A_1^i$ . We denote them  $Code(A_1^i) \subseteq U_A$ . In the same way, we determine the vertex sets  $Code(A_2^i) \subseteq L_A$ ,  $Code(B_1^i) \subseteq U_B$ , and  $Code(B_2^i) \subseteq L_B$  by using the same  $\sigma$ . Since  $\sigma$  is an injection, it holds that  $Code(A_1^i) \neq Code(A_1^j)$ ,  $Code(A_2^i) \neq Code(A_2^j)$ ,  $Code(B_1^i) \neq Code(B_1^j)$ , and  $Code(B_2^i) \neq Code(B_2^j)$  for  $i \neq j$ .

In addition, we add the following edges.

- For any  $i, j \in [n]$ , add edges between  $u \in A_1^i, v \in A_1^j$  if and only if  $i \neq j$ .
- For any  $i, j \in [n]$ , add edges between  $u \in B_2^i, v \in B_2^j$  if and only if  $i \neq j$ .

If  $\ell \geq 2$ , we add the following edges.

- For any  $i, j \in [n]$ , add edges between  $u \in A_2^i, v \in A_2^j$  if and only if  $i \neq j$ .
- For any  $i, j \in [n]$ , add edges between  $u \in B_1^i, v \in B_1^j$  if and only if  $i \neq j$ .

## 4.2 Creating $G_{x,y}$

Note that for  $\ell = 1$ , the fixed part of  $G_{x,y}$  in this section is exactly the same as the fixed part of graphs for induced 8-cycles in Section 3. Hence, we only describe the case  $\ell \geq 2$ . Given two binary strings  $x, y \in \{0, 1\}^{n^2}$ , we add the following edges:

- For  $i, j \in [n]$ , add edges  $\{(a_1^{i,k+1}, a_2^{j,k}) \mid k \in [\ell - 1]\} \cup \{(a_1^{i,1}, a_2^{j,\ell})\}$ , if and only if  $x_{i,j} = 1$ .
- For  $i, j \in [n]$ , add edges  $\{(b_1^{i,k}, b_2^{j,k}) \mid k \in [\ell]\}$ , if and only if  $y_{i,j} = 1$ .

This concludes the description of  $G_{x,y}$ . We show the following theorem which says that  $\{G_{x,y}\}$  is a family of lower bound graphs. Due to space constraint, the proof is moved to the Appendix.

► **Theorem 11.**  $G_{x,y}$  contains an induced  $8\ell$ -cycle if and only if  $\text{DISJ}_{n^2}(x, y) = 0$ .

Having constructed a family of lower bound graphs, we are now ready to prove Theorem 2.

**Proof of Theorem 2.** Theorem 11 implies that a family of graphs

$$\left\{G_{x,y} = (V_A \cup V_B, E_{x,y}) \mid x, y \in \{0, 1\}^{n^2}\right\}$$

where  $V_A = A_1 \cup A_2 \cup U_A \cup L_A$ ,  $V_B = B_1 \cup B_2 \cup U_B \cup L_B$  is a family of lower bound graphs with respect to the set disjointness function  $\text{DISJ}_{n^2}$  and the graph predicate is whether the graph has a copy of an induced  $C_{8\ell}$  or not with cut size  $\ell n^{1/\ell}$ . To bound the diameter of the network to 3, we add nodes  $c_A$  to  $V_A$  and  $c_B$  to  $V_B$  such that  $c_A$  is connected to all nodes in  $V_A$  and  $c_B$  is connected to all nodes in  $V_B$ . Finally, we add an edge  $(c_A, c_B)$ . The above modification does not effect to the existence of induced  $8\ell$ -cycles: If we choose  $c_A$  as one of the cycle nodes, then we cannot choose more than two  $V_A$  nodes as cycle nodes. However, we cannot choose more than  $8\ell - 4$  nodes from  $V_B$  due to Lemma 16 of Appendix A, which also holds after this modification. The theorem is proved by applying Theorem 9 (for  $m = 0$ ).

Slightly modifying the graphs gives the same complexity for the case of  $k = 8\ell + m$  where  $m \in \{1, 2, \dots, 7\}$ :

- Replace each edge  $e \in U_A \times U_B$  by a path of length  $\lfloor m/2 \rfloor$ .
- Replace each edge  $e \in L_A \times L_B$  by a path of length  $\lceil m/2 \rceil$ . ◀

## 5 Limitation of the Two-Party Communication Framework

Since no  $\tilde{O}(n)$ -round algorithm for detecting an induced  $k$ -cycle for  $k \geq 5$  is known, the main question is whether our lower bound can be improved or not. In this section, we show that the family of lower bound graphs cannot derive any better lower bounds for detecting an induced  $k$ -cycle for  $k \leq 7$ , by giving a two-party communication protocol for listing  $k$ -cycles for  $k \leq 7$  in the vertex partition model which is defined as follows.

► **Definition 12** (Vertex Partition Model, [11]). *Given a graph  $G = (V_A \cup V_B, E_A \cup E_B \cup E_{cut})$  where  $E_A = E(V_A, V_A)$ ,  $E_B = E(V_B, V_B)$  and  $E_{cut} = E(V_A, V_B)$ , the vertex partition model is a two-party communication model in which Alice receives  $G_A = (V_A, E_A \cup E_{cut})$  as the input and Bob receives  $G_B = (V_B, E_B \cup E_{cut})$  as the input. For any graph  $H$ , the  $O(k)$  communication protocol for induced  $H$  listing is a protocol such that*

- The players communicate  $O(k)$  bits in the protocol.
- At the end of the protocol, the players have their lists of  $H$ , denoted by  $A_H, B_H$ , such that all of copies of  $H$  in the input graph  $G$  are contained in either  $A_H$  or  $B_H$ .

► **Theorem 13.** *There is a two-party communication protocol in the vertex partition model for listing all induced  $k$ -cycles for  $k \leq 7$  that uses  $\tilde{O}(n|E_{cut}|)$  bits of communication where  $E_{cut}$  is the set of cut edges in the input graph.*

**Proof.** Let  $V'_A(V'_B)$  be a set of  $V_A(V_B)$  vertices which are incident to some cut edge. The protocol is as follows:

1. Bob sends all edges  $E_B \cap \{V'_B \times V_B\}$  to Alice in  $\tilde{O}(n|E_{cut}|)$  bits since the number of edges Bob sends to Alice is less than

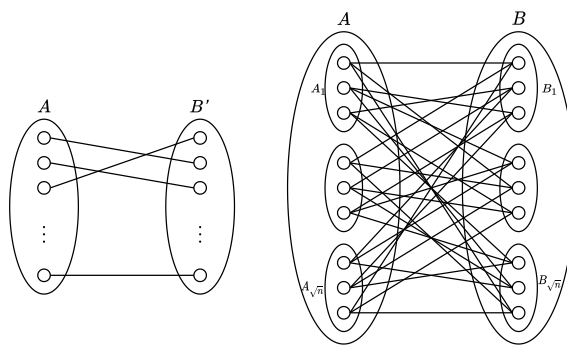
$$\sum_{v \in V'_B} \deg_{V_B}(v) \leq \sum_{v \in V'_B} n = n|E_{cut}|.$$

2. Alice sends all edges  $E_A \cap \{V'_A \times V_A\}$  to Bob in  $\tilde{O}(n|E_{cut}|)$  bits since the number of edges Alice sends to Bob is less than

$$\sum_{v \in V'_A} \deg_{V_A}(v) \leq \sum_{v \in V'_A} n = n|E_{cut}|.$$

Consider Alice has to list all induced  $k$ -cycles such that at least  $\lfloor k/2 \rfloor$  vertices of them are in  $V_A$ . Let  $U$  be a set of vertices in a copy of an induced  $k$ -cycle Alice should list in the input graph  $G$ . Since  $k \leq 7$ ,  $U$  contains at most three vertices in  $V_B$ . If  $U$  has at least one vertex in  $V_B$ , then the  $k$ -cycle induced by  $U$  has two cut edges. Therefore, we have  $U \times U \subseteq E_A \cup E_{cut} \cup \{E_B \cap \{V'_B \times V_B\}\}$ . Step 1 of the protocol enables Alice to list all induced  $k$ -cycles she should list. Similarly, step 2 of the protocol enables Bob to list all induced  $k$ -cycles he should list. Now all induced  $k$ -cycles of the input graph  $G$  are in either the list of Alice or the list of Bob. ◀

► **Theorem 4** (Formal statement). *For any  $\varepsilon > 0$ , no family of lower bound graphs gives an  $\tilde{\Omega}(n^{1+\varepsilon})$  lower bound of induced  $k$ -cycle detection for  $k \leq 7$ .*



■ **Figure 4** The cut edges in the family of lower bound graphs for listing diamonds. Many edges are omitted for clarity.

**Proof.** For the family of lower bound graphs

$$\{G_{x,y} = (V_A \cup V_B, E_A \cup E_B \cup E_{cut}) \mid x, y \in \{0, 1\}^K\}$$

for  $f : \{0, 1\}^K \times \{0, 1\}^K \rightarrow \{0, 1\}$  and the property which says that the graph contains an induced  $k$ -cycle, we can show an  $\tilde{\Omega}(CC^R(f)/|E_{cut}|)$  lower bound for induced  $k$ -cycle detection. On the other hand, we can solve  $f$  by  $\tilde{O}(n|E_{cut}|)$  bits of communication through the protocol of the vertex partition model in Theorem 13. Then it holds  $|E_{cut}| = \tilde{\Omega}(CC^R(f)/n)$ , implying that for any  $\varepsilon > 0$ , we cannot derive an  $\tilde{\Omega}(n^{1+\varepsilon})$  lower bound for induced  $k$ -cycle listing by the family of lower bound graphs. ◀

## 6 Lower Bound for Diamond Listing

We know that the round complexity of 4-clique detection is  $\tilde{\Theta}(\sqrt{n})$ , and induced 4-cycle detection is  $\tilde{\Theta}(n)$ . The only four-node graph that lies between 4-clique and 4-cycle is the *diamond*, which is the four-node graph obtained by removing one edge from a 4-clique. Intuitively, the complexity of diamond detection seems to be somewhere between the complexity of 4-clique and the complexity of 4-cycle. In this section, we make this intuition precise, and we show a lower bound for induced diamond listing (this result is complemented by the upper bound of Theorem 7 shown in Appendix D). Our construction of the family of lower bound graphs is similar to [11], but has the following differences. The graphs of [11] have two sets of vertices  $A$  and  $B$ , and edges between  $A$  and  $B$  are added randomly so that the size of the cut edges is  $O(n^{3/2})$ . In this random graph, *w.h.p.*, the number of tuples of the form  $(a_1, a_2, b_1, b_2)$  where  $a_1, a_2 \in A$  and  $b_1, b_2 \in B$  which corresponds to the  $i$ -th bit of the input strings  $x, y$  is  $\Omega(n^2)$ : a tuple  $(a_1, a_2, b_1, b_2)$  induces a 4-clique if and only if  $x_i = y_i = 1$ . However, in the case of diamonds, the number of tuples which correspond to the input is  $o(n^2)$ . To avoid this, we construct the family of lower bound graphs in a different way. This makes it much easier to analyze the properties of the graph.

**The fixed graph construction.** We refer to Figure 4 for an illustration. The set of vertices is  $V = A \cup B \cup B'$  such that  $A, B$  and  $B'$  are sets of  $n$  vertices. Each vertex is denoted as follows:

- $A = A_1 \cup A_2 \cup \dots \cup A_{\sqrt{n}}$ , where  $A_i = \{a_i^j \mid j \in [\sqrt{n}]\}$  for all  $i \in [\sqrt{n}]$ .
- $B = B_1 \cup B_2 \cup \dots \cup B_{\sqrt{n}}$ , where  $B_i = \{b_i^j \mid j \in [\sqrt{n}]\}$  for all  $i \in [\sqrt{n}]$ .
- $B' = \{b'_i \mid i \in [n]\}$ .

We add the edge set  $\{(a_i^j, b'_{j+(i-1)\sqrt{n}}) \mid i, j \in [\sqrt{n}]\}$ . For any  $i, j \in [\sqrt{n}]$ , we choose a bijection uniform randomly from all possible bijection  $\sigma : A_i \rightarrow B_j$ , and denote it  $\sigma_{i,j} : A_i \rightarrow B_j$ . Then, we add the edge set  $\{(a_i^k, \sigma_{i,j}(a_i^k)) \mid i, j \in [\sqrt{n}], k \in [\sqrt{n}]\}$ .

**Creating  $G_{x,y}$ .** We call a pair  $(a_i^j, a_k^\ell)$  is good iff  $|N(a_i^j) \cap N(a_k^\ell)| = 1$ , where  $N(u)$  is a set of neighbors of a vertex  $u$ . Let  $P_A$  be the set of good pairs in  $A \times A$ . That is,  $P_A := \{(a_i^j, a_k^\ell) \mid |N(a_i^j) \cap N(a_k^\ell)| = 1, i, j, k, \ell \in [\sqrt{n}]\}$ .

► **Lemma 14.** *There is a graph  $G$  created by the above procedure, in which it holds that  $|P_A| = \Omega(n^2)$ .*

The proof of Lemma 14 can be found in Appendix B. Consider the graph  $G$  in which  $|P_A| = \Omega(n^2)$ . Let  $\mathcal{H} = \emptyset$ . We partition  $A$  randomly into two sets  $A^*$  and  $A \setminus A^*$  so that  $|A^*| = n/2$ . For a pair  $(a_1, a_2) \in P_A$ , there is only one vertex  $b_1 \in N(a_1) \cap N(a_2)$ . For  $a_1$ , there is only one vertex  $b_2 \in N(a_1) \cap B'$ . We add a quadruple  $(a_1, a_2, b_1, b_2)$  to  $\mathcal{H}$  iff  $|\{a_1, a_2\} \cap A^*| = 1$ . This condition holds with probability  $\frac{1}{2}$ . Then, we remove  $(a_1, a_2)$  from  $P_A$ . We continue this operation until  $P_A$  becomes the empty set. Therefore, after this process, there are  $|\mathcal{H}| = \Omega(n^2)$  quadruples in  $\mathcal{H}$  with high probability, using Chernoff bound. We label  $\mathcal{H}$  as  $\mathcal{H} = \{h_1, h_2, \dots, h_{|\mathcal{H}|}\}$  and relabel quadruples as  $h_k = (a_{k,1}, a_{k,2}, b_{k,1}, b_{k,2})$ . Consider two bit strings  $x, y \in \{0, 1\}^{|\mathcal{H}|}$ . We create a graph  $G_{x,y}$  by adding edges to  $G$  as follows:

- If  $x_k = 1$ , we add an edge between  $a_{k,1}$  and  $a_{k,2}$ .
- If  $y_k = 1$ , we add an edge between  $b_{k,1}$  and  $b_{k,2}$ .

For a quadruple  $D = (u_1, u_2, u_3, u_4)$  of vertices, we say that  $D$  is an  $(i, j)$ -diamond when

- $(u_1, u_2, u_3, u_4)$  induces a diamond,
- $|A \cap \{u_1, u_2, u_3, u_4\}| = i$  and  $|(B \cup B') \cap \{u_1, u_2, u_3, u_4\}| = j$ .

► **Lemma 15.**  *$G_{x,y}$  contains a  $(2,2)$ -diamond if and only if there exists a pair of indices  $i, j \in [\sqrt{|\mathcal{H}|}]$  such that  $x_{ij} = y_{ij} = 1$ .*

**Proof.** It is clear that if  $x_k = y_k = 1$ , then  $h_k = (a_{k,1}, a_{k,2}, b_{k,1}, b_{k,2})$  induces a diamond. Consider four vertices  $a_1, a_2 \in A, b_1, b_2 \in B \cup B'$  that induce a  $(2,2)$ -diamond. If it holds that  $(b_1, b_2) \notin E$ , then  $(a_1, a_2) \in E$ . Thus, a pair  $(a_1, a_2)$  is good. It contradicts that vertices  $a_1, a_2, b_1$ , and  $b_2$  induce a diamond. Assume that  $(b_1, b_2) \in E$ . Without loss of generality, we assume  $b_1 \in B, b_2 \in B'$ . Since a pair  $(a_1, a_2)$  is good,  $b_1 = N(a_1) \cap N(a_2)$  and  $(a_1, a_2) \in E$ . Then, there is an index  $k \in [|\mathcal{H}|]$  such that  $h_k = (a_1, a_2, b_1, b_2)$  since  $(a_2, b_2) \notin E$  holds. ◀

**Proof of Theorem 5.** Consider that Alice and Bob construct the graph  $G_{x,y}$  where  $V_A = A, V_B = B \cup B'$ . By simulating an  $r$ -round CONGEST algorithm  $\mathcal{A}$  that solves listing all diamonds, they can compute the set disjointness function of size  $|\mathcal{H}| = \Omega(n^2)$ : Bob tells Alice if there exists a  $(2,2)$ -diamond in the output of vertices simulated by Bob by sending 1 bit. Then, from Lemma 15, Alice knows  $\text{DISJ}_{|\mathcal{H}|}(x, y)$  since Alice can know whether  $G_{x,y}$  contains a  $(2,2)$ -diamond. The number of edges between Alice and Bob is  $n^{3/2} + n = \Theta(n^{3/2})$ . Hence,  $O(rn^{3/2} \log n) = \Omega(|\mathcal{H}|)$  and this means  $r = \Omega(\sqrt{n}/\log n)$ . ◀

---

**References**

---

- 1 Amir Abboud, Keren Censor-Hillel, and Seri Khoury. Near-linear lower bounds for distributed distance computations, even in sparse networks. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC 2016)*, pages 29–42, 2016.
- 2 Keren Censor-Hillel, Yi-Jun Chang, François Le Gall, and Dean Leitersdorf. Tight distributed listing of cliques. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2878–2891, 2021.
- 3 Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(6):41–57, 2019.
- 4 Keren Censor-Hillel, Orr Fischer, Tzlil Gonen, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Fast distributed algorithms for girth, cycles and small subgraphs. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC 2020)*, pages 33:1–33:17, 2020.
- 5 Keren Censor-Hillel, François Le Gall, and Dean Leitersdorf. On distributed listing of cliques. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC 2020)*, pages 474–482, 2020.
- 6 Keren Censor-Hillel, Petteri Kaski, Janne H Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019.
- 7 Keren Censor-Hillel, Seri Khoury, and Ami Paz. Quadratic and near-quadratic lower bounds for the CONGEST model. In *Proceedings of the 31st International Symposium on Distributed Computing (DISC 2017)*, pages 10:1–10:16, 2017.
- 8 Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 821–840, 2019.
- 9 Yi-Jun Chang and Thatchaphol Saranurak. Improved distributed expander decomposition and nearly optimal triangle enumeration. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 66–73, 2019.
- 10 Derek G. Corneil, Yehoshua Perl, and Lorna K Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.
- 11 Artur Czumaj and Christian Konrad. Detecting cliques in CONGEST networks. *Distributed Computing*, 33(6):533–543, 2020.
- 12 Danny Dolev, Christoph Lenzen, and Shir Peled. “tri, tri again”: Finding triangles and small subgraphs in a distributed setting. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC 2012)*, pages 195–209, 2012.
- 13 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *Proceedings of the 33rd ACM Symposium on Principles of Distributed Computing (PODC 2014)*, pages 367–376, 2014.
- 14 Talya Eden, Nimrod Fiat, Orr Fischer, Fabian Kuhn, and Rotem Oshman. Sublinear-time distributed algorithms for detecting small cliques and even cycles. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC 2019)*, pages 15:1–15:16, 2019.
- 15 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.
- 16 Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2018)*, pages 153–162, 2018.
- 17 Pierre Fraigniaud, Pedro Montealegre, Dennis Olivetti, Ivan Rapaport, and Ioan Todinca. Distributed subgraph detection. *arXiv preprint*, 2017. [arXiv:1706.03996](https://arxiv.org/abs/1706.03996).
- 18 Pierre Fraigniaud and Dennis Olivetti. Distributed detection of cycles. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–20, 2019.



- 19 Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC 2016)*, 2016.
- 20 Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
- 21 Taisuke Izumi, François Le Gall, and Frédéric Magniez. Quantum distributed algorithm for triangle finding in the CONGEST model. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, pages 23:1–23:13, 2020.
- 22 Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the 36th ACM Symposium on Principles of Distributed Computing (PODC 2017)*, pages 381–389, 2017.
- 23 Janne H. Korhonen and Amir Nikabadi. Beyond distributed subgraph detection: Induced subgraphs, multicolored problems and graph parameters, 2021. [arXiv:2109.06561](https://arxiv.org/abs/2109.06561).
- 24 Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In *Proceedings of the 21th International Conference on Principles of Distributed Systems (OPODIS 2017)*, pages 4:1–4:16, 2017.
- 25 Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM Journal on Discrete Mathematics*, 27(2):892–909, 2013.
- 26 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in CONGEST networks. In *Proceedings of the 37th ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 337–346, 2018.
- 27 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 28 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *Proceedings of the 30th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2018)*, pages 405–414, 2018.
- 29 Alexander A Razborov. On the distributional complexity of disjointness. In *Proceedings of the 17th International Colloquium on Automata, Languages, and Programming (ICALP 1990)*, pages 249–253, 1990.
- 30 Virginia Vassilevska Williams, Joshua R Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the 24th annual ACM-SIAM symposium on Discrete algorithms (SODA 2014)*, pages 1671–1680, 2014.

## A Proof of Theorem 11

We first show the following two lemmas.

► **Lemma 16.** *Any subset of vertices  $\mathcal{C} \subseteq V$  of size  $8\ell$  in  $G_{x,y}$  which induces  $C_{8\ell}$  contains at most  $\ell$  vertices in  $S$  where  $S \in \{A_1, A_2, B_1, B_2\}$ .*

**Proof.** It is enough to check the case of  $S = A_1$ . Let  $c$  be the number of index  $i \in [n]$  such that  $|\mathcal{C} \cap A_1^i| > 0$ . At first, observe that if it holds that  $|\mathcal{C} \cap A_1^i| \geq 1$ ,  $|\mathcal{C} \cap A_1^j| \geq 1$ , and  $|\mathcal{C} \cap A_1^k| \geq 1$  for some distinct  $i, j, k \in [n]$ ,  $\mathcal{C}$  induces a triangle. Hence, we have  $c \leq 2$ . The proof is completed by the following case analysis.

1. **The case of  $\ell \geq 3$ :** Suppose that  $|\mathcal{C} \cap A_1| > \ell$ . Then we have  $c = 2$ . Let  $i, j \in [n]$  be the indices such that  $|\mathcal{C} \cap A_1^i| \geq |\mathcal{C} \cap A_1^j| > 0$ . If  $|\mathcal{C} \cap A_1^i| = 1$ , then  $|\mathcal{C} \cap A_1^j| \geq 3$ . This is not possible since the vertices in  $\mathcal{C} \cap A_1^j$  are connected at least three vertices in  $\mathcal{C} \cap A_1^i$ . If  $|\mathcal{C} \cap A_1^i| \geq 2$ , then  $|\mathcal{C} \cap A_1^j| \geq 2$ . This is not possible since  $\mathcal{C}$  contains a 4-cycle in this case.

2. **The case of  $\ell = 2$ :** Suppose that  $|\mathcal{C} \cap A_1| > \ell = 2$ . Then we have  $c = 2$  and let  $i, j \in [n]$  be the indices such that  $|\mathcal{C} \cap A_1^i| \geq |\mathcal{C} \cap A_1^j| > 0$ . If  $|\mathcal{C} \cap A_1^i| = 2$  and  $|\mathcal{C} \cap A_1^j| = 2$ , then  $\mathcal{C}$  induces  $C_4$ . Hence, we consider  $|\mathcal{C} \cap A_1^i| = 2$  and  $|\mathcal{C} \cap A_1^j| = 1$ . Denote  $\{a_1^{i,1}, a_1^{i,2}\} = \mathcal{C} \cap A_1^i, \{u\} = \mathcal{C} \cap A_1^j$ . Then,  $(a_1^{i,1}, u), (a_1^{i,2}, u) \in E \cap \{\mathcal{C} \times \mathcal{C}\}$ . The other edges which incident on  $a_1^{i,1}, a_1^{i,2}$  are both in  $A_2$  or both in  $U_A$ .
- a. In the former case, we have that  $\mathcal{C} \cap A_2 = A_2^k = \{a_2^{k,1}, a_2^{k,2}\}$  for some  $k \in [n]$ , otherwise  $\mathcal{C}$  includes an induced 5-cycle. Observe that due to our construction of  $G_{x,y}$ , six vertices in  $\mathcal{C}$  are automatically determined to  $Code(A_2^k), Code(B_2^k)$ , and  $B_2^k$ . It can be easily checked that no matter how we choose the remaining vertices,  $\mathcal{C}$  does not induce a 16-cycle.
  - b. In the latter case, it is automatically determined that  $\mathcal{C}$  includes  $Code(A_1^i), Code(B_1^i)$ , and  $B_1^i$ , due to our construction of  $G_{x,y}$ . In addition,  $\mathcal{C}$  includes  $B_2^k, Code(B_2^k), Code(A_2^k)$  for some  $k \in [n]$ . Then,  $\mathcal{C}$  does not induce a 16-cycle since two vertices of  $Code(A_2^k)$  do not share neighbors. ◀

► **Lemma 17.** Any subset of vertices  $\mathcal{C} \subseteq V$  of size  $8\ell$  in  $G_{x,y}$  which induces  $C_{8\ell}$  contains  $\ell$  vertices in  $S$ , where  $S \in \{A_1, A_2, B_1, B_2, U_A, U_B, L_A, L_B\}$ .

**Proof.** For  $S \subseteq V$ , we denote  $z(S) = |\mathcal{C} \cap S|$ . Observe that the number of edges in  $E_{\mathcal{C}}$  between  $A_1$  and  $U_A$  is at least  $z(U_A)$  since each vertex in  $\mathcal{C} \cap U_A$  has at least one neighbor in  $\mathcal{C} \cap A_1$ . On the other hand, the number of edges in  $E_{\mathcal{C}}$  between  $A_1$  and  $U_A$  is at most  $z(A_1)$  since each vertex in  $\mathcal{C} \cap A_1$  has at most one neighbor in  $\mathcal{C} \cap U_A$ . Hence, we have that  $z(A_1) \geq z(U_A)$ . Similar observation shows that  $z(A_2) \geq z(L_A)$ ,  $z(B_1) \geq z(U_B)$ , and  $z(B_2) \geq z(L_B)$ . Then, it holds that

$$\begin{aligned} 8\ell &= z(A_1) + z(A_2) + z(U_A) + z(L_A) + z(B_1) + z(B_2) + z(U_B) + z(L_B) \\ &\leq 2(z(A_1) + z(A_2) + z(B_1) + z(B_2)). \end{aligned}$$

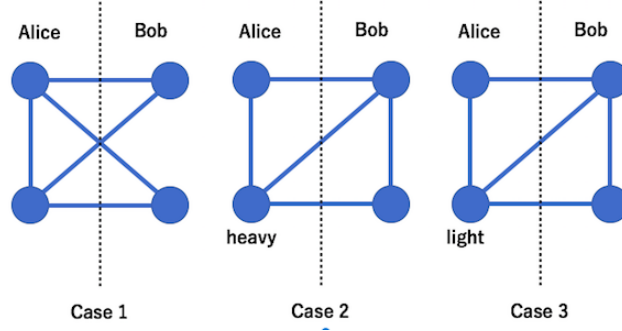
From Lemma 16, we have  $z(A_1) = z(A_2) = z(B_1) = z(B_2) = \ell$ . We also have  $z(U_A) = z(L_A) = z(U_B) = z(L_B) = \ell$  since

$$4\ell = z(U_A) + z(L_A) + z(U_B) + z(L_B) \blacktriangleright$$

**Proof of Theorem 11.** Let  $\mathcal{C} \subseteq V$  be a set of  $8\ell$  vertices which induces an  $8\ell$ -cycle in  $G_{x,y}$ . From Lemma 16 and Lemma 17,  $\mathcal{C}$  contains  $\ell$  vertices in each  $A_1^i, A_2^j, B_1^s$ , and  $B_2^t$  for some  $i, j, s, t \in [n]$ . Since  $A_1^i$  must be connected to  $\mathcal{C} \cap U_A$ , it holds  $Code(A_1^i) = \mathcal{C} \cap U_A$ . Similarly, we have that  $Code(A_2^j) = \mathcal{C} \cap L_A$ ,  $Code(B_1^s) = \mathcal{C} \cap U_B$ , and  $Code(B_2^t) = \mathcal{C} \cap L_B$ . It can be easily checked that  $\mathcal{C} = A_1^i \cup A_2^j \cup B_1^s \cup B_2^t \cup Code(A_1^i) \cup Code(A_2^j) \cup Code(B_1^s) \cup Code(B_2^t)$  induces  $C_{8\ell}$  iff  $i = s, j = t$ , and  $x_{i,j} = y_{i,j} = 1$ . ◀

## B Proof of Lemma 14

Let  $N(a_i^j) = \{b_1, b_2, \dots, b_{\sqrt{n}}\}$  be the set of neighbors of  $a_i^j$  in  $B$ . Consider  $A_k$  such that  $k \neq i$ . For any  $b_l \in N(a_i^j)$ , just one vertex that is connected to  $b_l$  is chosen uniformly at random from  $A_k$ . For  $a \in A_k$ , let  $X(a)$  be the indicator variable of the event “the pair  $(a, a_i^j)$  forms a good pair”. Then, the expected value of  $X(a)$  is  $E(X(a)) = \sqrt{n} \cdot \frac{1}{\sqrt{n}} \cdot \left(\frac{\sqrt{n}-1}{\sqrt{n}}\right)^{\sqrt{n}-1} \geq 1/e$ . Hence, the expected value of the number of vertices in  $A_k$  that form good pairs with  $a_i^j$  is greater than  $\sqrt{n}/e$  by linearity of expectation. The expected value of the number of vertices that form good pairs with  $a_i^j$  is  $\sqrt{n}/e \times (\sqrt{n} - 1) = \Omega(n)$ . Again, by using linearity of expectation, the expected value  $E(|P_A|) \geq \Omega(n) \cdot n/2 = \Omega(n^2)$ . This means that there exists a graph with the condition holds. ◀



■ **Figure 5** Three types of diamonds which have exactly two vertices in  $V_A$ .

### C Proof of Theorem 6

We show the two-party communication protocol for listing diamonds by modifying the protocol for listing cliques in [11]. More precisely, we show the following theorem.

► **Theorem 18.** *There is a two-party communication protocol in the vertex partition model for listing all diamonds that uses  $\tilde{O}(\sqrt{n}|E_{cut}|)$  communication where  $E_{cut}$  is a set of cut edges in the input graph.*

**Proof.** If  $|E_{cut}| \geq n^{3/2}$ , Alice can send  $E_A$  to Bob within  $O(\sqrt{n}|E_{cut}|)$  bits of communication since it holds  $\sqrt{n}|E_{cut}| \geq n^2$ . Suppose that  $|E_{cut}| < n^{3/2}$ . Since Alice (and Bob) can list all diamonds in which three or four vertices in Alice's side without communication, we only care about diamonds in which exactly two vertices are in Alice's side. Let  $V_A^{heavy} = \{v \in V_A : \deg_{V_B}(v) > \deg_{V_A}/\sqrt{n}\}$  and  $V_A^{light} = V_A \setminus V_A^{heavy}$ . As shown in Figure 5, there are three possible cases.

- To list diamonds of case 1, we can use the protocol for listing cliques in [11]. This requires  $O(\sqrt{n}|E_{cut}|)$  bits.
- To list diamonds of case 2, Alice sends edges  $E_A \cap \{V_A^{heavy} \times V_A\}$  to Bob. This requires  $O(\sqrt{n}|E_{cut}|)$  bits since the number of edges Alice sends to Bob is less than

$$\sum_{v \in V_A^{heavy}} \deg_{V_A}(v) \leq \sum_{v \in V_A^{heavy}} \sqrt{n} \cdot \deg_{V_B}(v) = \sqrt{n}|E_{cut}|.$$

- To list diamonds of case 3, for every  $v \in V_A^{light}$ , Bob sends edges  $E_B \cap \{N_{V_B}(v) \times N_{V_B}(v)\}$  to Alice. This requires  $O(\sqrt{n}|E_{cut}|)$  bits since the number of edges Bob sends to Alice is less than

$$\sum_{v \in V_A^{light}} (\deg_{V_B}(v))^2 \leq \sum_{v \in V_A^{light}} \frac{\deg_{V_A}(v)}{\sqrt{n}} \cdot \deg_{V_B}(v) \leq \sqrt{n} \sum_{v \in V_A^{light}} \deg_{V_B}(v) \leq \sqrt{n}|E_{cut}|. \blacktriangleleft$$

► **Theorem 6 (Formal statement).** *No family of lower bound graphs gives an  $\tilde{\Omega}(n^{1/2+\varepsilon})$  lower bound of induced diamond listing for any  $\varepsilon > 0$ .*

**Proof.** Exactly the same as how Theorem 4 was proved from Theorem 13. ◀

## D Sublinear-round listing of induced diamonds (Theorem 7)

Assume that, for some edge subset  $E' \subseteq E$  where  $|E'| = c|E|$  for some constant  $c > 0$ , all cliques that contain at least one edge from  $E'$  are listed by a procedure  $\mathcal{A}$ . We recursively apply  $\mathcal{A}$  for  $E \setminus E'$  since remaining cliques are the ones whose edges are in  $E \setminus E'$ . After  $O(\log n)$  levels of recursion of  $\mathcal{A}$ , all cliques in the original graph are listed since removing edges does not increase the number of cliques. Fastest (and optimal) clique listing algorithms [9, 2] use this recursive method. On the other hand, this cannot be used for induced subgraphs since removing edges may increase the number of induced subgraphs (e.g., removing one edge from a 4-clique creates an additional diamond). Instead, we can use  $K_4$  listing algorithm of [14] to list induced diamonds. The algorithm begins by computing the decomposition of edge set, in which the edge set are decomposed into two subsets: edges that induce clusters with low mixing time and edges between clusters. The clusters satisfy the following:

► **Definition 19** ( $\delta$ -cluster). *For an  $n$ -node graph  $G = (V, E)$  and a subgraph  $G' = (V', E')$  of  $G$ ,  $G'$  is called a  $\delta$ -cluster if the following condition holds:*

1. *Mixing time of  $G'$  is  $O(\text{poly log}(n))$ ,*
2. *For any  $v \in V'$ ,  $\deg_{E'}(v) = \Omega(n^\delta)$ .*

► **Lemma 20** (Expander Decomposition, Lemma 9 of [14]). *For a  $n$ -node CONGEST network  $G = (V, E)$ , we can find, w.h.p., in  $\tilde{O}(n^{1-\delta})$  rounds, a decomposition of  $E$  to  $E = E_m \cup E_s$  satisfying the following conditions.*

1.  $E_m$  is the union of at most  $s = O(\log n)$  sets,  $E_m = \bigcup_{i=1}^s E_m^i$ , where each  $E_m^i$  is the vertex-disjoint union of  $O(n^{1-\delta})$   $\delta$ -clusters,  $C_i^1, \dots, C_i^{k_i}$ .  
The set  $E_m^i$  is called  $i$ -th level of the decomposition. We say that a node  $u$  belongs to cluster  $C_i^j$  if at least one of  $u$ 's edges is in  $C_i^j$ .
2. Each level- $i$  cluster  $C_i^j$  has a unique identifier, which is a pair of the form  $(i, x)$  where  $x \in [n^{1-\delta}]$ , and an unique leader node, which is some node in the cluster. Each node  $u$  knows the identifier of all the clusters  $C_i^j$  to which  $u$  belongs, the leaders for those clusters, and it knows which of its edges belong to which clusters.
3.  $E_s = \bigcup_{v \in V} E_{s,v}$ , where  $E_{s,v}$  is a subset of edges incident to  $v$  and  $|E_{s,v}| \leq n^\delta \log n$ . Each vertex  $v$  knows  $E_{s,v}$ .

The reason that the expander decomposition is used in the distributed subgraph detection is that  $\delta$ -clusters can simulate CONGESTED CLIQUE style algorithms efficiently:

► **Lemma 21** (Lemma 13 of [14]). *For a constant  $0 < \varepsilon \leq 1$ , suppose an edge set  $E'$  is partitioned between the nodes of a  $\delta$ -cluster  $C$ , so that each node  $u \in C$  initially knows a subset  $E'_u$  of size at most  $O(n^{2-\varepsilon})$ . Then a simulation of  $t$  rounds of the CONGESTED CLIQUE algorithm on  $G' = (V, E')$  can be performed in  $\tilde{O}(n^{2-\delta-\varepsilon} + t \cdot n^{2-2\delta})$  rounds, with success probability  $1 - \frac{1}{n^2}$ .*

Roughly speaking, a  $\delta$ -cluster can simulate 1 round of a CONGESTED CLIQUE style algorithm in  $\tilde{O}(n^{2-2\delta})$  rounds.

After doing the decomposition  $E = E_m \cup E_s$ , the high-level approach of  $K_4$  listing algorithm of [14] is as follows:

1. To list  $K_4$  in  $E_s$ , we can use the trivial algorithm since the subgraph induced by  $E_s$  is sparse.

2. To list  $K_4$  which contains at least one edge in  $E_m$ , each cluster  $C$  gathers outside edges which are incident to a cluster node in  $\tilde{O}(n)$  rounds so that the number of edges gathered by each node of  $C$  is  $\tilde{O}(n^2)$ . All nodes that are outside of  $C$  with many neighbors in  $C$  send all edges incident to them. This can be done efficiently since they have enough communication bandwidth to  $C$ . Then,  $C$  simulates  $K_4$  listing algorithm of the CONGESTED CLIQUE model.
3. Note that each outside node  $u$  with small bandwidth to  $C$  have no ability to send all edges incident to it in  $\tilde{O}(n)$  rounds. However,  $u$  can quickly gather all cluster edges that would belong to some  $K_4$  which contains  $u$  since the number of cluster neighbors of  $u$  is small.

The difference is that in the case of diamonds, there are several types of diamonds which are listed in step 2 and step 3 of above algorithm (see Figure 6). For a cluster  $C$ , we call a node  $v$  is  $C$ -heavy when  $v$  does not belong to  $C$  and has more than  $n^\varepsilon$  neighbors in  $C$ . The algorithm for listing induced diamonds is as follows:

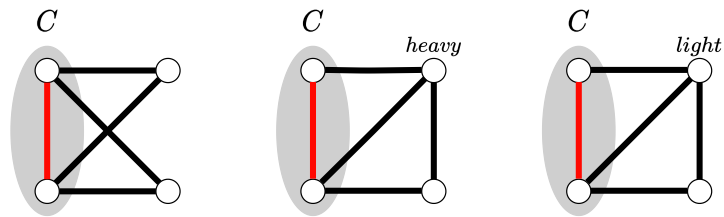
1. First, we run the expander decomposition of Lemma 20 in  $\tilde{O}(n^{1-\delta})$  rounds.
2. To list diamonds in  $E_s$ , each node  $u$  sends the edges of  $E_s$  that are incident on  $u$  to all neighbors. This requires  $\tilde{O}(n^\delta)$  rounds.
3. Each  $C$ -heavy node  $v$  sends  $N(v)$  to  $C$  in  $O(n^{1-\varepsilon})$  rounds by partitioning  $N(v)$  into  $|N(v) \cap C| = \Omega(n^\varepsilon)$  subsets of size  $|N(v)|/|N(v) \cap C| = O(n^{1-\varepsilon})$  and sending each subset to a different neighbor in  $C$ . Then, since each  $C$ -node  $u$  receive at most  $n \cdot n^{1-\varepsilon} = n^{2-\varepsilon}$  edges, we can list all induced diamonds which contains at least one edge from some cluster  $C$ , and contains a  $C$ -heavy node, in  $\tilde{O}(n^{2-\delta-\varepsilon} + \sqrt{n} \cdot n^{2-2\delta})$  rounds by the algorithm of Lemma 21.
4. Each  $C$ -light node  $u$  sends  $N(u) \cap C$  to all its neighbors in  $O(n^\varepsilon)$  rounds. Then, each node  $v$  received  $N(u) \cap C$  from  $u$  compute the following set locally:

$$\begin{aligned} \mathcal{L}_v^1 &= \{ \{u, c_1, c_2\} \mid u \in N(v) \text{ is } C\text{-light}, c_1, c_2 \in C, c_1 \in N(v) \cap N(u), c_2 \in N(u) \setminus N(v) \}, \\ \mathcal{L}_v^2 &= \{ \{u, c_1, c_2\} \mid u \notin N(v) \text{ is } C\text{-light}, c_1, c_2 \in C, c_1, c_2 \in N(v) \cap N(u), \\ &\quad (u, c_1), (u, c_2), (v, c_1), \text{ and } (v, c_2) \in E_s. \}, \end{aligned}$$

where  $\mathcal{L}_v^1$  and  $\mathcal{L}_v^2$  correspond to the rightmost and leftmost diamonds in Figure 6, respectively. Note that we do not have to care about the leftmost diamond of Figure 6 which contains an edge from another cluster  $C'$ : Among the leftmost diamonds  $\{u, v, c_1, c_2\}$  of Figure 6, where  $c_1, c_2$  belong to  $C$  and  $u, v$  are  $C$ -light nodes, we only need to enumerate the diamonds whose four edges  $(u, c_1), (u, c_2), (v, c_1)$  and  $(v, c_2)$  are  $E_s$ -edges. This is because, for instance, if  $(u, c_1)$  is  $C'$ -edge for some cluster  $C'$ , then this diamond is treated by the cluster  $C'$  as the middle or rightmost diamond in Figure 6. Since each node sends its  $E_s$ -edges to all its neighbors in step 2,  $v$  knows the four edges  $(u, c_1), (u, c_2), (v, c_1)$  and  $(v, c_2)$  even when  $u \notin N(v)$ . Then,  $v$  construct the following list of edge queries locally:

$$\mathcal{Q}_{v, c_1} = \{ \{c_1, c_2\} \mid \exists u \in N(v) : \{u, c_1, c_2\} \in \mathcal{L}_v^1 \text{ or } \exists u \notin N(v) : \{u, c_1, c_2\} \in \mathcal{L}_v^2 \}.$$

We have  $|\mathcal{Q}_{v, c_1}| = O(n^\varepsilon)$ : if  $c_1$  received  $\{c_1, c_2\} \in \mathcal{Q}_{v, c_1}$ , then  $c_2 \in N(v) \cap C$  by definition. On the other hand, since  $v$  is  $C$ -light,  $|N(v) \cap C| = O(n^\varepsilon)$  holds. Each node  $v$  sends  $\mathcal{Q}_{v, c_1}$  to  $c_1$ , and  $c_1$  responds with  $\mathcal{Q}_{v, c_1} \cap (\{c_1\} \times N(c_1))$  in  $O(n^\varepsilon)$  rounds. Therefore, in  $O(n^\varepsilon)$  rounds, we can list all induced diamonds which contains at least one edge from some cluster  $C$ , and does not contain  $C$ -heavy nodes.



■ **Figure 6** Three types of diamonds that contain at least one edge from  $E_m$ . Red edges in the figure correspond to  $E_m$ -edges, i.e., edges belong to some cluster  $C$ . Heavy node represents a  $C$ -heavy node, i.e., a node which does not belong to  $C$ , but has more than  $n^\varepsilon$  neighbors in  $C$ . Light node represents a  $C$ -light node.

Taking parameters  $\delta = 5/6$  and  $\varepsilon = 1/2$  (same as in the algorithm of [14]), we get the following theorem.

► **Theorem 7.** *Listing all induced diamonds can be done in  $\tilde{O}(n^{5/6})$  rounds with high probability in the CONGEST model.*






# Distributed Approximations of $f$ -Matchings and $b$ -Matchings in Graphs of Sub-Logarithmic Expansion

Andrzej Czygrinow ✉

School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ, USA

Michał Hanćkowiak ✉ 

Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Poland

Marcin Witkowski ✉ 

Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Poznań, Poland

---

## Abstract

---

We give a distributed algorithm which given  $\epsilon > 0$  finds a  $(1 - \epsilon)$ -factor approximation of a maximum  $f$ -matching in graphs  $G = (V, E)$  of sub-logarithmic expansion. Using a similar approach we also give a distributed approximation of a maximum  $b$ -matching in the same class of graphs provided the function  $b : V \rightarrow \mathbb{Z}^+$  is  $L$ -Lipschitz for some constant  $L$ . Both algorithms run in  $O(\log^* n)$  rounds in the LOCAL model, which is optimal.

**2012 ACM Subject Classification** Theory of computation → Distributed computing models

**Keywords and phrases** Distributed algorithms,  $f$ -matching,  $b$ -matching

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.59

**Funding** *Andrzej Czygrinow*: Research supported in part by Simons Foundation Grant # 521777.

## 1 Introduction

A matching in a graph  $G = (V, E)$  is a set of edges  $M \subseteq E$  such that every vertex  $v \in V$  belongs to at most one edge  $e \in M$ . Although the concept of a matching can be generalized in a few different ways, two natural and useful notions that have been considered in this context are that of an  $f$ -matching and a  $b$ -matching. Let  $G = (V, E)$  be a (simple) graph and let  $f : V \rightarrow \mathbb{Z}^+$ . A set  $M \subseteq E$  is called an  $f$ -matching in  $G$  if for every  $v \in V$ ,  $v$  is incident to at most  $f(v)$  edges from  $M$ . Given a function  $b : V \rightarrow \mathbb{Z}^+$ , a  $b$ -matching in  $G$  is a function  $x : E \rightarrow \mathbb{N}$  such that for every vertex  $v \in V$ ,  $\sum_{y \in N(v)} x(vy) \leq b(v)$ . Sometimes an  $f$ -matching is called a simple  $b$ -matching (a capacitated  $b$ -matching with bound one for capacities) [15].

In particular, in the case when  $f = 1$  ( $b = 1$ ), then an  $f$ -matching ( $b$ -matching) is simply a matching. The main difference between  $f$ -matchings and  $b$ -matchings is that in the case of  $b$ -matchings  $x : E \rightarrow \mathbb{N}$  and the corresponding assignment  $x$  for  $f$ -matching satisfies  $x : E \rightarrow \{0, 1\}$ . We shall use  $\nu_f(G)$  and  $\nu_b(G)$  to denote the maximum  $f$ -matching and  $b$ -matching, that is  $\nu_f(G) = \max\{|M| : M \text{ is an } f\text{-matching in } G\}$  and  $\nu_b(G) = \max\{\sum_{e \in E} x(e) : x \text{ is a } b\text{-matching in } G\}$ . In addition, we use  $\nu(G)$  for the size of a maximum matching in  $G$ .

We will use the LOCAL model from Peleg's book [14]. In this model a distributed network is modeled as an undirected graph with vertices corresponding to computational units and edges representing bidirectional links between them. The computations are synchronized, and in each round every vertex can send and receive messages from its neighbors, and, in addition, can perform some local computations. Neither the size of messages sent nor the amount of computations is restricted in any way. In addition, vertices have unique identifiers from  $\{1, \dots, n\}$  where  $n$  is the order of the graph.



© Andrzej Czygrinow, Michał Hanćkowiak, and Marcin Witkowski;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 59; pp. 59:1–59:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1.1 Related Work

Although distributed algorithms for matchings have been studied extensively, little is known about  $f$ -matchings and  $b$ -matchings.

A maximal matching gives a  $1/2$ -factor approximation of a maximum and there has been some success in designing distributed algorithms that find a maximal matching. Seminal papers of Hanćkowiak, Karoński and Panconesi [8, 9] give deterministic poly-logarithmic algorithms for the maximal matching problem. Recently, a faster distributed algorithm for the maximal matching problem was given by Fischer [5]. This algorithm runs in  $O(\log^2 |V| \log \Delta)$  rounds in graphs  $G = (V, E)$  of maximum degree  $\Delta$ . At the same time Kuhn et al. [12] showed that finding a maximal matching in a graph  $G$  of order  $n$  and maximum degree  $\Delta$  requires  $\Omega(\log \Delta / \log \log \Delta + \sqrt{\log n / \log \log n})$  rounds.

Distributed approximations for the maximum matching problem which run in a poly-logarithmic number of rounds are known [2, 3]. Quite recently, Ghaffari, Harris and Kuhn [6] gave a  $O(\log^2 n \log^5(\Delta/\epsilon^9))$  algorithm that finds a  $(1 - \epsilon)$ -factor approximation of a maximum matching in a graph on  $n$  vertices with maximum degree  $\Delta$ , and Harris [10] gave fast approximation algorithms for weighted maximum matching and hypergraph matching.

As impressive as these algorithms are, their time complexity is often prohibitively high. Not surprisingly, there has been a lot of interest in designing faster distributed algorithms for special classes of graphs. For planar graphs, there is a distributed deterministic algorithm [4] which given  $\epsilon > 0$  finds in a graph  $G$  of order  $n$  in  $O(\log^* n)$  rounds, a matching  $M$  such that  $|M| \geq (1 - \epsilon)\nu(G)$ . This algorithm proceeds in two main steps. First, an ad-hoc procedure is used to reduce a given graph  $G = (V, E)$  to a graph  $G' = (V', E')$  such that (1)  $|V'|$  is proportional to the size of a maximum matching and (2) a maximum matching in  $G'$  is also maximum in  $G$  and (3)  $G'$  is still planar. Second, a clustering algorithm is used to partition  $V'$  into sets  $V_1, \dots, V_i$  so that it is possible to quickly find optimal solutions in graphs  $G'[V_i]$  and combine them to obtain a good approximation in the whole graph  $G'$  and so  $G$ .

The notions of an  $f$ -matching and  $b$ -matching are significantly more general. Very little is known about the distributed complexity of  $f$ -matchings except what can be concluded from the case  $f = 1$ . Using the Tutte's construction (see for example [15]) one can reduce  $b$ -matchings to matchings but this reduction infuses potentially large complete bipartite graphs.

Hanćkowiak [7] gave a poly-logarithmic running time algorithm for the maximal  $f$ -matching problem in general graphs which builds on the approach for matchings [8]. In addition, extending the approach for matchings, Fischer [5] managed to give a deterministic distributed algorithm for  $(1/2 - \epsilon)$ -approximating a maximum  $f$ -matching in general graphs that runs in  $O(\log^2 \Delta \log 1/\epsilon + \log^* |V|)$  rounds. In the case of a maximum  $b$ -matching it is possible to reduce it to the problem of a maximum matching but the reduction leads to graphs which can potentially contain large complete bipartite graphs. Consequently, the graph obtained after reduction can have minors of large cliques. As a result, maximum matching algorithms that exploit sparseness conditions cannot be invoked in the graph obtained by the reduction.

## 1.2 Results

We will give approximation algorithms for  $f$ -matchings and  $b$ -matchings. Although our primary interest comes from graphs that are  $K_m$ -minor-free for a fixed integer  $m$ , it is possible to phrase the results in terms of graphs of sub-logarithmic expansion which generalize the former class. (See Section 2 for definitions.)

Our first contribution is a distributed algorithm which given a graph  $G = (V, E)$  of sub-logarithmic expansion and  $\epsilon > 0$  finds a  $(1 - \epsilon)$ -factor approximation of a maximum  $f$ -matching in  $G$ . The algorithm runs in  $O(\log^* |V|)$  rounds (Theorem 7). The algorithm proceeds in two main steps, the first, described in detail in what follows, reduces the input graph to a graph in which a maximum  $f$ -matching is proportional to the number of vertices and the second invokes the clustering procedure of Amiri et al. [1].

The case of  $b$ -matchings is surprisingly more subtle in the realm of sparse graphs and our approach requires an additional assumption about function  $b$ . Specifically, we shall require that  $b$  is  $L$ -Lipschitz for some constant  $L$ , which is known to the algorithm. Under the same regime as  $f$ -matchings, we again give a  $O(\log^* |V|)$ -time distributed  $(1 - \epsilon)$ -factor approximation (Theorem 17).

It is known [4] that finding a constant approximation of a maximum matching or maximum independent set in a cycle on  $n$  vertices requires  $\Omega(\log^* n)$  rounds and so the running time in Theorem 7 and Theorem 17 cannot be improved.

Algorithms for both theorems use in their second main step, i.e. the clustering procedure, the procedure of Amiri et al. [1] which is based on the algorithm of Czygrinow et al. [4]. However the first main step of both algorithms, i.e. the reduction phase is new and relies on the Gallai-Edmonds theorem [11]. The rest of the paper is structured as follows. In Section 2 we introduce necessary terminology. In Section 3 we discuss  $f$ -matchings and Section 4 is devoted to  $b$ -matchings.

## 2 Preliminaries

For a positive integer  $r$  and a graph  $H$ , we say that  $H$  is a *minor of depth  $r$*  of graph  $G = (V, E)$  if for some subgraph  $G' = (V', E')$  of  $G$  it is possible to partition  $V'$  into sets  $V_1, \dots, V_l$  so that for every  $i$ ,  $G'[V_i]$  has radius at most  $r$  and the graph obtained by contracting each  $V_i$  to a vertex is isomorphic to  $H$ . We will set  $\nabla_r(G) = \max_H \frac{|E(H)|}{|V(H)|}$  where the maximum is taken over all minors  $H$  of depth  $r$  of  $G$ . A graph  $G$  is said to have a *bounded expansion* if there exists  $g : \mathbb{N} \rightarrow \mathbb{N}$  such that for every  $r \in \mathbb{N}$ ,  $\nabla_r(G) \leq g(r)$ . Note that surprisingly many classes of graphs have bounded expansion and we refer to [13] for an extensive discussion. In this paper, we will consider graphs  $G$  of sub-logarithmic expansion, that is graphs  $G$  such that  $\nabla_r(G) \leq g(r)$  for some  $g(r) \in o(\log r)$ . The class was introduced by Amiri et al. [1] where it is shown that the clustering algorithm from [4] works in this more general setting. In the case graph  $G$  is  $K_m$ -minor-free for some fixed  $m$ ,  $\nabla_r(G)$  can be bounded from above by a constant which depends on  $m$  only and so graphs of sub-logarithmic expansion include graphs that are  $K_m$ -minor-free for a fixed  $m$ . In addition, we obviously have  $\nabla_r(G) \leq \nabla_{r+1}(G)$ .

Our analysis relies on the Gallai-Edmonds theorem. To state it we need some additional terminology. A  $k$ -factor of a graph  $G$  is a spanning  $k$ -regular subgraph of  $G$ . In particular, a 1-factor is a perfect matching. For a graph  $H$ , we use  $\mathcal{C}_H$  to denote the set of (connected) components of  $H$ . A graph  $H$  is called *factor-critical* if, for every  $v \in V(H)$ , graph  $H - v$  has a 1-factor. Note that a component with only one vertex is factor-critical. Gallai-Edmonds theorem (see for example Kotlov [11]) will play a major role in our analysis. For a graph  $G = (V, E)$  let  $A$  be the set of vertices  $v \in V$  such that there is a maximum matching in  $G$  that does not cover  $v$ . Let  $B := N(A) = \{v \in V \setminus A \mid \exists w \in A vw \in E\}$ , and let  $C := V \setminus (A \cup B)$ . Clearly,  $\{A, B, C\}$  is a partition  $V$  (although some sets can be empty) which we will call a *Gallai-Edmonds decomposition* of graph  $G$ . We have the following theorem.

► **Theorem 1** (Gallai-Edmonds Theorem). *Let  $G = (V, E)$  be a graph and let  $\{A, B, C\}$  be a Gallai-Edmonds decomposition of  $G$ . Then the following conditions hold.*

- (a) *Every odd component  $H$  of  $G - B$  is factor-critical and  $V(H) \subseteq A$ .*
- (b) *Every even component  $H$  of  $G - B$  has a perfect matching and  $V(H) \subseteq C$ .*
- (c) *For every non-empty subset  $X \subseteq B$ ,  $N(X)$  has vertices from more than  $|X|$  odd components of  $G - B$ .*

In particular, in view of Hall's theorem,  $B$  is matchable to the set of odd components in  $G - B$ .

Fix  $g : \mathbb{N} \rightarrow \mathbb{N}$  such that  $g(r) = o(\log r)$  and  $\nabla_r(G) \leq g(r)$ . It will be convenient to define  $D := 2 \cdot g(1)$ . Although the algorithms do not need to have perfect information about  $g$ , they certainly need to know  $D$ . By definition, for every subgraph  $H$  of  $G$  we have

$$\sum_{v \in V(H)} \deg_H(v) \leq D \cdot |V(H)|. \quad (1)$$

We will call a clique  $S$  an  $i$ -clique if  $|S| = i$ .

► **Lemma 2.** *Let  $G$  be a graph of order  $n$ ,  $D$  be such that  $\nabla_0(G) \leq D/2$  and let  $i \geq 2$ . Then there is a vertex  $v \in V(G)$  that belongs to at most  $\binom{D}{i-1}$   $i$ -cliques.*

**Proof.** Note that the statement is true when  $i = 2$ . For  $i \geq 3$ , let  $v$  be a vertex of minimum degree which by (1) is at most  $D$ . Then every  $i$ -clique containing  $v$  is a subset of  $N[v]$ . ◀

### 3 $f$ -matchings

Assume that  $G$  satisfies  $\nabla_r(G) \leq g(r)$  and  $D = 2 \cdot g(1)$ . We call  $u, v \in V(G)$   $i$ -clones if  $N(u) = N(v)$  and  $|N(u)| = i$ . Note that we consider open neighborhoods and so if  $uv$  is an edge then  $u$  and  $v$  are not clones. The relation of being  $i$ -clones,  $\sim_i$ , is symmetric, transitive, and reflexive on the set  $V_i := \{v \mid \deg(v) = i\}$ . Let  $v_1, \dots, v_l$  be  $i$ -clones such that  $N(v_1) = \{u_1, \dots, u_i\}$  and  $l \geq i + 1$ . Since for every  $j \in \{1, \dots, i\}$ ,  $\deg(u_j) \geq l \geq i + 1$ ,  $u_j$  cannot be an  $i$ -clone of any other vertex.

In the first phase of the algorithm we trim graph  $G$ , discard some edges, and delete isolated vertices so that for the trimmed graph  $G'$  we have  $\nu_f(G') = \nu_f(G)$  and  $\nu_f(G') = \Omega(|G'|)$ . Phase 1 is split into two procedures DISCARD EDGES and TRIMMING.

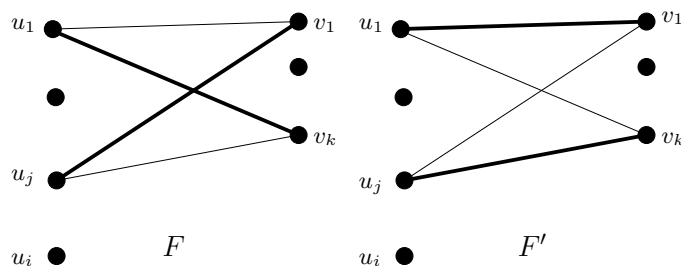
■ **Procedure** DISCARD EDGES  $(G, i)$ .

- 
- Let  $G'$  be the graph obtained from  $G$  by deleting the following edges. For every equivalence class  $\{v_1, \dots, v_l\}$  of  $\sim_i$  in  $V_i$  which satisfies  $f(v_1) \geq f(v_2) \geq \dots \geq f(v_l)$ :
    - If  $l \geq D$  and  $N(v_1) = \{u_1, \dots, u_i\}$  then, assuming  $f(u_1) \leq f(u_j)$  for every  $j \in \{1, \dots, i\}$ , delete edges  $u_1 v_j$  for  $j > \max\{f(u_1), D\}$ .
  - Return  $G'$ .
- 

We will first show that graph  $G'$  obtained by DISCARD EDGES has the same size of a maximum  $f$ -matching as  $G$  does.

► **Lemma 3.** *Let  $i \in \mathbb{Z}^+$  and let  $G'$  denote the graph returned by DISCARD EDGES  $(G, i)$ . Then*

$$\nu_f(G') = \nu_f(G).$$



■ **Figure 1** Case 2 of the proof of Lemma 3.

**Proof.** Clearly  $\nu_f(G') \leq \nu_f(G)$ . To prove the opposite inequality consider  $f$ -matching  $F$  in  $G$  of size  $\nu_f(G)$  that contains as few edges from  $E(G) \setminus E(G')$  as possible. Note that only edges incident to  $u_1$  are removed from  $G$ . Suppose  $F$  contains  $u_1v_k$  for some  $k > \max\{f(u_1), D\}$ . Thus from all edges  $u_1v_1, \dots, u_1v_{f(u_1)}$  that are in  $G'$ , at least one of them, say  $u_1v_1$ , is not in  $F$ .

**Case 1:** If  $u_jv_1 \in F$  for at most  $f(v_1) - 1$  edges  $u_jv_1$ , then consider  $F' = F - u_1v_k + u_1v_1$  and note that  $F'$  is an  $f$ -matching of size  $|F|$  with fewer edges from  $E(G) \setminus E(G')$  than  $F$ .

**Case 2:** For some  $j \in \{1, \dots, i\}$ ,  $u_jv_1 \in F$  but  $u_jv_k \notin F$ . Then consider  $F' = F - u_1v_k - u_jv_1 + u_1v_1 + u_jv_k$ .

If none of the cases 1 and 2 hold, then  $v_1$  is incident to  $f(v_1)$  edges  $v_1u_j \in F$  such that  $j \geq 2$  and  $v_k$  is incident to  $f(v_1) + 1$  edges from  $F$  because we have that  $u_jv_k \in F$  whenever  $u_jv_1 \in F$  and additionally  $u_jv_k \in F$ . Thus  $f(v_k) > f(v_1)$ , but by definition of the algorithm,  $f(v_k)$  is at most  $f(v_j)$  for every  $j \leq \max\{f(u_1), D\}$ . ◀

We now consider the main trimming procedure.

■ **Procedure** TRIMMING( $G$ ).

- 
- For  $i = D - 1$  downto 1, let  $G := \text{DISCARD\_EDGES}(G, i)$ .
  - Let  $H$  be obtained from  $G$  by deleting all isolated vertices.
  - Return  $H$ .
- 

Let  $H$  be the graph returned by TRIMMING( $G$ ) for the original graph  $G$ . In addition, we use  $G_i$  to denote graph  $G$  for which we call DISCARD EDGES( $G, i$ ) in the  $i$ th iteration of the loop in step one. Before proving our main lemma, we show the following observation.

► **Lemma 4.** Let  $1 \leq k \leq D - 1$ , and let  $Y = \{y_1, \dots, y_j\}$ ,  $|Y| = j$ , be  $k$ -clones in  $H$  such that  $N_H(y_1) = \{u_1, \dots, u_k\}$ . Then

$$j \leq D \cdot f^*,$$

where  $f^* = \min_{1 \leq i \leq k} \{f(u_i)\}$ .

**Proof.** We claim that  $y_1, \dots, y_j$  are  $k$ -clones in  $G_k$ . First note that in view of the main step of DISCARD EDGES, if  $x$  is a vertex in  $G$  such that  $\text{deg}_{G_l}(x) \geq D$  for some  $l < D$ , then  $\text{deg}_H(x) \geq D$ . Indeed,  $v_j$  has degree less than  $D$  as  $l < D$ , moreover for  $u_1$  edges  $u_1v_j$  can be deleted, but DISCARD EDGES always keeps at least  $D$  edges incident to  $u_1$ .

We clearly have  $N_H(y_l) \subseteq N_{G_k}(y_l)$  for every  $l = 1, \dots, j$ . Suppose  $k = \deg_H(y_l) < \deg_{G_k}(y_l)$  for some  $l$ . Then  $y_l$  had some edges deleted from the set of edges incident to  $y_l$  in  $G_k$  in iteration  $i$  for some  $i \leq k$ . However in iteration  $i$ , we delete edges incident to vertices of degree  $i$  or vertices of degree larger than  $D$ , keeping at least  $D$  of them. Thus either  $\deg_H(y_l) < k$ , which is not possible, or  $\deg_H(y_l) \geq D$ , contradicting  $k \leq D - 1$ .

Since  $y_1, \dots, y_j$  are  $k$ -clones in  $G_k$ , we have  $j \leq \max\{f^*, D\}$  after step two of DISCARD EDGES( $G_k, k$ ). Consequently,  $j \leq D \cdot f^*$  because  $f^*$  and  $D$  are at least one.  $\blacktriangleleft$

By Lemma 3, we have

$$\nu_f(H) = \nu_f(G), \quad (2)$$

and  $H$  has no isolated vertices.

We will now establish our second fact which shows that  $\nu_f(H)$  is proportional to the order of  $H$ .

**► Lemma 5.** *Let  $G$  be a graph such that  $\nabla_1(G) \leq D/2$  and let  $H$  be obtained by calling TRIMMING( $G$ ). Then  $\nu_f(H) \geq c_D |V(H)|$  for some  $c_D$  which depends on  $D$  only.*

**Proof.** We will use Theorem 1. Let  $\{A, B, C\}$  be a Gallai-Edmonds decomposition of  $H$ . Then the size of a maximum matching in  $H$  is  $\Omega(|B| + \sum_W |W|)$  where the sum is taken over components  $W$  of  $H - B$  of order at least two. Since a maximum  $f$ -matching has size larger than or equal to the size of a maximum matching,

$$\nu_f(H) = \Omega(|B| + \sum_W |W|).$$

Let  $X$  be the set of components of  $H - B$  of size one. We will identify each component from  $X$  with the vertex it contains. Since  $H$  has no isolated vertices, every component from  $X$  has at least one neighbor in  $B$ . Let  $Y$  be the set of components in  $X$  which have at least  $D$  neighbors in  $B$ . We have  $D \cdot |Y| \leq |E_H(Y, B)| \leq D(|Y| + |B|)/2$  from (1), and so  $|Y| \leq |B|$ . Thus  $\nu_f(H) = \Omega(|Y|)$ . For  $i = 1, \dots, D - 1$ , let  $X_i \subseteq X$  denote the set of components that have exactly  $i$  neighbors in  $B$ . In the rest of the argument we will show that for  $i \in \{1, \dots, D - 1\}$ ,  $\nu_f(H) = \Omega_D(|X_i|)$ . Given that this is the case, we can conclude

$$\nu_f(H) = \Omega(|B| + \sum_W |W| + |Y| + \sum_{i < D} |X_i|) = \Omega_D(|V(H)|).$$

**▷ Claim 6.** For  $i \in \{1, \dots, D - 1\}$ ,  $\nu_f(H) = \Omega_D(|X_i|)$ .

**Proof.** Let  $H^* := (B, \emptyset)$  be the edge-less graph on the set  $B$ . Order the vertices in  $X_i$  and proceed one by one for as long as possible using the following procedure. Take  $y \in X_i$ . If  $N_H(y)$  is not an  $i$ -clique in  $H^*$ , then take two non-adjacent vertices  $v_1, v_2$  from  $N_H(y) \subseteq B$ , add an edge between them to  $H^*$ , and delete  $y$  from  $X_i$ . Formally, we delete all edges incident to  $y$  except  $yv_1, yv_2$  and contract the edge  $yv_1$ . Let  $X_i^*$  denote the set of deleted vertices. Then  $|X_i^*| = |E(H^*)| \leq \nabla_1(G)|B| \leq D|B|/2$ . We will now bound the number of remaining vertices. Unlike the previous part of the argument, we will show that there is an  $f$ -matching  $F$  in  $H$  such that  $|F| \geq \Omega_D(|X_i \setminus X_i^*|)$ .

For every vertex  $y \in X_i \setminus X_i^*$ ,  $N_H(y)$  is an  $i$ -clique in  $H^*$ , and if  $y, y' \in X_i \setminus X_i^*$  are such that  $N_H(y) = N_H(y')$ , then  $y$  and  $y'$  are  $i$ -clones. We will put weights on these cliques. Specifically, the weight of an  $i$ -clique  $T$  is the number of vertices  $y$  in  $X_i \setminus X_i^*$  such that  $N_H(y) = T$ . By Lemma 4, for every  $i$ -clique  $T$  in  $H^*$ , the weight of  $T$  satisfies  $\omega(T) \leq D \cdot f_T$ , where  $f_T = \min_{u \in T} f(u)$ . In addition, we have

$$\sum_T \omega(T) = |X_i \setminus X_i^*|. \quad (3)$$

We will now assign  $i$ -cliques to vertices from  $B$  so that the following conditions hold:

- If  $T$  is assigned to  $v$ , then  $v \in T$ ;
- There are at most  $\binom{D}{i-1}$  cliques assigned to every vertex  $v$ ;
- Every  $i$ -clique in  $H^*$  is assigned to exactly one vertex from  $B$ .

This is possible, because by Lemma 2, there is a vertex  $v$  in  $B$  which is in at most  $\binom{D}{i-1}$   $i$ -cliques  $T$  in  $H^*$ . We assign these cliques to  $v$  and continue the process (note that  $\nabla_0(H^* - v) \leq \nabla_0(H^*)$ ). Let  $E_v$  denote the set of cliques assigned to  $v$ . For every  $T \in E_v$ , the weight satisfies  $\omega(T) \leq D \cdot f_T \leq D \cdot f(v)$  because  $v \in T$ . Let  $T_v \in E_v$  be an  $i$ -clique in  $E_v$  of the largest weight. Then, by (3),

$$\sum_v \omega(T_v) \geq \sum_v \sum_{T \in E_v} \omega(T) / \binom{D}{i-1} = \frac{\sum_T \omega(T)}{\binom{D}{i-1}} = |X_i \setminus X_i^*| / \binom{D}{i-1}.$$

Since  $f(x) \geq 1$  for every  $x$ , we can select  $\lceil \omega(T_v)/D \rceil \leq f(v)$  vertices  $y \in X_i \setminus X_i^*$  to get star  $F_v$  with center at  $v$  and selected vertices  $y$  as its leaves. Since  $F_u \cap F_v = \emptyset$  when  $u \neq v$ ,  $\bigcup F_v$  is an  $f$ -matching in  $H$  of size at least  $|X_i \setminus X_i^*| / (D \cdot \binom{D}{i-1})$ . Consequently,

$$|X_i| = |X_i^*| + |X_i \setminus X_i^*| = O_D(|B|) + O_D(\nu_f(H)) = O_D(\nu_f(H)). \quad \triangleleft$$

That concludes the proof of Lemma 5. ◀

We will now proceed to Phase 2 of the algorithm. For this phase we need to know that the graph  $G$  has sub-logarithmic expansion  $g$ . Note that to use the algorithm in Phase 2, constant  $c_D$  (or a bound for it) must be provided and, in addition, the algorithm from [1] which is used in step 2 requires some knowledge of the function  $g$ .

■ **Procedure** APPROXIMATION( $G, \epsilon, g$ ).

- 
- Use TRIMMING( $G$ ) to obtain graph  $H$ .
  - Use the algorithm from [1] to find a partition of  $V(H)$  into sets  $V_1, \dots, V_i$  such that  $\text{diam}(V_i) = O_{\epsilon, D}(1)$  and the number of edges with endpoints in different sets  $V_i$  is at most  $\epsilon \cdot c_D |H|$ .
  - Find an optimal solution  $F_i$  in  $H[V_i]$  and return  $\bigcup F_i$ .
- 

► **Theorem 7.** *There is a distributed algorithm which given a graph  $G$  on  $n$  vertices of expansion  $g$  such that  $g(r) = o(\log r)$  and  $\epsilon > 0$  finds an  $f$ -matching  $F$  in  $G$  such that  $|F| \geq (1 - \epsilon)\nu_f(G)$ . The algorithm runs in  $O_{\epsilon, D}(\log^* n)$  rounds.*

**Proof.** Use APPROXIMATION( $G, \epsilon, g$ ). By (2),  $\nu_f(H) = \nu_f(G)$  and by Lemma 5,  $\nu_f(H) \geq c_D |H|$ . We have

$$\nu_f(G) \leq \sum_i |F_i| + \epsilon \cdot c_D |V(H)| \leq |F| + \epsilon \nu_f(G).$$

TRIMMING runs in the number of rounds which depends on  $D$  only and the running time of the algorithm from [1] is  $O(\log^* n)$ . ◀



## 4 $b$ -matchings

In this section, we will discuss maximum  $b$ -matchings. All graphs are finite and simple. As before, we will consider graphs with sub-logarithmic expansion  $g$  and set  $D := 2 \cdot g(1)$ .

► **Definition 8.** Given a (simple) graph  $G = (V, E)$  and a function  $b : V \rightarrow \mathbb{Z}^+$ , let  $\nu_b(G) = \max\{\sum_{e \in E} x(e)\}$  where the maximum is taken over all functions  $x : E \rightarrow \mathbb{N}$  such that for every vertex  $v \in V$ ,  $\sum_{y \in N(v)} x(vy) \leq b(v)$ .

In our reduction algorithm, we will assume that  $b$  is  $L$ -Lipschitz for some given constant  $L$ , that is  $b : (V, d_1) \rightarrow (\mathbb{Z}^+, d_2)$  satisfies  $d_2(b(u), b(v)) \leq Ld_1(u, v)$  for any  $u, v \in V$ , where  $d_1$  is the metric determined by the distance in graph  $G = (V, E)$  and  $d_2(a, b) = |a - b|$ . The condition will be of no relevance if  $u$  and  $v$  are in different connected components of  $G$ .

To start the analysis we have the following observation.

► **Fact 9.** Let  $G = (V, E)$  be a graph with  $\nabla_0(G) \leq D/2$  and let  $b : V \rightarrow \mathbb{Z}^+$ . Define  $\omega : E \rightarrow \mathbb{Z}^+$  as

$$\omega(uv) := \min\{b(u), b(v)\}. \quad (4)$$

Then (a)  $\omega(E) \leq D \cdot b(V)$  and (b)  $\nu_b(G) \leq \omega(E)$ .

**Proof.** We shall prove (a) by induction on  $|V|$ . The base case is obvious. For the inductive step, by (1), the average degree of  $G$  is at most  $D$  and so there is a vertex  $v \in V$  of degree at most  $D$ . By induction,  $\omega(E(G - v)) \leq D \cdot b(V \setminus \{v\})$  and the weight on edges incident to  $v$  is at most  $D \cdot b(v)$  by (4). For (b), if  $x : E \rightarrow \mathbb{N}$  is such that  $x(E) = \nu_b(G)$ , then for every  $e \in E$ ,  $x(e) \leq \omega(e)$ . ◀

The general idea behind the approach is the same as in the case of  $f$ -matchings. We first reduce graph  $G$  using the notion of  $i$ -clones and then apply clustering. However, the reduction and the fact that it accomplishes the desired result (Lemma 16) require additional care.

■ **Procedure** `MODIFY( $G, b, i$ )`.

- 
- Let  $G'$  be the graph obtained from  $G$  by deleting the following edges. For every equivalence class  $\{v_1, \dots, v_l\}$  of  $\sim_i$  in  $V_i$  in parallel:
    - If  $l > D$  and  $N(v_1) = \{u_1, \dots, u_i\}$ , then delete vertices  $v_{D+1}, \dots, v_l$ . Set  $b'(v_D) := b(v_D) + \dots + b(v_l)$  and  $b'(w) := b(w)$  for any other vertex  $w$ .
  - Return  $(G', b')$ .
- 

For a graph  $G = (V, E)$  and  $v \in V$ , we used  $E_G(v)$  to denote the set of edges  $e \in E$  such that  $v \in e$ .

► **Fact 10.** Let  $(G', b')$  denote the pair returned by `MODIFY( $G, b, i$ )`.

(a) Then  $\nu_{b'}(G') = \nu_b(G)$ .

(b) For every  $u \in V(G')$  either  $\deg_{G'}(u) \geq D$  or  $N_{G'}(u) = N_G(u)$ .

**Proof.** We will first show part (a). Let  $x : E(G) \rightarrow \mathbb{N}$  be a maximum  $b$ -matching in  $G$  and let  $a_p = \sum_{k=1}^i x(v_p u_k)$ . Let  $x'$  be obtained by setting  $x'(v_D u_k) := \sum_{p=D}^l x(v_p u_k)$  and  $x'(v_j u_k) := x(v_j u_k)$  for  $j < D$ . Then  $x' : E(G') \rightarrow \mathbb{N}$  and for every  $k \in \{1, \dots, i\}$ ,

$$\sum_{e \in E_G(u_k)} x(e) = \sum_{e \in E_{G'}(u_k)} x'(e)$$

as the total value of  $x$  on edges between  $u_k$  and  $v_1, \dots, v_l$  stays the same. Obviously, for  $j < D$ , we have  $\sum_{k=1}^i x'(v_j u_k) = \sum_{k=1}^i x(v_j u_k)$  and, in addition,

$$\sum_{k=1}^i x'(v_D u_k) = \sum_{p=D}^l a_p \leq \sum_{p=D}^l b(v_p) = b'(v_D)$$

as  $a_p \leq b(v_p)$ . Thus  $x'$  is a  $b'$ -matching in  $G'$  of the same value as  $x$ .

Similarly, let  $x' : E(G') \rightarrow \mathbb{N}$  be a maximum  $b'$ -matching in  $G'$ . Then  $\sum_{e \in E_{G'}(u_k)} x'(e) \leq b'(u_k) = b(u_k)$  for every  $k \in \{1, \dots, i\}$ , and  $\sum_{k=1}^i x'(v_D u_k) \leq b'(v_D)$ . For  $p > D$ , taking  $a_p$  to be the maximum value such that  $0 \leq a_p \leq b(v_p)$  and  $\sum_{k=1}^i x'(v_D u_k) - a_p \geq 0$ , we can write  $a_p = \sum_{k=1}^i a_{pi}$  where  $0 \leq a_{pi} \leq x'(v_D u_k)$  and assign  $a_{pi}$  to  $v_p u_k$  so that the total of  $a_p$  is reassigned from  $E(\{v_D\}, \{u_1, \dots, u_i\})$  to  $E(\{v_p\}, \{u_1, \dots, u_i\})$ . Proceeding one by one with  $p = D + 1, \dots, l$  gives  $x : E(G) \rightarrow \mathbb{N}$  such that  $x(E(G)) = x'(E(G'))$  and  $x$  is a  $b$ -matching in  $G$ .

For part (b), removing vertices  $v_j$  for  $j > D$  only affects the neighborhoods of vertices  $u_1, \dots, u_i$ . However, for every  $k \in \{1, \dots, i\}$ ,  $\deg_{G'}(u_k) \geq D$  as we keep  $D$  of  $i$ -clones from  $v_1, \dots, v_l$ .  $\blacktriangleleft$

After  $\text{MODIFY}(G, b, i)$  we obtain a graph  $(G', b')$ , where some vertices were removed from  $G$  and for some vertices  $v$ , we have  $b(v) < b'(v)$ . In this cases  $v$  will be called *special*. Note that it is easy to reverse  $\text{MODIFY}$  and obtain  $x$  on  $G$  from  $x'$  on  $G'$  by making special vertices distribute  $x'$  to deleted vertices as described in the proof.

We will now obtain a reduction of  $G$  which will be used in further computations.

**Procedure**  $\text{REDUCTION}(G, b)$ .

- 
1. For  $i = D - 1$  downto 1, let  $(G, b) := \text{MODIFY}(G, b, i)$ .
  2. Return  $(G, b)$ .
- 

**► Lemma 11.** *Let  $(G', b')$  denote the pair returned by  $\text{REDUCTION}(G, b)$ . Then we have the following:*

- (a)  $\nu_{b'}(G') = \nu_b(G)$ ;
- (b) *For every  $i < D$ , if  $S \subseteq V$  has size  $i$ , then there are at most  $D$   $i$ -clones  $v$  in  $G'$  such that  $N_{G'}(v) = S$ .*

**Proof.** The first part follows from Fact 10 (a). For the second part, if  $w_1, \dots, w_l$  are  $i$ -clones in  $G'$  and  $N_{G'}(w_1) = S$ , then, by Fact 10 (b), we have  $N_{G'}(w_k) = S$  for every  $k \in \{1, \dots, l\}$ . Consequently,  $w_1, \dots, w_l$  are  $i$ -clones in the original graph  $G$  and all graphs obtained in step one of  $\text{REDUCTION}(G, b)$ . Therefore, in the  $i$ th iteration, all but at most  $D$  of  $w_1, \dots, w_l$  are removed, and so  $l \leq D$ .  $\blacktriangleleft$

To prove our main fact (Lemma 16) we need some additional preparation. In the proof of the main lemma, we will use the Tutte's construction that reduce the problem of  $b$ -matchings to matchings.

**► Definition 12.** *Let  $\tilde{G} = (\tilde{V}, \tilde{E})$  be obtained from  $G = (V, E)$  as follows. Replace vertex  $v$  from  $V$  with an independent set  $U_v$  of size  $|U_v| = b(v)$  so that for  $i \neq j$ ,  $U_v \cap U_w = \emptyset$ . If  $vw \in E$  then add all edges  $xy$  to  $\tilde{E}$  for every  $x \in U_v$  and  $y \in U_w$ .*

The following fact is easy to see [16]:

► **Fact 13.**  $\nu_b(G) = \nu(\tilde{G})$ .

Let  $\{\tilde{A}, \tilde{B}, \tilde{C}\}$  be a Gallai-Edmonds decomposition of  $\tilde{G}$ . Recall that properties of the decomposition are given before Theorem 1.

► **Lemma 14.** *Let  $G = (V, E)$  and  $\tilde{G} = (\tilde{V}, \tilde{E})$ . Then partition  $\{U_v \mid v \in V\}$  of  $\tilde{V}$  is a refinement of  $\{\tilde{A}, \tilde{B}, \tilde{C}\}$ .*

**Proof.** Let  $v \in V$  and let  $U_v = \{v_1, \dots, v_k\}$ .

- Suppose for some  $i$ ,  $v_i \in \tilde{A}$  and let  $\tilde{M}$  be a maximum matching in  $\tilde{G}$  that does not cover  $v_i$ . If  $v_j$  is covered by  $\tilde{M}$ , then  $v_j w \in \tilde{M}$  for some  $w \in \tilde{V}$ . By the construction,  $v_i w \in \tilde{E}$  and so  $\tilde{M} - v_j w + v_i w$  is a maximum matching that does not cover  $v_j$ . Consequently,  $v_j \in \tilde{A}$ .
- If  $v_i \in \tilde{B}$  then  $v_i \notin \tilde{A}$  and  $v_i w \in \tilde{E}$  for some  $w \in \tilde{A}$ . For any  $j \in \{1, \dots, k\}$ , by the previous observation  $v_j \notin \tilde{A}$  and by the construction  $v_j w \in \tilde{E}$ . Thus  $v_j \in \tilde{B}$ .
- If  $v_i \in \tilde{C}$  then for any  $j$ ,  $v_j \in \tilde{C}$ , because otherwise  $v_j \in \tilde{A} \cup \tilde{B}$  which in view of previous observations gives  $v_i \in \tilde{A} \cup \tilde{B}$ . ◀

Lemma 14 easily gives the following fact.

► **Fact 15.** *Let  $\tilde{H}$  be a component of  $\tilde{G} - \tilde{B}$  that satisfies  $|V(\tilde{H})| \geq 2$  and let  $v \in V(G)$ . If  $U_v \cap V(\tilde{H}) \neq \emptyset$ , then  $U_v \subseteq V(\tilde{H})$ .*

**Proof.** Suppose  $u \in U_v \cap V(\tilde{H})$ . Since  $|V(\tilde{H})| \geq 2$ , there is  $w \in V(\tilde{H})$  such that  $uw \in \tilde{E}$ . By construction, for every  $u' \in U_v$ ,  $u'w \in \tilde{E}$ . By Theorem 1,  $u \notin \tilde{B}$  and so by Lemma 14,  $u' \notin \tilde{B}$ . Consequently, since  $\tilde{H}$  is a component in  $\tilde{G} - \tilde{B}$ ,  $u' \in V(\tilde{H})$ . ◀

We can now prove the main lemma. We will define  $\omega$  on the edges set of  $G'$  where  $(G', b')$  is returned by  $\text{REDUCTION}(G, b)$  for the original graph  $G$  and function  $b$ , that is  $\omega : E(G') \rightarrow \mathbb{Z}^+$  as  $\omega(uv) = \min\{b'(u), b'(v)\}$ .

► **Lemma 16.** *For every  $D, L \in \mathbb{Z}^+$  there is  $c_{D,L} > 0$  such that the following holds. Let  $G$  be a graph with  $\nabla_1(G) \leq D/2$ , let  $b : V \rightarrow \mathbb{Z}^+$  be  $L$ -Lipschitz, and let  $(G', b')$  be the pair returned by  $\text{REDUCTION}(G, b)$ . Then*

$$\nu_{b'}(G') \geq c_{D,L} \cdot \omega(E(G')). \quad (5)$$

**Proof.** Consider  $\tilde{G}$  obtained from  $G'$  and let  $\{\tilde{A}, \tilde{B}, \tilde{C}\}$  be the Gallai-Edmonds decomposition of  $\tilde{G}$ . By Lemma 14, suppressing each set  $U_v$  to  $v$  gives partition  $\{A, B, C\}$  of  $V(G)$  defined as  $A = \bigcup_{U_v \subseteq \tilde{A}} \{v\}$ ,  $B = \bigcup_{U_v \subseteq \tilde{B}} \{v\}$ ,  $C = \bigcup_{U_v \subseteq \tilde{C}} \{v\}$ .

Let  $\tilde{H}$  be a component of  $\tilde{G} - \tilde{B}$  such that  $|V(\tilde{H})| \geq 2$  and let  $H$  be obtained by suppressing each  $U_v \subseteq V(\tilde{H})$  to  $v$ . If  $|V(\tilde{H})|$  is even then  $\tilde{H}$  has a perfect matching  $\tilde{M}$  by Theorem 1, and by Lemma 14,  $H$  contains a  $b'$ -matching  $x_H : E(H) \rightarrow \mathbb{N}$  such that for each vertex  $v \in V(H)$ ,  $\sum_{e \in E_H(v)} x_H(e) = b'(v)$ . Thus

$$x_H(E(H)) = \sum_{v \in V(H)} b'(v). \quad (6)$$

If  $|V(\tilde{H})|$  is odd, then  $|V(\tilde{H})| \geq 3$  and  $\tilde{H}$  is factor-critical. Consequently, there is a  $b'$ -matching  $x_H$  in  $H$  such that  $\sum_{e \in E_H(v)} x_H(e) = b'(v) - 1$  for any specified vertex  $v \in V(H)$  and  $\sum_{e \in E_H(w)} x_H(e) = b'(w)$  for every vertex  $w \in V(H) \setminus \{v\}$ . Therefore,

$$x_H(E(H)) = \left( \sum_{v \in V(H)} b'(v) \right) - 1 > \frac{1}{2} \sum_{v \in V(H)} b'(v). \quad (7)$$

In addition, by Lemma 1, we know that  $\tilde{B}$  is matchable to the set of odd components in  $\tilde{G} - \tilde{B}$ , and so, by Lemma 14, there is a  $b'$ -matching  $x_B$  in  $G'$  such that

$$x_B(E(B, V \setminus B)) = |\tilde{B}| = \sum_{v \in B} b'(v). \quad (8)$$

Let  $G_1 := G'[B \cup \bigcup V(H)]$  where the union is over all components  $H$  of  $G' - B$  that satisfy  $|V(\tilde{H})| \geq 2$ . By (6), (7), there is  $b'$ -matching  $x$  in  $G_1$  such that

$$x(E(G')) \geq \frac{1}{2} \sum_H \sum_{v \in V(H)} b'(v) \quad (9)$$

where the sum is taken over all components  $H$  in  $G' - B$  of order at least two. Since  $\frac{1}{2}(c + d) \leq \max\{c, d\}$ , by (8) and (9), there is a  $b'$ -matching  $x$  in  $G'$  (not necessarily  $G_1$ ), such that

$$x(E(G')) > \frac{1}{2} \left( x_B(E(B, V \setminus B)) + \frac{1}{2} \sum_H \sum_{v \in V(H)} b'(v) \right) \geq b'(V(G_1))/4 > \omega(E(G_1))/(4 \cdot D)$$

where the last inequality follow from Fact 9 (a).

We will now bound the weight of the edges of  $G'$  that are not in  $G_1$ . Let  $\tilde{U}$  be the set of components of size one in  $\tilde{G} - \tilde{B}$ , and let  $U$  be obtained by suppressing each  $U_v \subseteq \tilde{U}$  to  $v$ . Note that each  $u \in U$  is a component of size one in  $G' - B$ .

Let  $U' \subseteq U$  denote the set of vertices  $u \in U$  such that  $\deg_{G'}(u) = |N_{G'}(u) \cap B| \geq D$ . From (1), for every set  $S \subseteq U'$ ,

$$D|S| \leq |E(G'[S, N_{G'}(S)])| \leq D(|N_{G'}(S)| + |S|)/2$$

which gives  $|S| \leq |N_{G'}(S)|$ . Thus, by Hall's theorem, there is a matching of  $U'$  in  $G'[U', B]$ . Let  $\{uv_u \mid u \in U', v_u \in B\}$  be such a matching, and let  $H := G'[U', B]$ . We have  $|U'| \leq |B|$  and there is orientation  $\vec{H}$  of the edges of  $H$  such that  $\Delta^+(\vec{H}) \leq D$ . By definition of  $\omega$ , for  $v \in B$ ,  $\omega(E(v, N^+(v))) \leq D \cdot b'(v)$ . Similarly for  $u \in U'$ ,

$$\omega(E(u, N^+(u))) \leq D \cdot b'(u) = D \cdot b(u) \leq D \cdot (b(v_u) + L) \leq D \cdot (b'(v_u) + L) \leq D(L + 1)b'(v_u)$$

because  $b \leq b'$ ,  $b$  is  $L$ -Lipschitz, and  $\text{REDUCTION}(G, b)$  does not change the values of  $b$  for vertices of degree at least  $D$ . Consequently,

$$\omega(E(H)) \leq D(L + 2) \sum_{v \in B} b'(v) = D(L + 2) \cdot x_B(E(B, V \setminus B)).$$

Let  $U'' := U \setminus U'$  and for  $1 \leq i < D$ , let  $U_i \subseteq U''$  denote the set of vertices  $u \in U''$  such that  $\deg_{G'}(u) = |N_{G'}(u) \cap B| = i$ . Let  $S_v = N_{G'}(v) \cap B$  for  $v \in U_i$ , and note that by Lemma 11 part (b) for any  $v \in U_i$  there are at most  $D$  vertices  $w \in U_i$  such that  $S_w = S_v$ . Out of these vertices  $w$  all but at most one satisfy  $b(w) = b'(w)$  and potentially there is one vertex, called special vertex,  $w$  such that  $b(w) < b'(w)$ .

Before continuing with the main line of the argument we observe that if  $w \in S_v$  for some  $v \in U_i$ , then  $b(w) = b'(w)$ . Indeed, for  $b'(w)$  to be larger than  $b(w)$ ,  $w$  would have to have at least  $D$  clones in  $G$  and  $v$  would be the neighbor of all of them. However,  $\deg_{G'}(v) = i < D$  and by Fact 10 (b),  $N_G(v) = N_{G'}(v)$  and so  $v$  cannot have  $D$  neighbors in  $B$ .

## 59:12 Distributed Approximations of $f$ -Matchings and $b$ -Matchings

We will now go back to the main line of the argument which is similar to the proof of Claim 6. Starting with  $H^* := (B, \emptyset)$ , we add edges to  $H^*$  as in the proof of Claim 6, i.e. if for  $u \in U_i$ , there are  $v, w \in N_{G'}(u) \subseteq B$  such that  $vw$  is not already in  $E(H^*)$ , then add  $vw$  to  $E(H^*)$  and remove  $u$  from  $U_i$ . We continue the process for as long as possible. After it ends,  $H^*$  satisfies  $|E(H^*)| \leq D|B|/2$  because  $\nabla_1(G) \leq D/2$ .

Let  $U'_i$  denote vertices from  $U_i$  that correspond to edges of  $H^*$  and let  $U''_i = U_i \setminus U'_i$ . As before, there is an orientation of the edges of  $H^*$  such that the maximum out-degree is at most  $D$ . If the arc  $(v_1, v_2)$  is obtained by suppressing vertex  $u$  from  $U_i$ , then we say that  $u$  belongs to  $v_1$ . Let  $W_v$  denote the set of vertices that belong to  $v$ . Then  $|W_v| \leq D$  and for every  $u \in W_v$ ,  $\omega(uv) \leq b'(v) = b(v)$  by the previous observation. Since  $b$  is  $L$ -Lipschitz, the total weight of edges incident to vertices from  $W_v$  is at most  $Di(2L+1)$  because if  $u \in W_v$  and  $uv' \in E(G')$  for some  $v' \in B$  then  $\omega(uv') \leq b'(v') = b(v') \leq b(v) + 2L \leq (2L+1)b(v) = (2L+1)b'(v)$ . Consequently, the total weight of edges incident to vertices from  $U'_i$  is at most  $Di(2L+1) \sum_{v \in B} b'(v) = Di(2L+1)x_B(E(B, V \setminus B))$ .

Now consider  $U''_i$ . By construction, for every  $u \in U''_i$ ,  $N_{H^*}(u)$  is a clique on  $i$  vertices. Since  $\nabla_0(H^*) < D/2$ , by Lemma 2, we can assign  $i$ -cliques in  $H^*$  to vertices from  $B$  so that each vertex has at most  $\binom{D}{i-1}$  cliques assigned to it. If  $K_1, \dots, K_l$  are assigned to  $v$ , then the number of vertices  $u$  in  $U''_i$  such that  $N_{H^*}(u) = K_j$  for some  $j \in \{1, \dots, l\}$  is at most  $Dl \leq D \binom{D}{i-1}$ , and the weight on edges incident to them is at most  $Di(2L+1) \binom{D}{i-1} b'(v)$ . Consequently, the weight of edges incident to vertices from  $U''_i$  is at most  $Di(2L+1) \binom{D}{i-1} \sum_{v \in B} b'(v) = Di(2L+1) \binom{D}{i-1} x_B(E(B, V \setminus B))$ . As a result, the weight of edges incident to vertices from  $U_i$  is at most  $Di(2L+1)(1 + \binom{D}{i-1})\nu_{b'}(G')$ . Summing over  $i = 1, \dots, D-1$  shows that the weight of edges of  $G'$  that are not in  $G_1$  is  $O(\nu_{b'}(G'))$ , completing the proof of (5).  $\blacktriangleleft$

As in the case of  $f$ -matchings, next algorithm uses constant  $c_{D,L}$  and so it needs to know  $L$  and  $D = 2 \cdot g(1)$ , but the clustering procedure from [1] also needs some information about  $g$ .

### ■ Procedure $b$ -MATCHING APPROXIMATION( $G, b, g, L, \epsilon$ ).

- 
- Use REDUCTION( $G, b$ ) to obtain graph  $G'$  and function  $b'$ .
  - Define the weights  $\omega$  on  $E(G')$  as in (4). Use the algorithm from [1] to find a partition of  $V(G')$  into sets  $V'_1, \dots, V'_l$  such that  $\text{diam}_{G'}(V'_i) = O_{\epsilon, L, D}(1)$  and the total weight of edges with endpoints in different sets  $V'_i$  is at most  $\epsilon \cdot c_{D,L} \omega(E(G'))$ .
  - Find an optimal solution  $x'_i$  in  $G'[V'_i]$  and let  $x' : E(G') \rightarrow \mathbb{N}$  be given by  $x'(e) := x'_i(e)$  if  $e \in E(G'[V'_i])$  and  $x'(e) := 0$  otherwise. This gives (using the formula in MODIFY) solution  $x$  in  $G$ . Return  $x$ .
- 

► **Theorem 17.** *There is a distributed algorithm which given graph  $G = (V, E)$  on  $n$  vertices,  $\epsilon > 0$  and two functions  $b$  and  $g$  such that  $\nabla_r(G) \leq g(r) = o(\log r)$ , and  $b : V' \rightarrow \mathbb{Z}^+$  is  $L$ -Lipschitz, finds a  $b$ -matching  $x$  in  $(G, b)$  such that  $\sum_{e \in E} x(e) \geq (1 - \epsilon)\nu_b(G)$ . The algorithm runs in  $O_{\epsilon, L, g}(\log^* n)$  rounds.*

**Proof.** Use  $b$ -MATCHING APPROXIMATION( $G, b, g, L, \epsilon$ ) and note that  $x'$  is a  $b'$ -matching in  $G'$ , and so  $x$  is a  $b$ -matching in  $G$ . By Lemma 11,  $\nu_{b'}(G') = \nu_b(G)$  and by Lemma 5,  $\nu_{b'}(G') \geq c_{D,L} \omega(E(G'))$ . Consequently,

$$\begin{aligned} \nu_b(G) &= \nu_{b'}(G') \leq \sum_i x'_i(E(G'[V'_i])) + \epsilon \cdot c_{D,L} \omega(E(G')) \\ &\leq \sum_{e \in E(G)} x(e) + \epsilon \nu_{b'}(G') = \sum_{e \in E(G)} x(e) + \epsilon \nu_b(G). \end{aligned}$$

REDUCTION runs in the number of rounds which depends on  $D$  only and the running time of the algorithm from [1] is  $O_{\epsilon,L,g}(\log^* n)$ . Obtaining solution  $x$  from  $x'$  requires only a constant number of steps. ◀

---

### References

- 1 S. A. Amiri, S. Schmid, and S. Siebertz. Distributed dominating set approximations beyond planar graphs. *ACM Transactions on Algorithms, Vol 15, Issue 3, 39:1–39:18.*, 2019.
- 2 A. Czygrinow and M. Hańkowiak. Distributed algorithm for better approximation of the maximum matching. *Proc. Computing and Combinatorics: 9th Annual International Conference, COCOON 2003, 242–251.*, 2003.
- 3 A. Czygrinow, M. Hańkowiak, and E. Szymańska. A fast distributed algorithm for approximating the maximum matching. *Proceedings of the Annual European Symposium on Algorithms (ESA), 2004, LNCS 3221, 252–263.*, 2004.
- 4 A. Czygrinow, M. Hańkowiak, and W. Wawrzyniak. Fast distributed approximations in planar graphs. *International Symposium on Distributed Computing, DISC, Arcachon, France, September 2008, LNCS 5218, 78–92.*, 2008.
- 5 M. Fischer. Improved deterministic distributed matching via rounding. *31st International Symposium on Distributed Computing (DISC 2017), LIPIcs 91, 17:1–17:15.*, 2017.
- 6 M. Ghaffari, D. Harris, and F. Kuhn. On derandomizing local distributed algorithms. *Proc. 59th IEEE Symposium on Foundations of Computer Science, 662–673.*, 2018.
- 7 M. Hańkowiak. Private communication.
- 8 M. Hańkowiak, M. Karoński, and A. Panconesi. On the distributed complexity of computing maximal matchings. *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 219–225.*, 1998.
- 9 M. Hańkowiak, M. Karoński, and A. Panconesi. A faster distributed algorithm for computing maximal matchings deterministically. *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC), 219–228.*, 1999.
- 10 D. Harris. Distributed local approximation algorithms for maximum matching in graphs and hypergraphs. *arXiv*, 2018. [arXiv:1807.07645](https://arxiv.org/abs/1807.07645).
- 11 A. Kotlov. Short proof of the gallai-edmonds structure theorem. *arXiv*, 2000. [arXiv:math/0011204](https://arxiv.org/abs/math/0011204).
- 12 F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally? *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC 2004). New York, USA: ACM Press, 300–309.*, 2004.
- 13 J. Nešetřil and P. Ossona de Mendez. Sparsity: Graphs, structures, and algorithms. *Algorithms and combinatorics 28, Springer, pp. I-XXIII, 1–457.*, 2012.
- 14 D. Peleg. Distributed computing: A locality-sensitive approach. *Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.*, 2000.
- 15 A. Schrijver. Combinatorial optimization. *Springer*, 2002.
- 16 W.T. Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of Mathematics 6, pp. 347–352.*, 1954.





# Inverse Suffix Array Queries for 2-Dimensional Pattern Matching in Near-Compact Space

Dhrumil Patel ✉

Division of Computer Science, Louisiana State University, Baton Rouge, LA, USA

Rahul Shah ✉

Division of Computer Science, Louisiana State University, Baton Rouge, LA, USA

---

## Abstract

In a 2-dimensional (2D) pattern matching problem, the text is arranged as a matrix  $M[1..n, 1..n]$  and consists of  $N = n \times n$  symbols drawn from alphabet set  $\Sigma$  of size  $\sigma$ . The query consists of a  $m \times m$  square matrix  $P[1..m, 1..m]$  drawn from the same alphabet set  $\Sigma$  and the task is to find all the locations in  $M$  where  $P$  appears as a (contiguous) submatrix. The patterns can be of any size, but as long as they are square in shape data structures like suffix trees and suffix array exist [5, 8] for the task of efficient pattern matching. These are essentially 2D counterparts of classic suffix trees and arrays known for traditional 1-dimensional (1D) pattern matching. They work based on linearization of 2D suffixes which would preserve the prefix match property (i.e., every pattern match is a prefix of some suffix).

The main limitation of the suffix trees and the suffix arrays (in 1D) was their space utilization of  $O(N \log N)$  bits, where  $N$  is the size of the text. This was suboptimal compared to  $N \log \sigma$  bits of space, which is information theoretic optimal for the text. With the advent of the field of succinct/compressed data structures, it was possible to develop compressed variants of suffix trees and array based on Burrows-Wheeler Transform and LF-mapping (or  $\Phi$  function) [7, 4, 15]. These data structures indeed achieve  $O(N \log \sigma)$  bits of space or better. This gives rise to the question: analogous to 1D case, can we design a succinct or compressed index for 2D pattern matching? Can there be a 2D compressed suffix tree? Are there analogues of Burrows-Wheeler Transform or LF-mapping? The problem has been acknowledged for over a decade now and there have been a few attempts at applying  $\Phi$  function [1] and achieving entropy based compression [10]. However, achieving the complexity breakthrough akin to 1D case has yet to be found.

In this paper, we still do not know how to answer suffix array queries in  $O(N \log \sigma)$  bits of space - which would have led to efficient pattern matching. However, for the first time, we show an interesting result that it is indeed possible to compute inverse suffix array (ISA) queries in near compact space in  $O(\text{polylog} n)$  time. Our 2D succinct text index design is based on two 1D compressed suffix trees and it takes  $O(N \log \log N + N \log \sigma)$  bits of space which is much smaller than its naive design that takes  $O(N \log N)$  bits.

Although the main problem is still evasive, this index gives a hope on the existence of a full 2D succinct index with all functionalities similar to that of 1D case.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Pattern matching

**Keywords and phrases** Pattern Matching, Succinct Data Structures

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.60

## 1 Introduction

In the classical pattern matching problem we are given a text  $T[1..n]$  over an alphabet  $\Sigma$ , which is a finite totally ordered set of size  $\sigma$  and a pattern  $P[1..m]$  drawn from the same alphabet set. The task is to find locations of all occurrences of pattern  $P$  in  $T$ . This has been a classic field of research since last 50 years and many algorithms were developed to achieve this task in optimal time complexity of  $O(n + m)$  [9]. In data structural sense, the problem becomes to index the text so that patterns can be taken as queries. Data structures like suffix trees were proposed for this task which took optimal  $O(n)$  (words of) space and optimal



© Dhrumil Patel and Rahul Shah;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 60; pp. 60:1–60:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$O(m)$  time for the query. It was seen that even though taking theoretically optimal space, the space utilization of suffix trees often times could be 50 times the size of original text data. Space saving structures like suffix arrays were introduced which showed substantial space savings at the cost of slightly worse query times. However, when measured in bits, these still take  $O(n \log n)$  bits as against the information theoretic optimal of  $O(n \log \sigma)$  bits. FM-Index [4] and compressed suffix array (CSA) [7] were the first to achieve this goal. Introduction of the compressed suffix tree (CST) ensured full-functionalities of suffix trees simulated in compressed space of  $O(n \log \sigma)$  bits (or even lower in entropy compressed sense) [15]. This led to the field of compressed text indexing which has seen a myriad of results in last two decades with many positive developments [13].

There are other variants of text-indexing problems where suffix trees and suffix arrays exist but their compressed counterparts have yet to be found. One of the problems which has proven to be hard in this context is the problem of 2D pattern matching.

In the 2D pattern matching problem, the text is arranged as a matrix  $M[1..n, 1..n]$  and consists of  $N = n \times n$  symbols drawn from the alphabet  $\Sigma$  of size  $\sigma$ . The query consists of  $m \times m$  square matrix  $P[1..m, 1..m]$  drawn from the same alphabet set  $\Sigma$  and the task is to find all the locations in  $M$  where  $P$  appears as a (contiguous) submatrix. The patterns can be of any size, but as long as they are square in shape the data structures like suffix trees and suffix array exist [5, 8]. The suffix starting from any location  $M[i, j]$  is the largest square matrix which fits within  $M$  and whose top-left corner is  $M[i, j]$ . The suffixes can be linearized [5] and indexed using a trie akin to the 1D suffix tree. The problem of designing an index for 2D pattern matching in compact  $O(N \log \sigma)$  space (based on suffix trees/arrays BWT or otherwise) has been long open. There were some attempts and partial results [1, 10] but they mainly focused on entropy compression, without first addressing the more fundamental problem of achieving the optimal space complexity (compact space). This gives rise to some fundamental questions: analogous to 1D case, can we design a succinct or compressed index for 2D pattern matching? Can there be a compressed suffix tree?

However, achieving the complexity breakthrough similar to 1D case has yet to be found, in this paper, we present a text index that can answer inverse suffix array (ISA) queries in near compact space in  $O(\text{polylog}(n))$  time. We show this by introducing a novel technique named LFISA-mapping that is an analogue of LF-mapping operation typically associated with Burrows–Wheeler Transform. This technique works with *linearization* scheme of Reference [5]. Our 2D succinct text index design is based on two 1D compressed suffix trees, and it takes  $O(N \log \log N + N \log \sigma)$  bits of space as compared to previous non-compact space of  $O(N \log N)$  bits.

## 2 Preliminaries

First, we show an overview of the classical pattern matching problem and its associated terminology. Next, we extend the same for the 2D pattern matching problem, where we provide additional definitions associated with the problem.

### 2.1 Classical Pattern Matching Problem

Let  $S = \{T[i..n] | 1 \leq i \leq n\}$  be the set of all the suffixes of  $T$ . The *suffix tree* (denoted by  $ST$ ) of  $T$  is an edge-labeled compact trie constructed from all the suffixes in  $S$  [12, 16, 3, 17]. In the suffix tree, concatenating all the edge labels on a particular root-to-leaf path, we get one of the suffixes in  $S$ . In other words, each leaf of  $ST$  corresponds to a suffix of  $T$ . Additionally, as each suffix  $T[i..n]$  in  $S$  is uniquely identified with its starting position  $i$  in  $T$ , we can map

text positions to leaves of ST. Upon traversal of the leaves from left-to-right, we get suffixes sorted lexicographically, and storing the corresponding text positions in an array gives an indexing data structure called *suffix array* (SA) [11]. Here by  $i = \text{SA}[r]$ , we mean that the leaf with its corresponding text position  $i$  is the  $r^{\text{th}}$  leftmost leaf ( $\ell_r$ ) in ST. In other words,  $r$  is the lexicographical order or *rank* of the suffix  $T[i..n]$ . Similarly, the *inverse suffix array* (ISA) is defined as  $\text{ISA}[i] = \text{SA}^{-1}[i] = r$ . In other words, the inverse suffix array maps each text position  $i$  to the leaf position  $r$  in ST.

The LF-mapping is the relation between the leaves  $\ell_r$  and  $\ell_{r'}$  ( $\text{LF}(r) = r'$ ) such that their corresponding text positions are  $i$  and  $i - 1$  respectively. Formally, LF-mapping is defined in terms of the suffix array as  $\text{LF}(r) = \text{SA}^{-1}[\text{SA}[r] - 1]$ . But the index such as the FM-index efficiently computes the LF-mapping using the *Burrows Wheeler Transform* (BWT) [2] of the original text along with some auxiliary counting data structures. This computation lies at the heart of BWT based text indexes that enables them to answer pattern matching queries without actually storing the costly suffix array and instead replacing it with a *sampled suffix array*.

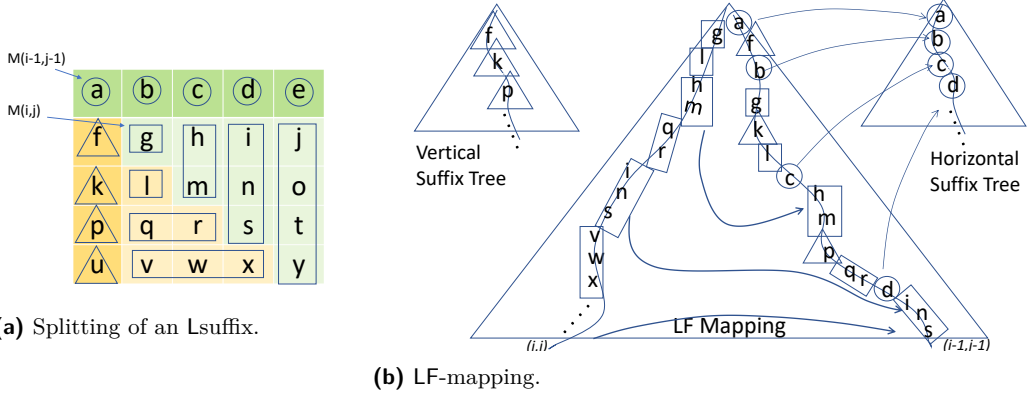
On the contrary, just storing a CSA in itself does not provide all the required functionalities that a full CST provide. Therefore, a CST with full functionalities is needed and is realised using three components: 1) its underlying CSA 2) the *compressed tree topology* that provides navigational operations where each operation takes  $O(1)$  time and 3) some auxiliary data structures providing the *longest common prefix* (LCP) information. Moreover, the full list of operations supported by CST is given in the Appendix. Out of which one of the most important operations is to answer inverse suffix array queries i.e. given CSA, an ISA entry can be decoded in  $O(\log^\epsilon N)$  time for some constant  $\epsilon > 0$ . Now in the ensuing subsection, we formally go over the 2D pattern matching problem and associated terminology.

## 2.2 2D Pattern Matching Problem

Let  $M$  be a square matrix of dimension  $N = n \times n$  where every element  $M[i, j]$  is taken from an alphabet  $\Sigma$  which is a finite totally ordered set of size  $\sigma$ . The query consists of a square pattern  $P[1..m, 1..m]$  also drawn from  $\Sigma$  and the goal is to find all the occurrences of  $P$  in  $M$ .

In a 1D text, an  $i^{\text{th}}$  suffix is the largest substring of the text starting from the  $i^{\text{th}}$  position i.e.  $T[i..n]$ . Similarly, this way of defining a *suffix* can be extended to 2D suffixes of a matrix. A 2D suffix  $S_{i,j}^{2D}$  defined for a position  $(i, j)$  is the largest square submatrix of  $M$  starting at  $(i, j)$  position i.e.  $M[i..i+l, j..j+l]$ , where  $l = n - \max(i, j)$ . Giancarlo [6] proposed a way of *linearization* of 2D suffixes such that they follow the constraints of *completeness* and *common prefix property* similar to 1D suffixes. The *completeness* constraint is that every square submatrix of  $M$  in the linear form must correspond to some *prefix* (whatever the definition of *prefix* is) of some suffix of  $M$  each represented linearly. The *common prefix constraint* is that a square submatrix of  $M$  should be a *prefix* of some suffixes of  $M$  after linearizing them. Giancarlo proposed *Lsuffix* which is a linear representation of a 2D suffix. Here  $L$  stands for *linear*. An *Lsuffix*  $S_{i,j}^L$  of a 2D suffix  $S_{i,j}^{2D}$  is the concatenation of strings  $a_0, a_1, a_2, \dots, a_l$  where  $a_0 = M[i, j]$  and  $a_k = M[i+k, j..j+k-1] \cdot M[i..i+k, j+k]$  which is of length  $2k+1$  and  $l = n - \max(i, j)$  for  $k \neq 0$  (see Figure 1 for example). Here  $\alpha \cdot \beta$  refers to the concatenation of the strings  $\alpha$  and  $\beta$ .

Let  $S^L$  be the set of all such *Lsuffixes* of  $M$ . Here  $|S^L| = N$  as there are total  $N$  suffixes. Let  $\text{ST}^L$  be the compact trie (suffix tree) constructed from *Lsuffixes* in  $S^L$  (also known as *Lsuffix tree*). The uncompressed version of  $\text{ST}^L$  [5] takes  $\Theta(N \log N)$  bits of space which is very large compared to the optimal space required to store the original matrix  $M$  i.e.  $N \lceil \log \sigma \rceil$ . Similarly, the uncompressed version of suffix array ( $\text{SA}^L$ ) [8] for such suffixes also



**Figure 1** Lsuffixes and LF-mapping. a) *Splitting of an Lsuffix*: The characters inside the circle are a part of the horizontal suffix  $S_{i-1, j-1}^H = abcde\dots$  and it resides on the horizontal suffix tree. Similarly, The characters inside the triangle are a part of the vertical suffix  $S_{i, j-1}^V = fkpu\dots$  and this suffix resides on the vertical suffix tree. Additionally, the linear form of the 2D suffix starting from the position  $(i, j)$  is formed by the characters inside the rectangle i.e.  $S_{i, j}^L = g \cdot l \cdot hm \cdot qr \cdot ins \cdot vwx \cdot joty$  and it resides on the Lsuffix tree (the biggest tree on the right). Here  $\alpha \cdot \beta$  denotes concatenation of strings  $\alpha$  and  $\beta$ . Now the Lsuffix starting at the position  $(i-1, j-1)$  is formed by characters of these three sequences i.e.  $S_{i-1, j-2}^L = a \cdot f \cdot bg \cdot kl \cdot chm \cdot pqr \cdot dins \cdot vwx \cdot ejoty$  b) *LF-mapping*: LF-mapping takes from the leaf corresponding to the Lsuffix starting at position  $(i, j)$  to that of Lsuffix starting at position  $(i-1, j-1)$ . A lot of new characters are introduced in doing so. Therefore, the LF-mapping operation in case of 2D pattern matching problem is not trivial to evaluate.

requires  $\Theta(N \log N)$  bits of space. The suffix array and inverse suffix array is defined in a similar fashion as defined in the linear case. For suffix array, given the rank  $r$ , it outputs the position in the matrix of the corresponding Lsuffix  $S_{i, j}^L$  i.e.  $SA^L[r] = (i, j)$ . Furthermore, inverse suffix array is defined as  $ISA^L[i, j] = r$ . Additionally, we introduce the LF-mapping with respect to the 2D case (LF<sup>L</sup>-mapping) and we define it as follows,

$$LF^L(r) = ISA^L[i-1, j-1], \text{ where } SA^L[r] = (i, j)$$

Here,  $ISA^L[0, j'] = ISA^L[i', 0] = \emptyset$ . In other words, LF<sup>L</sup>-mapping operation outputs the rank of the Lsuffix  $S_{i-1, j-1}^L$  given the rank of Lsuffix  $S_{i, j}^L$  (i.e. it goes diagonally above). Figure 1 shows an example of a particular LF-mapping operation and how new characters get introduced when going from Lsuffix  $S_{i, j}^L$  to  $S_{i-1, j-1}^L$  in contrast to the addition of only one character (in front) in the case of 1D suffixes i.e. going from  $T[i..n]$  to  $T[i-1..n]$ . This is the reason why it is not trivial to evaluate LF-mapping for the 2D case.

As the LF<sup>L</sup>-mapping is related to the SA<sup>L</sup>, we introduce a similar mapping for ISA which we call *LF-mapping for ISA* (LFISA<sup>L</sup>). We define it as,

$$LFISA^L(i, j) = ISA^L[i-1, j-1]$$

Here, for computational purposes, we provide  $ISA^L[i, j]$  as an additional parameter. The pseudocode for computing  $LFISA^L(i, j, ISA^L[i, j])$  is given in Section 7. In other words, given the position and the rank of the Lsuffix  $S_{i, j}^L$ , LFISA<sup>L</sup>-mapping outputs the rank of the Lsuffix  $S_{i-1, j-1}^L$  (diagonally above). Now, in order to compute the value of any ISA<sup>L</sup> entry, as storing the entire ISA<sup>L</sup> takes much space, we sample it and store only those  $ISA^L[i, j]$  values such that  $i = 1 + (k-1)\Delta$  where  $k = \{1, 2, \dots, \lceil \frac{\sqrt{N}}{\Delta} \rceil\}$ . This reduces the problem of computing an ISA<sup>L</sup> value to computing at most  $\Delta$  LFISA<sup>L</sup>-mapping operations. Now, in the latter sections,

we show how to compute LFISA<sup>L</sup>-mapping in  $t_{\text{LFISA}} = O((\log N / \log \log N)^3)$  time using our  $O(N \log \sigma + N \log \log N)$ -bit index. Therefore, ISA<sup>L</sup> value for any position in the matrix can be calculated in  $t_{\text{ISA}} = \Delta \cdot t_{\text{LFISA}} = O(\log N \cdot t_{\text{LFISA}})$  time as we take  $\Delta = O(\log N)$  for our case.

Till now, succinct versions of 2D text indexes have been evasive because the intuition behind the pattern matching in 1D case does not extend directly to the 2D case due to the non trivial nature of 2D LF-mapping. However, this does not restrict us from asking the fundamental question as to whether it is possible to design a compact text index for matrices. In this paper, we propose the design of a text index that can atleast answer inverse suffix array queries in near compact space using LFISA<sup>L</sup>-mapping. Although, a solution to the main question evades us, this is a ray of hope. Now, the following theorem states the objective of the paper more formally as,

► **Theorem 1.** *The text index for matrix  $M$  of size  $N = n \times n$  can be encoded in  $O(N \log \sigma + N \log \log N)$ -bit space and any entry in the inverse suffix array ISA<sup>L</sup> can be decoded in time  $O(\log N \cdot t_{\text{LFISA}})$  where  $t_{\text{LFISA}} = O((\log N / \log \log N)^3)$*

**Proof.** See Sections 8.1 and 8.2 for the proof. ◀

**Our approach.** The intuition here is that we split the Lsuffix for which we need the ISA value into three subsequences, and thereby solve the problem for each subsequence to eventually solve for the main Lsuffix. We discuss this splitting in detail in the later section. But in order to understand this dividing strategy, first we define what we call *horizontal* and *vertical suffixes* (or in short Hsuffix and Vsuffix respectively) and also how they relate to Lsuffixes.

### 3 Horizontal and Vertical Suffixes

Firstly, given a matrix  $M$ , we linearize it horizontally by concatenating all the rows of  $M$  one after another to get a single 1D text  $T^H$  of length  $N$ . The set of all the suffixes of  $T^H$  is defined as  $S^H = \{T^H[i..N] | 1 \leq i \leq N\}$ . We denote such suffixes as horizontal or Hsuffixes. Let  $ST^H$  be the compressed suffix tree obtained from all the Hsuffixes of text  $T^H$ . Secondly, by concatenating all the columns into a single text  $T^V$  we linearize  $M$  vertically. The set of all the suffixes of  $T^V$  is defined as  $S^V = \{T^V[i..N] | 1 \leq i \leq N\}$ . Such suffixes are denoted as vertical or Vsuffixes. Here, let  $ST^V$  be the compressed suffix tree constructed from such Vsuffixes of  $T^V$ . From the context of  $M$ , Hsuffix and Vsuffix starting from  $M[i, j]$  are written as

$$S_{i,j}^H = M[i, j..n] \cdot M[i + 1, 1..n] \cdot M[i + 2, 1..n] \cdot \dots \cdot M[n, 1..n]$$

$$S_{i,j}^V = M[i..n, j] \cdot M[1..n, j + 1] \cdot M[1..n, j + 2] \cdot \dots \cdot M[1..n, n]$$

Finally, as  $ST^H$  and  $ST^V$  are the compact versions of the original suffix trees, they only occupy  $O(N \log \sigma)$  bits of space which is very close to the space required by the original matrix [15]. Their full functionalities are provided in the Appendix. Next, we relate all these defined suffixes.

### 4 Splitting of an Lsuffix

In this section, we show how to split an Lsuffix into three different subsequences. Given an Lsuffix  $S_{i,j}^L$  in the 2D form, we can split it into three subsequences: 1) The horizontal subsequence (i.e. the first row  $M[i, j..n]$ ), 2) The vertical subsequence (i.e. the first column  $M[i + 1..n, j]$ ) and 3) The subsequence (linear form) of the remaining square submatrix

i.e.  $S_{i+1,j+1}^L$ . An example of such a splitting is provided in Figure 1. Here,  $M[i, j..n]$  and  $M[i+1..n, j]$  subsequences come from the Hsuffix  $S_{i,j}^H$  and Vsuffix  $S_{i+1,j}^V$  respectively. Let us denote  $h_k$  and  $v_k$  as the  $(k+1)^{th}$  characters of  $M[i, j..n]$  and  $M[i+1..n, j]$  respectively. Now, as mentioned before an Lsuffix  $S_{i,j}^L$  is the concatenation of strings  $a_0, a_1, a_2, \dots, a_l$  where  $a_0 = M[i, j]$  and  $a_k = M[i+k, j..j+k-1] \cdot M[i..i+k, j+k]$  which is of length  $2k+1$  and  $l = n - \max(i, j)$  for  $k \neq 0$ . Similarly, let  $S_{i+1,j+1}^L$  be the concatenation of strings  $b_0, b_1, b_2, \dots, b_{l-1}$  where  $b_0 = M[i+1, j+1]$  and  $b_k = M[(i+1)+k, (j+1)..(j+1)+k-1]M[(i+1)..(i+1)+k, (j+1)+k]$  when  $k \neq 0$ . For simplicity we break each  $a_k$  and  $b_k$  into two parts as follows,

$$\begin{aligned} a'_k &= M[i+k, j..j+k-1] \\ a''_k &= M[i..i+k, j+k] \\ b'_k &= M[(i+1)+k, (j+1)..(j+1)+k-1] \\ b''_k &= M[(i+1)..(i+1)+k, (j+1)+k] \end{aligned}$$

We can write  $a_k$  in terms of  $h_k$  and  $v_k$   $b_k$  as follows,

$$\begin{aligned} a'_k &= M[i+k, j..j+k-1] = v_{k-1}b'_{k-1} \\ a''_k &= M[i..i+k, j+k] = h_k b''_{k-1} \end{aligned}$$

Therefore, we can say that  $a_k$  is the concatenation of strings  $v_{k-1}, b'_{k-1}, h_k, b''_{k-1}$  where  $b'_0 = \emptyset$  and  $b''_0 = b_0$  and  $a_0 = h_0$  as  $h_0 = M[i, j]$ . We want to redirect the reader's attention to Figure 1 where we showcase an example that helps in better understanding of the above concept.

Now, given  $a_k$  we can get  $v_{k-1}, b'_{k-1}, h_k, b''_{k-1}$  as  $v_{k-1}$  and  $h_k$  are characters and  $b'_{k-1}$  and  $b''_{k-1}$  are the strings of length  $k-2$  and  $k-1$  respectively. Here  $v_{k-1}$  and  $h_k$  can be thought of as delimiters of the string  $a_k$  and these two uniquely breaks down  $a_k$  into its constituents. Now since we know that given  $a_k$  we can get  $v_{k-1}, b'_{k-1}, h_k, b''_{k-1}$  and vice versa, we denote the *horizontal component* of the entire Lsuffix  $S_{i,j}^L$  by  $hc(S_{i,j}^L) = h_0 h_1 h_2 \dots h_l$ . Similarly, we denote the *vertical component* by  $vc(S_{i,j}^L) = v_0 v_1 v_2 \dots v_{l-1}$  and the *square component* by  $sc(S_{i,j}^L) = b_0 b_1 b_2 \dots b_{l-1}$ . Likewise, we can define the same for any prefix  $pf$  of the Lsuffix  $S_{i,j}^L$ . We can state the following fact about the relation between the length of the three components of the prefix  $pf$  of  $S_{i,j}^L$  and its length. Here, by length, we mean the length of the string.

► **Fact 2.**  $\text{length}(pf) = \text{length}(hc(pf)) + \text{length}(vc(pf)) + \text{length}(sc(pf))$

Now, intuitively we use such a splitting to evaluate a single LFISA-mapping operation. Next, we go over some of the basic terminologies of a suffix tree that will be needed in understanding the construction stage of our text index.

## 5 Terminology of a Suffix Tree (ST)

A suffix tree is an edge-labelled compact trie. We call any character on the edge of the suffix tree be represented as a *point*. Given any point  $c$  on the ST,  $\text{string}(c)$  represents the concatenation of all the characters from root to that point (including  $c$ ) along the root to  $c$  path of ST. The string depth of a point  $c$  on a path of ST is given by the length of  $\text{string}(c)$  i.e.  $\text{depth}(c) = \text{length}(\text{string}(c))$ . A node of the ST is also a point as that node is represented by the character just above it. The locus  $u$  of a point  $c$  is the highest node of ST such that  $\text{string}(c)$  is the prefix of  $\text{string}(u)$  (lets denote it as  $u = \text{locus}(c)$ ). Now, we define whether



a point  $c$  is marked or not as  $\text{marked}(c) = 1$  or  $0$  respectively. The leftmost and rightmost leaves in the subtree of a particular point  $c$  are given as  $\text{lleaf}(c)$  and  $\text{rleaf}(c)$  respectively. Here  $\text{lleaf}(c)$  and  $\text{rleaf}(c)$  give the leaf rank from left in  $\text{ST}$ . In general, we denote  $r^{\text{th}}$  leftmost leaf of  $\text{ST}$  as  $\ell_r$ . Let  $\text{lca}(c_1, c_2)$  be the *lowest common ancestor* of points  $c_1$  and  $c_2$ . The lowest common ancestor as the name suggests is the common ancestor node of two points and is the farthest from the root.

## 6 Computing LFISA-mapping in time $O((\log N / \log \log N)^3)$ using Compact Space

Just to recall, we have three suffix trees based on three different types of suffix definitions as shown before, i.e.  $\text{ST}^{\text{L}}$ ,  $\text{ST}^{\text{H}}$  and  $\text{ST}^{\text{V}}$ . Here, we store  $\text{ST}^{\text{H}}$  and  $\text{ST}^{\text{V}}$  as compressed suffix trees (CST) [15] with full functionalities (See Theorem 10 and 11 in Appendix) and they together occupy only  $O(N \log \sigma) + O(N \log \sigma) = O(N \log \sigma)$  bits of space. On the contrary, we won't be storing the entire  $\text{ST}^{\text{L}}$  but only the compressed topology of the tree that has navigational functionalities each supported in constant time and occupies  $4N + o(N)$  bits of space (See Theorem 10 in Appendix). In the ensuing subsection, firstly we show a scheme of marking some relevant points on these trees (construction stage) and then explain how this will help in computing LFISA-mapping.

### 6.1 Marking Scheme and Mapping

Firstly, we mark some nodes on Lsuffix tree  $\text{ST}^{\text{L}}$ . We mark a node  $v_i^{\text{L}}$  of  $\text{ST}^{\text{L}}$  such that  $v_i^{\text{L}} = \text{lca}(\ell_{(i-1)g+1}, \ell_{ig})$ , where  $i = \{1, 2, \dots, \lceil \frac{N}{g} \rceil\}$  and  $g$  is the *grouping factor*. Furthermore, we define  $G_i = [(i-1)g+1, ig]$  as the *grouping interval*. For our case, we shall use  $g = \lceil \log^3 N \rceil$ . Hence, the total number of marked nodes on  $\text{ST}^{\text{L}}$  is bounded by  $O(\frac{N}{\log^3 N})$ . Now, we define *marked ancestor*, *lowest marked ancestor*, *cover* of a leaf and *coveredby*( $v^{\text{L}}$ ) set of a marked node as follows:

► **Definition 3** (Marked Ancestor). *A marked node  $v^{\text{L}}$  is the marked ancestor of a leaf  $\ell$  if  $v^{\text{L}}$  lies on the path from root to leaf  $\ell$  in the suffix tree.*

► **Definition 4** (Lowest Marked Ancestor). *A node  $v^{\text{L}}$  is the lowest marked ancestor of the leaf  $\ell$  if it is the lowest (one with the maximum string depth) among all the marked ancestors of  $\ell$ .*

► **Definition 5** (Cover). *A node  $v^{\text{L}}$  is the cover of the leaf  $\ell$  if it is the lowest marked ancestor of  $\ell$ .*

► **Definition 6** (coveredby( $v^{\text{L}}$ ) set). *A coveredby( $v^{\text{L}}$ ) set is the set of the leaves for which  $v^{\text{L}}$  is the cover.*

As mentioned before in Section 4 showcasing the splitting of an Lsuffix, given a marked node  $v^{\text{L}}$ , its associated string i.e.  $\text{string}(v^{\text{L}})$  can be split into its horizontal, vertical and square components i.e.  $\text{hc}(\text{string}(v^{\text{L}}))$ ,  $\text{vc}(\text{string}(v^{\text{L}}))$  and  $\text{sc}(\text{string}(v^{\text{L}}))$  respectively. For a marked node  $v^{\text{L}}$  in  $\text{ST}^{\text{L}}$ , we mark a point  $p^{\text{H}}$  in  $\text{ST}^{\text{H}}$  corresponding to its horizontal component such that  $\text{string}(p^{\text{H}}) = \text{hc}(\text{string}(v^{\text{L}}))$ . Similarly, we mark points  $p^{\text{V}}$  and  $p^{\text{L}}$  corresponding to its vertical and square components in  $\text{ST}^{\text{V}}$  and  $\text{ST}^{\text{L}}$  respectively. We call them the *shadow* points. Just to recall, a point is any character on the edge of the suffix tree. Note that  $v^{\text{L}}$  is not the same marked node as  $p^{\text{L}}$  even though they are marked on the same tree (see Figure 2). We repeat the above process for every marked node on  $v^{\text{L}}$  in  $\text{ST}^{\text{L}}$ .



At the end of the marking process, let  $MP^H$ ,  $MP^V$  and  $MP^L$  be the sets of all the shadow points on  $ST^H$ ,  $ST^V$  and  $ST^L$  respectively. Hence, a marked node  $v^L$  in  $ST^L$  can be viewed as a unique triplet of shadow points in  $ST^H$ ,  $ST^V$  and  $ST^L$  i.e.  $v^L = (p^H, p^V, p^L)$ . Therefore, the total number of shadow points in each tree is bounded. Formally, we have  $|MP^H|$ ,  $|MP^V|$  and  $|MP^L|$  as bounded by  $O(\frac{N}{\log^3 N})$ , where  $|X|$  is the cardinality of the set  $X$ . Due to this one-to-one correspondence between a marked node and the triplet of shadow points, we define a set  $U \subseteq MP^H \times MP^V \times MP^L$  which consists of only those triplets of shadow points which come from the marked nodes.

Now, we state our central task as follows,

Given  $ISA^L[i, j]$ , compute  $ISA^L[i - 1, j - 1]$ .

We shall preprocess the text and construct data structures that will take near compact space and achieve this task in  $O(\text{polylog}N)$  time. The main step in this is computing  $LFISA^L(i, j, ISA^L[i, j])$ . In the following subsection, we show the details on how to achieve this, and thereafter we outline the pseudocode for the same as  $LFISA^L(i, j, ISA^L(i, j))$  in the subsection 6.2.3.

## 6.2 Computing $LFISA^L(\cdot)$

In the section, we show the details for the evaluation of  $LFISA^L(i, j, ISA^L[i, j])$  given the matrix position  $(i, j)$  and  $ISA^L[i, j]$ . Firstly, using the  $\text{inverse}(\cdot)$  function of  $ST^H$  and  $ST^V$  (See Theorem 10 in Appendix), we evaluate the inverse suffix array values  $ISA^H[i - 1, j - 1]$  and  $ISA^V[i, j - 1]$  respectively. For simplicity, let  $ISA^L[i, j] = r$ ,  $ISA^H[i - 1, j - 1] = h$ ,  $ISA^V[i, j - 1] = v$ ,  $ISA^L[i - 1, j - 1] = LFISA^L(i, j, ISA^L[i, j]) = s$ .

As inverse suffix array values are related to the leaves of the suffix tree, let  $\ell_h$ ,  $\ell_v$  and  $\ell_r$  be  $h^{th}$ ,  $v^{th}$  and  $r^{th}$  leftmost leaves in their respective suffix trees. The aim here is to find the leaf  $\ell_s$  in  $ST^L$  using the information provided by the shadow points of our index along the root-to-leaf paths of  $\ell_h$ ,  $\ell_v$  and  $\ell_r$  in  $t_{LFISA}$  time. We shall use some auxiliary data structures that we introduce in the latter subsections.

Given  $(h, v, r)$ , we define a set as  $A = \{(p^H, p^V, p^L) \in U \mid \ell_h, \ell_v \text{ and } \ell_r \text{ lie in the respective subtrees of } p^H, p^V \text{ and } p^L\}$ . To put it another way,  $A$  is a set of valid triplets of shadow points that lie on the root-to-leaf paths of  $\ell_h$ ,  $\ell_v$  and  $\ell_r$  in their respective trees. Out of all the valid triplets that are in  $A$ , let a specific triplet or its corresponding marked node  $v_{max}^L$  be defined as follows,

$$v_{max}^L = \underset{v^L = (p^H, p^V, p^L) \in A}{\text{argmax}} (\text{depth}(\text{string}(v^L))).$$

Recall that there is a one-to-one correspondence between the marked nodes in  $ST^L$  and triplets in  $U$ .

Lemma 7 proves that the marked node  $v_{max}^L$  is the *lowest marked ancestor (or cover)* of the leaf  $\ell_s$ . Therefore, the marked node  $v_{max}^L$  along with some augmenting information shown in later subsection, will lead us to the leaf  $\ell_s$  which is what we are interested in as. Hence, we call the above query as *lowest marked ancestor query*.

► **Lemma 7.** *The marked node  $v_{max}^L$  in  $ST^L$  is the lowest marked ancestor (or cover) of the leaf  $\ell_s$ .*

**Proof.** Firstly, we prove that any valid triplet  $v^L = (p^H, p^V, p^L) \in A$  is the marked ancestor of the leaf  $\ell_s$ . As  $p^H$  is the shadow point on the root-to-leaf path of  $\ell_h$ ,  $\text{string}(p^H)$  is the prefix of the horizontal suffix  $S_{i-1, j-1}^H$  as we have  $\text{ISA}^H[i-1, j-1] = h$ . Similarly,  $\text{string}(p^V)$  and  $\text{string}(p^L)$  are the prefixes of the vertical suffix  $S_{i, j-1}^V$  and  $\text{Lsuffix } S_{i, j}^L$  respectively.

Furthermore, as the  $\text{string}(p^H)$ ,  $\text{string}(p^V)$  and  $\text{string}(p^L)$  are the horizontal, vertical and square components of the  $\text{string}(v^L)$  respectively (as per the marking scheme), one of the occurrences of  $\text{string}(v^L)$  in its 2D form is at matrix position  $(i-1, j-1)$ . Therefore, it is a prefix of the suffix starting at the position  $(i-1, j-1)$  which in its linear form is represented as  $S_{i-1, j-1}^L$ . Therefore, the triplet node  $v^L$  lies on the root-to-leaf path of the leaf representing the  $\text{Lsuffix } S_{i-1, j-1}^L$  and that leaf is  $\ell_s$ . Hence,  $v^L$  is a *marked ancestor* of  $\ell_s$ . The same is true for  $\forall v^L \in A$ .

Moreover, as  $v_{max}^L \in A$  and is the output of the lowest marked ancestor query that maximizes over string depth over all triplets  $v^L \in A$ , it is the *lowest marked ancestor or cover* of  $\ell_s$ . ◀

Now as we are interested in obtaining cover  $v_{max}^L$ , we reduce the above lowest marked ancestor query to a *stabbing-max* query. This reduction is interesting and useful in our context due to the result mentioned in Theorem 8. The details concerning this reduction is discussed in the next subsection. Furthermore, after finding the cover  $v_{max}^L$ , in order to uniquely go to the correct leaf  $\ell_s$  we store additional augmenting information [discussed in latter subsection]. This shows the computation of an  $\text{LFISA}^L$  operation. The time or query complexity of such an operation is discussed in the Section 7.

### 6.2.1 Reduction to 3-dimensional (3D) Stabbing-Max Query

In this section, we show how to reduce that the aforementioned lowest marked ancestor query to a *3D stabbing-max* query. In [14], the authors proves the following theorem,

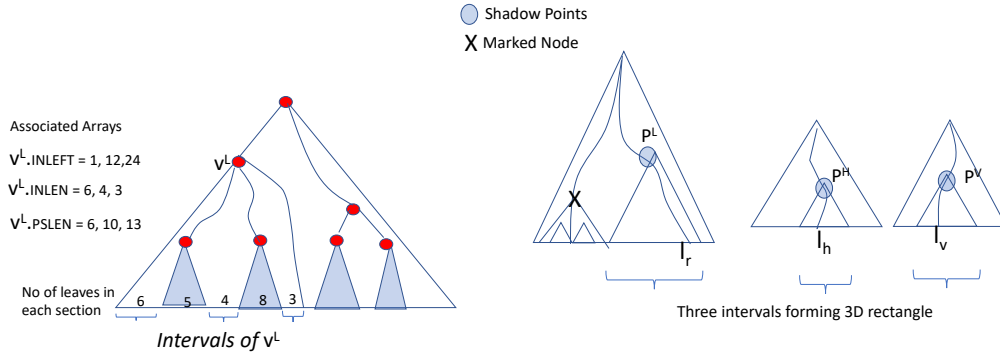
► **Theorem 8** ([14]). *Given a set  $I$  of  $n$  3D rectangles in  $\mathbb{R}^3$ , where each rectangle  $rec$  has a weight  $w(rec)$  associated to it, finding a rectangle with maximum weight containing (or stabbed by) the 3D query point  $q$  can be done in  $O((\frac{\log n}{\log \log n})^3)$  time using a data structure occupying  $O(n(\frac{\log n}{\log \log n})^2)$  space.*

We define the sides of the 3D rectangle  $rec$  for each marked node  $v^L = (p^H, p^V, p^L)$  in  $U$  as follows:

$$\begin{aligned} (x_{left}, x_{right}) &= (\text{lleaf}(p^H), \text{rleaf}(p^H)) \\ (y_{up}, y_{down}) &= (\text{lleaf}(p^V), \text{rleaf}(p^V)) \\ (z_{front}, z_{back}) &= (\text{lleaf}(p^L), \text{rleaf}(p^L)) \\ w(rec) &= \text{depth}(\text{string}(v^L)) \end{aligned}$$

This shows that each triplet in  $U$  or its corresponding marked node  $v^L$  in  $\text{ST}^L$  is uniquely represented as a weighted rectangle.

Next, we define the 3D query point as  $q = (h, v, r)$ . Therefore, the output of this 3D stabbing-max query is the rectangle with maximum weight i.e. the rectangle corresponding to the cover of the leaf  $\ell_s$  ( $v_{max}^L$ ). Furthermore, after obtaining the cover of the leaf, we shall provide details on what augmenting information to store in order to get the desired leaf uniquely, i.e.  $\ell_s$  in the next subsection.



■ **Figure 2** For a particular marked node  $v^L$  in  $ST^L$  (shown in red color), the array `INLEFT` corresponding to  $v^L$  stores the start of its associated intervals. Likewise, the array `INLEN` stores the length of such intervals and the array `PSLEN` is the prefix-sum array of `INLEN`. The points  $(p^L, p^H, p^V)$  are the shadow points of  $v^L$  (shown as X in the figure shown on the right side).

### 6.2.2 Augmenting Information for getting $\ell_s$ from its Cover

In this section, we explain the procedure of obtaining the correct leaf  $\ell_s$  from its cover  $v^L$  by storing the leaf's rank  $q$  (say). Here, we define the task for this section as: Given  $q$  and  $v^L$ , find the  $q^{th}$  leftmost leaf in `coveredby`( $v^L$ ) (See Definition 6 of `coveredby`( $v^L$ )).

Now the challenge lies due to the fact that  $v^L$  may have multiple marked nodes in its subtree and due to that there may be leaves in its subtree whose lowest marked ancestor or cover is not  $v^L$ . Therefore, the set of leaves for which  $v^L$  is the cover i.e. `coveredby`( $v^L$ ) can be represented as a set of contiguous intervals. Let us denote it as  $CI = \{I_1, I_2, \dots, I_k\}$ . Here,  $I_i = [a_i, b_i]$  where  $i \in \{1, 2, \dots, k\}$  and all the leaves between  $\ell_{a_i}$  and  $\ell_{b_i}$  belongs to `coveredby`( $v^L$ ). Here  $CI$  denotes covered intervals.

Lemma 9 proves that the total number of such intervals is bounded by  $O(\sigma)$ , i.e.  $k = O(\sigma)$ . Additionally, it establishes that the total number of leaves for which  $v^L$  is the cover is bounded by  $\sum_{a=1}^k |I_a| = O(k \log^3 N) = O(\sigma \log^3 N)$  where  $|I|$  is the length of the interval  $I$ . Furthermore, as there is a one-to-one correspondence between each marked node and a rectangle as shown before, we store the augmenting information for each rectangle rather than storing it explicitly for the marked node. Let the rectangle associated with  $v^L$  be denoted as  $rec$ . Therefore, let  $CI_{rec} = CI$ .

► **Lemma 9.** *The total number of intervals in  $CI_{rec}$  is bounded by  $O(\sigma)$  and the total number of leaves for which any marked node (here  $v^L$ ) is the cover is bounded by  $O(\sigma \log^3 N)$ .*

**Proof.** Let  $c^L$  be one of the child nodes of  $v^L$ . Assume that  $lleaf(c^L)$  and  $rleaf(c^L)$  lie inside the intervals  $I_i$  and  $I_j$  respectively. First, we prove that  $I_i$  and  $I_j$  are consecutive intervals.

Suppose there is an interval  $I_k$  between  $I_i$  and  $I_j$ . This means that  $I_k$  is entirely contained inside the subtree of  $c^L$ . In other words, there is an interval of leaves  $I_k$  completely inside the subtree of  $c^L$  for which  $v^L$  is the cover. This implies that there is at least one grouping interval of leaves completely contained inside the subtree of  $c^L$  for which  $v^L$  is the lowest common ancestor (lca) of its leftmost and rightmost leaves (See marking scheme for details). But this is not possible as for  $v^L$  to be the lca, the leftmost and rightmost leaves of that grouping interval should exist on two separate downward branches of  $v^L$ . This is the contradiction. Therefore, this means that there is no grouping interval completely contained inside the subtree of the child node  $c^L$ . Hence, there is no  $I_k$  that is entirely contained inside the subtree of  $c^L$ .

This means  $I_i$  and  $I_j$  are consecutive intervals. Therefore, the subtree of a child node of  $v^L$  overlaps with at most 2 consecutive intervals in  $CI_{rec}$ . Furthermore, there are at most  $\sigma$  child nodes of  $v^L$ . Hence, the total number of intervals in  $CI_{rec}$  is bounded by  $O(\sigma)$ .

Secondly, there are at most  $O(\sigma)$  grouping intervals under the subtree of  $v^L$  for which  $v^L$  is the lca of its leftmost and rightmost leaves, as each grouping interval need to span over two separate downward branches of  $v^L$  for  $v^L$  to be that lca. Additionally, the total number of leaves in all such grouping intervals combined is bounded by  $O(\sigma \cdot g) = O(\sigma \cdot \log^3 N)$  where  $g$  is the grouping factor. This implies that the total number of leaves for which  $v^L$  is the lowest marked ancestor or the cover is bounded by the same factor i.e.  $O(\sigma \log^3 N)$ . ◀

As the set of leaves for which  $v^L$  is the cover, is divided into contiguous intervals of leaves (as shown above), to go from the cover  $v^L$  to the output leaf  $\ell_s$ , first we store some information to retrieve which interval that leaf belongs to and then where exactly that leaf is inside that interval.

For each marked node (here  $v^L$  or its associated  $rec$ ) firstly we store the start of each interval in an array  $INLEFT_{rec}[\cdot]$ . Additionally, we store the size of such intervals in another array  $INLEN_{rec}[\cdot]$ . Moreover, we store the prefix-sum array of  $INLEN_{rec}[\cdot]$  in an array  $PSLEN_{rec}[\cdot]$  (See Figure 2 for example). Now as we are not storing the entire  $ISA^L[\cdot, \cdot]$  because it requires  $O(\log N)$  bits for each leaf instead we store what we call a  $miniISA^L[\cdot, \cdot]$ , where we store just a  $O(\log \sigma + \log \log^3 N)$ -bit number for each matrix position  $(i, j)$ . This is because each entry in the  $miniISA^L[i, j]$  is the lexicographical rank of the leaf associated with  $ISA^L[i, j]$  under its lowest marked ancestor and the total number of leaves for which a marked node is the lowest marked ancestor is bounded by  $O(\sigma \log^3 N)$  (Lemma 9). Now let  $miniISA^L[i, j] = q$ . First we do binary search of  $q$  in  $PSLEN_{rec}[\cdot]$  and get the index  $e$  such that the value of  $PSLEN_{rec}[e]$  is the largest number smaller than  $q$ . Now return the final output  $s = INLEFT_{rec}[e] + (q - PSLEN_{rec}[e])$ .

### 6.2.3 Pseudocode of LFISA<sup>L</sup>-mapping Operation

Now, we outline the pseudocode for LFISA<sup>L</sup>-mapping operation.

■ **Algorithm 1** LFISA<sup>L</sup>( $i, j, ISA^L(i, j)$ ).

---

```

1:  $h = ST^H.inverse(i, j)$ 
2:  $v = ST^V.inverse(i, j)$ 
3:  $s = ISA^L[i, j]$ 
4:  $rec = 3d\_stabbing\_max(h, v, s)$ 
5:  $q = miniISA^L[i, j]$ 
6:  $e = binary\_search(PSLEN_{rec}, q)$ 
7:  $s = INLEFT_{rec}[e] + (q - PSLEN_{rec}[e])$ 
8: return  $s$ 

```

---

## 7 Space and Time Complexity Analysis

### 7.1 Space Complexity

After the end of the construction phase, we have three suffix trees in our index based on three different types of suffix definitions, along with some auxiliary structures that are actually stored. The horizontal and vertical suffix trees i.e.  $ST^H$  and  $ST^V$  are stored as compact suffix trees (See Theorem 10 and 11 in Appendix) which together occupy

$O(N \log \sigma) + O(N \log \sigma) = O(N \log \sigma)$  bits of space. On the contrary, we only store the compressed topology for the Lsuffix tree  $ST^L$  rather than storing the entire suffix tree (See Theorem 10 in Appendix). This compressed topology provides navigational functionalities, and overall it occupies  $4N + o(N)$  bits of space.

As previously mentioned in the marking scheme section, the number of marked nodes on  $ST^L$  is bounded by  $O(N/\log^3 N)$ . Thus, the number of their corresponding shadow points on  $ST^L$ ,  $ST^H$  and  $ST^V$  are also bounded by  $O(N/\log^3 N)$ . Additionally, due to one-to-one correspondence between marked nodes and 3D rectangles, the number of such rectangles is also bounded by the same factor.

Each rectangle has a set of arrays associated with it. The length of each of these arrays ( $INLEFT_{rec}[\cdot]$ ,  $INLEN_{rec}[\cdot]$ ,  $PSLEN_{rec}[\cdot]$ ) is the number of intervals under the marked node of that rectangle. As per the marking scheme, the number of grouping intervals is bounded by  $O(N/\log^3 N)$ . Therefore, the total number of intervals across all the rectangles is also bounded by  $O(N/\log^3 N)$  [Implication from Lemma 9]. Each number in these auxiliary data structures take  $O(\log N)$  bits to store. Identifiers for each marked node or shadow points also take at most  $O(\log N)$  bits. Thus, the storage space for all the auxiliary structures is bounded by  $O(N/\log^2 N) = o(N)$  bits.

If there are  $t$  rectangles, the data structure for stabbing-max query takes  $O(t(\log t/\log \log t)^2)$  [14] which is  $O(t \log^2 t)$  space. By taking  $t = O(N/\log^3 N)$ , we get that stabbing-max data structure takes  $O(N/\log N)$  words of space which is bounded by  $O(N)$  bits of space.

Finally, for our  $miniSA^L$  structure, we simply store a matrix of dimensions  $n \times n$ , with each entry  $miniSA^L[i, j]$  taking  $O(\log \sigma + \log \log N)$  bits. This is because any entry in  $miniSA^L$  writes a position of the desired leaf among at most  $\sigma \log^3 N$  leaves which have the same lowest marked node. Thus, in total we get  $O(N \log \sigma + N \log \log N)$  bits for this part. Additionally, we store the sampled inverse suffix array which has  $O(N/\log N)$  elements where each element takes  $O(\log N)$  bits. Therefore, in total it takes  $O(N)$ -bits of space.

After summing up all five parts that are considered, we get  $O(N \log \sigma) + o(N) + O(N) + O(N \log \sigma + N \log \log N) + O(N)$  bits. This simplifies to  $O(N \log \sigma + N \log \log N)$  bits as claimed in Theorem 1.

## 7.2 Time Complexity

For the time complexity of query evaluation, as a key component, we first focus on computing  $LFISA^L$ -mapping operation. We follow the pseudocode step by step for this. The first two steps take  $t_{inverse}$  as given by CST which is  $O(\log^\epsilon n)$  (See Theorem 11 in Appendix). The third step is constant time since the value is provided as a part of the function. The main time consuming part is the stabbing-max data structure which takes  $O((\log N/\log \log N)^3)$  time. Finding corresponding marked node can be done in  $O(1)$  time using succinct tree data structure and searching for prefix sum in the array associated with the rectangle can be done via binary search in  $O(\log N)$  time. Thus, our dominating and main query bound for  $LFISA^L$ -mapping operation is  $O((\log N/\log \log N)^3)$ . Finally, considering that our query algorithm for  $ISA^L$  can have at most  $\log N$  applications of  $LFISA^L$ , we get our query-time bound as  $O(\log^4 N/(\log \log N)^3)$  (as claimed in Theorem 1).

## 8 Conclusion

To conclude, we provide an  $O(N \log \sigma + N \log \log N)$ -bit index that supports inverse suffix array queries in  $O(\log^4 N / (\log \log N)^3)$  time. Even though the main goal of developing 2D text index which can allow pattern matching i.e. to compute suffix array (SA) value or LF values efficiently is not achieved, we think this is a significant step forward in understanding the structure of the problem. Exploring the inter-relations here may lead us to better tools to compute LF operation efficiently in compact space.

---

### References

- 1 Jeffrey Scott Vitter Ankur Gupta, Roberto Grossi. Entropy-compressed indexes for multi-dimensional pattern matching. In *DIMACS working group on Burrows-Wheeler Transform*, 2004.
- 2 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Equipment Corporation (now part of Hewlett-Packard, Palo Alto, CA), 1994.
- 3 Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.
- 4 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. An extended abstract appeared in *FOCS 2000* under the title “Opportunistic Data Structures with Applications”. doi:10.1145/1082036.1082039.
- 5 Raffaele Giancarlo. A generalization of the suffix tree to square matrices, with applications. *SIAM J. Comput.*, 24(3):520–562, 1995. doi:10.1137/S0097539792231982.
- 6 Raffaele Giancarlo and Roberto Grossi. Suffix tree data structures for matrices. In Alberto Apostolico and Zvi Galil, editors, *Pattern Matching Algorithms*, pages 293–340. Oxford University Press, 1997.
- 7 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. An extended abstract appeared in *STOC 2000*. doi:10.1137/S0097539702402354.
- 8 Dong Kyue Kim, Yoo Ah Kim, and Kunsoo Park. Constructing suffix arrays for multi-dimensional matrices. In Martin Farach-Colton, editor, *Combinatorial Pattern Matching, 9th Annual Symposium, CPM 98, Piscataway, New Jersey, USA, July 20-22, 1998, Proceedings*, volume 1448 of *Lecture Notes in Computer Science*, pages 126–139. Springer, 1998. doi:10.1007/BFb0030786.
- 9 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.
- 10 Veli Mäkinen and Gonzalo Navarro. On self-indexing images – image compression with added value. In *2008 Data Compression Conference (DCC 2008), 25-27 March 2008, Snowbird, UT, USA*, pages 422–431. IEEE Computer Society, 2008. doi:10.1109/DCC.2008.47.
- 11 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.
- 12 Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976. doi:10.1145/321941.321946.
- 13 Gonzalo Navarro. *Compact Data Structures – A Practical Approach*. Cambridge University Press, 2016. URL: <http://www.cambridge.org/de/academic/subjects/computer-science/algorithmics-complexity-computer-algebra-and-computational-g/compact-data-structures-practical-approach?format=HB>.
- 14 Yakov Nekrich. A dynamic stabbing-max data structure with sub-logarithmic query time. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation – 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, volume 7074 of *Lecture Notes in Computer Science*, pages 170–179. Springer, 2011. doi:10.1007/978-3-642-25591-5\_19.

- 15 Kuniyiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.
- 16 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.
- 17 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.

## A

 Appendix

A fully functional compact/compressed suffix tree is realized using three components, 1) its *compressed tree topology* that supports navigational functionalities [See Theorem 10] 2) the *compressed suffix array* [See Theorem 11] and 3) auxiliary data structures that supports *longest common prefix* (LCP) information.

► **Theorem 10** (Fully-Functional Succinct Suffix Tree [15]). *The topology of a suffix tree can be encoded in  $4N + o(N)$  bits to support the following operations in  $O(1)$  time.*

- $\text{pre-order}(u)/\text{post-order}(u)$ : *pre-order/post-order rank of node  $u$*
- $\text{parent}(u)$ : *parent of node  $u$*
- $\text{nodeDepth}(u)$ : *number of edges on the path from the root to  $u$*
- $\text{child}(u, q)$ :  *$q$ th leftmost child of node  $u$*
- $\text{sibRank}(u)$ : *number of children of  $\text{parent}(u)$  to the left of  $u$*
- $\text{lca}(u, v)$ : *lowest common ancestor (LCA) of two nodes  $u$  and  $v$*
- $\text{lleaf}(u)/\text{rleaf}(u)$ : *leftmost/rightmost leaf in the subtree of  $u$*
- $\text{levelAncestor}(u, d)$ : *ancestor of  $u$  such that  $\text{nodeDepth}(u) = d$*

► **Theorem 11** (Compressed Suffix Array [15]). *The compressed suffix array part of the above compressed suffix tree can be encoded in  $O(N \log \sigma)$  bits to support the following operations.*

- $\text{lookup}(r)$ : *returns  $\text{SA}[r]$  in time  $O(\log^\epsilon N)$*
- $\text{inverse}(i)$ : *returns  $r = \text{SA}^{-1}[i]$  in time  $O(\log^\epsilon N)$ .*



# Dynamic Boolean Formula Evaluation

**Rathish Das**

Cheriton School of Computer Science, University of Waterloo, Canada

**Andrea Lincoln** ✉

University of California, Berkeley, CA, USA

Simons NTT Fellow, Berkeley, CA, USA

**Jayson Lynch** ✉

Cheriton School of Computer Science, University of Waterloo, Canada

**J. Ian Munro** ✉

Cheriton School of Computer Science, University of Waterloo, Canada

---

## Abstract

We present a linear space data structure for Dynamic Evaluation of  $k$ -CNF Boolean Formulas which achieves  $O(m^{1-1/k})$  query and variable update time where  $m$  is the number of clauses in the formula and clauses are of size at most a constant  $k$ . Our algorithm is additionally able to count the total number of satisfied clauses. We then show how this data structure can be parallelized in the PRAM model to achieve  $O(\log m)$  span (i.e. parallel time) and still  $O(m^{1-1/k})$  work. This parallel algorithm works in the stronger Binary Fork model.

We then give a series of lower bounds on the problem including an average-case result showing the lower bounds hold even when the updates to the variables are chosen at random. Specifically, a reduction from  $k$ -Clique shows that dynamically counting the number of satisfied clauses takes time at least  $n^{\frac{2\omega-3}{6}\sqrt{2k}-1-o(\sqrt{k})}$ , where  $2 \leq \omega < 2.38$  is the matrix multiplication constant. We show the Combinatorial  $k$ -Clique Hypothesis implies a lower bound of  $m^{(1-k^{-1/2})(1-o(1))}$  which suggests our algorithm is close to optimal without involving Matrix Multiplication or new techniques. We next give an average-case reduction to  $k$ -clique showing the prior lower bounds hold even when the updates are chosen at random. We use our conditional lower bound to show any Binary Fork algorithm solving these problems requires at least  $\Omega(\log m)$  span, which is tight against our algorithm in this model. Finally, we give an unconditional linear space lower bound for Dynamic  $k$ -CNF Boolean Formula Evaluation.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Data Structures, SAT, Dynamic Algorithms, Boolean Formulas, Fine-grained Complexity, Parallel Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.61

**Funding** *Rathish Das*: Supported by the Canada Research Chairs Programme and NSERC Discovery Grants.

*Andrea Lincoln*: This work is supported partly by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship. This work was also supported by the Simons NTT research fellowship.

*Jayson Lynch*: Supported by the Canada Research Chairs Programme and NSERC Discovery Grants.

*J. Ian Munro*: Supported by the Canada Research Chairs Programme and NSERC Discovery Grants.

## 1 Introduction

Boolean formula evaluation is a fundamental problem in computer science. There are many cases when one might want to evaluate the formula multiple times on related inputs. For example some SAT solving algorithms evaluate all inputs within some small Hamming Ball around certain variable settings, requiring many evaluations of a Boolean formula on very similar inputs [39, 22, 35]. Another example is systems safety monitoring where one needs



© Rathish Das, Andrea Lincoln, Jayson Lynch, and J. Ian Munro;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 61; pp. 61:1–61:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to check whether certain safety constraints have been violated [32, 18]. One may wonder whether the new values in these settings can be calculated significantly faster than the time it takes to reevaluate the entire formula.

In this paper we define a dynamically updating version of evaluating formulas on Boolean variables in which variables are allowed to change value and the data-structure must maintain whether the formula is still true and, in a harder version, the total number of currently satisfied clauses. In particular, we are given a Boolean expression in  $k$ -CNF (where  $k$  is taken to be a constant) with  $n$  variables and with  $m$  clauses. We achieve a sub-linear time update cost for a generalized version of this problem, including counting the number of satisfied clauses, and show a work-efficient parallelization (Section 3). We give lower bounds on the space and time complexity of this problem, including a fine-grained average-case lower bound when all the updates are given at random (Section 4). We find it exciting that these average-case lower bounds are as strong as our worst case lower bounds, and that only a small gap remains between those and our algorithmic upper bound.

The computational complexity of evaluating Boolean formulas has been well studied, showing the problem is in  $\text{ALOGTIME}$  [17] and the related Boolean Circuit Evaluation problem is complete for  $NC^1$  [16]. There is even a classification of Boolean Formulas under various families of allowed connectives [38]. Since we deal with formulas in  $k$ -CNF form, this restriction of the problem would lie in  $\text{AC}0$ , as it is by definition depth 2. However, when we consider the counting version in which we keep track of the number of clauses currently satisfied, it must be outside  $\text{AC}0$  since it can trivially solve the Parity problem [25].

There has been significant work in the fine-grained complexity of dynamic problems which give conditional lower bounds for a variety of problems such as dynamic shortest path, graph connectivity, bipartite matching, max-flow, and graph diameter [37, 36, 3, 28, 20, 29]. Our results on the fine-grained complexity of Dynamic Boolean Formula Evaluation add to this body of knowledge, more importantly it brings the field of average-case fine-grained complexity to dynamic problems and data-structures. Although Alberts-Henzinger give algorithms for some average-case dynamic problems [5], but we're not aware of any work on average-case lower bounds.

Common fine-grained complexity assumptions were used to establish the average-case hardness of evaluating certain types of polynomials over finite fields in [7]. This was used to show the average-case hardness of counting the number of  $k$ -cliques in certain easy to sample random graphs [27]. This was then adapted to Erdős-Rényi random graphs in [14] and to counting bicliques in [30]. More recently, hardness for evaluating lower dimensional polynomials has been established from even weaker assumptions than those standard in fine-grained complexity and these have been used to give average-case hardness for a variety of problems such as Edit Distance and Max-Flow [21].

One of the goals of average-case fine-grained complexity is the development of fine-grained cryptography where one hopes to actually prove cryptographic security from fine-grained complexity assumptions and may offer cryptographic protocols that remain secure even if more common cryptographic assumptions turn out to be false. One-way functions and public key cryptography which is unconditionally secure against  $\text{AC}0$  circuits was shown in [24]. In [8] fine-grained proof of work was shown from standard fine-grained complexity assumptions. Finally, one-way functions and public key encryption were built based on the average-case complexity of zero- $k$ -clique [34].

## 2 Preliminaries

We study the problem of dynamically updating queries to a Boolean formula. We define this problem formally and then define the hypotheses from which we get lower bounds for this dynamic problem. Note that in this paper we treat  $k$  as a constant.

► **Definition 1.** In *Dynamic Boolean Formula Evaluation* you are given a fixed  $k$ -CNF formula  $\phi$  with  $m$  clauses and  $n$  variables. Further, the variables are given as an array  $\vec{x}$  and initially set to true. The objective is to maintain a data-structure which can answer whether the formula evaluated on the current setting of the variables evaluates to true, subject to updates which flip the value of a single variable. Specifically an update is given as a single index  $i \in [0, n - 1]$  indicating  $\vec{x}[i] = !\vec{x}[i]$ .

In the counting version of the problem, *Counting Dynamic  $k$ -CNF Boolean Formula*, the query instead asks how many clauses are set false. In the *parity* version of the dynamic formula evaluation problem the queries ask for the parity of the number of clauses set false.

### 2.1 Computational Hypotheses

We get computational lower bounds on the dynamic formula evaluation problem from variants of the  $k$ -clique problem. In this context  $\omega$  is the matrix multiplication constant, that is, the smallest real number such that an algorithm exists with running time  $O(n^{\omega+o(1)})$  exists for matrix multiplication. It is known that  $\omega \in [2, 2.37286]$  [6].

► **Definition 2.** The  $k$ -clique hypothesis states that the  $k$ -clique problem requires  $n^{\omega k/3 - o(k)}$  time with randomized or deterministic algorithms [2].

The next hypothesis has to do with combinatorial algorithms. These are an informally defined set of algorithms which use only “combinatorial methods”, specifically excluding fast matrix multiplication. Although informal, these lower bounds can help inform algorithm design. In our case, the hypothesis will also allow for a better presentation of the reduction and analysis. For a discussion of this hypothesis see [1].

► **Definition 3.** The combinatorial  $k$ -clique hypothesis states that for combinatorial algorithms the  $k$ -clique problem requires  $n^{k - o(1)}$  time with randomized or deterministic algorithms [1].

There exist decision to parity reductions for the clique problem. Goldreich defines  $CC_2^{(\ell)}(G)$  as the parity of the count of the number of  $\ell$  cliques in  $G$ .

► **Theorem 4** (Decision to parity  $\ell$ -clique [26, Theorem 1]). For every integer  $\ell \geq 3$ , there is a randomized reduction of determining whether a given  $n$ -vertex graph contains an  $\ell$ -clique to computing  $CC_2^{(\ell)}$  on  $n$ -vertex graphs such that the reduction runs in time  $O(n^2)$ , and makes  $\exp(\ell^2)$  queries, and has error probability at most  $1/3$ .

### 2.2 Binary-Forking Model

The Binary-forking model [12, 4, 9, 10, 11, 23], formally defined in [12], is designed to capture the performance of algorithms in the modern multicore shared-memory machines. In this model, the computation starts with a single thread, and as the computation progresses threads are created dynamically and *asynchronously*. The binary-forking model better captures the asynchronous events such as cache misses, varying clock speed, interrupts, etc., than the well-studied and stronger PRAM model [31] where computation progresses synchronously. Since modern multicore architectures employ multiple caches, processor

pipelining, branch prediction, hyper-threading, etc., many asynchronous events arise in the system, thus demanding the development of a parallel computation model where computation can proceed asynchronously.

Computations in the binary-forking model can be described as max-degree 3 series-parallel directed acyclic graphs (DAGs). A node in the DAG represents a thread's instruction and each node has at most two children. The root of the DAG is the first instruction of the starting thread. If node  $u$  denotes the  $i$ -th instruction of thread  $t$  and  $u$  has only one child  $v$ , then  $v$  denotes the  $(i + 1)$ -th instruction of the same thread  $t$ . If node  $u$  has two children  $v$  and  $w$ , then  $v$  represents the  $(i + 1)$ -th instruction of the same thread  $t$  and  $w$  represents the first instruction of the new forked thread  $t'$ . Note that whether  $w$  represents the 1-st thread  $t'$  or the  $(i + 1)$ -th instruction of thread  $t$  is arbitrary, we just want to model the overall structure of the computation. The binary-forking model includes “join” instructions to join the forking threads which are modeled as nodes with two incoming edges.

The work of the computation is the number of nodes in the series-parallel DAG and the span of the computation is the length of the longest path in the DAG assuming unbounded resources such as processors and space.

### 3 Algorithm

In this section we first describe our main algorithm which achieves an update time of  $O(m^{1-\frac{1}{k}})$ , and then we describe how to parallelize it in the PRAM model with total work  $O(m^{1-\frac{1}{k}})$  and span (parallel time)  $O(\log m)$ . We assume  $k$  is a constant.

#### 3.1 Main Algorithm

The high level idea is to take as input a formula  $\phi$  and handle the variables which appear in many clauses (high frequency) and variables that appear in relatively few clauses (low frequency) with different methods. For a low frequency variable we simply update each clause in which it is involved. For high frequency variables we group all clauses with roughly the same structure together and simply track the number of clauses with that structure. Fundamentally, we will try to track for each clause how many variables are set true and how many are set false.

► **Theorem 5.** *Given a  $k$ -CNF formula  $\phi$  with  $m$  clauses and  $n$  variables there is an algorithm which takes  $O(km)$  preprocessing time and every further update takes  $O(m^{1-1/k})$  time to solve the dynamic formula evaluation problem as well as the counting and parity variants of the dynamic formula evaluation problem.*

**Proof.** Given the formula  $\phi$  with  $m$  clauses split the variables into two sets:  $H$  for high frequency variables and  $L$  for low frequency variables. Variables in  $H$  appear in at least  $\lambda$  clauses. Variables in  $L$  appear in fewer than  $\lambda$  clauses. Let  $\vec{a}$  be the current assignment, without loss of generality assume the starting assignment is the all true assignment ( $\vec{a} = \vec{1}$ ).

Our goal will be to spend  $O(\lambda)$  time updating each variable in  $L$  when it flips value. We want to build a structure to make updates of variables in  $H$  faster than the number of clauses that contain them. First, we will have  $k + 1$  variables  $count_0, \dots, count_k$ . The variable  $count_i$  will track the count of the number of clauses with  $i$  literals set true. Now we will build  $k$  arrays  $A_1, \dots, A_k$ . The  $A_i$  array will keep track of clauses with  $i$  high frequency variables. An entry  $A_i[x_1][x_2] \dots [x_i][j]$  will hold a number. That number will be the count of the number of clauses that contain literals  $x_1, \dots, x_i \in H$  and have  $j$  variables (not necessarily from  $x_1, \dots, x_i$ ) set true. Note, although we construct the  $A_i$  based only on high frequency variables, there may be many clauses with a mix of high and low frequency variables and

these arrays help track all of those clauses. So each  $A_i$  has a total of  $(2|H|)^i \cdot k$  entries, each entry has a number in  $[0, m]$ . There are  $2|H|$  literals for the high-frequency variables (both  $x$  and  $\bar{x}$ ), each of the  $i$  chosen literals can be any one of those  $2|H|$  literals, and for each of these combinations we can have up to  $k$  literals true in a clause. Since ordering is equivalent, this is actually an over-count of what is needed, but we work with this number for convenience. For convenience and uniqueness, we give all variables an arbitrary order and only fill out entries where  $x_1 < x_2 < \dots < x_i$ . We maintain another helpful Boolean array  $S$  that tracks the current settings of all variables:  $S[x_i]$  indicates the current value of  $x_i$ . Yet another array allows the variables in  $L$  to have quick update times. We build the array  $B$  of size  $2|L|$  where  $B[x]$  contains a representation of the at most  $\lambda$  clauses in which the literal  $x \in L$  appears. This is  $2|L|$  because we have both  $x$  and  $\bar{x}$ . We want these representations to include a marking for which of the variables are high frequency variables.

**Preprocessing.** We fill in  $B$  with the representations of all clauses for low frequency variables. Recall that we mark each clause with which of the variables in the clause are high frequency. Set  $S[x_i] = \text{True}$  for all  $x_i$ . For all the  $A_i$  we go through all  $m$  clauses in  $\phi$  and update the counts. Consider the following example and a clause  $c = (x_1 \vee x_2 \vee \dots \vee x_k)$  where the  $x_i$  are literals. Suppose our initial assignment  $\vec{a} = \vec{1}$  sets  $\ell$  literals to true and  $i$  of the literals are high frequency variables,  $x_1, \dots, x_i \in H$ , then increment  $A_i[x_1] \dots [x_i][\ell]$  by one. We also want to update  $count_0, \dots, count_k$ ; note that  $count_\ell$  is simply a sum of all entries  $A_i[*] \dots [*][\ell]$  for all  $i$ . We can also compute these in the preprocessing stage in  $O(km + (2|H|)^k \cdot k^2)$  time to initialize the arrays and by simply evaluating each clause. We will later see that this is  $O(km)$  with our selection of the size of  $|H|$ .

Now every entry  $A_i[x_1] \dots [x_i][\ell]$  is a count of how many clauses with the associated literals have  $\ell$  of those literals set to true. We want to maintain this.

**Updates for variables in  $L$ .** If we flip a variable  $x_L \in L$  then go to  $S[x_i]$  to see the current setting, call it  $b$ . Flip it so that now  $S[x_L] = \bar{b}$ . Next read  $B[x_L]$  and  $B[\bar{x}_L]$  to get all clauses containing  $x_L$ . We will deal with each clause differently depending on how many high frequency variables it has.

If the clause  $c$  has no high frequency variables, then read the  $k$  entries  $S$  for the settings of the variable to determine how many literals were set true before the flip. Let  $\ell$  denote this number. Now, by flipping  $x_L$  we either increment or decrement the number of literals in  $c$  that are set true (either  $+1$  or  $-1$  depending on the original setting and the literal). Suppose the new number of literals set true is  $\ell + \Delta$ , then we decrement  $count_\ell$  and increment  $count_{\ell+\Delta}$  (the clause is moving from being an  $\ell$  literal true clause to an  $\ell + \Delta$  clause). All of this takes  $O(k)$  time.

If the clause  $c$  has  $i > 0$  high frequency variables, then we have to make changes to  $A_i$  as well. First, we read the  $k$  entries for variable settings in  $S$  to evaluate the clause. Suppose the high frequency variables are  $x_1, \dots, x_i$  and that  $c$  had  $\ell$  literals set true before the flip. Suppose that after the flip of  $x_L$  the clause  $c$  has  $\ell + \Delta$  literals set true. Then we decrement  $A_i[x_1] \dots [x_i][\ell]$  and increment  $A_i[x_1] \dots [x_i][\ell + \Delta]$ . As before we decrement  $count_\ell$  and increment  $count_{\ell+\Delta}$ . This takes  $O(k)$  time. With at most  $\lambda$  clauses, the total time is  $O(k\lambda)$ .

The variable  $count_0$  now holds the correct total for the number of unsatisfied clauses.

**Updates for variables in  $H$ .** Now suppose we are flipping a variable in  $H$ ,  $x_H$ . For a variable in  $H$  we cannot look at all of its clauses. However, we can look at all of the relevant entries in  $A_i$  arrays. First in  $O(1)$  time read our previous setting  $b = S(x_H)$  and update  $S(x_H) = \bar{b}$ .

Now update the  $A_i$  arrays. For all  $i \in [1, k]$  we will read all the entries that involve our variable  $x_H$ . Note that there are  $2 \cdot (2|H|)^{i-1} \cdot k$  entries with  $x_H$  in each  $A_i$ , and so  $O((2n)^{k-1})$  entries in total since  $|H| < n$  and  $k$  is assumed to be constant. Without loss of generality assume that  $b = True$  and we are flipping the variable from false to true. Also, for convenience of notation assume that  $x_H$  is the first variable in our arbitrary order (if instead  $x_H$  appears elsewhere simply access with the variables in order). Then for all  $(2n)^{i-1}$  choices of variables  $x_2, \dots, x_i$  we will do the following (from  $v = 0$  to  $v = k$ ):

$$A_i[x_H][x_2] \dots [x_i][v] = \begin{cases} 0 & \text{if } v = k \\ A_i[x_H][x_2] \dots [x_i][v+1] & \text{if } v < k \end{cases}$$

and (from  $v = k$  to  $v = 0$ )

$$A_i[\bar{x}_H][x_2] \dots [x_i][v] = \begin{cases} 0 & \text{if } v = 0 \\ A_i[\bar{x}_H][x_2] \dots [x_i][v-1] & \text{if } v > 0 \end{cases}.$$

The above procedure simply moves the appropriate number of clauses up or down a bucket depending on whether the literal is now true or false. If  $x_H$  appears, it used to be true and is now false. So, every clause with  $x_H$  now has one fewer literal true, we can move all those clauses into the bucket of the array one below it. If  $\bar{x}_H$  appears, it used to be false and is now set true. So we can move all those clauses to the buckets above them. We also have to update our counts. If you move  $t$  clauses from  $A_i[*] \dots [v]$  to  $A_i[*] \dots [v + \Delta]$  then decrement  $count_v$  by  $t$  and increment  $count_{v+\Delta}$  by  $t$ .

The variable  $count_0$  now holds the correct total for the number of unsatisfied clauses.

Each entry  $A_i[*] \dots [v]$  takes  $O(1)$  time to update (or  $O(\lg(n))$  time in the RAM model instead of word RAM). We need to update a total of  $O(k(2|H|)^{k-1})$  entries.

**Overall Runtime and Space.** The space usage is  $m+n+k(2|H|)^k$ . The runtime is (assuming  $k$  is a constant)  $O(\lambda + |H|^{k-1})$ . Now, we can bound  $|H| < km/\lambda$  and so the runtime is  $O(\lambda + (m/\lambda)^{k-1})$ . Optimizing, we choose  $\lambda = (m/\lambda)^{k-1}$ , and get  $\lambda = m^{(k-1)/k} = m^{1-1/k}$ . For a total claimed runtime of  $O(m^{1-1/k})$ . ◀

This algorithm gives a non-trivial improvement over the naive algorithm which requires  $\Theta(m)$  time updates. We will discuss lower bounds in Section 4, where we are able to show this runtime is close to optimal if the  $k$ -clique hypothesis is true (for example a runtime of  $O(n)$  are ruled out).

### 3.2 A Parallel Algorithm

We now show how to parallelize the algorithm of Section 3.1 in the PRAM CREW (concurrent read exclusive write) model [31]. Our results also hold in a more restrictive parallel model called the binary-forking model [12].

► **Theorem 6.** *Given a  $k$ -CNF formula  $\phi$  with  $m$  clauses and  $n$  variables there is a parallel algorithm (in PRAM CREW model and the binary-forking model) which for each variable-update performs  $O(m^{1-\frac{1}{k}})$  work in expectation and achieves  $O(\log m)$  span (parallel runtime) with high probability (in the number of clauses) to solve the dynamic formula evaluation problem.*

**Proof.** We use the same arrays in the parallel algorithm as in the serial algorithm to achieve the work and span bounds.

**Updates for low frequency variables.** There are at most  $\lambda$  clauses where low frequency variable  $x_L$  might appear. For each clause  $c$  where  $x_L$  appears, we do the following. Without loss of generality assume that clause  $c$  has  $i$  high-frequency variables  $x_1, x_2, \dots, x_i$  and has  $\ell$  true literals before flipping  $x_L$ . Let  $\ell + \Delta$  be the number of true literals in  $c$  after the flip.

We update  $A_i[x_1] \dots [x_i][\ell], A_i[x_1] \dots [x_i][\ell + \Delta]$ ,  $\text{count}_\ell$ , and  $\text{count}_{\ell+\Delta}$  for each clause  $c$  where  $x_L$  appears.

It might happen that multiple clauses trigger updates to the same entry  $A_i$  or  $\text{count}_\ell$ , since multiple clauses may have the same set of high-frequency variables and the same number of true literals. When multiple updates try to edit the same entry in parallel, write-write conflicts (race conditions) arise which lead to incorrect answers. We resolve race condition using  $O(\lambda)$  extra space as follows.

Since  $x_L$  is a low-frequency variable,  $x_L$  could appear in  $\lambda$  clauses at most. Suppose that each  $A_i[x_1] \dots [x_i][\ell]$  has an ID. We allocate two arrays each of size  $O(\lambda)$ . One array is used to compute the  $\text{count}_\ell$ s and the other array is used to compute the  $A_i$ s. For each of the  $\lambda$  clauses, we allocate a designated  $O(1)$  space in each array. Recall that  $\text{count}_\ell$  denotes the number of clauses with  $\ell$  literals set true. We would like to update  $\text{count}_\ell$  after flipping variable  $x_L$ .

We now describe how we decrement  $\text{count}_\ell$  and increment  $\text{count}_{\ell+\Delta}$  for each clause  $c$ . Each  $c$  computes corresponding  $\text{count}_\ell$  and  $\text{count}_{\ell+\Delta}$  (as in the serial version) in  $O(k)$  time ( $k$  is a constant here) and write them in the designated location in the array allocated to compute  $\text{count}_\ell$  (see Figure 1 in the Appendix A). We perform *semisort* [13] (i.e. collect equal values in groups) on the array so that all the  $\text{count}_i$  comes before all the  $\text{count}_j$  where  $i < j$ . The randomized semisort algorithm [13] performs  $O(\lambda)$  work in expectation and takes  $O(\log \lambda)$  parallel time with high probability. After that we do a binary reduction to get the correct  $\text{count}_i$ s. We do this binary reduction in  $O(\log \lambda)$  parallel time. Note that we avoid race conditions by updating  $\text{count}_i$  in different locations.

Similar analysis holds to calculate the  $A_i$ s where we store the the value of the  $A_i$ s according to their IDs. Hence, a low frequency is updated in  $O(\log \lambda)$  parallel time and uses  $O(\lambda)$  extra space.

**Updates for high frequency variables.** We do not parse the clauses where the high frequency variable occurs. Instead, we update the corresponding  $A_i$ s and  $\text{count}_i$ s as follows.

(from  $v = 0$  to  $v = k$ ):

$$A_i[x_H][x_2] \dots [x_i][v] = \begin{cases} 0 & \text{if } v = k \\ A_i[x_H][x_2] \dots [x_i][v + 1] & \text{if } v < k \end{cases}$$

and (from  $v = k$  to  $v = 0$ )

$$A_i[\bar{x}_H][x_2] \dots [x_i][v] = \begin{cases} 0 & \text{if } v = 0 \\ A_i[\bar{x}_H][x_2] \dots [x_i][v - 1] & \text{if } v > 0 \end{cases}.$$

In short, if  $x_H$  appears it used to be true and is now false. So, every clause with  $x_H$  now has one fewer literal true, we can move all those clauses into the bucket of the array one below it. If  $\bar{x}_H$  appears it used to be false and is now set true. So we can move all those clauses to the buckets above them.

We can do all these updates in parallel in  $O(1)$  parallel time. We update the  $\text{count}_\ell$  after we update the  $A_i$ s. We update  $O(k \cdot 2|H|^{k-1})$  entries of  $A_i$  where  $x_H$  is present. If we move  $t$  clauses from  $A_i[*] \dots [v]$  to  $A_i[*] \dots [v + \Delta]$  then we decrement  $\text{count}_v$  by  $t$  and



increment  $count_{v+\Delta}$  by  $t$ . As multiple  $A_i$  entries may update the same  $count_i$  entry, we use an array of  $O(k \cdot 2|H|^{k-1})$  extra space to avoid the race condition for parallel updates. Like before, we semisort the array based on the  $v$  of  $A_i[*] \dots [*][v]$ . Then we perform a parallel sum of the the entries with the same  $v$  and then add/subtract this sum to  $count_i$ . Combining all these the span to update a high frequency variable is  $O(\log |H|) = O(\log n)$ .

As we set  $\lambda = m^{1-\frac{1}{k}}$ , the span to update a low frequency variable is  $O(\log \lambda) = O(\log m)$  and a high frequency variable is  $O(\log n)$ . Since we do not increase the work asymptotically, the work in this parallel algorithm remains the same as in the serial version. Hence, total work performed is  $O(\lambda) = O(m^{1-\frac{1}{k}})$  in expectation. The span is  $O(\log m)$  w.h.p. due to semisort. ◀

We could replace the randomized semisort algorithm with a deterministic sorting algorithm. Cole-Ramachandran's [19] deterministic sorting algorithm performs  $O(x \log x)$  work and takes  $O(\log x \log \log x)$  span to sort  $x$  items in the binary-forking model (in PRAM CREW model also) giving the following theorem.

► **Theorem 7.** *Given a  $k$ -CNF formula  $\phi$  with  $m$  clauses and  $n$  variables there is a deterministic parallel algorithm (in PRAM CREW model and the binary-forking model) which for each variable update performs  $O(m^{1-\frac{1}{k}} \log m)$  work and achieves  $O(\log m \log \log m)$  span (parallel running time) to solve the dynamic formula evaluation problem.*

## 4 Lower Bounds for Dynamic Formula Evaluation

In this section we present lower bounds for the dynamic formula evaluation problem on  $k$ -CNF formulas. First, we present an unconditional linear lower bound on the space of the data-structure. Next, we present a series of conditional lower bounds on the running times of preprocessing, updates, and queries. These are based on the  $k$ -Clique Hypothesis. This culminates in an average-case lower bound for random updates to the variables based on counting  $k$ -cliques in Erdős-Rényi graphs. Finally, we present a conditional lower bound on the span of the algorithm in the binary forking model. Recall that we are treating  $k$  as a constant in this section.

### 4.1 Linear Space Lower Bound

We show a space lower bound linear in the number of variables by a reduction from INDEX [33]. Details can be found in the Appendix.

► **Theorem 8.** *Every randomized algorithm for dynamic-formula-evaluation which correctly decides that the formula for each variable flip (with  $n$  variables,  $O(n)$  clauses, and clause size at least 2) is satisfied or not with probability strictly larger than  $1/2$ , uses  $\Omega(n)$  space in the worst case.*

### 4.2 The $k$ -Clique Hypotheses and a General Reduction

We give these lower bounds from hypotheses related to the  $\ell$ -clique problem. We will use  $k$  for the  $k$ -CNF Dynamic Counting Boolean Formula problem we reduce to. First, let us present a generic reduction from all  $\ell$ -clique instances to a dynamic formula evaluation formula  $\phi$  on  $n = |V|$  variables, where each given instance of  $\ell$ -clique will correspond to a single setting of the variables  $X$ .

► **Lemma 9.** *Consider a graph  $G$ , where we have not yet decided what edges exist or don't. There are  $|V|$  vertices and  $\binom{|V|}{2}$  potential edges. There exists a formula  $\phi_\ell$  with  $n = \binom{|V|}{2}$  variables, width  $k = \binom{\ell}{2}$  clauses, and a total of  $m = \binom{|V|}{\ell}$  clauses such that when the  $n$  variables  $X$  are set with a 1 if the edge exists in  $G$  and 0 if the edge doesn't exist then the number of false clauses in  $\phi_\ell(X)$  corresponds to the number of  $k$  cliques in  $G$ .*

Next we want a useful lemma which will help us bound the preprocessing time. We will use many smaller instances of the clique problem so that our number of update calls to our data-structure is large compared to its initial size. Variations on this lemma are folklore, but a proof is provided for completeness.

► **Lemma 10.** *Assume (existence/counting/parity) (combinatorial)  $\ell$ -clique requires  $|V|^{c\ell(1-o(1))}$  time for some constant  $c$ . Let  $T(\cdot)$  be the time to solve a single instance when given a list of instances. Then giving correct answers to  $g^\ell$  instances each of size  $|U| = \ell|V|/g$  takes  $g^\ell T(|U|)$  time which is at least  $|V|^{c\ell(1-o(1))}$  time.*

*Consider value  $0 < d < c \leq 1$  where we insist that  $(\ell|V|/g)^\ell = \Theta(|V|^{d\ell})$  then  $T(|U|)$  requires at least  $|U|^{(c-\frac{1+d}{d})\ell(1-o(1))}$  time.*

**Proof.** Partition the vertex set into  $V_1, \dots, V_g$  each partition of size  $|V|/g$ . Now we form  $g^\ell$  instances each of size at most  $|U| = \ell|V|/g$ . These instances are all possible choices of at most  $\ell$  of our vertex sets merged together. Using inclusion exclusion we can count the number of cliques in the original graph using calls to these problems. This inclusion and exclusion also allows parity to solve parity. We will find a clique in these instances if and only if there is at least one clique in the original graph.

Given this reduction we can solve a (existence/counting/parity) (combinatorial)  $\ell$ -clique problem with answers to  $g^\ell$  instances each of size  $|U| = \ell|V|/g$ , giving the first statement.

For the second statement note that  $g = \Theta(|V|^{1-d})$ . Then note that  $|U| = \Theta(|V|^d)$ . Now we can say that  $T(|U|)$  must be at least  $|V|^{(c-1+d)\ell(1-o(1))}$  time. So, in terms of  $|U|$ ,  $T(|U|)$  must be at least

$$|U|^{(c-\frac{1+d}{d})\ell(1-o(1))}. \quad \blacktriangleleft$$

Note that when  $c = 1$  you get no loss in efficiency regardless of the constant  $d < 1$  you pick. This case of  $c = 1$  happens for combinatorial  $\ell$ -clique. It is why we use it as an example. It simplifies the reduction to not worry about the loss that comes from a large  $d$ .

### 4.3 From the Combinatorial $k$ -Clique Hypothesis

Using the reduction in the prior section, we give a lower bound from the combinatorial  $\ell$ -clique hypothesis. Note how we choose the value of  $d$  for the reduction from Lemma 10. We want  $d$  to be small so that our preprocessing time is small. This is what will cause inefficiency in the future reductions when  $c \neq 1$ ; we lose efficiency in the reduction when  $d$  is small.

► **Theorem 11.** *Assume  $\ell$  is a constant. If the combinatorial  $\ell$ -clique hypothesis is true then any combinatorial algorithm  $A$  for the Dynamic Counting Boolean Evaluation problem,  $U(n, m, k)$ , with polynomial preprocessing time  $P(n, m, k) = \text{poly}(n) \cdot \text{poly}(m) \cdot \text{poly}(k)$  requires  $m^{(1-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U(n, m, k)$  requires at least  $n^{(1-o(1))(\sqrt{2k}-2)/2}$  time.*

#### 4.4 From the $k$ -Clique Hypothesis

We will now use the same reduction style as the previous section. However, now  $c$  from Lemma 10 will be  $c = \omega/3$  instead of 1. Here  $\omega$  is the matrix multiplication exponent constant. The analysis proceeds similarly to that of Section 4.3. Details are in the Appendix.

► **Theorem 12.** *Assume  $\ell$  is a constant. If the  $\ell$ -clique hypothesis is true then any algorithm  $\mathcal{A}$  for the Dynamic Counting Boolean Evaluation problem,  $U(n, m, k)$ , with polynomial preprocessing time  $P(n, m, k) = O(m)$  requires  $m^{\left(\frac{2\omega-3}{3} - k^{-1/2}\right)(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U(n, m, k)$  requires at least  $n^{\frac{2\omega-3}{6}\sqrt{2k}-1-o(\sqrt{k})}$  time.*

#### 4.5 Random Input Implications

Our goal is to show that with a worst-case fixed formula, specifically our  $k$ -clique formula, it is still hard to evaluate updates to the formula when the *updates are random*. We will show lower bounds against algorithms which always correctly implement updates, but, are allowed error when responding to queries. The queries this dynamic formula evaluation algorithm must support are queries on the number of falsified clauses.

For this result we will use a theorem from [26]<sup>1</sup>. At a high level, they show that the counting/parity  $l$ -clique problem requires just as long over the uniform average-case distribution as it does in the worst-case when  $l$  is a constant. First we will define the model of average-case inputs we are considering.

► **Definition 13.** *Let Average-Case counting dynamic formula evaluation (AC#DFE) be the problem where you are given a worst-case formula to evaluate but random (so average-case) updates on the variables. More specifically, you are given a worst-case formula  $\phi$ , and a series of updates where each update flips the assignment of a uniformly random chosen variable. A data structure  $D$  for AC#DFE with error probability  $\epsilon$  takes as input a worst-case  $\phi$  and after each random update answers the count of the number of false clauses in the formula with probability  $1 - \epsilon$ .*

► **Definition 14 (Erdős-Rényi graphs).** *Create a graph with  $|V|$  vertices and no edges to start with. Now, for every pair of vertices in the graph iid include an edge between them with probability  $1/2$ . Thus, every potential edge in the graph appears with probability  $1/2$  independently from all other edges.*

Now we can describe one theorem from [26]. First we start by using a piece of their notation. Define  $CC_2^{(\ell)}(G)$  as the parity of the count of the number of  $\ell$  cliques in  $G$ .

► **Theorem 15 (Average-Case  $\ell$ -clique [26, Theorem 2]).** *For every integer  $\ell \geq 3$ , there is a randomized reduction of computing  $CC_2^{(\ell)}$  on the worst-case  $n$ -vertex graph to correctly computing  $CC_2^{(\ell)}$  on at least a  $1 - \exp(-\ell^2)$  fraction of the  $n$ -vertex graphs such that the reduction runs in time  $O(n^2)$ , makes  $\exp(\ell^2)$  queries, and has error probability at most  $1/3$ .*

So, up to sub-polynomial factors, average-case counting and parity  $\ell$ -clique are just as hard on average as worst-case parity  $\ell$ -clique. Now, we can use these reductions to show lower bounds for our problem with random inputs. First, we will show that if you start

<sup>1</sup> For an alternate presentation of the parity proof of this result see [15]. The formulation from [26] is easier to build on in this case.

from a graph and make  $\binom{n}{2} \lg^2(n)$  random edge flips the resulting graph is drawn from a distribution that has total variation distance less than  $2^{-\Omega(\lg^2(n))}$  from an Erdős-Rényi graph. Note that, on an intuitive level, this means that when we sample from our edge flipping distribution we “look like” we are sampling an Erdős-Rényi graph a  $1 - 2^{-\Omega(\lg^2(n))}$  fraction of the time. Since our reduction represents edges as variables, flipping a random variable is like flipping an edge in the original graph, and we want to argue that after not too many flips we have a random looking graph.

► **Lemma 16.** *Given a graph  $G$  let the procedure  $F$  be the process of selecting a uniformly random pair of vertices  $(u, v)$  and either deleting edge  $(u, v)$  if it exists or inserting  $(u, v)$  if it doesn't exist.*

*If we run the procedure  $F \Theta\left(\binom{n}{2} \lg^2(n)\right)$  times on any graph  $G$  the resulting distribution over graphs has total variation distance at most  $n^{2-\Omega(\lg(n))}$  from the distribution over Erdős-Rényi graphs.*

The high level idea for the reduction is to take the worst-case reduction from the prior section and at the last step add the randomized reduction. Specifically, in the worst case we show that solving a list of instances  $G_1, \dots, G_x$  is still hard. Now, for the average-case we will use the [26] reduction to turn each  $G_i$  into  $\exp(\ell^2)$  queries on random graphs, call these  $G_i^{(1)}, \dots, G_i^{(\exp(\ell^2))}$ . Note that these graphs each look random, but have correlations. So, we can't ask the same AC#DFE data structure about two of these. However, we can spin up  $\exp(\ell^2)$  instances of data structures for AC#DFE:  $D_1, \dots, D_{\exp(\ell^2)}$ . We will, implicitly, give the data structure  $D_j$  graphs  $G_1^{(j)}, \dots, G_x^{(j)}$ . These will be totally uncorrelated (each are randomized under different random bits). This will correspond to random updates on variables (which in our worst-case reduction correspond to edges in the graph).

We will show in this next theorem that even with random updates, the same lower bounds hold (up to factors in time of  $2^{2k}$ , which if  $k$  is a constant is simply a constant).

► **Theorem 17.** *Let  $k$  be a constant. Say there is a data structure  $D$  that can solve the AC#DFE problem on worst-case  $k$ -CNF formulas  $\phi$  with error  $\epsilon < \exp(-2k)/3$  with  $P_D(n, m, k)$  pre-processing time and  $U_D(n, m, k)$  time per query and update. We will note the lower bounds for combinatorial  $D$  from the combinatorial  $\ell$ -clique hypothesis. We will also note the lower bound for general  $D$  from the  $\ell$ -clique hypothesis.*

- *If  $D$  is a combinatorial data structure, the combinatorial  $\ell$ -clique hypothesis is true, and  $P_D(n, m, k) = \text{poly}(n) \cdot \text{poly}(m) \cdot \text{poly}(k)$ , then  $U_D(n, m, k)$  requires  $m^{(1-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U_D(n, m, k)$  requires at least  $n^{(1-o(1))(\sqrt{2k}-2)/2}$  time.*
- *If the  $\ell$ -clique hypothesis is true and  $P_D(n, m, k) = O(m)$  then  $U_D(n, m, k)$  requires  $m^{(\frac{2\omega-3}{3}-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U_D(n, m, k)$  requires at least  $n^{\frac{2\omega-3}{6}\sqrt{2k}-1-o(\sqrt{k})}$  time.*

**Proof.** First, let us describe the general reduction regardless of the hypothesis we start with. We take the list of  $|V|^{\ell(1-d)}$  sub-problems  $G_1, \dots, G_{|V|^{\ell(1-d)}}$  and on each instance individually we run the reduction from Theorem 15. So we take each problem,  $G_i$ , of size  $|V|^d$  and use Theorem 15 to make a list of  $s = \exp(\ell^2)$  queries  $G_i^{(1)}, \dots, G_i^{(s)}$ . Now, we spin up  $s$  different instances of  $D$ , call them  $D_1, \dots, D_s$ . Consider sampling the updates from  $G_i^{(j)}$  to  $G_{i+1}^{(j)}$  uniformly at random from all random series of updates of length  $s = \Theta(n^2 \lg^2(n))$  that flip edges to  $G_{i+1}^{(j)}$ . Because every variable corresponds to an edge in the graph flipping a variable and flipping an edge is a one to one relationship. So, after this series of flips that look uniformly random up to a TVD of  $n^{-\Omega(\lg(n))}$  (by Lemma 16) we get the answer to the number of  $\ell$ -cliques in  $G_{i+1}^{(j)}$ ! By the claims of the Theorem statement each  $D_j$  has an error rate of

$\epsilon < \exp(-2k)/3$  on the random updates of the average-case distribution of AC#DFE our TVD tells us that the error rate of  $D_j$  on this “nearly random” set of updates is at most  $\epsilon + n^{\Omega(\lg(n))}$ . Now note that if each instance  $D_1, \dots, D_s$  gives us the correct answer we can correctly answer our query on all  $G_i$  by Theorem 15. So, we get a correct answer on the worst-case  $G_i$  with probability  $2/3 - \exp(\ell^2) n^{-\Omega(\lg(n))}$ , note this is large enough that this procedure can be repeated to boost this probability. Note this means we get the same lower bounds as we did in the worst-case, but with an overhead of  $\exp(\ell^2)$  which is a constant from our perspective.

First we discuss the combinatorial  $k$ -clique case. Start by taking the reduction from Theorem 11. So if  $D$  is a combinatorial data structure, the combinatorial  $\ell$ -clique hypothesis is true, and  $P_D(n, m, k) = \text{poly}(n) \cdot \text{poly}(m) \cdot \text{poly}(k)$ , then  $U_D(n, m, k)$  requires  $m^{(1-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U_D(n, m, k)$  requires at least  $n^{(1-o(1))(\sqrt{2k}-2)/2}$  time.

Finally, we will address the  $\ell$ -clique hypothesis. As before we will get the same lower bounds, up to a factor of  $\exp(\ell^2)$  which is a constant from our perspective. So if the  $\ell$ -clique hypothesis is true and  $P_D(n, m, k) = O(m)$  then  $U_D(n, m, k)$  requires  $m^{(\frac{2\omega-3}{3}-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U_D(n, m, k)$  requires at least  $n^{\frac{2\omega-3}{6}\sqrt{2k}-1-o(\sqrt{k})}$  time. ◀

#### 4.6 Span Lower Bound in the Binary-Forking Model

In the Appendix we prove an  $\Omega(\log n)$  span lower bound in the binary forking model. This lower bound only requires that the work of the algorithm is  $\Omega(n^c)$  for some constant  $c$ , and is thus a weaker condition than the  $k$ -Clique Hypothesis.

► **Theorem 18.** *The span per update for dynamic boolean formula evaluation is  $\Omega(\log n)$  in the binary forking model.*

## 5 Conclusion and Open Problems

Defining Dynamic Boolean Formula Evaluation and giving a data-structure with sub-linear update time is an important first step in characterizing the complexity of this problem. Our data-structure is simple enough and provides a large enough benefit for small  $k$  that we hope it will inspire practical implementations. On the theoretical side, although we give upper and lower bounds that show roughly what the complexity of this problem should be, there is still a significant gap between them and plenty of room to generalize both types of results.

With a more careful analysis both our algorithms and lower bounds should hold for some small super-constant  $k$ . Improving this analysis may be of interest, but more importantly finding both algorithms and lower bounds that work for a greater range of  $k$ , especially when  $k = m^c$  for some constant  $c$ .

Closing the gap between our lower and upper bounds is one obvious open question. The conditional lower bound giving  $\frac{1}{\sqrt{k}}$  comes directly cliques containing  $l^2$  edges. Thus perhaps one should be trying to use graph problems with sparse structures rather than dense ones. However, one needs to be careful because there is only a single  $k$ -clique on  $k$  vertices whereas there can be many isomorphic sparse structures. The larger issues comes from the factor of the matrix multiplication constant in our lower bound. We see this disappear with the combinatorial  $l$ -clique conjecture which suggests either 1) Dynamic Boolean Formula Evaluation can be solved much faster, but doing so will likely require linear-algebraic techniques or some other fundamentally different approach, 2) we need a slightly more clever reduction. A common solution to strengthening “combinatorial” fine-grained

assumptions is to move to a weighted version of the problem. However, the OR of variables is symmetric under permutation and thus does not appear to be something that supports any sort of arithmetic at the clause level. Further, our algorithms fundamentally require this symmetry and will fail when trying to generalize in this direction.

Further, it would be interesting to have conditional lower bounds for the pure Dynamic Boolean Formula problem rather than the counting version which seems much stronger. Both giving lower bounds for more restricted versions of the problem and generalizing the algorithm to handle more general formulas seems of interest. Generalizing the depth of the formula (no longer requiring CNF form) is another obvious direction.

We believe that a  $\log(n)$ -span lower bound should hold in a stronger model such as CREW PRAM. Both improved lower bounds and whether there are better algorithms in stronger concurrent data-structures models remains an interesting question to explore. Depth 2 Boolean formulas being in AC0 strongly suggests at least this special case should be able to be parallelized more efficiently, perhaps at the cost of work-efficiency.

One obvious problem to consider is a version of Dynamic Boolean Formula Evaluation in which the formula itself can be altered, instead of or in addition to the values of the variables. These problems are equivalent in the DynamicSAT case, but looking for sub-linear update cost we have much less leeway in the efficiency of this reduction. We believe minor alterations to our current algorithm will allow it to handle the insertion and deletion of literals and clauses with the same amortized worst-case cost by simply rebuilding the table whenever variables pass some threshold for their frequency of appearing in clauses. However, de-amortizing this does not seem at all obvious.

---

## References

- 1 Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 192–203. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.26.
- 2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant’s parser. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 98–117. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.16.
- 3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014.
- 4 Umut A Acar, Guy E Blelloch, and Robert D Blumofe. The data locality of work stealing. In *ACM symposium on Parallel algorithms and architectures*, pages 1–12, 2000.
- 5 David Alberts and Monika Rauch Henzinger. Average case analysis of dynamic graph algorithms. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 312–321. ACM/SIAM, 1995. URL: <http://dl.acm.org/citation.cfm?id=313651.313712>.
- 6 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. *CoRR*, abs/2010.05846, 2020. arXiv:2010.05846.
- 7 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 483–496, 2017.
- 8 Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of work from worst-case assumptions. In *Annual International Cryptology Conference*, pages 789–819. Springer, 2018.

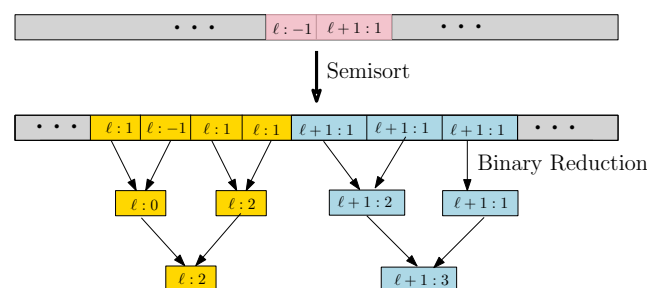


- 9 Naama Ben-David, Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 145–156, 2016.
- 10 Guy E Blelloch, Rezaul Alam Chowdhury, Phillip B Gibbons, Vijaya Ramachandran, Shimin Chen, and Michael Kozuch. Provably good multicore cache performance for divide-and-conquer algorithms. In *SODA*, volume 8, pages 501–510. Citeseer, 2008.
- 11 Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, and Harsha Vardhan Simhadri. Scheduling irregular parallel computations on hierarchical caches. In *ACM symposium on Parallelism in algorithms and architectures*, pages 355–366, 2011.
- 12 Guy E Blelloch, Jeremy T Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 89–102, 2020.
- 13 Guy E Blelloch, Jeremy T Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 89–102, 2020.
- 14 Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1256–1280. IEEE, 2019.
- 15 Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1256–1280. IEEE Computer Society, 2019.
- 16 S Buss, S Cook, Arvind Gupta, and Vijaya Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992.
- 17 Samuel R Buss. The boolean formula value problem is in alogtime. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 123–131, 1987.
- 18 Feng Chen and Grigore Roşu. Towards monitoring-oriented programming: A paradigm combining specification and implementation. *Electronic Notes in Theoretical Computer Science*, 89(2):108–127, 2003.
- 19 Richard Cole and Vijaya Ramachandran. Resource oblivious sorting on multicores. *ACM Transactions on Parallel Computing (TOPC)*, 3(4):1–31, 2017.
- 20 Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
- 21 Mina Dalirrooyfard, Andrea Lincoln, and Virginia Vassilevska Williams. New techniques for proving fine-grained average-case hardness. *arXiv preprint*, 2020. [arXiv:2008.06591](https://arxiv.org/abs/2008.06591).
- 22 Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic  $(2 - 2/(k+1))^n$  algorithm for  $k$ -SAT based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002. doi:10.1016/S0304-3975(01)00174-8.
- 23 Rathish Das, Shih-Yu Tsai, Sharmila Duppala, Jayson Lynch, Esther M Arkin, Rezaul Chowdhury, Joseph SB Mitchell, and Steven Skiena. Data races and the discrete resource-time tradeoff problem with resource reuse over paths. In *ACM Symposium on Parallelism in Algorithms and Architectures*, pages 359–368, 2019.
- 24 Akshay Degwekar, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Fine-grained cryptography. In *Annual International Cryptology Conference*, pages 533–562. Springer, 2016.
- 25 Merrick Furst, James B Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, 1984.
- 26 Oded Goldreich. On counting  $\$t$ -cliques mod 2. *Electron. Colloquium Comput. Complex.*, 27:104, 2020. URL: <https://ecc.weizmann.ac.il/report/2020/104>.



- 27 Oded Goldreich and Guy Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 77–88. IEEE, 2018.
- 28 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.
- 29 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 30 Shuichi Hirahara and Nobutaka Shimizu. Nearly optimal average-case complexity of counting bicliques under seth. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2346–2365. SIAM, 2021.
- 31 J. JaJa. *An Introduction to Parallel Algorithms*. Addison Wesley, 1997. URL: <https://books.google.com/books?id=9BpYtwAACAAJ>.
- 32 Moonjoo Kim, Sampath Kannan, Insup Lee, Oleg Sokolsky, and Mahesh Viswanathan. Javamac: a run-time assurance tool for java programs. *Electronic Notes in Theoretical Computer Science*, 55(2):218–235, 2001.
- 33 Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.
- 34 Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. Public-key cryptography in the fine-grained setting. In *Annual International Cryptology Conference*, pages 605–635. Springer, 2019.
- 35 Andrea Lincoln and Adam Yedidia. Faster random  $k$ -cnf satisfiability. *arXiv preprint*, 2019. [arXiv:1903.10618](https://arxiv.org/abs/1903.10618).
- 36 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610, 2010.
- 37 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *European Symposium on Algorithms*, pages 580–591. Springer, 2004.
- 38 Henning Schnoor. The complexity of the boolean formula value problem. Technical report, Technical report, Theoretical Computer Science, University of Hannover, 2005.
- 39 Uwe Schöning. A probabilistic algorithm for  $k$ -SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414, 1999. doi:10.1109/SFFCS.1999.814612.

## A Parallel Algorithm Figure



■ **Figure 1** Clause  $c$  writes whether count $_{\ell}$  and count $_{\ell+1}$  increment or decrement in their designated locations (in color pink). We then semisort so that all the updates corresponding to count $_{\ell}$  come in adjacent locations (in color yellow for count $_{\ell}$ ). After that we do a binary reduction to get the net update to count $_{\ell}$ .

## B Lower Bounds Omitted Proofs

### B.1 Proof of Linear Space Lower Bound

► **Theorem 19.** *Every randomized algorithm for dynamic-formula-evaluation which correctly decides that the formula for each variable flip (with  $n$  variables,  $O(n)$  clauses, and clause size at least 2) is satisfied or not with probability strictly larger than  $1/2$ , uses  $\Omega(n)$  space in the worst case.*

**Proof.** We prove this using communication complexity, specifically, reducing the problem to the INDEX problem.

**INDEX problem.** Alice gets an  $n$ -bit string  $x \in \{0, 1\}^n$  and Bob gets an integer  $i \in \{1, n\}$ . Alice can communicate to Bob, but Bob cannot send messages to Alice. The goal is to find out  $x_i$ , the  $i$ -th index of  $x$ .

We know the lower bound of the randomized one-way communication complexity of INDEX is  $\Omega(n)$  [33].

We now do the following reduction. Given  $x \in \{0, 1\}^n$ , Alice constructs a CNF formula  $F$  as follows. There are  $n$  clauses and  $2n$  variables in  $F$ . For  $1 \leq j \leq n$ , clause  $C_j = (v_{j,1} \wedge v_{j,2})$ . If  $x_j = 1$ , we assign  $v_{j,1} = \text{TRUE}$  and  $v_{j,2} = \text{FALSE}$ . Otherwise, we assign  $v_{j,1} = \text{FALSE}$  and  $v_{j,2} = \text{TRUE}$ . Note that in formula  $F$ , every clause is initially satisfied and only one variable is TRUE in every clause.

Now given index  $i$ , we flip variable  $v_{i,1}$ . If the formula remains satisfied, then  $x_i = 0$  in Alice's string, otherwise,  $x_i = 1$ . ◀

### B.2 Proofs about $k$ -Clique reduction General Reduction

► **Lemma 20.** *Consider a graph  $G$ , where we have not yet decided what edges exist or don't. There are  $|V|$  vertices and  $\binom{|V|}{2}$  potential edges. There exists a formula  $\phi_k$  with  $n = \binom{|V|}{2}$  variables, width  $k = \binom{\ell}{2}$  clauses, and a total of  $m = \binom{|V|}{\ell}$  clauses such that when the  $n$  variables  $X$  are set with a 1 if the edge exists in  $G$  and 0 if the edge doesn't exist then the number of false clauses in  $\phi_k(X)$  corresponds to the number of  $k$  cliques in  $G$ .*

**Proof.** In this reduction we will use variables to represent possible edges in the graph and we will use clauses to detect cliques. Let us index our variables  $X$  with two numbers  $i, j$  such that  $X[i, j]$  is a variable if  $i < j$  and  $i, j \in [|V|]$ . We will treat  $X[i, j]$  as the variable corresponding to if the edge  $(i, j)$  exists in  $G$ . Now, for all  $i_1 < \dots < i_\ell$  where  $i_1, \dots, i_\ell \in [|V|]$  add the following clause to  $\phi_k$ :

$$(\bar{X}[i_1, i_2] \vee \bar{X}[i_1, i_3] \vee \dots \vee \bar{X}[i_1, i_\ell] \vee \bar{X}[i_2, i_3] \vee \dots \vee \bar{X}[i_{\ell-1}, i_\ell]).$$

Note that if the nodes  $i_1, \dots, i_\ell$  in  $G$  are a  $\ell$ -clique given the setting of edges implied by  $X$ , then this clause is false if any of the edges among vertices 1 to  $\ell$  are missing.

Each clause has size  $k = \binom{\ell}{2}$ . There are a total of  $m = \binom{|V|}{\ell}$  clauses. We need one variable per potential edge in the graph, so there are  $n = \binom{|V|}{2}$  variables. Given a setting of  $X$  we have a corresponding graph  $G$ , and the number of false clauses in  $\phi_k$  is exactly equal to the number of  $\ell$ -cliques in  $G$ . ◀

► **Theorem 21.** *Assume  $\ell$  is a constant. If the combinatorial  $\ell$ -clique hypothesis is true then any combinatorial algorithm  $\mathcal{A}$  for the Dynamic Counting Boolean Evaluation problem,  $U(n, m, k)$ , with polynomial preprocessing time  $P(n, m, k) = \text{poly}(n) \cdot \text{poly}(m) \cdot \text{poly}(k)$  requires  $m^{(1-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U(n, m, k)$  requires at least  $n^{(1-o(1))(\sqrt{2k-2})/2}$  time.*

**Proof.** We are given an instance  $G$  of the  $\ell$ -clique problem with  $|V|$  vertices and  $|E| = O(|V|^2)$  edges. By the combinatorial  $\ell$ -clique hypothesis this problem requires  $|V|^{\ell(1-o(1))}$  time.

First note that, with Lemma 10, we want  $c = 1$  from our combinatorial  $k$ -clique hypothesis. Next assume that  $P(n, m, k) = (n \cdot m \cdot k)^f$  for some constant  $f$ . Now set  $d = \frac{(1-\epsilon)\ell}{(\ell+2)f}$  for some constant  $\epsilon > 0$ . Recall, with these settings, we can say that solving all  $|V|^{\ell(1-d)}$  clique problems requires  $|V|^{\ell(1-o(1))}$  time. This can be stated as a time for each of the problems of size  $|V|^d$ , they each require  $(|V|^d)^\ell$  time.

Now we will use Lemma 9 to produce our formula  $\phi$  on our problems of size  $|V'| = |V|^d$ . Note that  $n = \binom{|V'|}{2} = \Theta(|V'|^2)$  and  $k = \binom{\ell}{2} = \ell(\ell+1)/2$  and  $m = \binom{|V'|}{\ell} = \Theta(|V'|^\ell)$ . Finally, observe  $P(n, m, k) = O((|V|^{\ell(\ell+2)})^f)$  which can be written as  $P(n, m, k) = O((|V|^{\ell(1-\epsilon)})^f)$ . Thus all of the updates we do to solve our clique problem must take  $|V|^{\ell(1-o(1))}$  time.

Note that with at most  $|E'| = O(|V|^{2d})$  variable updates we can cause the number of false clauses in  $\phi_\ell$  to be equal to the number of  $\ell$ -cliques in a new instance. So  $|V|^{2d}$  updates should take at least  $(|V|^d)^\ell$  time. Further,  $U(n, m, k)$  must be at least  $|V|^{d(\ell-2)(1-o(1))} = |V'|^{(\ell-2)(1-o(1))}$ . Now, lets re-state this in terms of  $n, m, k$ . First lets add our value of  $m$  into this equation:

$$|V'|^{(\ell-2)(1-o(1))} = m^{(1-o(1))(\ell-2)/\ell} = m^{(1-2/\ell)(1-o(1))}.$$

We can also write in terms of  $n$ :

$$|V'|^{(\ell-2)(1-o(1))} = n^{(1-o(1))(\ell-2)/2}.$$

Now note that  $2\sqrt{k} > \ell > \sqrt{2k}$ . So we can re-write our above equations as

$$|V'|^{(\ell-2)(1-o(1))} = m^{(1-1/\sqrt{k})(1-o(1))},$$

and

$$|V'|^{(\ell-2)(1-o(1))} = n^{(1-o(1))(\sqrt{2k}-2)/2}. \quad \blacktriangleleft$$

► **Theorem 22.** *Assume  $\ell$  is a constant. If the  $\ell$ -clique hypothesis is true then any algorithm  $A$  for the Dynamic Counting Boolean Evaluation problem,  $U(n, m, k)$ , with polynomial preprocessing time  $P(n, m, k) = O(m)$  requires  $m^{(\frac{2\omega-3}{3}-k^{-1/2})(1-o(1))}$  time. We can also state the lower bound in terms of the number of variables  $n$ , in which case  $U(n, m, k)$  requires at least  $n^{\frac{2\omega-3}{6}\sqrt{2k}-1-o(\sqrt{k})}$  time.*

Using in  $\omega = 2.37286$  (from the best current upper bound [6]) these lower bounds per update are:  $m^{(0.5819-k^{-1/2})(1-o(1))}$  and  $n^{0.5819\sqrt{2k}-1-o(\sqrt{k})}$ .

Using  $\omega = 2$  (the smallest value  $\omega$  could be) the lower bounds per update are:  $m^{(1/3-k^{-1/2})(1-o(1))}$  and  $n^{\sqrt{2k}/3-1-o(\sqrt{k})}$ .

**Proof.** We are given an instance  $G$  of the  $\ell$ -clique problem with  $|V|$  vertices and  $|E| = O(|V|^2)$  edges. By the  $\ell$ -clique hypothesis we have that this problem requires  $|V|^{(1-o(1))\omega\ell/3}$  time.

First note that with Lemma 10, we want  $c = \omega/3$  from our  $l$ -clique hypothesis. Next we assumed that  $P(n, m, k) = O(m)$ . Now set  $d = \frac{(1-\epsilon)\omega}{3}$  for some constant  $\epsilon > 0$  to be set later. Recall, with these settings we can say that solving all  $|V|^{\ell(1-d)}$  clique problems requires  $|V|^{(1-o(1))\omega\ell/3}$  time. For convenience call  $\delta = \epsilon\omega/3$ . This can be stated as a time for each of the problems of size  $|V|^d$ , they each require time

$$R(|V|) = (|V|^d)^{\frac{c-1+d}{d}\ell(1-o(1))} = (|V|^d)^{\frac{2\omega-3-\delta}{3-\delta}\ell(1-o(1))}.$$

## 61:18 Dynamic Boolean Formula Evaluation

We will use the reduction from Lemma 9. Let  $|V'| = |V|^d$ . Now let's note that  $n = \binom{|V'|}{2} = \Theta(|V'|^2)$  and  $k = \binom{\ell}{2} = \ell(\ell+1)/2$  and  $m = \binom{|V'|}{\ell} = \Theta(|V'|^\ell)$ .

Now note that as before if  $P(n, m, \ell) = O(m)$  then it is also  $O(|V|^{d\ell})$  which can be restated as  $|V|^{(1-\epsilon)\omega\ell/3}$ . This is small enough that the updates must take  $|V|^{(1-o(1))\omega\ell/3}$  time.

We require  $O(|V|^{2d})$  updates per instance. And each instance requires  $R(|V|)$  time. So each update requires  $R(|V|)|V|^{-2d}$  time which we can expand to

$$(|V|^d)^{\left(\frac{2\omega-3-\delta}{3-\delta}\right)\ell(1-o(1))-2}$$

time. Now note that  $2\sqrt{k} > \ell > \sqrt{2k}$ . If we re-state this with  $m$  we get:

$$m^{\left(\frac{2\omega-3-\delta}{3-\delta}\right)(1-o(1))-1/\sqrt{k}}$$

If we treat  $\epsilon$  as a parameter we tune to be arbitrarily small we can re-state as follows:

$$m^{\frac{2\omega-3}{3}-1/\sqrt{k}-\epsilon'}$$

where  $\epsilon'$  is some other arbitrarily small constant (a function of  $\epsilon$  implicitly). So, if someone claims they have update time

$$m^{\frac{2\omega-3}{3}-1/\sqrt{k}-\gamma}$$

for a fixed constant  $\gamma$  then we can tune  $\epsilon$  sufficiently small for that to give a contradiction from the  $k$ -clique hypothesis.

Now, if we state in terms of  $n = O(|V'|^2)$

$$n^{\left(\frac{2\omega-3-\delta}{6-2\delta}\right)\ell(1-o(1))-1}$$

we can use the same trick with  $\epsilon'$  to get the simplified version

$$n^{\left(\frac{2\omega-3}{6}-\epsilon'\right)\ell-1} = \Omega\left(n^{\left(\frac{2\omega-3}{6}-\epsilon'\right)\sqrt{2k}-1}\right)$$

which gives a lower bound of

$$n^{\frac{2\omega-3}{6}\sqrt{2k}-1-o(\sqrt{k})}$$

time per update. ◀

► **Lemma 23.** *Given a graph  $G$  let the procedure  $F$  be the process of selecting a uniformly random pair of vertices  $(u, v)$  and either deleting edge  $(u, v)$  if it exists or inserting  $(u, v)$  if it doesn't exist.*

*If we run the procedure  $F$   $\Theta\left(\binom{n}{2} \lg^2(n)\right)$  times on any graph  $G$  the resulting distribution over graphs has total variation distance at most  $n^{-\Omega(\lg(n))}$  from the distribution over Erdős-Rényi graphs.*

**Proof.** Let  $p = \binom{n}{2}^{-1}$ , the probability any given edge is selected to be flipped by  $F$ . Let  $s = \Theta\left(\binom{n}{2} \lg^2(n)\right)$ , the number of times we run  $F$ .

For any given edge  $(u, v)$  it is flipped a Binomially distributed number of times  $B(s, p)$ . The  $\Pr[B(s, p) \equiv 0 \pmod{2}] = 1/2 + (1-2p)^s/2$ , which is the probability that  $(u, v)$  stays an edge if it started with one and continues to have no edge if started without one. If this were an Erdős-Rényi graph the probability an edge exists is  $1/2$  iid. So on each edge the TVD from the probability of  $1/2$  is  $(1-2p)^s$ . Now note that  $(1-2p)^s = (1-2p)^{(2/(2p))\Theta(\lg^2(n))} < e^{-\Omega(\lg^2(n))} < n^{-\Omega(\lg(n))}$ .

We can now union bound over all edges to get a TVD between running the procedure  $F$   $\binom{n}{2} \lg^2(n)$  times and Erdős-Rényi graphs of at most  $n^{-\Omega(\lg(n))} = 2^{-\Omega(\lg^2(n))}$ . ◀

### B.3 Proof of Span Lower Bound in the Binary-Forking Model

► **Theorem 24.** *The span per update for dynamic boolean formula evaluation is  $\Omega(\log n)$  in the binary forking model.*

**Proof.** We first show the work lower bound for a single update (or, the time required per update), and then using the work lower bound we compute the span lower bound in the binary-forking model. From Theorem 12, we know that the lower bound of work per update is  $n^c$  where  $c = \Theta(1)$  for constant  $k$ .

In the binary forking model [12], it takes  $\Omega(\log t)$  time to launch  $t$  threads. This is due to spawning  $t$  threads in a binary tree fashion where there are  $t$  leaves in the binary tree and the height is  $\log t$ . The height  $\log t$  specifies the span of this launching process.

Let  $p$  be the number of processors used for the dynamic formula evaluation problem with  $n$  variables. The span  $T_\infty(n)$  in the binary forking model is lower bounded by  $n^c/p$  and  $\log p$ . Combining them, we get the following:

$$T_\infty(n) = \Omega(n^c/p + \log p)$$

Minimizing  $T_\infty(n)$  over values of  $p$ , we get  $T_\infty(n) = \Omega(\log n)$ . ◀



# Shortest Beer Path Queries in Outerplanar Graphs

Joyce Bacic ✉

Carleton University, Ottawa, Canada

Saeed Mehrabi

University of Massachusetts Lowell, MA, USA

Michiel Smid ✉

Carleton University, Ottawa, Canada

---

## Abstract

A *beer graph* is an undirected graph  $G$ , in which each edge has a positive weight and some vertices have a beer store. A *beer path* between two vertices  $u$  and  $v$  in  $G$  is any path in  $G$  between  $u$  and  $v$  that visits at least one beer store.

We show that any outerplanar beer graph  $G$  with  $n$  vertices can be preprocessed in  $O(n)$  time into a data structure of size  $O(n)$ , such that for any two query vertices  $u$  and  $v$ , (i) the weight of the shortest beer path between  $u$  and  $v$  can be reported in  $O(\alpha(n))$  time (where  $\alpha(n)$  is the inverse Ackermann function), and (ii) the shortest beer path between  $u$  and  $v$  can be reported in  $O(L)$  time, where  $L$  is the number of vertices on this path. Both results are optimal, even when  $G$  is a beer tree (i.e., a beer graph whose underlying graph is a tree).

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis

**Keywords and phrases** shortest paths, outerplanar graph

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.62

**Related Version** *Full Version*: <https://arxiv.org/abs/2110.15693>

**Funding** *Joyce Bacic*: Supported by an NSERC Undergraduate Student Research Award.

*Michiel Smid*: Supported by NSERC

## 1 Introduction

Imagine that you are going to visit a friend and, not wanting to show up empty handed, you decide to pick up some beer along the way. In this paper we determine the fastest way to go from your place to your friend’s place while stopping at a beer store to buy some drinks.

A *beer graph* is a undirected graph  $G = (V, E)$ , in which each edge  $(u, v)$  has a positive weight  $\omega(u, v)$  and some of the vertices are beer stores. For two vertices  $u$  and  $v$  of  $G$ , we define the *shortest beer path* from  $u$  to  $v$  to be the shortest (potentially non-simple) path that starts at  $u$ , ends at  $v$ , and visits at least one beer store. We denote this shortest path by  $SP_B(u, v)$ . The *beer distance*  $\text{dist}_B(u, v)$  between  $u$  and  $v$  is the weight of the path  $SP_B(u, v)$ , i.e., the sum of the edge weights on  $SP_B(u, v)$ .

Observe that even though the shortest beer path from  $u$  to  $v$  may be a non-simple path, it is always composed of two simple paths: the shortest path from  $u$  to a beer store and the shortest path from this same beer store to  $v$ . Thus, when looking at the shortest beer path problem, we often need to consider the shortest path between vertices. We denote the shortest path in  $G$  from  $u$  to  $v$  by  $SP(u, v)$  and we use  $\text{dist}(u, v)$  to denote the weight of this path. We also say that  $\text{dist}(u, v)$  is the *distance* between  $u$  and  $v$  in  $G$ .

To the best of our knowledge, the problem of computing shortest beer paths has not been considered before. Let  $s$  be a fixed source vertex of  $G$ . Recall that Dijkstra’s algorithm computes  $\text{dist}(s, v)$  for all vertices  $v$ , by maintaining a “tentative distance”  $\delta(v)$ , which is the weight of the shortest path from  $s$  to  $v$  computed so far. If we also maintain a “tentative beer distance”  $\delta_B(v)$  (which is the weight of the shortest beer path from  $s$  to  $v$  that has been found so far), then a modification of Dijkstra’s algorithm allows us to compute  $\text{dist}_B(s, v)$  for all vertices  $v$ , in  $O(|V| \log |V| + |E|)$  total time.



© Joyce Bacic, Saeed Mehrabi, and Michiel Smid;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 62; pp. 62:1–62:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



As far as we know, no non-trivial results are known for beer distance queries. In this case, we want to preprocess the beer graph  $G$  into a data structure, such that, for any two query vertices  $u$  and  $v$ , the shortest beer path  $\text{SP}_B(u, v)$ , or its weight  $\text{dist}_B(u, v)$ , can be reported.

## 1.1 Our Results

We present data structures that can answer shortest beer path queries in outerplanar beer graphs. Recall that a graph  $G$  is *outerplanar*, if  $G$  can be embedded in the plane, such that all vertices are on the outer face, and no two edges cross.

Our first result is stated in terms of the inverse Ackermann function. We use the definition as given in [3]: Let  $A_0(i) = i + 1$  and, for  $\ell \geq 0$ ,  $A_{\ell+1}(i) = A_\ell^{(i+1)}(i + 8)$ , where  $A_\ell^{(i+1)}$  is the function  $A_\ell$  iterated  $i + 1$  times. We define  $\alpha(m, n)$  to be the smallest value of  $\ell$  for which  $A_\ell(\lfloor m/n \rfloor) > n$ , and we define  $\alpha(n) = \alpha(n, n)$ .

► **Theorem 1.** *Let  $G$  be an outerplanar beer graph with  $n$  vertices. For any integer  $m \geq n$ , we can preprocess  $G$  in  $O(m)$  time into a data structure of size  $O(m)$ , such that for any two query vertices  $u$  and  $v$ , both  $\text{dist}(u, v)$  and  $\text{dist}_B(u, v)$  can be computed in  $O(\alpha(m, n))$  time.*

By taking  $m = n$ , both the preprocessing time and the space used are  $O(n)$ , and for any two query vertices  $u$  and  $v$ , both  $\text{dist}(u, v)$  and  $\text{dist}_B(u, v)$  can be computed in  $O(\alpha(n))$  time.

As another example, let  $\log^* n$  be the number of times the function  $\log$  must be applied, when starting with the value  $n$ , until the result is at most 1, and let  $\log^{**} n$  be the number of times the function  $\log^*$  must be applied, again starting with  $n$ , until the result is at most 1. Let  $m = n \log^{**} n$ . Since  $\alpha(m, n) = O(1)$ , we obtain a data structure with space and preprocessing time  $O(n \log^{**} n)$  that can answer both distance and beer distance queries in  $O(1)$  time.

As we mentioned before, beer distance queries have not been considered for any class of graphs. In fact, the only result on (non-beer) distance queries in outerplanar graphs that we are aware of is by Djidjev *et al.* [5]. They show that an outerplanar graph with  $n$  vertices can be preprocessed in  $O(n \log n)$  time into a data structure of size  $O(n \log n)$ , such that any distance query can be answered in  $O(\log n)$  time. Our result in Theorem 1 significantly improves their result.

We also show that the result in Theorem 1 is optimal for beer distance queries, even if  $G$  is a *beer tree* (i.e., a beer graph whose underlying graph is a tree). We do not know if the query time is optimal for (non-beer) distance queries.

Our second result is on reporting the shortest beer path between two query vertices.

► **Theorem 2.** *Let  $G$  be an outerplanar beer graph with  $n$  vertices. We can preprocess  $G$  in  $O(n)$  time into a data structure of size  $O(n)$ , such that for any two vertices  $u$  and  $v$ , the shortest beer path from  $u$  to  $v$  can be reported in  $O(L)$  time, where  $L$  is the number of vertices on this beer path.*

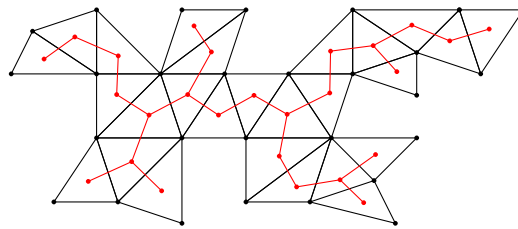
Observe that the query time in Theorem 2 does not depend on the number  $n$  of vertices of the graph. Again, we are not aware of any previous work on reporting shortest beer paths. Djidjev *et al.* [5] show that, after  $O(n \log n)$  preprocessing and using  $O(n \log n)$  space, the shortest (non-beer) path between two query vertices can be reported in  $O(\log n + L)$  time, where  $L$  is the number of vertices on the path.

## 1.2 Preliminaries and Organization

Throughout this paper, we only consider outerplanar beer graphs  $G$ . The number of vertices of  $G$  is denoted by  $n$ . It is well known that  $G$  has at most  $2n - 3$  edges. As in [5], we say that  $G$  satisfies the *generalized triangle inequality*, if for every edge  $(u, v)$  in  $G$ ,  $\text{dist}(u, v) = \omega(u, v)$ , i.e., the shortest path between  $u$  and  $v$  is the edge  $(u, v)$ .

The outerplanar graph  $G$  is called *maximal*, if adding an edge between any two non-adjacent vertices of  $G$  results in a graph that is not outerplanar. In this case, the number of edges is equal to  $2n - 3$ . A maximal outerplanar graph  $G$  is 2-connected, each internal face of  $G$  is a triangle and the outer face of  $G$  forms a Hamiltonian cycle. In such a graph, edges on the outer face will be referred to as *external* edges, where all other edges will be referred to as *internal* edges.

The *weak dual* of a maximal outerplanar graph  $G$  is the graph  $D(G)$  whose node set is the set of all internal faces of  $G$ , and in which  $(F, F')$  is an edge if and only if the faces  $F$  and  $F'$  share an edge in  $G$ ; see Figure 1. For simplicity, we will refer to  $D(G)$  as the dual of  $G$ . Observe that  $D(G)$  is a tree with  $n - 2$  nodes, each of which has degree at most three.



■ **Figure 1** A maximal outerplanar graph shown in black. Its dual is shown in red.

If  $H$  is a subgraph of the beer graph  $G$ , and  $u$  and  $v$  are vertices of  $H$ , then  $\text{dist}(u, v, H)$  and  $\text{dist}_B(u, v, H)$  denote the distance and beer distance between  $u$  and  $v$  in  $H$ , respectively. The shortest beer path in  $H$  between  $u$  and  $v$  must be entirely within  $H$ . Observe that we use the shorthand  $\text{dist}(u, v)$  for  $\text{dist}(u, v, G)$ , and  $\text{dist}_B(u, v)$  for  $\text{dist}_B(u, v, G)$ .

It will not be surprising that the algorithms for computing shortest beer paths use the dual  $D(G)$ . Thus, our algorithms will need some basic data structures on trees. These data structures will be presented in Section 2.

In Section 3, we will prove Theorem 1 for maximal outerplanar beer graphs. We also prove that the result in Theorem 1 is optimal, even for beer trees. The proof of Theorem 2, again for maximal outerplanar beer graphs, will be presented in Section 4. The extensions of these theorems to arbitrary outerplanar beer graphs will be given in the full version of this paper. The full version will also present an  $O(n)$ -time algorithm for computing the single-source shortest beer path tree for any given source vertex.

## 2 Query Problems on Trees

Our algorithms for computing beer shortest paths in an outerplanar graph  $G$  will use the dual of  $G$ , which is a tree. In order to obtain fast implementations of these algorithms, we need to be able to solve several query problems on this tree. In this section, we present all query problems that will be used in later sections.

► **Lemma 3.** *Let  $T$  be a tree with  $n$  nodes that is rooted at an arbitrary node. We can preprocess  $T$  in  $O(n)$  time, such that each of the following queries can be answered in  $O(1)$  time:*

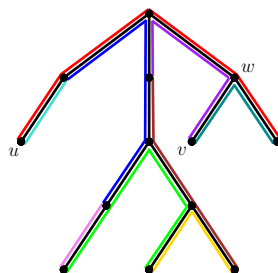
1. *Given a node  $u$  of  $T$ , return its level, denoted by  $\text{level}(u)$ , which is the number of edges on the path from  $u$  to the root.*
2. *Given two nodes  $u$  and  $v$  of  $T$ , report their lowest common ancestor, denoted by  $\text{LCA}(u, v)$ .*
3. *Given two nodes  $u$  and  $v$  of  $T$ , decide whether or not  $u$  is in the subtree rooted at  $v$ .*
4. *Given two distinct nodes  $u$  and  $v$  of  $T$ , report the second node on the path from  $u$  to  $v$ .*
5. *Given three nodes  $u, v$ , and  $w$ , decide whether or not  $w$  is on the path between  $u$  and  $v$ .*

**Proof.** The first claim follows from the fact that by performing an  $O(n)$ -time pre-order traversal of  $T$ , we can compute  $level(u)$  for each node  $u$ . A proof of the second claim can be found in Harel and Tarjan [6] and Bender and Farach-Colton [2]. The third claim follows from the fact that  $u$  is in the subtree rooted at  $v$  if and only if  $LCA(u, v) = v$ . A proof of the fourth claim can be found in Chazelle [4, Lemma 15]. The fifth claim follows from the following observations. Assume that  $u$  is in the subtree rooted at  $v$ . Then  $w$  is on the path between  $u$  and  $v$  if and only if  $LCA(u, w) = w$  and  $w$  is in the subtree rooted at  $v$ . The case when  $v$  is in the subtree rooted at  $u$  is symmetric. Assume that  $LCA(u, v) \notin \{u, v\}$ . Then  $w$  is on the path between  $u$  and  $v$  if and only if  $w$  is on the path between  $u$  and  $LCA(u, v)$  or  $w$  is on the path between  $v$  and  $LCA(u, v)$ . ◀

## 2.1 Closest-Colour Queries in Trees

Let  $T$  be a tree with  $n$  nodes and let  $\mathcal{C}$  be a set of *colours*. For each colour  $c$  in  $\mathcal{C}$ , we are given a path  $P_c$  in  $T$ . Even though these paths may share nodes, each node of  $T$  belongs to at most a constant number of paths. This implies that the total size of all paths  $P_c$  is  $O(n)$ . We assume that each node  $u$  of  $T$  stores the set of all colors  $c$  such that  $u$  is on the path  $P_c$ .

In a *closest-colour query*, we are given two nodes  $u$  and  $v$  of  $T$ , and a colour  $c$ , such that  $u$  is on the path  $P_c$ . The answer to the query is the node on  $P_c$  that is closest to  $v$ . Refer to Figure 2 for an illustration.



■ **Figure 2** A tree  $T$  and a collection of coloured paths. For a query with nodes  $u$  and  $v$ , and color “red”, the answer is the node  $w$ .

► **Lemma 4.** *After an  $O(n)$ -time preprocessing, we can answer any closest-colour query in  $O(1)$  time.*

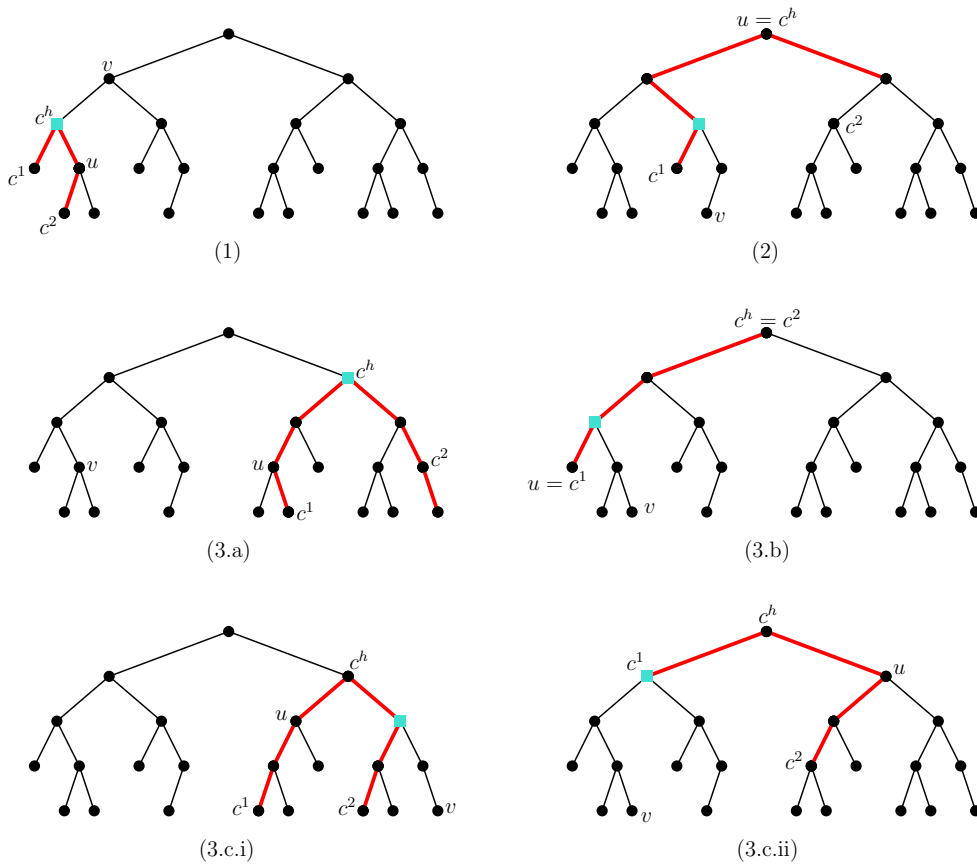
**Proof.** We take an arbitrary node of  $T$  and make it the root. Then we preprocess  $T$  such that each of the queries in Lemma 3 can be answered in  $O(1)$  time.

For each colour  $c$ , let  $c^1$  and  $c^2$  be the end nodes of the path  $P_c$ , and let  $c^h$  be the highest node on  $P_c$  in the tree (i.e., the node on  $P_c$  that is closest to the root). With each node of  $P_c$ , we store pointers to  $c^1$ ,  $c^2$ , and  $c^h$ .

Since each node of  $T$  is in a constant number of coloured paths, we can compute the pointers for all the coloured paths in  $O(n)$  total time.

The query algorithm does the following. Let  $u$  and  $v$  be two nodes of  $T$ , and let  $c$  be a colour such that  $u$  is on the  $c$ -coloured path  $P_c$ .

If  $u = v$  or  $v$  is also on  $P_c$ , then we return the node  $v$ . From now on, assume that  $u \neq v$  and  $v$  is not on  $P_c$ . Below, we consider all possible cases, which are illustrated in Figure 3.



■ **Figure 3** Illustrating all possible cases in the proof of Lemma 4. The path  $P_c$  is red and the blue square indicates the node that is returned by the closest-colour query.

1. If  $\text{LCA}(u, v) = v$ , then  $u$  is in the subtree rooted at  $v$ . In this case, we return  $c^h$ , the highest  $c$ -coloured node.
2. Assume that  $\text{LCA}(u, v) = u$ . Then  $v$  is in the subtree rooted at  $u$ . The closest  $c$ -coloured node to  $v$  is either  $\text{LCA}(v, c^1)$  or  $\text{LCA}(v, c^2)$ . Since  $v$  is lower than  $u$  in the tree, we know that the closest  $c$ -coloured node to  $v$  is at  $\text{level}(u)$  or greater. If  $\text{level}(\text{LCA}(v, c^1)) > \text{level}(\text{LCA}(v, c^2))$ , then  $\text{LCA}(v, c^1)$  is lower in the tree and closer to  $v$ , so we return  $\text{LCA}(v, c^1)$ . Otherwise,  $\text{LCA}(v, c^2)$  is lower in  $T$  or equal to both  $\text{LCA}(v, c^1)$  and  $u$ , so we return  $\text{LCA}(v, c^2)$ .
3. Assume that  $\text{LCA}(u, v) \neq u$  and  $\text{LCA}(u, v) \neq v$ . Then  $u$  and  $v$  are in different subtrees of  $\text{LCA}(u, v)$ .
  - a. If  $\text{level}(c^h) > \text{level}(\text{LCA}(u, v))$ , then we return  $c^h$ .
  - b. If  $\text{level}(c^h) < \text{level}(\text{LCA}(u, v))$ , then we return  $\text{LCA}(u, v)$ .
  - c. Assume that  $\text{level}(c^h) = \text{level}(\text{LCA}(u, v))$ . Observe that exactly one end node of the  $c$ -coloured path is in the subtree rooted at  $u$ .
    - i. If  $c^1$  is in the subtree rooted at  $u$ , then we return  $\text{LCA}(v, c^2)$ .
    - ii. If  $c^2$  is in the subtree rooted at  $u$ , then we return  $\text{LCA}(v, c^1)$ .

Using Lemma 3, each of these case takes  $O(1)$  time. Therefore, the entire query algorithm takes  $O(1)$  time. ◀

## 2.2 Path-Sum Queries in Trees

Let  $(W, \oplus)$  be a semigroup. Thus,  $W$  is a set and  $\oplus : W \times W \rightarrow W$  is an associative binary operator. We assume that for any two elements  $s$  and  $s'$  in  $W$ , the value of  $s \oplus s'$  can be computed in  $O(1)$  time.

Let  $T$  be a tree with  $n$  nodes in which each edge  $e$  stores a value  $s(e)$ , which is an element of  $W$ . For any two distinct nodes  $u$  and  $v$  in  $T$ , we define their *path-sum*  $\text{PS}(u, v)$  as follows: Let  $e_1, e_2, \dots, e_k$  be the edges on the path in  $T$  between  $u$  and  $v$ . Then we define  $\text{PS}(u, v) = \oplus_{i=1}^k s(e_i)$ .

Chazelle [4] considers the problem of preprocessing the tree  $T$ , such that for any two distinct query nodes  $u$  and  $v$ , the value of  $\text{PS}(u, v)$  can be reported. (See also Alon and Schieber [1], Thorup [8], and Chan *et al.* [3].) Chazelle's result is stated in terms of the inverse Ackermann function; see Section 1.1.

► **Lemma 5.** *Let  $T$  be a tree with  $n$  nodes in which each edge stores an element of the semigroup  $(W, \oplus)$ . For any integer  $m \geq n$ , we can preprocess  $T$  in  $O(m)$  time into a data structure of size  $O(m)$ , such that any path-sum query can be answered in  $O(\alpha(m, n))$  time.*

► **Remark 6.** Assume that  $(W, \oplus)$  is the semigroup, where  $W$  is the set of all real numbers and the operator  $\oplus$  takes the minimum of its arguments. In this case, we will refer to a query as a *path-minimum query*. For this semigroup, the result of Lemma 5 is optimal: Any data structure that can be constructed in  $O(m)$  time has worst-case query time  $\Omega(\alpha(m, n))$ . To prove this, assume that we can answer any query in  $o(\alpha(m, n))$  time. Then the on-line minimum spanning tree verification problem on a tree with  $n$  vertices and  $m \geq n$  queries can be solved in  $o(m \cdot \alpha(m, n))$  time, by performing a path-maximum query for the endpoints of each edge  $e$  and checking that the weight of  $e$  is larger than the path-maximum. This contradicts the lower bound for this problem proved by Pettie [7].

## 3 Beer Distance Queries in Maximal Outerplanar Graphs

Let  $G$  be a maximal outerplanar beer graph with  $n$  vertices that satisfies the generalized triangle inequality. We will show how to preprocess  $G$ , such that for any two vertices  $u$  and  $v$ , the weight,  $\text{dist}_B(u, v)$ , of a shortest beer path between  $u$  and  $v$  can be reported. Our approach will be to define a special semigroup  $(W, \oplus)$ , such that each element of  $W$  “contains” certain distances and beer distances. With each edge of the dual  $D(G)$ , we will store one element of the set  $W$ . As we will see later, a beer distance query can then be reduced to a path-sum query in  $D(G)$ . Thus, by applying the results of Section 2.2, we will obtain a proof of Theorem 1.

We will need the first claim in the following lemma. The second claim will be used in Section 4.

► **Lemma 7.** *Consider the beer graph  $G$  as above.*

1. *In  $O(n)$  total time, we can compute  $\text{dist}_B(u, u)$  for each vertex  $u$  of  $G$ , and  $\text{dist}_B(u, v)$  for each edge  $(u, v)$  in  $G$ .*
2. *After an  $O(n)$ -time preprocessing of  $G$ , we can report,*
  - a. *for any query edge  $(u, v)$  of  $G$ , the shortest beer path between  $u$  and  $v$  in  $O(L)$  time, where  $L$  is the number of vertices on this path,*
  - b. *for any query vertex  $u$  of  $G$ , the shortest beer path from  $u$  to itself in  $O(L)$  time, where  $L$  is the number of vertices on this path.*

**Proof.** We choose an arbitrary face  $R$  of  $G$  and make it the root of  $D(G)$ . Let  $(u, v)$  be any edge of  $G$ . This edge divides  $G$  into two outerplanar subgraphs, both of which contain  $(u, v)$  as an edge. Let  $G_{uv}^R$  be the subgraph that contains the face  $R$ , and let  $G_{uv}^{-R}$  denote the other subgraph. Note that if  $(u, v)$  is an external edge, then  $G_{uv}^R = G$  and  $G_{uv}^{-R}$  consists of the single edge  $(u, v)$ . By the generalized triangle inequality, the shortest beer path between  $u$  and  $v$  is completely in  $G_{uv}^R$  or completely in  $G_{uv}^{-R}$ . The same is true for the shortest beer path from  $u$  to itself. Thus, for each edge  $(u, v)$  of  $G$ ,

$$\text{dist}_B(u, v) = \min(\text{dist}_B(u, v, G_{uv}^R), \text{dist}_B(u, v, G_{uv}^{-R})),$$

$$\text{dist}_B(u, u) = \min(\text{dist}_B(u, u, G_{uv}^R), \text{dist}_B(u, u, G_{uv}^{-R})).$$

By performing a post-order traversal of  $D(G)$ , we can compute  $\text{dist}_B(u, v, G_{uv}^R)$  and  $\text{dist}_B(u, u, G_{uv}^{-R})$  for all edges  $(u, v)$ , in  $O(n)$  total time. After these values have been computed, we perform a pre-order traversal of  $D(G)$  and obtain  $\text{dist}_B(u, v, G_{uv}^R)$  and  $\text{dist}_B(u, u, G_{uv}^R)$ , again for all edges  $(u, v)$ , in  $O(n)$  total time. The details will be given in the full version of this paper. ◀

In the rest of this section, we assume that all beer distances in the first claim of Lemma 7 have been computed.

For any two distinct internal faces  $F$  and  $F'$  of  $G$ , let  $Q_{F,F'}$  be the union of the two sets

$$\{(u, v, \text{dist}(u, v), D) \mid u \text{ is a vertex of } F, v \text{ is a vertex of } F'\}$$

and

$$\{(u, v, \text{dist}_B(u, v), \text{BD}) \mid u \text{ is a vertex of } F, v \text{ is a vertex of } F'\},$$

where the “bits”  $D$  and  $\text{BD}$  indicate whether the tuple represents a distance or a beer distance. In words,  $Q_{F,F'}$  is the set of all shortest path distances and all shortest beer distances between a vertex in  $F$  and a vertex in  $F'$ . Since each internal face has three vertices, the set  $Q_{F,F'}$  has exactly 18 elements.

▶ **Observation 8.** *Let  $u$  and  $v$  be vertices of  $G$ , and let  $F$  and  $F'$  be internal faces that contain  $u$  and  $v$  as vertices, respectively.*

1. *If  $F = F'$ , then we can determine both  $\text{dist}(u, v)$  and  $\text{dist}_B(u, v)$  in  $O(1)$  time.*
2. *If  $F \neq F'$  and we are given the set  $Q_{F,F'}$ , then we can determine both  $\text{dist}(u, v)$  and  $\text{dist}_B(u, v)$  in  $O(1)$  time.*

**Proof.** First assume that  $F = F'$ . If  $u = v$ , then  $\text{dist}(u, v) = 0$  and  $\text{dist}_B(u, v)$  has been precomputed. If  $u \neq v$ , then  $(u, v)$  is an edge of  $G$  and, thus,  $\text{dist}(u, v) = \omega(u, v)$  and  $\text{dist}_B(u, v)$  has been precomputed.

Assume that  $F \neq F'$ . If we know the set  $Q_{F,F'}$ , then we can find  $\text{dist}(u, v)$  and  $\text{dist}_B(u, v)$  in  $O(1)$  time, because these two distances are in  $Q_{F,F'}$ . ◀

In the rest of this section, we will show that Lemma 5 can be used to compute the set  $Q_{F,F'}$  for any two distinct internal faces  $F$  and  $F'$ .

▶ **Lemma 9.** *For any edge  $(F, F')$  of  $D(G)$ , the set  $Q_{F,F'}$  can be computed in  $O(1)$  time.*

**Proof.** Let  $u$  be a vertex of  $F$  and let  $v$  be a vertex of  $F'$ . Consider the subgraph  $G[F, F']$  of  $G$  that is induced by the four vertices of  $F$  and  $F'$ ; this subgraph has five edges. By the generalized triangle inequality,  $\text{dist}(u, v) = \text{dist}(u, v, G[F, F'])$ . Thus,  $\text{dist}(u, v)$  can be computed in  $O(1)$  time.

We now show how  $\text{dist}_B(u, v)$  can be computed in  $O(1)$  time. If  $u = v$  or  $(u, v)$  is an edge of  $G$ , then  $\text{dist}_B(u, v)$  has been precomputed. Assume that  $u \neq v$  and  $(u, v)$  is not an edge of  $G$ . Let  $w$  and  $w'$  be the two vertices that are shared by  $F$  and  $F'$ . Since any path in  $G$  between  $u$  and  $v$  contains at least one of  $w$  and  $w'$ ,  $\text{dist}_B(u, v)$  is the minimum of

1.  $\text{dist}_B(u, w) + \omega(w, v)$ ,
2.  $\omega(u, w) + \text{dist}_B(w, v)$ ,
3.  $\text{dist}_B(u, w') + \omega(w', v)$ ,
4.  $\omega(u, w') + \text{dist}_B(w', v)$ .

Since  $(u, w)$ ,  $(w, v)$ ,  $(u, w')$ , and  $(w', v)$  are edges of  $G$ , all terms in these four sums have been precomputed. Therefore,  $\text{dist}_B(u, v)$  can be computed in  $O(1)$  time.

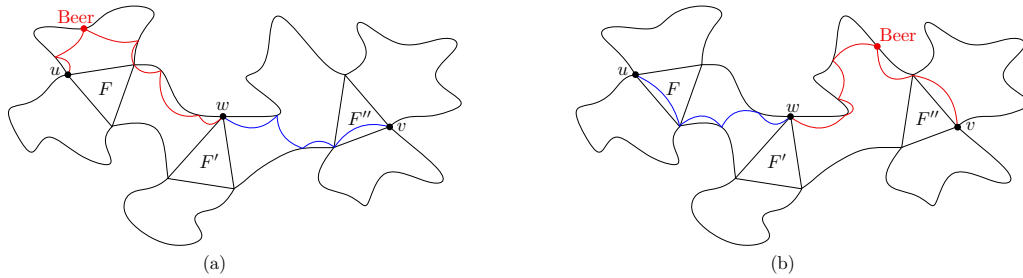
We have shown that each of the 18 elements of  $Q_{F, F'}$  can be computed in  $O(1)$  time. Therefore, this entire set can be computed in  $O(1)$  time. ◀

► **Lemma 10.** *Let  $F$ ,  $F'$ , and  $F''$  be three pairwise distinct internal faces of  $G$ , such that  $F'$  is on the path in  $D(G)$  between  $F$  and  $F''$ . If we are given the sets  $Q_{F, F'}$  and  $Q_{F', F''}$ , then the set  $Q_{F, F''}$  can be computed in  $O(1)$  time.*

**Proof.** Let  $u$  be a vertex of  $F$  and let  $v$  be a vertex of  $F''$ . Since  $G$  is an outerplanar graph, any path in  $G$  between  $u$  and  $v$  must contain at least one vertex of  $F'$ . It follows that

$$\text{dist}(u, v) = \min\{\text{dist}(u, w) + \text{dist}(w, v) \mid w \text{ is a vertex of } F'\}.$$

Thus, since  $(u, w, \text{dist}(u, w), D) \in Q_{F, F'}$  and  $(w, v, \text{dist}(w, v), D) \in Q_{F', F''}$ , the value of  $\text{dist}(u, v)$  can be computed in  $O(1)$  time.



■ **Figure 4** Any beer path from  $u$  to  $v$  contains at least one vertex of  $F'$ . In (a), we consider the shortest beer path from  $u$  to  $w$ , followed by the shortest path from  $w$  to  $v$ . In (b), we consider the shortest path from  $u$  to  $w$ , followed by the shortest beer path from  $w$  to  $v$ .

By a similar argument,  $\text{dist}_B(u, v)$  is equal to (refer to Figure 4)

$$\min\{\min(\text{dist}_B(u, w) + \text{dist}(w, v), \text{dist}(u, w) + \text{dist}_B(w, v)) : w \text{ is a vertex of } F'\}.$$

All values  $\text{dist}(u, w)$ ,  $\text{dist}(w, v)$ ,  $\text{dist}_B(u, w)$ , and  $\text{dist}_B(w, v)$  are encoded in the sets  $Q_{F, F'}$  and  $Q_{F', F''}$ . Therefore, we can compute  $\text{dist}_B(u, v)$  in  $O(1)$  time.

Thus, since each of the 18 elements of  $Q_{F, F''}$  can be computed in  $O(1)$  time, the entire set can be computed in  $O(1)$  time. ◀

We define

$$W = \{Q_{F, F'} \mid F \text{ and } F' \text{ are distinct internal faces of } G\} \cup \{\perp\},$$

where  $\perp$  is a special symbol. We define the operator  $\oplus : W \times W \rightarrow W$  in the following way.



1. If  $F$  and  $F'$  are distinct internal faces of  $G$ , then  $Q_{F,F'} \oplus Q_{F,F'} = Q_{F,F'}$ .
2. If  $F$ ,  $F'$ , and  $F''$  are pairwise distinct internal faces of  $G$  such that  $F'$  is on the path in  $D(G)$  between  $F$  and  $F''$ , then  $Q_{F,F'} \oplus Q_{F',F''} = Q_{F,F''}$ .
3. In all other cases, the operator  $\oplus$  returns  $\perp$ .

It is not difficult to verify that  $\oplus$  is associative, implying that  $(W, \oplus)$  is a semigroup. By Lemma 9, we can compute  $Q_{F,F'}$  for all edges  $(F, F')$  of  $D(G)$ , in  $O(n)$  total time.

Recall from Lemma 3 that, after an  $O(n)$ -time preprocessing, we can decide in  $O(1)$  time, for any three internal faces  $F$ ,  $F'$ , and  $F''$  of  $G$ , whether  $F'$  is on the path in  $D(G)$  between  $F$  and  $F''$ . Therefore, using Lemma 10, the operator  $\oplus$  takes  $O(1)$  time to evaluate for any two elements of  $W$ .

Finally, let  $F$  and  $F'$  be two distinct internal faces of  $G$ , and let  $F = F_0, F_1, F_2, \dots, F_k = F'$  be the path in  $D(G)$  between  $F$  and  $F'$ . Then  $Q_{F,F'} = \oplus_{i=0}^{k-1} Q_{F_i, F_{i+1}}$ . Thus, if we store with each edge of the tree  $D(G)$ , the corresponding element of the semigroup, then computing  $Q_{F,F'}$  becomes a path-sum query as in Section 2.2.

To summarize, all conditions to apply Lemma 5 are satisfied. As a result, we have proved Theorem 1 for maximal outerplanar graphs that satisfy the generalized triangle inequality.

### The Result in Theorem 1 is Optimal

In Section 2.2, see also Remark 6, we have seen path-minimum queries in a tree, in which each edge  $e$  stores a real number  $s(e)$ . In such a query, we are given two distinct nodes  $u$  and  $v$ , and have to return the smallest value  $s(e)$  among all edges  $e$  on the path between  $u$  and  $v$ . Lemma 5 gives a trade-off between the preprocessing and query times when answering such queries.

Let  $D$  be an arbitrary data structure that answers beer distance queries in any beer tree. Let  $P(n)$ ,  $S(n)$ , and  $Q(n)$  denote the preprocessing time, space, and query time of  $D$ , respectively, when the beer tree has  $n$  nodes. We will show that  $D$  can be used to answer path-minimum queries.

Consider an arbitrary tree  $T$  with  $n$  nodes, such that each edge  $e$  stores a real number  $s(e)$ . We may assume without loss of generality that  $0 < s(e) < 1$  for each edge  $e$  of  $T$ .

By making an arbitrary node the root of  $T$ , the number of edges on the path in  $T$  between two nodes  $u$  and  $v$  is equal to

$$\text{level}(u) + \text{level}(v) - 2 \cdot \text{level}(\text{LCA}(u, v)).$$

Thus, by Lemma 3, after an  $O(n)$ -time preprocessing, we can compute the number of edges on this path in  $O(1)$  time.

We create a beer tree  $T'$  as follows. Initially,  $T'$  is a copy of  $T$ . For each edge  $e = (u, v)$  of  $T'$ , we introduce a new node  $x_e$  and replace  $e$  by two edges  $(u, x_e)$  and  $(v, x_e)$ ; we assign a weight of 1 to each of these two edges. In the current tree  $T'$ , none of the nodes has a beer store. For every node  $x_e$  in  $T'$ , we introduce a new node  $x'_e$ , add the edge  $(x_e, x'_e)$ , assign a weight of  $s(e)$  to this edge, and make  $x'_e$  a beer store. Finally, we construct the data structure  $D$  for the resulting beer tree  $T'$ . Since  $T'$  has  $n + 2(n - 1) = 3n - 2$  nodes, it takes  $P(3n - 2) + O(n)$  time to construct  $D$  from the input tree  $T$ . Moreover, the amount of space used is  $S(3n - 2) + O(n)$ .

Let  $u$  and  $v$  be two distinct nodes in the original tree  $T$ , let  $\pi$  be the path in  $T$  between  $u$  and  $v$ , and let  $\ell$  be the number of edges on  $\pi$ . The corresponding path  $\pi'$  in  $T'$  between  $u$  and  $v$  has weight  $2\ell$ .

For any edge  $e$  of  $T$ , let  $\pi'_e$  be the beer path in  $T'$  that starts at  $u$ , goes to  $x_e$ , then goes to  $x'_e$  and back to  $x_e$ , and continues to  $v$ .

If  $e$  is an edge of  $\pi$ , then the weight of  $\pi'_e$  is equal to  $2\ell + 2 \cdot s(e)$ , which is less than  $2\ell + 2$ . On the other hand, if  $e$  is an edge of  $T$  that is not on  $\pi$ , then the weight of  $\pi'_e$  is at least  $2\ell + 2 + 2 \cdot s(e)$ , which is larger than  $2\ell + 2$ . It follows that the shortest beer path in  $T'$  between  $u$  and  $v$  visits the beer store  $x'_e$ , where  $e$  is the edge on  $\pi$  for which  $s(e)$  is minimum.

Thus, by computing  $\ell$  and querying  $D$  for the beer distance in  $T'$  between  $u$  and  $v$ , we obtain the smallest value  $s(e)$  among all edges  $e$  on the path in  $T$  between  $u$  and  $v$ . The query time is  $Q(3n - 2) + O(1)$ .

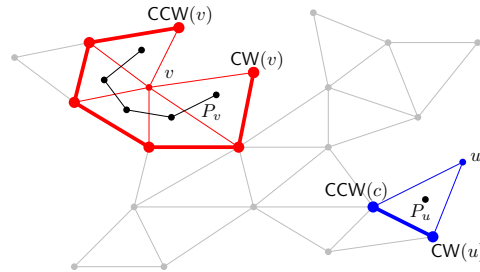
By combining this reduction with Remark 6, it follows that the result of Theorem 1 is optimal.

#### 4 Reporting Shortest Beer Paths in Maximal Outerplanar Graphs

Let  $G$  be a maximal outerplanar beer graph with  $n$  vertices that satisfies the generalized triangle inequality. In this section, we show that, after an  $O(n)$ -time preprocessing, we can report, for any two query vertices  $s$  and  $t$ , the shortest beer path  $\text{SP}_B(s, t)$  from  $s$  to  $t$ , in  $O(L)$  time, where  $L$  is the number of vertices on this path. As before,  $D(G)$  denotes the dual of  $G$ .

► **Observation 11.** *Let  $v$  be a vertex of  $G$ . The faces of  $G$  containing  $v$  form a path of nodes in  $D(G)$ .*

Define  $P_v$  to be the path in  $D(G)$  formed by the faces of  $G$  containing the vertex  $v$ . Let  $G[P_v]$  be the subgraph of  $G$  induced by the faces of  $G$  containing  $v$ . Note that  $G[P_v]$  has a fan shape. Let  $\text{CW}(v)$  denote the clockwise neighbor of  $v$  in  $G[P_v]$  and let  $\text{CCW}(v)$  denote the counterclockwise neighbor of  $v$  in  $G[P_v]$ . We will refer to the clockwise path from  $\text{CW}(v)$  to  $\text{CCW}(v)$  in  $G[P_v]$  as the  $v$ -chain and denote it by  $\rho_v$ . (Refer to Figure 5.)



■ **Figure 5** A maximal outerplanar graph  $G$ . The subgraphs  $G[P_v]$  and  $G[P_u]$  are shown in red and blue, respectively. Both the  $v$ -chain  $\rho_v$  and the  $u$ -chain  $\rho_u$  are shown in bold. Both paths  $P_v$  and  $P_u$  are shown in black. Observe that  $P_u$  is a single node.

► **Lemma 12.** *After an  $O(n)$ -time preprocessing, we can answer the following queries, for any three query vertices  $v$ ,  $u$ , and  $w$ , such that both  $u$  and  $w$  are on the  $v$ -chain  $\rho_v$ :*

1. Report the weight  $\text{dist}(u, w, \rho_v)$  of the path from  $u$  to  $w$  along  $\rho_v$  in  $O(1)$  time.
2. Report the path  $\text{SP}(u, w, \rho_v)$  from  $u$  to  $w$  along  $\rho_v$  in  $O(L)$  time, where  $L$  is the number of vertices on this path.

**Proof.** For any vertex  $v$  and any vertex  $u$  on  $\rho_v$ , we store the weight of the path from  $u$  to  $\text{CW}(v)$  along  $\rho_v$ . Observe that

$$\text{dist}(u, w, \rho_v) = |\text{dist}(u, \text{CW}(v), \rho_v) - \text{dist}(w, \text{CW}(v), \rho_v)|.$$

Any exterior edge in  $G$  is in exactly one chain and any interior edge in  $G$  is in exactly two chains. Thus, the sum of the number of edges on each chain is proportional to the number of edges of  $G$ , which is  $O(n)$ . ◀

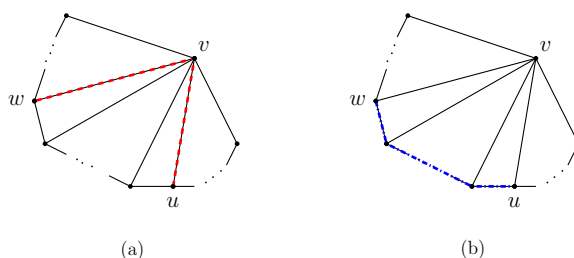
► **Lemma 13.** *After an  $O(n)$ -time preprocessing, we can answer the following query in  $O(1)$  time: Given three query vertices  $v$ ,  $u$ , and  $w$ , such that both  $u$  and  $w$  are vertices of  $G[P_v]$ , report  $\text{dist}(u, w)$ , i.e., the distance between  $u$  and  $w$  in  $G$ .*

**Proof.** We get the following cases; the correctness follows from the generalized triangle inequality:

1. If  $u = w$  then  $\text{dist}(u, w) = 0$ .
2. If  $u = v$  then  $(u, w)$  is an edge and we return  $\omega(u, w)$ . Similarly if  $w = v$ , we return  $\omega(u, w)$ .
3. Otherwise  $u$  and  $w$  are both on  $\rho_v$  and we return  $\min(\text{dist}(u, w, \rho_v), \omega(u, v) + \omega(v, w))$ . ◀

► **Lemma 14.** *After an  $O(n)$ -time preprocessing, we can report, for any three vertices  $v$ ,  $u$ , and  $w$ , such that both  $u$  and  $w$  are vertices of  $G[P_v]$ ,  $\text{SP}(u, w)$  in  $O(L)$  time, where  $L$  is the number of vertices on the path.*

**Proof.** Using Lemma 13, we can determine in  $O(1)$  if the shortest path from  $u$  to  $w$  goes through  $v$  or follows the  $v$ -chain  $\rho_v$ . (Refer to Figure 6). If it goes through  $v$ , then  $\text{SP}(u, w) = (u, v, w)$ . Otherwise,  $\text{SP}(u, w)$  takes the path along  $\rho_v$  and by Lemma 12, we can find this path in  $O(L)$  time. ◀



■ **Figure 6** Two possible cases for the shortest path between  $u$  and  $w$ : (a) it goes through vertex  $v$  (shown in dashed red), or (b) it goes through the vertices of the  $v$ -chain between  $u$  and  $w$  (shown in dashed blue).

► **Lemma 15.** *After an  $O(n)$ -time preprocessing, we can report, for any three vertices  $v$ ,  $u$  and  $w$ , such that both  $u$  and  $w$  are vertices of  $G[P_v]$ , the beer distance  $\text{dist}_B(u, w)$  in  $O(1)$  time. The corresponding shortest beer path  $\text{SP}_B(u, w)$  can be reported in  $O(L)$  time, where  $L$  is the number of vertices on the path.*

**Proof.** Recall from Lemma 7 that we can compute  $\text{dist}_B(u, v)$  for every edge  $(u, v)$  in  $G$ , and  $\text{dist}_B(v, v)$  for every vertex  $v$  in  $G$ , in  $O(n)$  time.

Let  $\rho_v = (\text{CW}(v) = u_1, u_2, \dots, u_N = \text{CCW}(v))$ . Let  $A_v[\ ]$  be an array of size  $N - 1$ . For  $i = 1, \dots, N - 1$ , we set  $A_v[i] = \text{dist}_B(u_i, u_{i+1}) - \omega(u_i, u_{i+1})$ . Recall that by the generalized triangle inequality,  $\omega(u_i, u_{i+1}) = \text{dist}(u_i, u_{i+1})$ . Therefore,  $A[i]$  holds the difference between the weights of the shortest path from  $u_i$  to  $u_{i+1}$  and the shortest beer path from  $u_i$  to  $u_{i+1}$ . After preprocessing the array  $A_v[\ ]$  in  $O(N)$  time, we can conduct range minimum queries

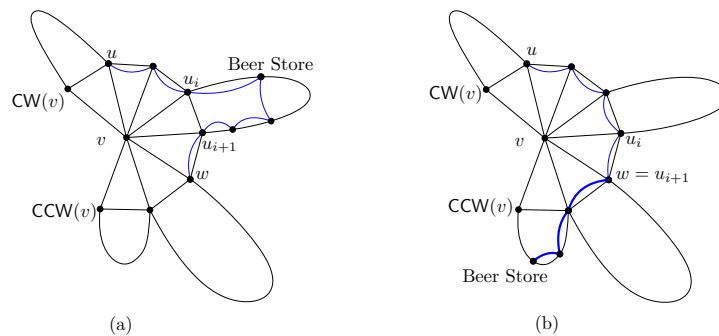
## 62:12 Shortest Beer Path Queries in Outerplanar Graphs

in  $O(1)$  time. (Bender and Farach-Colton [2] show that these queries are equivalent to LCA-queries in the Cartesian tree of the array.) Thus, for each  $v$ -chain of  $N$  nodes, we spend  $O(N)$  time processing the  $v$ -chain. Since every edge is in at most two chains, processing all  $v$ -chains takes  $O(n)$  time and space.

Given two vertices  $u$  and  $w$  of  $G[P_v]$ , we determine the beer distance  $\text{dist}_B(u, w)$  as follows:

1. If  $u = w$  then  $\text{dist}_B(u, w)$  has already been computed by Lemma 7.
2. If  $u = v$  or  $w = v$ , then there is an edge from  $v$  to the other vertex. Thus,  $\text{dist}_B(u, w)$  has already been computed by Lemma 7.
3. Otherwise,  $u, w$  and  $v$  are three distinct vertices. Assume without loss of generality that  $w$  is clockwise from  $u$  on the  $v$ -chain. We take the minimum of the following two cases:
  - a. The shortest beer path from  $u$  to  $w$  that goes through  $v$ . Since a beer store must be visited before or after  $v$ , this beer path has a weight of  $\min(\text{dist}_B(u, v) + \omega(v, w), \omega(u, v) + \text{dist}_B(v, w))$ .
  - b. The shortest beer path through the vertices of the  $v$ -chain. Note that this beer path will visit each vertex on the  $v$ -chain between  $u$  and  $w$ , but may go off the  $v$ -chain to visit a beer store. On  $\text{SP}_B(u, w)$ , there is one pair of vertices,  $u_i$  and  $u_{i+1}$ , such that a beer path is taken between  $u_i$  and  $u_{i+1}$ , and  $u_i$  and  $u_{i+1}$  are adjacent on the  $v$ -chain; refer to Figure 7. The shortest path is taken between all other pairs of adjacent vertices on the  $v$ -chain. From Lemma 12, we can compute  $\text{dist}(u, w, \rho_v)$  in  $O(1)$  time. The shortest beer path through the vertices of the  $v$ -chain has a weight of  $\text{dist}(u, w, \rho_v) + A_v[i]$ , where  $A_v[i]$  is the additional distance needed to visit a beer store between  $u_i$  and  $u_{i+1}$ . Let  $u$  be the  $j^{\text{th}}$  vertex on  $\rho_v$  and let  $w$  be the  $k^{\text{th}}$  vertex in  $\rho_v$ . Then  $A_v[i]$  is the minimum value in  $A_v[j, \dots, k - 1]$ . We can determine  $A_v[i]$  in constant time using a range minimum query.

Note that in case 1 and case 2,  $\text{SP}_B(u, w)$  can be constructed in  $O(L)$  time by Lemma 7. For case 3 (a) let  $p = (u, v, w)$  and for case 3 (b) let  $p = \text{SP}(u, w, \rho_v)$ . Let  $u_i, u_{i+1}$  be the pair of adjacent vertices on  $p$  between which a beer path was taken. Using Lemma 7 we can find  $\text{SP}_B(u_i, u_{i+1})$  in  $O(L)$  time. We obtain  $\text{SP}_B(u, w)$  by replacing the edge  $(u_i, u_{i+1})$  in  $p$  with  $\text{SP}_B(u_i, u_{i+1})$ . ◀



■ **Figure 7** Both figures show a shortest beer path from  $u$  to  $w$  through the vertices on the  $v$ -chain. Thicker edges on the blue beer path are edges that are traversed twice; once in each direction.

### Answering Shortest Beer Path Queries

Recall that, for any vertex  $v$  of  $G$ ,  $P_v$  denotes the path in  $D(G)$  formed by the faces of  $G$  containing  $v$ . Moreover,  $G[P_v]$  denotes the subgraph of  $G$  induced by these faces.

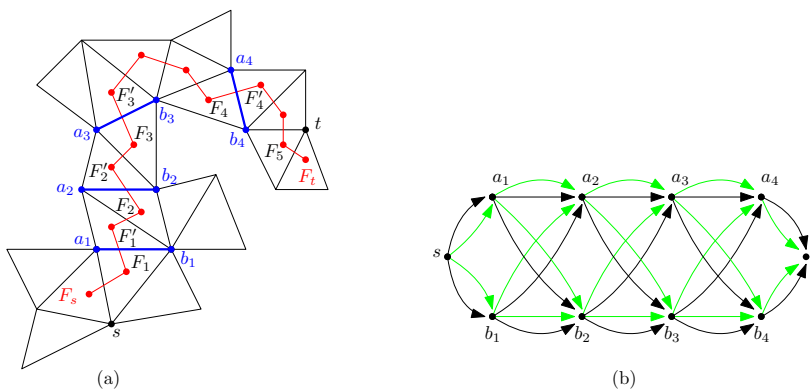
Consider two query vertices  $s$  and  $t$  of  $G$ . Our goal is to compute the shortest beer path  $SP_B(s, t)$ .

Let  $F_s$  and  $F_t$  be arbitrary faces containing  $s$  and  $t$ , respectively. If  $t$  is in  $G[P_s]$  then, by Lemma 15, we can construct  $SP_B(s, t)$  in  $O(L)$  time. For the remainder of this section, we assume that  $t$  is not in  $G[P_s]$ . To find  $SP_B(s, t)$ , we start by constructing a directed acyclic graph (DAG),  $H$ . In this DAG, vertices will be arranged in columns of constant size, and all edges go from left to right between vertices in adjacent columns. In  $H$ , each column will contain one vertex that is on  $SP_B(s, t)$ . First we will construct  $H$  and then we will show how we can use  $H$  to construct  $SP_B(s, t)$ . The entire construction is illustrated in Figure 8.

► **Observation 16.** Any interior edge  $(a, b)$  of  $G$  splits  $G$  into two subgraphs such that if  $s$  is in one subgraph and  $t$  is in the other, then any path in  $G$  from  $s$  to  $t$  must visit at least one of  $a$  and  $b$ .

Let  $P$  be the unique path between  $F_s$  and  $F_t$  in  $D(G)$ . Consider moving along  $P$  from  $F_s$  to  $F_t$ . Let  $F_1$  be the node on  $P_s$  that is closest to  $F_t$ , and let  $F'_1$  be the successor of  $F_1$  on  $P$ . Note that, by Lemmas 3 and 4, we can find  $F_1$  and  $F'_1$  in  $O(1)$  time.<sup>1</sup> Let  $e_1 = (a_1, b_1)$  be the edge in  $G$  shared by the faces  $F_1$  and  $F'_1$ . Since  $SP_B(s, t)$  must visit both of these faces, by Observation 16, at least one of  $a_1$  or  $b_1$  is on the shortest beer path.

We place  $s$  in the first column of  $H$  and  $a_1$  and  $b_1$  in the second column of  $H$ . We then add two directed edges from  $s$  to  $a_1$ , one with weight  $\text{dist}(s, a_1)$  and the other with weight  $\text{dist}_B(s, a_1)$ . Similarly, we add two directed edges from  $s$  to  $b_1$  with weights  $\text{dist}(s, b_1)$  and  $\text{dist}_B(s, b_1)$ .



■ **Figure 8** An outerplanar graph  $G$  (a) and the DAG  $H$  constructed for the shortest beer path query from  $s$  to  $t$  (b). The path  $P$  from  $F_s$  to  $F_t$  is shown in red. Each edge  $e_i = (a_i, b_i)$  such that  $e_i$  is shared by  $F_i$  and  $F'_i$  is shown in blue. The green edges of  $H$  represent the beer edges.

When  $i \geq 2$  we construct the  $(i + 1)^{th}$  column of  $H$  in the following way. Let  $e_{i-1} = (a_{i-1}, b_{i-1})$  be the edge shared by the faces  $F_{i-1}$  and  $F'_{i-1}$ . The  $i^{th}$  column of  $H$  contains the vertices  $a_{i-1}$  and  $b_{i-1}$ . Note that  $F'_{i-1}$  is in both  $P_{b_{i-1}}$  and  $P_{a_{i-1}}$ . Using Lemma 4, we find the node  $F_i^b$  on  $P_{b_{i-1}}$  that is closest to  $F_t$ . If the vertex  $a_{i-1}$  is not in  $F_i^b$ , then we let  $F_i = F_i^b$ . Otherwise, we let  $F_i$  be the node on  $P_{a_{i-1}}$  that is closest to  $F_t$ .

<sup>1</sup> To apply Lemma 4, we consider each vertex of  $G$  to be a colour. For each vertex  $v$  of  $G$ , the  $v$ -coloured path in the tree  $D(G)$  is the path  $P_v$ . The face  $F_1$  is the answer to the closest-colour query with nodes  $F_s$  and  $F_t$  and colour  $s$ .

## 62:14 Shortest Beer Path Queries in Outerplanar Graphs

If  $t$  is not a vertex of  $F_i$ , then let  $F'_i$  be the node that follows  $F_i$  on  $P$ ; we find  $F'_i$  using Lemma 3. Let  $e_i = (a_i, b_i)$  be the edge of  $G$  shared by the faces  $F_i$  and  $F'_i$ . In the  $(i+1)^{th}$  column, we place  $a_i$  and  $b_i$ . For each  $u \in \{a_{i-1}, b_{i-1}\}$  and each  $v \in \{a_i, b_i\}$  we add two directed edges  $(u, v)$  to the DAG, one with weight  $\text{dist}(u, v)$  and the other with weight  $\text{dist}_B(u, v)$ . If  $F_i$  is in  $P_{a_{i-1}}$ , all these vertices are in  $G[P_{a_{i-1}}]$ ; otherwise,  $F_i$  is in  $P_{b_{i-1}}$ , and all these vertices are in  $G[P_{b_{i-1}}]$ . Thus, by Lemmas 13 and 15, we can find the distances and beer distances to assign to these edges in constant time.

If  $t$  is in  $F_i$ , then in the  $(i+1)^{th}$  column we only place the vertex  $t$ . In this case, for each  $u \in \{a_{i-1}, b_{i-1}\}$ , we add two directed edges  $(u, t)$  to the DAG with weights  $\text{dist}(u, t)$  and  $\text{dist}_B(u, t)$ . At this point we are done constructing  $H$ .

We define a *beer edge* to be an edge of  $H$  that was assigned a weight of a beer path during the construction of  $H$ . We find the beer distance from  $s$  to  $t$  in  $G$  using the following dynamic programming approach in  $H$ .

Let  $M$  denote the number of columns in  $H$ . For  $i = 3, \dots, M$  and for all  $u$  in the  $i^{th}$  column of  $H$ , compute

$$\text{dist}_B(s, u) = \min \begin{cases} \text{dist}_B(s, a_{i-2}) + \text{dist}(a_{i-2}, u) \\ \text{dist}(s, a_{i-2}) + \text{dist}_B(a_{i-2}, u) \\ \text{dist}_B(s, b_{i-2}) + \text{dist}(b_{i-2}, u) \\ \text{dist}(s, b_{i-2}) + \text{dist}_B(b_{i-2}, u) \end{cases}$$

and

$$\text{dist}(s, u) = \min \begin{cases} \text{dist}(s, a_{i-2}) + \text{dist}(a_{i-2}, u) \\ \text{dist}(s, b_{i-2}) + \text{dist}(b_{i-2}, u) \end{cases}$$

The vertices  $a_{i-2}$  and  $b_{i-2}$  occur in the  $(i-1)^{th}$  column. Thus,  $\text{dist}_B(s, a_{i-2})$ ,  $\text{dist}_B(s, b_{i-2})$ ,  $\text{dist}(s, a_{i-2})$ , and  $\text{dist}(s, b_{i-2})$  will be computed before computing the values for the  $i^{th}$  column. We get  $\text{dist}(a_{i-2}, u)$ ,  $\text{dist}_B(a_{i-2}, u)$ ,  $\text{dist}(b_{i-2}, u)$  and  $\text{dist}_B(b_{i-2}, u)$  from the weights of the DAG-edges between the  $(i-1)^{th}$  and  $i^{th}$  columns of  $H$ .

By keeping track of which expression produced  $\text{dist}_B(s, u)$  and  $\text{dist}(s, u)$ , we can backwards reconstruct the shortest beer path in the DAG. Knowing the shortest beer path in the DAG enables us to construct the corresponding beer path in  $G$  as follows.

1. Define  $P_{st}$  to be an empty path.
2. For each edge  $(w, v)$  of the shortest beer path in the DAG.
  - a. If  $(w, v)$  was a beer edge, let  $P_{wv} = \text{SP}_B(w, v)$ , which can be constructed in time proportional to its number of vertices via Lemma 15.
  - b. Otherwise, let  $P_{wv} = \text{SP}(w, v)$  which can be constructed in time proportional to its number of vertices as seen in Lemma 14.

Let  $P_{st} = P_{st} \cup P_{wv}$ .

3. Return  $P_{st}$ , which is equal to  $\text{SP}_B(w, v)$ .

Let  $L$  denote the number of vertices on  $\text{SP}_B(s, t)$ . In order for the above query algorithm to take  $O(L)$  time, the size of the DAG must be  $O(L)$ . The following three lemmas will show this to be true.

► **Lemma 17.** For  $2 \leq i < M - 1$ ,  $F_i$  contains either  $a_{i-1}$  or  $b_{i-1}$ , but not both.

**Proof.** Recall that we defined  $F_i^b$  to be the last node on  $P$  that is also on  $P_{b_{i-1}}$ . We similarly define  $F_i^a$  to be the last node on  $P$  that is also on  $P_{a_{i-1}}$ . From the way we choose  $F_i$ ,  $F_i$  is either  $F_i^b$  or  $F_i^a$ . We only choose  $F_i = F_i^b$  after having checked that  $a_{i-1}$  is not in  $F_i^b$ ; thus in this case we can be sure that  $F_i$  only contains  $b_{i-1}$ .

Assume for the purpose of contradiction that we choose  $F_i = F_i^a$  and  $b_{i-1}$  is also in  $F_i$ . Let the third vertex of  $F_i$  be  $c$ . Let the face on  $P$  immediately following  $F_i$  be  $F'_i$ . The edge shared by  $F_i$  and  $F'_i$  is either  $(b_{i-1}, c)$  or  $(a_{i-1}, c)$ . If  $(b_{i-1}, c)$  is the shared edge, then  $F'_i$  is a face closer to  $F_t$  that contains  $b_{i-1}$  and not  $a_{i-1}$ , so we would have chosen  $F_i = F_i^b$ , which is a contradiction. Otherwise,  $(a_{i-1}, c)$  is the edge shared by  $F_i$  and  $F'_i$ , which implies that there is a face containing  $a_{i-1}$  closer to  $F_t$  in  $P$  than  $F_i^a$ , which contradicts the definition of  $F_i^a$ . ◀

► **Lemma 18.** *Every vertex of  $G$  appears in at most one column of  $H$ .*

**Proof.** Since  $(a_1, b_1)$  is an edge shared by both the last face of  $P$  containing  $s$  and the first face of  $P$  that does not contain  $s$  it is not possible for either of these vertices to be the vertex  $s$ . Thus,  $s$  will only be represented by the vertex in the first column of  $H$ . By stopping the construction of  $H$  as soon as we add a vertex representing  $t$ , we ensure that  $H$  only contains one vertex corresponding to the vertex  $t$  in  $G$ .

For  $2 \leq i \leq M - 2$ , consider the vertex  $a_{i-1}$  in  $G$  represented by a vertex in the  $i^{\text{th}}$  column of  $H$ . If  $F_i = F_i^a$  then by definition of  $F_i^a$ ,  $F'_i$  does not contain  $a_{i-1}$ . Since  $(a_i, b_i)$  is an edge of  $F'_i$ ,  $a_i \neq a_{i-1}$  and  $b_i \neq a_{i-1}$ . Because the face  $F'_i$  is closer to  $F_t$  than  $F_i^a$ ,  $a_{i-1}$  is not a vertex on any of the faces on the path from  $F'_i$  to  $F_t$ . Thus, subsequent columns of  $H$  will not contain vertices representing the vertex  $a_{i-1}$  in  $G$ .

If  $F_i = F_i^b$  then by Lemma 17,  $a_{i-1}$  is not in  $F_i$  and since  $(a_i, b_i)$  is an edge of  $F_i$ ,  $a_i \neq a_{i-1}$  and  $b_i \neq a_{i-1}$ . Because  $F_i$  is a face on  $P$  closer to  $F_t$  than  $F_{i-1}$  (a face that contains  $a_{i-1}$ ) it follows from Observation 11 that none of the faces on  $P$  from  $F_{i-1}$  to  $F_t$  will have the vertex  $a_{i-1}$  on their face and, thus,  $a_{i-1}$  will not be represented by vertices in subsequent columns of  $H$ .

By switching the roles of  $a_{i-1}$  with  $b_{i-1}$  in the above reasoning we can see that this also holds for  $b_{i-1}$ . ◀

► **Lemma 19.** *The number of vertices and edges of  $H$  is  $O(L)$ .*

**Proof.** By Observation 16 and Lemma 18, the number of columns of  $H$  is at most  $L$ . Since each column has at most two vertices, each of which having at most four outgoing edges, the total number of vertices and edges of  $H$  is  $O(L)$ . ◀

Observe that the total preprocessing time is  $O(n)$ . For two query vertices  $s$  and  $t$ , the DAG,  $H$ , can be constructed in  $O(L)$  time. Finally, the dynamic programming algorithm on  $H$  takes  $O(L)$  time. Thus, we have proved Theorem 2 for maximal outerplanar graphs that satisfy the generalized triangle inequality.

---

## References

- 1 N. Alon and B. Schieber. Optimal preprocessing for answering on-line product queries. Technical Report 71/87, Tel-Aviv University, 1987.
- 2 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94, Berlin, 2000. Springer-Verlag.
- 3 T. M. Chan, M. He, J. I. Munro, and G. Zhou. Succinct indices for path minimum, with applications. *Algorithmica*, 78(2):453–491, 2017.
- 4 B. Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.



## 62:16 Shortest Beer Path Queries in Outerplanar Graphs

- 5 H. Djidjev, G. E. Pantziou, and C. D. Zaroliagis. Computing shortest paths and distances in planar graphs. In *Automata, Languages and Programming, 18th International Colloquium, ICALP91*, volume 510 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 1991.
- 6 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.
- 7 S. Pettie. An inverse-Ackermann type lower bound for online minimum spanning tree verification. *Combinatorica*, 26(2):207–230, 2006.
- 8 M. Thorup. Parallel shortcutting of rooted trees. *Journal of Algorithms*, 32:139–159, 1997.

# Space-Efficient Algorithms for Reachability in Directed Geometric Graphs

Sujoy Bhore  

Indian Institute of Science Education and Research, Bhopal, India

Rahul Jain  

Fernuniversität in Hagen, Germany

---

## Abstract

The problem of graph REACHABILITY is to decide whether there is a path from one vertex to another in a given graph. In this paper, we study the REACHABILITY problem on three distinct graph families - intersection graphs of Jordan regions, unit contact disk graphs (penny graphs), and chordal graphs. For each of these graph families, we present space-efficient algorithms for the REACHABILITY problem.

For intersection graphs of Jordan regions, we show how to obtain a “good” *vertex separator* in a space-efficient manner and use it to solve the REACHABILITY in polynomial time and  $O(m^{1/2} \log n)$  space, where  $n$  is the number of Jordan regions, and  $m$  is the total number of crossings among the regions. We use a similar approach for chordal graphs and obtain a polynomial time and  $O(m^{1/2} \log n)$  space algorithm, where  $n$  and  $m$  are the number of vertices and edges, respectively. However, for unit contact disk graphs (penny graphs), we use a more involved technique and obtain a better algorithm. We show that for every  $\epsilon > 0$ , there exists a polynomial time algorithm that can solve REACHABILITY in an  $n$  vertex directed penny graph, using  $O(n^{1/4+\epsilon})$  space. We note that the method used to solve penny graphs does not extend naturally to the class of geometric intersection graphs that include arbitrary size cliques.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** Reachability, Geometric intersection graphs, Space-efficient algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.63

**Related Version** *Full Version:* <https://arxiv.org/abs/2101.05235> [7]

## 1 Introduction

Given a directed graph  $G = (V, E)$  and two of its vertices  $s$  and  $t$ , the problem of REACHABILITY is to decide if there exists a path from  $s$  to  $t$  in  $G$ . REACHABILITY is one of the fundamental problems in theoretical computer science, and for directed and undirected graphs the problem is known to be complete for the classes NL and L, respectively (see [26, 28]). The famous open question  $L \stackrel{?}{=} NL$  essentially asks if there exists a deterministic log-space algorithm for REACHABILITY or not. Note that REACHABILITY can be solved in  $\Theta(n \log n)$  space and optimal time by using standard graph traversal algorithms such as DFS and BFS. Furthermore, it is known that this problem can be solved in  $\Theta(\log^2 n)$ -space and  $n^{\Theta(\log n)}$  time [32].

In the realm of space-efficient algorithms, the primary objective is to optimize the space-complexity of an algorithm while maintaining a polynomial-time bound. Wigderson asked in his survey of REACHABILITY problems [33], whether there is an algorithm for REACHABILITY that runs in  $O(n^{1-\epsilon})$  space (for some  $\epsilon > 0$ ) and polynomial time. Barnes et al. [6] partially answered this question and showed that REACHABILITY on general graphs can be solved in polynomial time and  $O(n/2^{\Theta(\sqrt{\log n})})$  space. This result is followed by numerous works on various restricted graph families.



© Sujoy Bhore and Rahul Jain;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 63; pp. 63:1–63:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Asano and Doerr [3] presented an algorithm for grid graphs that uses  $O(n^{1/2+\varepsilon})$ -space, for any small  $\varepsilon > 0$ . Imai et al. [23] achieved the similar space bound for planar graphs. Later, Asano et al. [4] improved the space bound to  $\tilde{O}(n^{1/2})$  for planar graphs. Recently, this bound has been improved to  $O(n^{1/4+\varepsilon})$  for grid graphs [24]. Besides, Chakraborty et al. [10] studied REACHABILITY for graphs with higher genus and gave an  $\tilde{O}(n^{2/3}g^{1/3})$ -space bound algorithm, and an  $\tilde{O}(n^{2/3})$  space algorithm for H minor-free graphs.

For layered planar graphs, Chakraborty and Tewari [11] showed that there is an  $O(n^\varepsilon)$ -space and polynomial algorithm. Gupta et al. [20] showed that given a pair  $\{s, t\}$  and an embedding of an  $O(\log n)$  genus graphs, REACHABILITY from  $s$  to  $t$  in  $G$  can be decided *unambiguously* in logspace (i.e. REACHABILITY is in the class UL).

Tree-decomposition and associated *treewidth* is an important notion of the graphs. Many problems which are computationally hard on general graphs are efficiently solvable on graphs of bounded Treewidth [1]. Graphs of small Treewidth also have small *vertex-separators*, which is a small set of vertices of the graph, removal of which divides the graph into pieces whose size are at most a fraction of the original graph. Recently, Jain and Tewari [25] showed that given an  $n$  vertex directed graph of treewidth  $w$  along with its tree decomposition, there exists an algorithm for REACHABILITY problem that runs polynomial time and  $O(w \log n)$  space. They achieved this result by using the vertex separator of small Treewidth graphs. In fact, they formalized the connection between *Vertex Separator* and REACHABILITY problems, which has several consequences. For the sake of completeness, we state their result below.

► **Theorem 1** ([25]). *Let  $\mathcal{G}$  be a class of directed graphs and  $w : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  be a function. If there exist an  $O(w(n, m) \log n)$  space and polynomial time algorithm, that given a graph  $G \in \mathcal{G}$  of  $n$  vertices and  $m$  edges, and a set  $U$  of  $V(G)$ , outputs a separator of  $U$  in underlying undirected graph of  $G^1$  of size  $O(w(n, m))$ , then there exists an algorithm to decide REACHABILITY in  $G$  that uses  $O(w(n, m) \log n)$  space and polynomial time.*

In this paper, we study the REACHABILITY problem on directed geometric graph families. Many important graph families can be described as intersection graphs of more restricted types of set families. More often than not, the geometric intersection graph families provide additional geometric structures that help to generate efficient algorithms. Many problems that are NP-complete on general graphs are tractable on geometric intersection graphs [13, 22]. Indeed, such advantages have been exploited for space-efficient algorithms as well. We refer to the survey on geometric algorithms with limited work-space [5].

In this work, we make progress towards answering the question raised by Wigderson in his survey [33]. We primarily design and use the *Vertex separator* of some geometric graphs space-efficiently. Previously, vertex separators in the context of geometric graphs have been studied by Fox and Pach [16], who established several geometric extensions of the famous Lipton-Tarjan separator theorem [27]. Recently, Carmi et al. [8], and Hoffmann et al. [21] established improved bounds on the size of separators for some restricted classes of geometric graphs.

## 1.1 Our Contribution

We study the REACHABILITY problem on three different graph families – intersection Graphs of Jordan Regions, Unit Contact Disk Graphs (Penny Graphs) and Chordal Graphs.

---

<sup>1</sup> For a directed graph  $G$ , its *underlying undirected graph* is the graph formed by removing the orientation of all the edges.

First, in Section 2, we show that given a collection of Jordan regions, there exists a polynomial time algorithm that computes a separator of size  $O(m^{1/2})$  using  $O(m^{1/2} \log n)$  space, where  $n$  is the number of Jordan regions and  $m$  is the total number of crossings among those regions. Then, by combining this with Theorem 1, we note that REACHABILITY on directed intersection graphs of Jordan regions can be solved in polynomial time and  $O(m^{1/2} \log n)$  space.

In Section 3, we present a space-efficient algorithm for REACHABILITY on penny graphs that uses  $O(n^{1/4+\epsilon})$  space and polynomial time. Since penny graphs are a subclass of planar graphs, REACHABILITY can be solved for penny graphs in  $O(n^{1/2} \log n)$  space. However, to reduce the space complexity, we use an involved technique. First, we use the axis-parallel separator of Carmi et al. [8] repeatedly to form rectangular subdivision such that each cell of a subdivision contains a bounded size subgraph of the input graph. Then using these subdivisions, we construct a smaller auxiliary graph that preserves REACHABILITY information. Finally, using a notion of *pseudo-separator*, we solve REACHABILITY in this auxiliary graph in a space-efficient manner. Note that, there exists a  $O(n^{1/4+\epsilon})$  space and polynomial time algorithm for reachability in grid graphs [24]. Since grid graphs are a subclass of penny graphs, our result is a generalization and improvement of previously known results.

Finally, in Section 4, we adopt the algorithm of [19] for REACHABILITY on chordal graphs to provide a space-efficient and polynomial time algorithm.

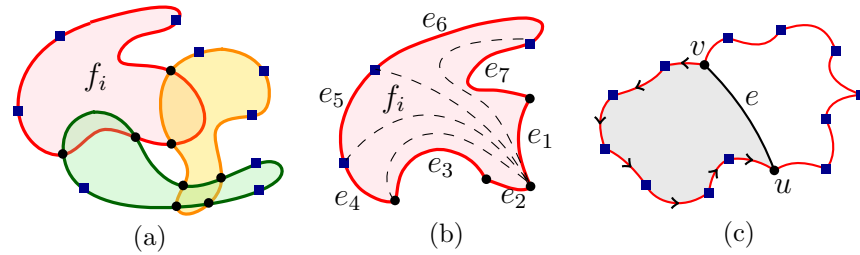
Even though chordal graphs seem to be an exception in the context of geometric graph families, we note that there exists work representing chordal graphs as a subfamily of string graphs [12]. Moreover, it is known that chordal graphs can be characterized as intersection graphs of sub-trees of a tree [18], and interval graphs are a subfamily of chordal graphs.

## 1.2 Preliminaries

Throughout the text, we denote the set  $\{1, 2, \dots, n\}$  as  $[n]$ . For a graph  $G = (V, E)$  and a subset  $U \subseteq V$ ,  $G[U]$  denotes the subgraph induced on  $U$ . Given a graph  $G = (V, E)$  with vertex set  $V$  and edge set  $E$ , a weight function  $w : V \rightarrow R_{\geq 0}$  is a non-negative function on  $V$  such that the sum of the weights is 1. For any subset  $S \subseteq V$ , the weight  $w(S)$  is defined to be  $\sum_{v \in S} w(v)$ . A separator in a graph  $G$  with respect to a weight function  $w$  is a subset  $S \subseteq V$  of vertices such that there is a partition  $V = S \cup V_1 \cup V_2$  such that  $w(V_1), w(V_2) \leq 2/3$  and there are no edges between  $V_1$  and  $V_2$ . If the weight function is not specified, it is assumed that  $w(v) = \frac{1}{|V|}$ , for every vertex  $v$ . We refer to Arora and Barak [2] for a basic understanding of the model and terminologies for space-efficient algorithms.

## 2 Intersection Graphs of Jordan Regions

In this section, we study the REACHABILITY problem on the intersection graphs of Jordan regions. Let  $\mathcal{C}$  be a set such that each element  $C$  of  $\mathcal{C}$  is a simply connected compact region in a plane bounded by a closed Jordan curve. Let  $G(\mathcal{C})$  be an intersection graph on  $\mathcal{C}$ , where two distinct elements  $C_1, C_2 \in \mathcal{C}$  are adjacent if and only if their intersection is not empty. Additionally, each edge of  $G(\mathcal{C})$  is a directed edge. A Jordan region  $A$  *contains* another Jordan region  $B$  if  $A \subseteq \text{int}(B)$ , where  $\text{int}(B)$  is the interior of  $B$ . We assume that no point is a boundary point of three elements of  $\mathcal{C}$ , and each element of  $\mathcal{C}$  intersects at least one other element of  $\mathcal{C}$ . Further, we may assume that the number of such intersection points is finite. In [16], Fox and Pach showed the existence of a  $O(m^{1/2})$ -size separator on a collection of Jordan regions, where  $m$  is the total number of crossings points in the boundary of Jordan regions.



■ **Figure 1** (a) An illustration of the planar embedding of the curves in  $\mathcal{C}$ . The black disk points are the intersection points from set  $A(\mathcal{C})$ , and the blue square points the three extra points from set  $B(\mathcal{C})$ . (b) An illustration of a face in the embedding and its corresponding planar triangulation. (c) Traversal of the left-face of the edge  $e = (u, v)$ .

For the purpose of computation, we work with those classes of Jordan regions that can be represented compactly, i.e., the set of  $n$  Jordan regions can be represented by  $\text{poly}(n)$  bits. Furthermore, we assume that basic operations, such as determining the intersection point of two Jordan regions and outputting a constant number of points of given Jordan regions, can be performed in log-space. We prove the following theorem about the separator. Note that any *vertex separator* of an intersection graph does not rely on the direction of the edges. Hence for ease of explanation, we drop the directions from the input graph  $G(\mathcal{C})$ .

► **Theorem 2.** *Let  $\mathcal{C}$  be a collection of Jordan regions, and let  $G(\mathcal{C})$  be an intersection graph on  $\mathcal{C}$ . Let  $w$  be a weight function on  $\mathcal{C}$ . There exists a polynomial time algorithm that takes as an input the set  $\mathcal{C}$  and outputs a separator of  $G(\mathcal{C})$  with respect to  $w$ , of size  $O(m^{1/2})$  using  $O(m^{1/2} \log n)$  space.*

We assume that the sum of the weights of the given Jordan regions is one. Let  $H(\mathcal{C})$  be the set of *heavy* regions in  $\mathcal{C}$  whose weight is more than  $1/m^{1/2}$ , and let  $L(\mathcal{C})$  be set of regions in  $\mathcal{C}$  that are involved<sup>2</sup> in at least  $\frac{1}{3}m^{1/2}$  containments with other elements of  $\mathcal{C}$ . Let  $I(\mathcal{C}) = \mathcal{C} \setminus (H(\mathcal{C}) \cup L(\mathcal{C}))$ . We define a planar graph  $G_P(\mathcal{C})$ . The vertex set of this graph is the union of two subsets, i.e.,  $A(\mathcal{C}) \cup B(\mathcal{C})$ , where  $A(\mathcal{C})$  is the set of all intersection points that lie on the boundary of at least one element of  $I(\mathcal{C})$  and  $B(\mathcal{C})$  is a collection of points not in  $A(\mathcal{C})$  such that the boundary of each  $C \in I(\mathcal{C})$  contains precisely three points in  $B(\mathcal{C})$ . There exists an edge between two vertices of  $G_P(\mathcal{C})$  if and only if there are consecutive points along the boundary of an element of  $I(\mathcal{C})$ ; see Figure 1(a).

To see that the defined graph is planar, we see that we have added a vertex at every point of intersection of the boundary of the given Jordan curves. We can now draw the edges along the boundary. We proceed with the following lemmas.

► **Lemma 3.** *There exists a log-space algorithm that takes as an input a set of Jordan regions  $\mathcal{C}$  and outputs a graph  $G_P(\mathcal{C})$ .*

**Proof.** First of all, note that there exists a log-space subroutine that on an input Jordan region  $C$  of  $\mathcal{C}$  determines if  $C$  is in  $L(\mathcal{C})$ . To do this, the subroutine iterates over all Jordan regions in  $\mathcal{C}$  and calculates the total number of containment with which  $C$  is involved. The weight of each Jordan region comes with the input. Hence it is enough for us to determine if a Jordan region  $C$  of  $\mathcal{C}$  belongs to the set  $I(\mathcal{C})$ . In order to construct the planar graph, we essentially need to know  $I(\mathcal{C})$ . Thereby concluding the proof of the Lemma. ◀

<sup>2</sup> We say that a region  $C_1$  is involved in a *containment* with a region  $C_2$  if either  $C_1$  contains  $C_2$  or  $C_2$  contains  $C_1$ .

Next, we triangulate  $G_P(\mathcal{C})$ , and denote the triangulated planar graph by  $G_T(\mathcal{C})$ . This triangulation can be obtained in log-space using the following lemma

► **Lemma 4.** *There exists an algorithm that takes a planar graph in the input and returns a triangulated planar graph in log-space.*

**Proof.** Consider a planar graph  $G_P$ . The embedding of a planar graph can be computed in log-space [15]. Note that an edge  $e_1 = (u_1, v_1)$  in  $G_P$  is part of two faces in the planar embedding of the graph. We call them the left and the right face, respectively. We explain how to traverse the left face of  $e_1$  in log-space. In order to do this, we start by traversing  $e_1$  to one of its endpoint (say  $v_1$ ) and take the edge clockwise next to  $e_1$ , that is incident on  $v_1$ . Let  $e_2$  be such an edge; see Figure 1. We continue the traversal along the edge  $e_3$ , which is the edge clockwise next to  $e_2$  from the endpoint  $v_2$  of  $e_2$ , and so on. In general, if we reach at the vertex  $v_i$  using the edge  $e_i$ , we continue along the edge clockwise next to  $e_i$  from  $v_i$ . Clearly, by following this procedure, we can traverse the boundary of the face. In order to prove this lemma, it is sufficient to show that given two vertices  $u_l$  and  $v_l$  of the input planar graph  $G_P$ , whether an edge  $e_l$  can be added between them in log-space as part of the triangulation. We assume that the vertices of the input graph  $G_P$  are indexed by an integer from 1 to  $k$ , for some  $k \in [n]$ . For each edge  $e$  that is incident with  $u$ , we first traverse the left face of  $e$  and see if (i)  $v$  is present in that face, (ii) either  $u$  or  $v$  is the lowest-indexed vertex of that face. There is a triangulated edge between  $u$  and  $v$  if and only if both these conditions are true for any such edge  $e$ ; see Figure 1(b). This concludes the proof of the lemma. ◀

Let  $d(\mathcal{C})$  be the number of points on the boundary of  $\mathcal{C}$  that belong to the vertex set of  $G_T(\mathcal{C})$ . For a vertex  $v$  in  $G_T(\mathcal{C})$ , we define a new weight function  $\text{weight}(v)$  as follows:

**Case 1:**  $v$  is part of two boundaries  $C_1$  and  $C_2$  of  $\mathcal{C}$ ,  $\text{weight}(v) = \frac{w(C_1)}{d(C_1)} + \frac{w(C_2)}{d(C_2)}$ .

**Case 2:**  $v$  is in the boundary of only one element  $C_1$  of  $\mathcal{C}$ ,  $\text{weight}(v) = \frac{w(C_1)}{d(C_1)}$ .

Fox and Pach [16] used the idea to find a cycle-separator in this triangulated graph  $G_T(\mathcal{C})$  and used it to construct a separator of the original geometric intersection graph. Instead, we will use the result of Imai et al. [23] to obtain such a separator in a space-efficient manner.

► **Lemma 5** ([23]). *Let  $G$  be a triangulated planar graph. There exists a polynomial time algorithm that uses  $O(\sqrt{n} \log n)$  space to output a separator of  $G$  of size  $O(\sqrt{n})$ .*

Now, consider the set of regions of  $\mathcal{C}$  whose boundary contains at least one of the points of the separator returned by the algorithm of Imai et al. on  $G_T(\mathcal{C})$ . We denote these regions by  $\text{sep}(\mathcal{C})$ . In the following, we show that this set is indeed the required separator. Moreover, this set can be calculated within the required space-time bounds.

► **Lemma 6.** *The set  $\text{sep}(\mathcal{C})$  is a separator of the intersection graph of  $\mathcal{C}$*

**Proof.** First of all, note that  $G_T(\mathcal{C})$  is a triangulated planar graph. Let  $S$  be the separator of this graph returned by the algorithm of Imai et al. [23]. For a triangulated graph, we observe that the separator  $S$  is a cycle. Let  $V_0$  be the set of all elements of  $\mathcal{C}$  that are either not in  $I(\mathcal{C})$ , or whose boundary contains a vertex of  $S$ . Let  $K_1$  and  $K_2$  be the set of vertices that are *inside* and *outside* the cycle  $S$ , respectively. Let  $V_i$  (for  $i \in \{1, 2\}$ ) be the set of elements of  $I(\mathcal{C})$ , such that all vertex of  $V(G_T(\mathcal{C}))$  which are on its boundary belong to the component  $K_i$ . It is easy to see that  $V_0$ ,  $V_1$ , and  $V_2$  are pairwise disjoint sets, and their union

is  $\mathcal{C}$ . If  $V_0$  has a weight of at least  $1/3$ , then we see that it act as a trivial separator. Hence, for the rest of this proof, we assume that the weight of  $V_0$  is less than  $1/3$ . Thus, we can also see that the weight of  $V_i$  is at most  $2/3$ . It only remains to show that there is no edge between a vertex of  $V_1$  and a vertex of  $V_2$ .

Let us assume, w.l.o.g., that the weight of  $V_2$  is greater than the weight of  $V_1$  and hence greater than  $1/3$ . In order to show that there is no edge between a vertex of  $V_1$  and a vertex of  $V_2$ , let us assume to the contrary that there is one such edge. Since  $V_1$  and  $V_2$  are on two different sides of a closed Jordan curve, there exists an element  $v$  of  $V_1$  that contains in its interior all the elements of  $V_2$ . The number of elements in  $V_2$  is at least  $\frac{1}{3}m^{1/2}$  contradicting the fact that  $v$  belongs to  $I(\mathcal{C})$ . ◀

Next, in the final lemma, we argue about the space-complexity of the above mentioned procedure.

► **Lemma 7.** *There exists a polynomial time algorithm that takes as an input  $\mathcal{C}$  and outputs the set  $\text{sep}(\mathcal{C})$  in  $O(m^{1/2} \log n)$  space.*

**Proof.** We first see that on input  $\mathcal{C}$ , the graph  $G_T(\mathcal{C})$  contains  $O(m)$  vertices. We can output this graph in log-space (see Lemma 4). Then, using the planar separator algorithm of Imai et al. [23], we can get a set of  $O(m^{1/2})$  vertices which acts as the separator of this graph. This process can be done in  $O(m^{1/2} \log n)$  space and polynomial time. Once we obtain this set, it is possible to construct  $\text{sep}(\mathcal{C})$  by using this set. ◀

This completes the proof of Theorem 2. Then, by combining Theorem 2 and Theorem 1, we conclude the following.

► **Corollary 8.** *There exists an algorithm that solves the REACHABILITY on the directed intersection graph of Jordan regions in polynomial time and  $O(m^{1/2} \log n)$  space.*

### 3 Unit Contact Disk Graphs (Penny Graphs)

We now study the REACHABILITY problem on directed unit contact disk graphs (penny graphs). Penny graphs are also known as unit coin graphs [9]. Let  $G = (D, E)$  be a directed penny graph, where  $D = \{d_1, \dots, d_n\}$  is a set of unit disks, and there is an edge  $e \in E$  between two disks  $d_i$  and  $d_j$  (for some  $i, j \in [n]$ ) if  $d_i$  and  $d_j$  touch each other. Moreover, each edge in  $e \in E$  is a directed edge.

We prove the following theorem.

► **Theorem 9.** *For every  $\varepsilon > 0$ , there exists a polynomial time algorithm that can solve REACHABILITY in an  $n$  vertex directed penny graph, using  $O(n^{1/4+\varepsilon})$  space.*

We know that given a set of  $n$  unit disks with  $m$  intersections, there exists an axis-parallel line intersecting  $O(\sqrt{m+n})$  disks such that each half-plane separated by that line contains at most  $4n/5$  disks [8]. They call such a separator a *balanced separator*. We first describe that how such a balanced separator can be obtained in a space-efficient manner.

Let  $R$  be a rectangular bounding box that contains the disks in  $D$ . Our algorithm to solve the REACHABILITY problem on penny graphs consists of four steps: (1) we find the balanced separators and use them to obtain a rectangular subdivision of the plane, (2) create an auxiliary graph, (3) obtain a pseudo-separator, (4) solve the REACHABILITY.



### 3.1 Obtaining Rectangular Subdivision

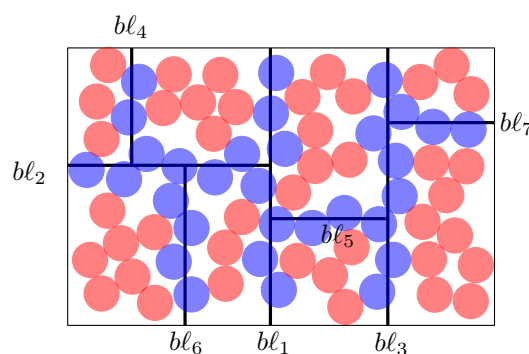
In this step, the direction of the edges of  $G$  is not relevant, so we simply consider the input as a set  $D$  of unit non-overlapping unit disks. The main idea is to divide the rectangle  $R$  into smaller rectangles such that each smaller rectangle contains at most  $n^{1-\varepsilon}$  unit disks, and the number of unit disks intersecting the boundary of any rectangle is at most  $n^{1/2+\varepsilon/2}$ .

In what follows, we describe a sweeping procedure. We discuss primarily in terms of vertical sweeping, and the the horizontal sweeping is done similarly.

We consider the disks in sorted order based on the x-coordinates of their corresponding centers, which can be easily done in log-space. Consider a vertical sweep line  $\ell$ . We start with the leftmost disk from  $D$  and sweep until we find a balanced separator that is intersecting  $O(\sqrt{m+n})$  many disks such that each half-plane separated by that line contains at most  $4n/5$  disks. If we do not find any such vertical separator, then we apply the same procedure with a horizontal sweep line (say  $\ell'$ ) sweeping from top to bottom. From Theorem 2 in [8], we know that there exists an axis-parallel balanced separator. Therefore, we shall obtain a balanced separator by doing this procedure.

When the sweep line  $\ell$  encounters or leaves a disk, we call it an event. There are precisely  $2n$  number of *events*. Note that it is possible to test by using  $\log n$  space whether a disk  $d_i$  intersects the line  $\ell$ . We maintain a counter  $c_\ell$  corresponding to the sweep line  $\ell$ . At each event  $k$  (for some  $k \in [n]$ ), we determine the number of disks that intersect  $\ell$ , by checking each disk that whether it is intersecting with  $\ell$  or not in log-space. Then, when the next event happens, we increase the value of the counter by 1 if it intersects a new disk, or decrease it by 1, otherwise. By using this procedure, we can find a separator line  $\ell$ , that is, a balanced separator. It is also clear that we can determine such a separator by using  $\log n$  space. Once we find such a separator, we only store the  $x$ -coordinate (resp.  $y$ -coordinate) of the vertical line  $\ell$  (resp. horizontal line  $\ell'$ ). Note that it is possible to find the actual set of disks that form such a separator in log-space when needed.

We subdivide the rectangles repeatedly until each of the rectangles has smaller than  $n^{1-\varepsilon}$  disks. Initially, we have the rectangle  $R$  containing all the disks. Let  $\mathcal{R}_0 = \{R\}$  be the initial set of rectangles. After step  $i$ , we have the rectangles  $\mathcal{R}_i$  be the set of rectangles, and We pick the rectangle with more than  $n^{1-\varepsilon}$  disks and subdivide it further using the above process to get  $\mathcal{R}_{i+1}$ . See Figure 2 for an illustration.



■ **Figure 2** An illustration of the rectangular subdivision by using the balanced separators. The blue disks are the ones intersected by the balanced separator lines, and the red disks are contained inside the rectangles.

We calculate the number of separators required to reach this termination point. From [8], we know that one  $\sqrt{m+n}$  size separator guarantees that on each side there are at most  $\frac{4n}{5}$  disks. Since the class of penny graphs is a subclass of planar graphs, the total number of

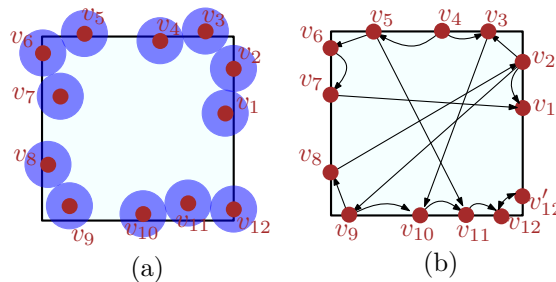
edges is at most  $3n - 6$ . Now at each step, we have obtained a *balanced separator* whenever it has satisfied the criteria. We need  $O(n^\varepsilon)$  many separators to have at most  $O(n^{1-\varepsilon})$  disks in each cell. To store these line separators, we need to use  $O(n^\varepsilon \log n)$  space.

The initial graph  $G$  is divided into  $n^\varepsilon \times n^\varepsilon$  rectangles obtained from the above procedure. Let  $\mathcal{Z}$  be the set of all rectangles. The idea is to reduce the size of the graph  $G$  by dropping the disks that are completely contained inside some rectangle and are not touched or intersected by its boundary line. However, while reducing the size of the number of disks, we need to ensure that the REACHABILITY information is fully preserved between any pair of disks in  $G$ . For that, we proceed to the next step and build an auxiliary graph.

### 3.2 Building Auxiliary Graph

For a rectangle  $R$ , let  $G_R$  be the graph defined as follows. The vertex set of  $G_R$  is the set of all disks which intersect at least one of the boundaries of  $R$ . We add an edge from a vertex  $u$  to a vertex  $v$  in  $G_R$  if there is a directed path from  $u$  to  $v$ , which contains only the disks present inside the rectangle  $R$ . Let  $v_1$  be an arbitrary disk, and let  $\{v_1, \dots, v_k\}$  be the sets of disks intersecting the boundary of  $R$  in the anti-clockwise order. We place the disk centers on the boundary while preserving (1) the order of them on the boundary, (2) each vertex  $v_i$  is on the side of the boundary that it intersects. However, if it is intersected by more than one side (one vertical and one horizontal) then we create an additional dummy vertex (say  $v'_i$ ) and assign  $v_i$  and  $v'_i$  to the horizontal and vertical side, respectively. Moreover, we add a bidirectional edge between  $v_i$  and  $v'_i$ . see Figure 3(b)) for an illustration.

The edges of  $G_R$  are drawn in the following manner. If there is an edge between two vertices that are on different sides, then we give a directed straight line edge. Otherwise, we join them by a directed circular arc. Moreover, we ensure that there is a crossing between two edges  $(v_i, v_k)$  and  $(v_l, v_j)$  in the drawing if and only if there is an ordering on the boundary which is one of the followings - 1.  $\{v_i \prec v_l \prec v_k \prec v_j\}$ , 2.  $\{v_j \prec v_l \prec v_k \prec v_i\}$ , 3.  $\{v_l \prec v_i \prec v_j \prec v_k\}$ , 4.  $\{v_l \prec v_i \prec v_j \prec v_k\}$ . Clearly, there exists such a drawing as their arc edges can be drawn arbitrarily close to the boundary lines.

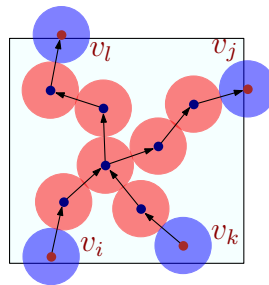


■ **Figure 3** An illustration of the representation of the graph  $G_R$ , (a) the blue disks are the vertices of  $G_R$  intersecting the boundary of the rectangle (b) drawing of  $G_R$ .

Now, by combining the graphs defined for each rectangle, we define the auxiliary graph  $Aux_\varepsilon(G)$ , for  $0 < \varepsilon < 1$ . The vertex set of  $Aux_\varepsilon(G)$  is  $\bigcup_{R \in \mathcal{Z}} V(G_R)$  and the edge set is  $\bigcup_{R \in \mathcal{Z}} E(G_R)$ . Notice that  $Aux_\varepsilon(G)$  might have parallel edges, since there exists paths between vertices in two adjacent rectangles, and in that case we keep both of these edges in their respective rectangles. The total number of vertices in each cell is  $O(n^{1-\varepsilon})$ . Hence the total number of vertices in  $Aux_\varepsilon(G)$  is  $O(n^{1/2+\varepsilon/2})$ . We point out that we do not store  $Aux_\varepsilon(G)$  explicitly because that requires too much space. Instead, we deal with each cell recursively when the subroutine queries for an edge in that cell of  $Aux_\varepsilon(G)$ . Now, we prove the following property about the auxiliary graph.

► **Lemma 10.** *Let  $G$  be a penny graph and  $e = (v_i, v_j)$  and  $e' = (v_k, v_l)$  be two edges in  $Aux_\varepsilon(G)$ . If  $e$  and  $e'$  cross each other, then  $Aux_\varepsilon(G)$  also contains the edges  $(v_i, v_l)$  and  $(v_k, v_j)$ .*

**Proof.** Consider two edges  $e = (v_i, v_j)$  and  $e' = (v_k, v_l)$  be two edges in  $Aux_\varepsilon(G)$ . From the definition, these edges corresponding to the directed paths in the input graph  $G$ . From the construction, we know that the ordering of the end vertices are one of the following -  $\{v_i \prec v_l \prec v_k \prec v_j\}$ ,  $\{v_j \prec v_l \prec v_k \prec v_i\}$ ,  $\{v_l \prec v_i \prec v_j \prec v_k\}$ ,  $\{v_l \prec v_i \prec v_j \prec v_k\}$ . Now, we know that the corresponding directed paths are fully embedded inside the grid cell. If two directed path intersects in the Auxiliary graph (based on our definition) of an input penny graph, that is embedded inside a rectangle, then they must have one common vertex; see Figure 4. Otherwise, it will not admit a planar embedding. Hence, there must be directed path from  $v_i$  to  $v_l$ , and  $v_k$  to  $v_j$ . Thereby proving the lemma. ◀



■ **Figure 4** An illustration of two directed paths inside a grid cell.

### 3.3 Constructing Pseudo-Separator

An essential property of a vertex separator is that, for any two vertices  $u$  and  $v$ , a path between them must contain a vertex from the separator if  $u$  and  $v$  lie in two different components with respect to the separator. We use a separator construction for the auxiliary graph  $Aux_\varepsilon(G)$ . However, note that  $Aux_\varepsilon(G)$  is not a planar graph anymore. Therefore, we need a special kind of separator, which we call a *pseudo-separator*.

The notion of the pseudo-separator was introduced by Jain and Tewari [24] in the context of grid graphs. However, since the class of Penny graphs is a superclass of grid graphs, it is not possible to use their idea directly.

Let  $G$  be a penny graph and  $H = (V_1, E_1)$  be a vertex induced subgraph of  $Aux_\varepsilon(G)$  with  $h$  vertices. Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  be a function. A subgraph  $C = (V_2, E_2)$  of  $H$  is said to be an  $f(h)$ -PseudoSeparator of  $H$ , if the size of every connected component in  $H \cdot C$  is at most  $f(h)$ , where the graph  $H \cdot C = (V_3, E_3)$  is defined as  $V_3 = V_1 \setminus V_2$  and  $E_3 = E_1 \setminus \{e \in E_1 \mid \exists e' \in E_2, e \text{ crosses } e'\}$ . The size of  $C$  is the total number of vertices and edges of  $C$  summed together.

The general idea of our approach is the following. Consider a vertex induced subgraph  $H$  of  $Aux_\varepsilon(G)$ . We choose a maximal subset of edges such that  $H$  is a planar graph, thus admits a planar embedding. Next, we triangulate this chosen sub-graph. We show that until this point, each operation can be performed in log-space. Then, we use the algorithm of Imai et al. [23] to obtain a separator of the triangulated graph. In what follows, we describe these procedures in detail.

We start with a maximal planar graph  $H(Aux_\varepsilon(G))$  of  $Aux_\varepsilon(G)$ . The vertex set of  $H(Aux_\varepsilon(G))$  is same as the vertex set of  $Aux_\varepsilon(G)$ . For each rectangle, we index the vertices in ascending order while traversing them in anti-clockwise direction. For each rectangle  $R$ , an

edge  $e_l = (u_l, v_l)$  of  $G_R$  is added to  $H(Aux_\varepsilon(G))$  if there is no edge  $e_k = (u_k, v_k)$  such that the ordering of the vertices is one of the following -  $\{u_k \prec u_l \prec v_k \prec v_l\}$ ,  $\{v_k \prec u_l \prec u_k \prec v_l\}$ ,  $\{u_k \prec v_l \prec v_k \prec u_l\}$ ,  $\{u_k \prec v_l \prec v_k \prec u_l\}$ . We prove that,  $H(Aux_\varepsilon(G))$  is indeed a maximal planar graph of  $Aux_\varepsilon(G)$ .

► **Lemma 11.**  $H(Aux_\varepsilon(G))$  is a maximal planar graph of  $Aux_\varepsilon(G)$ .

Proof of Lemma 11 can be found in the full version [7].

Next, we triangulate  $H(Aux_\varepsilon(G))$  by adding the boundary edges. Then for each rectangle, we consider a face of  $H(Aux_\varepsilon(G))$  and add the edges to complete the triangulation by a similar procedure described in Lemma 4 from Section 2. Moreover, the direction of the edges are arbitrary. Let  $\hat{H}(Aux_\varepsilon(G))$  be the triangulated graph. Note that this procedure can be done in log-space. Next, we use a Lemma from [23] that is stated below.

► **Lemma 12** ([23]). *For each  $\beta > 0$ , there exists a polynomial time algorithm and  $\tilde{O}(h^{1/2+\beta/2})$  space algorithm that takes a  $h$ -vertex planar graph  $P$  as input and outputs a set of vertices  $S$ , such that  $|S|$  is  $O(h^{1/2+\beta/2})$  and removal of  $S$  disconnects the graph into components of size  $O(h^{1-\beta})$ .*

Next, we construct the pseudo-separator  $S_H(Aux_\varepsilon(G))$  by using the following steps. First, by Lemma 12 we find a set  $S$  in  $\hat{H}(Aux_\varepsilon(G))$  that divides it into components of size  $O(h^{1-\beta})$ , where  $h$  is the number vertices in  $\hat{H}(Aux_\varepsilon(G))$ . We add the vertices and edges of  $S$  to the vertex and edge set of  $S_H(Aux_\varepsilon(G))$ , respectively. However, there is a small caveat to use Lemma 12 on  $\hat{H}(Aux_\varepsilon(G))$ . In order to triangulate the graph, we have added edges that were originally not part of the auxiliary graph. Therefore, for each edge  $e_k = (u_k, v_k)$  of the triangulation that is present in some rectangle  $\mathcal{R}$ , we consider a set of at most four edges of  $Aux_\varepsilon(G)$  that form a so-called *shield* around the edge  $e_k$ . Two of these edges start from  $u_k$  ending at two vertices  $v_p, v'_p$ , where  $v_p$  and  $v'_p$  are the closest point to the left and closest point to the right of  $u_k$ , respectively, in the total ordering of the boundary vertices. The other two edges start from  $v_k$  and end at two vertices  $u_q, u'_q$ , where  $u_q$  and  $u'_q$  are the closest points to the right and the left of  $v_k$  respectively, in the total ordering of boundary vertices. See Figure 5(a) for an illustration. Later, we argue that if these edges do not exist, and the cycle separator intersects the corresponding triangulation edge, then there must be other edges chosen in the maximal planar graph intersecting that triangulation edge, and hence this gives a contradiction.

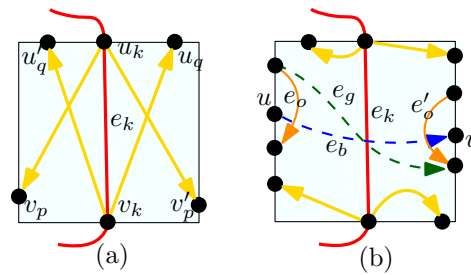
In order to prove that  $S_H(Aux_\varepsilon(G))$  is indeed a pseudo-separator, we need a property of triangulated graphs from [25].

► **Lemma 13** ([25]). *Let  $G$  be a triangulated planar graph and  $S$  be a subset of its vertices. For every pair of vertex  $u, v$  which belong to different components of  $G \setminus S$ , there exists a cycle in  $G[S]$ , such that  $u$  and  $v$  belong to different sides of this cycle.*

Next, we prove the Lemma about pseudo-separator.

► **Lemma 14** (Pseudo-Separator Lemma). *Let  $G$  be a penny graph and  $H(Aux_\varepsilon(G))$  be a vertex induced subgraph of  $Aux_\varepsilon(G)$ . The graph  $S_H(Aux_\varepsilon(G))$  is a  $h^{1-\beta}$ -pseudo-separator of  $H(Aux_\varepsilon(G))$ .*

**Proof.** Let  $S$  be a set of vertices chosen from  $\hat{H}(Aux_\varepsilon(G))$  by using Lemma 12. The claim is that if two vertices  $u$  and  $v$  belong to two different components in  $\hat{H}(Aux_\varepsilon(G)) \setminus S$ , then a path between  $u$  and  $v$  in  $G$  either takes a vertex of  $S_H(Aux_\varepsilon(G))$  or crosses an edge of



■ **Figure 5** An illustration of the pseudo-separator. The red edge is an edge of the triangulation and part of the red cycle separator. The yellow edges are chosen to form the shield (a) when the proper shield exists (b) otherwise.

$S_H(Aux_\varepsilon(G))$ . Note that due to Lemma 13, we know that  $u$  and  $v$  are on two different sides of a cycle of  $S$ . Thus, any edge drawn in the plane connecting  $u$  and  $v$  (not necessarily a straight line edge) crosses the cycle of  $S$ . If the path crosses non-triangulation edge or an edge which is shielded properly (see Figure 5(a)), we are done as the path will intersect an edge of  $S_H(Aux_\varepsilon(G))$ . The other situation is when such a path intersects a triangulation edge. We prove by contradiction that this can not happen. Assume that it does happen (see Figure 5(b)). Let  $e_r$  be the triangulation edge intersected by the edge  $e_b$  of the path from  $u$  to  $v$ . Now, since  $e_b$  is not chosen in the maximal planar graph, this means there exist other edges chosen in the  $H(Aux_\varepsilon(G))$ . Let  $e_o$  and  $e'_o$  be two such edges with maximum width on either side of the triangulated edge (see the orange edges in Figure 5(b)). By Lemma 10, there exists an edge  $e_g$  from the tail of  $e_o$  to the head of  $e'_o$  (the green path in Figure 5(b)). Since this edge  $e_g$  also crosses the triangulation edge  $e_r$ , it is not present in the maximal planar graph. Any edge that is present in maximal planar graph and crosses  $e_g$  contradicts the fact that  $e_o$  and  $e'_o$  are of maximum width. There are other cases when either  $e_o$  or  $e'_o$  is not present, but they can be handled similarly. Thereby concluding the proof of the Lemma. ◀

### 3.4 The Algorithm

Let  $H$  be a vertex induced subgraph of an auxiliary graph. We first explain how to solve REACHABILITY in  $H$ . Initially,  $H$  is the whole auxiliary graph, and we wish to find the REACHABILITY between given two vertices  $s$  and  $t$  of  $H$ .

By using Lemma 14, we find a pseudo-separator  $S$  of  $H$ . W.l.o.g., assume that  $s$  and  $t$  are both in  $S$ . The pseudo-separator divides the graph into components  $C_1, C_2, \dots, C_k$ , for  $k \in [h]$ . We maintain a set of vertices  $M$  that we call *marked* vertex set. We use an array of size at most  $|S|$  to mark a set of vertices  $M_1$  of the pseudo-separator. Additionally, for each edge of the pseudo-separator we have at most one associated vertex, and they form the set  $M_2$ . Then,  $M = M_1 \cup M_2$  is called a *marked* vertex set.

Throughout the algorithm, we maintain that if a vertex is marked, then there is a path from  $s$  to that vertex in the auxiliary graph. Initially, only the vertex  $s$  is marked. We then perform  $h$  iterations. In each iteration, we update the set of marked vertices as follows:

**Step 1.** For every vertex of  $S$ , we mark it if there is a path from an already marked vertex to it such that the internal vertices of that path all belong to only one component  $C_i$ , for  $i \in [h]$ . We check this by recursively running our algorithm on the subgraph of  $H$  induced by the vertex set  $C_i$ .

**Step 2.** For each edge  $e$  of  $S$ , the algorithm sets the associated marked vertex to  $u$  if the following three conditions are satisfied: (a) There exists an edge  $e' = (u, v)$  which crosses  $e$ , (b) there exists a path from a marked vertex to  $u$  such that the internal vertices of that path all belong to only one component  $C_i$  (we check this by recursively running our algorithm on the vertex induced subgraph of  $C_i$ ), (c)  $e'$  is the closest such edge to  $u$ .

Finally, we output 'YES' if and only if  $t$  is marked at the end of these iterations.

### 3.5 Correctness

Let  $P$  be a path from  $s$  to  $t$  in  $H$ . Suppose  $P$  passes through the components  $C_{\sigma_1}, C_{\sigma_2}, \dots, C_{\sigma_l}$  (each  $\sigma_i$  is a value in  $[k]$ ) in this order. By the definition of a pseudo-separator, the path can go from one component,  $C_{\sigma_i}$  to the next component  $C_{\sigma_{i+1}}$  in the following ways only:

**Case 1:** The path exits  $C_{\sigma_i}$  and enters  $C_{\sigma_{i+1}}$  through a vertex  $w$  of the pseudo-separator.

**Case 2:** The path exits  $C_{\sigma_i}$  and enters  $C_{\sigma_{i+1}}$  through an edge  $e' = (u, v)$  whose tail is in  $C_{\sigma_i}$  and head is in  $C_{\sigma_{i+1}}$ . This edge will cross an edge  $e = (x, y)$  of the pseudo-separator.

We see that after the  $i$ -th iteration, our algorithm will traverse the fragment of the path in the component  $C_{\sigma_i}$  and either mark in **Case 1** its endpoint or in **Case 2** a vertex  $u'$  such that the edge  $(u', v)$  exist. Thus,  $t$  will be marked after  $l$  iterations if and only if there is a path from  $s$  to  $t$  in  $H$ . Since  $l$  can be at most  $h$ , these many iterations would suffice.

From the above discussion, we know how to solve REACHABILITY on the auxiliary graph with the desired space-bound. To solve the problem on the input graph, we find the rectangular subdivision of the input graph. Then, we form the auxiliary graph by solving each rectangle recursively. Next, we solve the REACHABILITY problem on the auxiliary graph by following the above procedure. We have shown earlier that the reachability information of the input graph is fully preserved in the auxiliary graph. Hence, it is possible to report the solution of the input directed penny graph from the solution on the auxiliary graph.

### 3.6 Space-Complexity

The space complexity of our algorithm is dominated by the space required to store the *marked* vertices. Since there can be only  $h^{1/2+\varepsilon/2}$  such vertices, we need  $O(h^{1/2+\varepsilon/2} \log n)$  space. It is easy to see that for every  $\varepsilon' > 0$ , there exists an  $\varepsilon > 0$  such that  $O(h^{1/2+\varepsilon/2} \log n) = O(h^{1/2+\varepsilon'})$ . Note that, for the input auxiliary graph  $H$ , our algorithm recurses on vertex induced subgraphs whose size is  $O(h^{1-\varepsilon})$ . Hence, the depth of recursion is bounded by a constant. This increases the space required by at most a constant factor. Since the number of vertices in the initial auxiliary graph was itself  $O(n^{1/2+\varepsilon/2})$ , we get the desired space bound.

From the correctness and the space-complexity analysis of the algorithm, we conclude the proof of Theorem 9

## 4 Chordal Graphs

In this section, we study the REACHABILITY problem on directed chordal graphs and design a space-efficient algorithm. A graph is said to be chordal if every cycle of length at least four has a *chord*, which is an edge joining two vertices that are not adjacent on the cycle. A *directed* chordal graph is a graph whose underlying undirected graph is chordal. See [14, 17, 31] for the basic theory on chordal graphs. We adopt the algorithm of Gilbert et al. [19] for *Vertex Separator* and analyze to obtain the desired space-bound. In [25], it was noted that REACHABILITY on chordal graphs could be solved in a space-efficient manner, however, without any formal explanation. Here we provide a detailed analysis of this claim.



Let  $G = (V, E)$  be a directed chordal graph. For the sake of completeness, we will state some of the known results that are relevant to us. Chordal graphs are also known as *triangulated graphs*, *monotone transitive graphs*, *rigid circuit graphs*, *perfect elimination graphs* in the literature.

### Finding Separator

Gilbert et al. [19] presented  $O(mn)$ -time algorithm for finding *Vertex Separator* on chordal graphs. Furthermore, a better algorithm of time complexity  $O(m)$  is also shown. We design a space-efficient algorithm for Vertex separator that uses  $O(m^{1/2} \log n)$  space and polynomial time. We adopt the  $O(mn)$ -time algorithm from [19] and analyze it to provide the desired space-bound. In order to do that, we allow time complexity that is much larger than  $O(mn)$  but remains polynomial in  $n$ . We proceed with the following lemma: A similar property is somewhat implicitly used by [19].

► **Lemma 15.** *Let  $G$  be a chordal graph and let  $C$  be a clique. Let  $A$  be the largest component in  $G \setminus C$ . Then, either of the following two statements is true:*

- *There exists a vertex  $v$  in  $C$  which is not adjacent to any vertex in  $A$ .*
- *There exists a vertex  $u$  in  $A$  which is adjacent to every vertex in  $C$ .*

We need several definitions and structural properties of the Chordal graphs in order to prove this Lemma. However, some of the properties are well-known results for chordal graphs. We proceed with the following definitions.

► **Definition 16.** *Let  $G$  be a graph and  $v$  be a vertex of  $G$ . The deficiency of  $v$ , denoted by  $D(v)$  is defined as follows:  $D(v) = \{\{u_1, u_2\} \mid \{u_1, v\} \in E(G), \{u_2, v\} \in E(G) \text{ and } \{u_1, u_2\} \notin E(G)\}$*

► **Definition 17.** *Let  $G$  be a graph and  $v$  be a vertex of  $G$ . We define the graph  $G_v$  as follows:  $G_v = (V(G) \setminus \{v\}, E(G[V(G) \setminus \{v\}]) \cup D(v))$ . We say that the graph  $G_v$  is formed by eliminating  $v$  from  $G$ .*

► **Definition 18.** *When a sequence of vertices is eliminated from a graph, the edges in the deficiencies that are added are called fill-in edges. A simplicial vertex of a graph is a vertex that has a deficiency of 0.*

We state the following known facts about the chordal graphs. The following Lemmas are due to [14, 17, 30, 29, 31]. For the sake of completion, we state them here in the form that we will be using and provide their proofs.

► **Lemma 19.** *Let  $G$  be a chordal graph and  $a$  and  $b$  be two vertices in  $V(G)$ . Let  $S$  be a set of vertices of  $G$  such that: 1)  $a$  and  $b$  are in different components of the graph  $G \setminus S$ ; 2) there exists no proper subset  $S' \subseteq S$  such that vertices  $a, b$  are in different components of the graph  $G \setminus S'$ . If these conditions hold, then the set  $S$  forms a clique in  $G$ .*

**Proof.** Let  $C_a$  and  $C_b$  be the components in  $G \setminus S$  containing  $a$  and  $b$ , respectively. Note that, each vertex  $s \in S$  is adjacent to some vertices in  $C_a$ , and some vertices in  $C_b$ . Consider two vertices  $x, y \in S$ . Let  $\mathcal{P}_a$  be the shortest path between  $x$  and  $y$  in  $G[C_a \cup \{x, y\}]$ . Similarly, let  $\mathcal{P}_b$  be the shortest path between  $x$  and  $y$  in  $G[C_b \cup \{x, y\}]$ . The paths  $\mathcal{P}_a$  and  $\mathcal{P}_b$  together forms a cycle. Hence, there must be an edge between  $x$  and  $y$ , since it is the only *chord* that is possible. This argument holds for any pair of vertices in  $S$ . This completes the proof. ◀

► **Lemma 20.** *Let  $G$  be a chordal graph and let  $C$  be any clique of  $G$ . Then, either  $G$  is a complete graph, or there is a vertex  $v \in G \setminus C$  that is simplicial.*



**Proof.** We prove this by induction on  $|G|$ . The base case  $|G| = 1$  is trivial. Let us assume that the Lemma holds for all  $G$  such that  $|G| \leq k$ , for some  $k \in [n]$ . Now,  $G$  be a graph such that  $|G| = k + 1$ . If  $G$  is not a complete graph then, let  $a, b$  be two vertices of  $G$ , that are not adjacent. Due to Lemma 19, there exists a set  $S$  that separates  $a$  and  $b$ . Let  $C_a, C_b$ , be the corresponding components  $G$  containing  $a$  and  $b$ , respectively. Note that, the vertices in  $C \setminus S$  should be in one component. W.l.o.g., assume that they belong to  $C_a$ . Consider the graph  $G_b = G[S \cup C_b]$ . We have  $|G_b| \leq k$ . Hence, by induction, either  $G_b$  is a clique or there is a vertex  $u \notin S$ , that is *simplicial* in  $G_b$ . In either of the cases, there exists a vertex  $u \notin S$  that is simplicial in  $G_b$  since  $G_b$  must contain at least one vertex, not in  $S$ . Note,  $u$  is a simplicial vertex in  $G$  since  $u$  is not adjacent to a vertex in any component other than  $C_b$ . This completes the proof.  $\blacktriangleleft$

► **Corollary 21.** *Let  $G$  be a chordal graph and  $C$  be a clique. Let  $A$  be the largest component in  $G \setminus C$ . If  $B$  is a non-empty subset of  $A$ , then  $B$  contains a vertex whose neighbours in  $G[B \cup C]$  forms a clique.*

► **Lemma 22** ([19]). *Let  $a_0, a_1, \dots, a_k$  be an elimination ordering for a graph  $G$ . Let  $v$  and  $w$  be nonadjacent vertices of  $G$ . Then  $\{v, w\}$  is a fill-in edge if and only if there is a path from  $v$  to  $w$  consisting of vertices that are eliminated earlier than both  $v$  and  $w$ .*

Now, we have the ingredients to prove the main Lemma 15.

**Proof.** Let  $G$  be a chordal graph, and let  $C$  be a clique. Let  $A$  be the largest component in  $G \setminus C$ . Let us assume that each vertex  $v \in C$  is adjacent to at least one vertex in  $A$ . We will show that under this assumption, there exists a vertex  $u$  in  $A$  that is adjacent to each vertex in  $C$ , thereby proving the Lemma.

It is known that a vertex induced subgraph of a chordal graph is also a chordal graph. Hence, the subgraph  $G_0 := G[A \cup C]$  is a chordal graph. Due to Lemma 20, we know that there is a vertex  $u_0 \in G_0 \setminus C$ , that is simplicial, whose neighbors form a clique. Let  $G_1 := G[(A \cup C) \setminus \{u_0\}]$ . Similarly, let  $u_1$  be a vertex in  $G_1 \setminus C$  that is simplicial. In general, let  $u_i$  be a vertex in  $G_i \setminus C$  whose neighbours forms a clique, where  $G_i$  is defined as  $G[(A \cup C) \setminus \{u_0, u_1, \dots, u_{i-1}\}]$ . Let  $k$  be an integer such that  $G_{k+1} = G[C]$ . We claim that  $u_k$  is adjacent to every vertex of  $C$ .

Consider a vertex  $x$  in  $C$ . Since  $A$  is connected and by our assumption,  $x$  is adjacent to a vertex of  $A$ , there is a path from  $x$  to  $u_k$  in  $G[A \cup C]$  that uses only vertices of  $A$ . Lemma 22 says that if  $\{x, u_k\}$  is not an edge of  $G[A \cup C]$ , then it is a fill-in edge. But a perfect elimination ordering has no fill-in, so  $x$  is adjacent to  $u_k$  in  $G[A \cup C]$ . Thus  $u_k$  is adjacent to every vertex of  $C$ . This completes the proof.  $\blacktriangleleft$

Given the above lemma, the algorithm for finding a separator in a chordal graph is rather straightforward, as also observed by [19]. The algorithm is described in the following Procedure 1.

The correctness of the algorithm directly comes from Lemma 15. We show that this algorithm uses  $O(m^{1/2} \log n)$  space.

## Space-Complexity

To see that the above algorithm can be implemented in  $O(m^{1/2} \log n)$ -space and polynomial time, we first recall that the separator  $S$  always forms a clique in the graph. Since the size of a clique is upper-bounded by  $m^{1/2}$ , it will require at most  $O(m^{1/2} \log n)$  space to store the clique.

■ **Algorithm 1** Vertex Separator in Chordal Graphs.

---

**Input:** A chordal graph  $G$   
**Output:** A separator  $S$  of  $G$  of size  $\sqrt{m}$

- 1  $S \leftarrow \emptyset$ ;
- 2 **while** *there exists a component  $A$  of  $G \setminus S$  that has weight more than  $n/2$*  **do**
- 3     **while** *there exists a vertex  $x$  of  $S$  that is adjacent to no vertex of  $A$*  **do**
- 4          $S \leftarrow S \setminus \{x\}$ ;
- 5     **end**
- 6      $v \leftarrow$  a vertex of  $A$  adjacent to every vertex of  $S$ ;
- 7      $S \leftarrow S \cup \{v\}$ ;
- 8 **end**
- 9 Return  $S$ ;

---

Now, to implement the algorithm, we need to calculate the weights of each of the connected components of  $G \setminus S$ . However, we cannot afford to store these components explicitly, since the number of vertices present in these components could be large. Instead, we identify a component with the lowest-index vertex present in it. We call the lowest-index vertex a *marker* of the component. In order to check that whether a vertex  $v$  is a *marker*, we run Reingold's undirected REACHABILITY algorithm to see if it is not connected with any vertex  $u$  in  $G \setminus S$  such that the index of  $u$  is lower than  $v$ . Also, to know the size of the connected component for a marker vertex  $v$ , we use Reingold's algorithm [28] to count the number of vertices that are connected to it. Therefore it is possible to count the weight of any component of  $G \setminus S$  in the desired space-bound. We conclude the following theorem.

► **Theorem 23.** *Given a chordal graph  $G = (V, E)$ , there exists an algorithm that computes a  $\sqrt{m}$  separator in polynomial time by using  $O(m^{1/2} \log n)$ -space.*

From Theorem 23 and Theorem 1, we have the following corollary.

► **Corollary 24.** *There exists an algorithm that solves the REACHABILITY problem for chordal graphs by using  $O(m^{1/2} \log n)$ -space and polynomial time.*

## 5 Conclusion

We studied REACHABILITY problem on three important graph families and obtained space-efficient algorithms for each of these classes. An interesting open problem is whether one can obtain a space-efficient algorithm for intersection graphs of Jordan regions when the embedding of the graph is not provided in the input. Another important open problem is to study the REACHABILITY for unit disk intersection graphs and obtain a space-efficient algorithm. Our method that we used for unit contact disk graphs does not generalize to the case of unit disk intersection graphs, as in the latter case, there can be arbitrarily large directed cliques, and it is not possible to obtain auxiliary graphs while preserving reachability information between every pair of vertices. However, we believe that our method can be used to solve to REACHABILITY for other classes of geometric contact graphs.

---

## References

- 1 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete applied mathematics*, 23(1):11–24, 1989.
- 2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

- 3 Tetsuo Asano and Benjamin Doerr. Memory-constrained algorithms for shortest path problem. In *CCCG*, 2011.
- 4 Tetsuo Asano, David Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe.  $\tilde{O}(\sqrt{n})$ -space and polynomial-time algorithm for planar directed graph reachability. In *International Symposium on Mathematical Foundations of Computer Science*, pages 45–56. Springer, 2014.
- 5 Bahareh Banyassady, Matias Korman, and Wolfgang Mulzer. Geometric algorithms with limited workspace: A survey. *CoRR*, abs/1806.05868, 2018. [arXiv:1806.05868](#).
- 6 Greg Barnes, Jonathan F Buss, Walter L Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed st connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998.
- 7 Sujoy Bhore and Rahul Jain. Space-efficient algorithms for reachability in geometric graphs. *CoRR*, abs/2101.05235, 2021. [arXiv:2101.05235](#).
- 8 Paz Carmi, Man-Kwun Chiu, Matthew J. Katz, Matias Korman, Yoshio Okamoto, André van Renssen, Marcel Roeloffzen, Taichi Shiitada, and Shakhar Smorodinsky. Balanced line separators of unit disk graphs. *Comput. Geom.*, 86, 2020.
- 9 Marcia R Cerioli, Luerbio Faria, Talita O Ferreira, and Fábio Protti. A note on maximum independent sets and minimum clique partitions in unit disk graphs and penny graphs: complexity and approximation. *RAIRO-Theoretical Informatics and Applications*, 45(3):331–346, 2011.
- 10 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPICs*, pages 585–595. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.
- 11 Diptarka Chakraborty and Raghunath Tewari. An  $o(n^\epsilon)$  space and polynomial time algorithm for reachability in directed layered planar graphs. *ACM Trans. Comput. Theory*, 9(4):19:1–19:11, 2018.
- 12 Steven Chaplick, Vít Jelínek, Jan Kratochvíl, and Tomáš Vyskočil. Bend-bounded path intersection graphs: Sausages, noodles, and waffles on a grill. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 274–285. Springer, 2012.
- 13 Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. In *Annals of Discrete Mathematics*, volume 48, pages 165–177. Elsevier, 1991.
- 14 Gabriel Andrew Dirac. On rigid circuit graphs. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 25, pages 71–76. Springer, 1961.
- 15 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 383–392, 2014.
- 16 Jacob Fox and János Pach. Separator theorems and turán-type results for planar intersection graphs. *Advances in Mathematics*, 219(3):1070–1080, 2008.
- 17 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.
- 18 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.
- 19 John R Gilbert, Donald J Rose, and Anders Edenbrandt. A separator theorem for chordal graphs. *SIAM Journal on Algebraic Discrete Methods*, 5(3):306–313, 1984.
- 20 Chetan Gupta, Vimal Raj Sharma, and Raghunath Tewari. Reachability in  $o(\log n)$  genus graphs is in unambiguous logspace. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- 21 Michael Hoffmann, Vincent Kusters, and Tillmann Miltzow. Halving balls in deterministic linear time. In *European Symposium on Algorithms*, pages 566–578. Springer, 2014.

- 22 Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, 1983.
- 23 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu Watanabe. An  $o(n^{1/2} + \sum)$ -space and polynomial-time algorithm for directed planar reachability. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 277–286. IEEE Computer Society, 2013.
- 24 Rahul Jain and Raghunath Tewari. An  $o(n^{(1/4+\epsilon)})$  space and polynomial algorithm for grid graph reachability. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPICs*, pages 19:1–19:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 25 Rahul Jain and Raghunath Tewari. Reachability in high treewidth graphs. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 26 Harry R Lewis and Christos H Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, 1982.
- 27 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.
- 28 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.
- 29 Donald J Rose. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications*, 32(3):597–609, 1970.
- 30 Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Elsevier, 1972.
- 31 Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.
- 32 Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.
- 33 Avi Wigderson. The complexity of graph connectivity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 112–132. Springer, 1992.



# Repetition- and Linearity-Aware Rank/Select Dictionaries

Paolo Ferragina   

Department of Computer Science, University of Pisa, Italy

Giovanni Manzini   

Department of Computer Science, University of Pisa, Italy

Giorgio Vinciguerra   

Department of Computer Science, University of Pisa, Italy

---

## Abstract

We revisit the fundamental problem of compressing an integer dictionary that supports efficient rank and select operations by exploiting two kinds of regularities arising in real data: *repetitiveness* and *approximate linearity*. Our first contribution is a Lempel-Ziv parsing properly enriched to also capture approximate linearity in the data and still be compressed to the  $k$ th order entropy. Our second contribution is a variant of the block tree structure whose space complexity takes advantage of both repetitiveness and approximate linearity, and results highly competitive in time too. Our third and final contribution is an implementation and experimentation of this last data structure, which achieves new space-time trade-offs compared to known data structures that exploit only one of the two regularities.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Data compression; Theory of computation  $\rightarrow$  Data structures design and analysis

**Keywords and phrases** Data compression, Compressed data structures, Entropy

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.64

**Supplementary Material** *Software*: <https://github.com/gvinciguerra/BlockEpsilonTree>  
archived at `swh:1:dir:a0f2406c8797804e8aec0afda6c06a80e945f85b`

**Funding** The authors have been supported in part by the Italian MUR PRIN project 2017WR7SHH: *Multicriteria data structures and algorithms*.

## 1 Introduction

We focus on the fundamental problem of representing an ordered dictionary  $A$  of  $n$  distinct elements drawn from the integer universe  $[u] = \{0, \dots, u\}$  while supporting the operation  $\text{rank}(x)$ , that returns the number of elements in  $A$  which are  $\leq x$ ; and  $\text{select}(i)$ , that returns the  $i$ th smallest element in  $A$ .

Rank/select dictionaries are at the heart of virtually any compact data structure [34], such as text indexes [15, 18, 20, 22, 30, 36], succinct trees and graphs [32, 41], hash tables [4], permutations [3], etc. Unsurprisingly, the literature is abundant in solutions, e.g. [2, 8, 21, 24, 31, 37, 40, 41]. Yet, the problem of designing theoretically and practically efficient rank/select structures is anything but closed. The reason is threefold. First, there is an ever-growing list of applications of compact data structures (in bioinformatics [13, 29], information retrieval [33], and databases [1], just to mention a few) each having different characteristics and requirements on the use of computational resources, such as time, space, and energy consumption. Second, the hardware is evolving [23], sometimes requiring new data structuring techniques to fully exploit it, e.g. larger CPU registers, new instructions, parallelism, next-generation memories such as PMem. Third, data may present different kinds of regularities, which require different techniques that exploit them to improve the space-time performance.



© Paolo Ferragina, Giovanni Manzini, and Giorgio Vinciguerra;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 64; pp. 64:1–64:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Among the latest of such regularities to be exploited, there is a geometric concept of *approximate linearity* [7]. Regard  $A$  as a sorted array  $A[1, n]$ , so that  $\text{select}(i)$  can be implemented as  $A[i]$ . The idea is to first map each element  $A[i]$  to a point  $(i, A[i])$  in the Cartesian plane, for  $i = 1, 2, \dots, n$ . Intuitively, any function  $f$  that passes through all the points in this plane can be thought of as an encoding of  $A$  because we can recover  $A[i]$  by querying  $f(i)$ . Now the challenge is to find a representation of  $f$  which is both fast to be computed and compressed in space. To this end, the authors of [7] implemented  $f$  via a piecewise linear model whose error, measured as the vertical distance between the prediction and the actual value of  $A$ , is bounded by a given integer parameter  $\varepsilon$ .

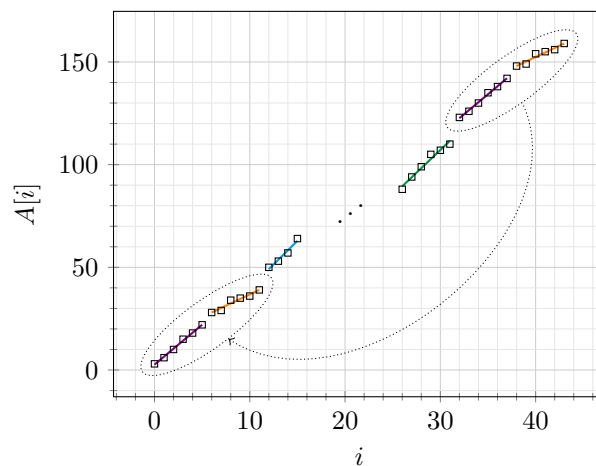
► **Definition 1.** A piecewise linear  $\varepsilon$ -approximation for the integer array  $A[1, n]$  is a partition of  $A$  into subarrays such that each subarray  $A[i, j]$  of the partition is covered by a segment, represented by a pair  $\langle \alpha, \beta \rangle$  of numbers, such that  $|(\alpha \cdot k + \beta) - A[k]| \leq \varepsilon$  for each  $k \in [i, j]$ .

Among all possible piecewise linear  $\varepsilon$ -approximations, one aims for the most succinct one, namely the one with the least amount of segments. This is a classical computational geometry problem that admits an  $\mathcal{O}(n)$ -time algorithm [38]. The structure introduced by [7], named LA-vector, uses this succinct piecewise linear  $\varepsilon$ -approximation as a lossy representation of  $A$ , and it mends the information loss by storing the vertical errors into an array  $C$  of  $\lceil \log(2\varepsilon + 1) \rceil$ -bit integers, called corrections (all logarithms are to the base two). To answer  $\text{select}(i)$ , the LA-vector uses a constant-time rank structure on a bitvector of size  $n$  to find the segment  $\langle \alpha, \beta \rangle$  covering  $i$ , and it returns the value  $\lfloor (\alpha \cdot i + \beta) \rfloor + C[i]$ . The  $\text{rank}(x)$  operation is implemented via a sort of binary search that exploits the information encoded in the piecewise linear  $\varepsilon$ -approximation [7]. In practical implementations, we allocate  $c \geq 0$  bits for each correction and set  $\varepsilon = 2^c - 1$ . The space usage in bits of an LA-vector consists therefore of a term  $\mathcal{O}(nc)$  accounting for the corrections array  $C$ , and a term  $\mathcal{O}(wm)$ , where  $w$  is the word size, that grows with the number of segments  $m$  in the piecewise linear  $\varepsilon$ -approximation. Despite the apparent simplicity of the piecewise linear representation, the experiments in [7] show that the LA-vector offers the fastest  $\text{select}$  and competitive rank performance with respect to several well-established structures implemented in the `sds1` library [19]. In addition to its good practical performance, recent results [14, 17] have shown there are also theoretical reasons that justify the effectiveness of the piecewise linear  $\varepsilon$ -approximation in certain contexts.

Despite their succinctness and power in capturing linear trends, piecewise linear  $\varepsilon$ -approximations still lack the capacity to find and exploit one fundamental source of compressibility arising in real data: *repetitiveness* [35]. Although the input consists in an array  $A$  of strictly increasing values, there can be significant repetitiveness in the differences between consecutive elements. Consider the gap-string  $S[1, n]$  defined as  $S[i] = A[i] - A[i - 1]$ , with  $A[0] = 0$ , and suppose the substring  $S[i, j]$  has been encountered earlier at  $S[i', i' + j - i]$  (we write  $S[i, j] \equiv S[i', i' + j - i]$ ). Then, instead of finding a new set of segments  $\varepsilon$ -approximating the subarray  $A[i, j]$ , we can use the segments  $\varepsilon$ -approximating the subarray  $A[i', j']$  properly shifted. Note that, even if  $A[i', j']$  is covered by many segments, the same shift will transform all of them into an approximation for  $A[i, j]$  (see example in Figure 1). Therefore, in this case, we would need to store only the *shift* and the *reference* to the segments of  $A[i', j']$ . The LA-vector is unable to take advantage of such regularities. In the extreme case where  $A$  consists of the concatenation of a small subarray  $A'$  shifted by some amounts  $\Delta_i$ s for  $k$  times, that is  $A = A', A' + \Delta_1, A' + \Delta_2, \dots, A' + \Delta_{k-1}$ , the overall cost of representing  $A$  with an LA-vector will be roughly  $k + 1$  times the cost of representing  $A'$ .

The goal of this paper is to harness the repetitions in the gap-string  $S$  to make the LA-vector repetition aware. In fact, the approximate linearity and the repetitiveness of a string are different proxies of its compressibility and therefore it is interesting to take both of





■ **Figure 1** The points in the top-right circle follows the same “pattern” (i.e. the same distance between consecutive points) of the ones in the bottom-left circle. A piecewise linear  $\varepsilon$ -approximation for the top-right set can be obtained by shifting the segments for the bottom-left set.

them into account. Take as an example an order- $h$  De Bruijn binary sequence  $B[1, 2^h]$  and define  $A[i] = 2i + B[i]$ , then the line with slope 2 and intercept 0 is a linear approximation of the entire array  $A$  with  $\varepsilon = 1$ . Conversely, following the argument above and considering the gap-string  $S[i] = A[i] - A[i - 1] = 2 + B[i] - B[i - 1]$ , we would not find repetitions longer than  $h - 1$  in  $S$ . The challenge is to devise techniques that are able to exploit both the presence of repetitions in the gap-string  $S$  and the presence of subarrays in  $A$  which can be linearly  $\varepsilon$ -approximated well, while still supporting efficient rank/select primitives on  $A$ . We point out that, although in this paper we consider only linear approximations, our techniques can be applied also to other data approximation functions, such as polynomials and rational functions.

**Our contribution in context.** The most common approach in the literature to design a *succinct* dictionary for a set of distinct integers  $A$  over the universe  $\{0, \dots, u\}$  is to represent  $A$  using the characteristic bitvector  $\mathbf{bv}(A)$ , which has length  $u + 1$  and is such that  $\mathbf{bv}(A)[i] = 1$  iff  $i \in A$ . In this paper, we use instead linear  $\varepsilon$ -approximations of  $A$  and the gap string  $S$ , and we show how to modify two known compression methods so that they can take advantage of approximate linearity. The first method is the Lempel-Ziv (LZ) parsing [26–28, 44], which is one of the best-known approaches to exploit repetitiveness [35]. The second method is the block tree [5], which is a recently proposed query-efficient alternative to LZ-parsing and grammar-based representations [6] suitable also for highly repetitive inputs since its space usage can be bounded in terms of the string complexity measure  $\delta$  (see [25, 35] for the definition and significance of this measure).

Our first contribution is a novel parsing scheme, the  $LZ_\varepsilon^\rho$  parsing, whose phrases are a combination of a backward copy and a linear  $\varepsilon$ -approximation, i.e., a segment and the corresponding correction values. The  $LZ_\varepsilon^\rho$  parsing encapsulates a piecewise linear  $\varepsilon$ -approximation of the array  $A$  and supports efficient rank/select primitives on  $A$ . Surprisingly, this combination uses space bounded by the  $k$ th order entropy  $H_k(S)$  of the gap-string  $S$  (see [26] for the definition and significance of  $k$ th order entropy). More precisely (Theorems 7 and 9), if  $\sigma$  denotes the number of distinct gaps in  $S$ , the  $LZ_\varepsilon^\rho$  parsing supports rank in  $\mathcal{O}(\log^{1+\rho} n + \log \varepsilon)$  time and select in  $\mathcal{O}(\log^{1+\rho} n)$  time using  $nH_k(S) + \mathcal{O}(n/\log^\rho n) + o(n \log \sigma)$

bits of space, for any positive  $\rho$  and  $k = o(\log_\sigma n)$ , plus the space to store the segments and the correction values that are used to advance the parsing (like the explicit characters in traditional LZ-parsing).

The best succinct data structure based on  $\text{bv}(A)$  is the one by Sadakane and Grossi [43] that supports constant-time **rank** and **select** in  $uH_k(\text{bv}(A)) + \mathcal{O}(u \log \log u / \log u)$  bits of space. This space bound cannot be compared to ours since it is given in terms of  $H_k(\text{bv}(A))$  instead of  $H_k(S)$ . To achieve space  $nH_k(S)$  one can use an entropy-compressed representation of  $S$  enriched with auxiliary data structures to support **rank/select** on  $A$ . For example, by sampling one value of  $A$  out of  $\log n$  and performing a binary search followed by a prefix sum of the gaps one can support  $\mathcal{O}(\log n)$ -time **rank** and **select** queries. Using the representation of [16], this solution uses  $nH_k(S) + \mathcal{O}(n \log u / \log n) + o(n \log \sigma) = nH_k(S) + o(n \log u)$  bits of space, which is worse in space than our solution but faster in query time. Other trade-offs are possible: the crucial point however is that none of the known techniques is able to exploit simultaneously the presence of exact repetitions and approximate linearity in the input data as instead our  $LZ_\varepsilon^\rho$  does. In the best scenario,  $LZ_\varepsilon^\rho$  parsing uses segments to quickly consume any approximate linearity in  $A$  thus potentially reducing significantly the number of LZ-phrases. On the other hand, if  $A$  cannot be linearly approximated, segments will be short and the overall space occupancy of  $LZ_\varepsilon^\rho$  parsing will be  $nH_k(S) + o(n \log \sigma)$  bits, i.e. no worse than a traditional LZ-parsing.

Our second contribution is the block- $\varepsilon$  tree, an orchestration of block trees [5, 25] and linear  $\varepsilon$ -approximations. The main idea is to build the block tree over the gap-string  $S$  and to prune the subtrees whose corresponding subarray can be covered more succinctly using a linear  $\varepsilon$ -approximation in place of a block (sub)tree. We show that this solution supports **rank** in  $\mathcal{O}(\log \log \frac{u}{\delta} + \log \frac{n}{\delta} + \log \varepsilon)$  time and **select** in  $\mathcal{O}(\log \frac{n}{\delta})$  time using  $\mathcal{O}(\delta \log \frac{n}{\delta} \log n)$  bits of space in the worst case, where  $\delta$  is the string complexity of  $S$  [25, 35].

A block tree built on  $\text{bv}(A)$ , instead, supports **rank** and **select** in  $\mathcal{O}(\log \frac{u}{\delta'})$  time using  $\mathcal{O}(\delta' \log \frac{u}{\delta'} \log u)$  bits of space, where  $\delta'$  is the string complexity measure on  $\text{bv}(A)$ . The time and space bounds achieved by the block tree and by our block- $\varepsilon$  tree are not comparable due to the use of  $\delta'$  instead of  $\delta$ . Therefore, as our third contribution, we provide an implementation of our block- $\varepsilon$  tree built on  $S$ , and we compare it with the standard block tree built on  $\text{bv}(A)$ . Our proposal turns out to be more space-efficient for some of the experimented sparse datasets and, as far as query time is concerned, it is  $2.19\times$  faster in **select**, and it is either faster ( $1.32\times$ ) or slower ( $1.27\times$ ) in **rank** than the block tree.

In the Conclusions, we comment on several research directions that naturally arise from the novel approaches described in this paper.

## 2 Tools

We use as a black box the Elias-Fano [9, 10] representation for compressing and random-accessing monotone integer sequences [34, §4.4].

► **Lemma 2** (Elias-Fano encoding). *We can store a sequence of  $n$  increasing positive integers over a universe of size  $u$  in  $n \lceil \log \frac{u}{n} \rceil + 2n + o(n) = n \log \frac{u}{n} + \mathcal{O}(n)$  bits and access any integer of the sequence in  $\mathcal{O}(1)$  time.*

Henceforth, we always assume that a piecewise linear  $\varepsilon$ -approximation for an input array  $A$  is the most succinct one in terms of the number of segments, or equivalently, that we always maximise the length  $\ell$  of the subarray  $A[i, i + \ell - 1]$  covered by a segment starting at  $i$ . This is possible thanks to the algorithm of O'Rourke [38], which in optimal  $\mathcal{O}(n)$  time computes the piecewise linear  $\varepsilon$ -approximation with the smallest number of segments for the set of points  $\{(i, A[i]) \mid i = 1, \dots, n\}$ .

Another key tool that we use is LZ-end [27]. Formally, the LZ-end parsing of a text  $T[1, n]$  is a sequence  $f_1, f_2, \dots, f_z$  of phrases, such that  $T = f_1 f_2 \dots f_z$ , built as follows. If  $T[1, i]$  has been parsed as  $f_1 f_2 \dots f_{q-1}$ , the next phrase  $f_q$  is obtained by finding the longest prefix of  $T[i+1, n]$  that appears also in  $T[1, i]$  ending at a phrase boundary, i.e. the longest prefix of  $T[i+1, n]$  which is a suffix of  $f_1 \dots f_r$  for some  $r \leq q-1$ . If  $T[i+1, j]$  is the prefix with the above property, the next phrase is  $f_q = T[i+1, j+1]$  (notice the addition of  $T[j+1]$  to the longest copied prefix). The occurrence in  $T[1, i]$  of the prefix  $T[i+1, j]$  is called the *source* of the phrase  $f_q$ .

Although LZ-end is less powerful than the classic LZ77 parsing, which allows the *end* of a source to be anywhere in  $T[1, i]$ , it compresses any text  $T$  up to its  $k$ th order entropy, and it is more efficient than LZ77 in extracting any substring of  $T$ .

With the advent of large datasets containing many repetitions, researchers have observed that the entropy does not always provide a meaningful lower bound to the information content of such datasets [27]. Recently, [35] has given a complete picture of several alternative measures of information content and has shown that they are all lower bounded by the complexity measure  $\delta$  defined as  $\max\{T(k)/k \mid 1 \leq k \leq n\}$ , where  $T(k)$  is the number of distinct length- $k$  substrings of  $T$  [42]. In [25], it is shown that using the block tree [5] it is possible to represent a text  $T[1, n]$  in space bounded in terms of  $\delta$  while supporting:  $\text{rank}_a(i)$ , which returns the number of occurrences of the character  $a$  in  $T[1, i]$ , and  $\text{select}_a(j)$ , which returns the position of the  $j$ th occurrence of  $a$  in  $T$ . Specifically, their block tree supports  $\text{rank}_a$  and  $\text{select}_a$  in  $\mathcal{O}(\log \frac{n}{\delta})$  time using  $\mathcal{O}(\sigma \delta \log \frac{n}{\delta} \log n)$  bits of space.

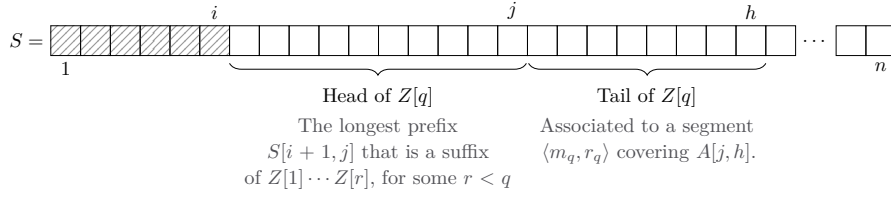
### 3 Two novel LZ-parsings: $\text{LZ}_\varepsilon$ and $\text{LZ}_\varepsilon^\rho$

Assume that  $A$  contains distinct positive elements and consider the gap-string  $S[1, n]$  defined as  $S[i] = A[i] - A[i-1]$ , where  $A[0] = 0$ . To make the LA-vector repetition aware, we parse  $S$  via a strategy that combines linear  $\varepsilon$ -approximation with LZ-end parsing. We generalise the phrases of the LZ-end parsing in a way that they are a “combination” of a backward copy ending at a phrase boundary (as in the classic LZ-end), computed over the gap-string  $S$ , plus a segment covering a subarray of  $A$  with an error of at most  $\varepsilon$  (unlike classic LZ-end, which instead adds a single trailing character). We call this parsing the  $\text{LZ}_\varepsilon$  parsing of  $S$ .

Suppose that  $\text{LZ}_\varepsilon$  has partitioned  $S[1, i]$  into  $Z[1], Z[2], \dots, Z[q-1]$ . We determine the next phrase  $Z[q]$  as follows (see Figure 2):

1. We compute the longest prefix  $S[i+1, j]$  of  $S[i+1, n]$  that is a suffix of the concatenation  $Z[1] \dots Z[r]$  for some  $r \leq q-1$  (i.e. the source must end at a previous phrase boundary).
2. We find the longest subarray  $A[j, h]$  that may be  $\varepsilon$ -approximated linearly, as well as the slope and intercept of such approximation. Note that using the algorithm of [38] the time complexity of this step is  $\mathcal{O}(h-j)$ , i.e. linear in the length of the processed array.

The new phrase  $Z[q]$  is then the substring  $S[i+1, j] \cdot S[j+1, h]$ . If  $h = n$ , the parsing is complete. Otherwise, we continue the parsing with  $i \leftarrow h+1$ . As depicted in Figure 2, we call  $S[i+1, j]$  the *head* of  $Z[q]$  and  $S[j+1, h]$  the *tail* of  $Z[q]$ . Note that the tail covers also the value  $A[j]$  corresponding to the head’s last position  $S[j]$ . When  $S[i+1, j]$  is the empty string (e.g. at the beginning of the parsing), the head is empty, and thus no backward copy is executed. In the worst case, the longest subarray we can  $\varepsilon$ -approximate has length 2, which nonetheless guarantees that  $Z[q]$  is nonempty. Experiments in [7] show that the average segment length ranges from 76 when  $\varepsilon = 31$  to 1480 when  $\varepsilon = 511$ .



■ **Figure 2** Computation of the next phrase  $Z[q]$  in the parsing of the gap-string  $S$  of the array  $A$ , where the prefix  $S[1, i]$  has already been parsed into  $Z[1], Z[2], \dots, Z[q-1]$ .

If the complete parsing consists of  $\lambda$  phrases, we store it using:

- An integer vector  $\text{PE}[1, \lambda]$  (Phrase Ending position) such that  $h = \text{PE}[q]$  is the ending position of phrase  $Z[q]$ , that is,  $Z[q] = S[i+1, h]$ , where  $i = \text{PE}[q-1] + 1$ .
- An integer vector  $\text{HE}[1, \lambda]$  (Head Ending position) such that  $j = \text{HE}[q]$  is the last position of  $Z[q]$ 's head. Hence,  $Z[q]$ 's head is  $S[\text{PE}[q-1] + 1, \text{HE}[q]]$ , and  $Z[q]$ 's tail is  $S[\text{HE}[q] + 1, \text{PE}[q]]$ .
- An integer vector  $\text{HS}[1, \lambda]$  (Head Source) such that  $r = \text{HS}[q]$  is the index of the last phrase in  $Z[q]$ 's source. Hence, the head of  $Z[q]$  is a suffix of  $Z[1] \cdots Z[r]$ . If the head of  $Z[q]$  is empty then  $\text{HS}[q] = 0$ .
- A vector of pairs  $\text{TL}[1, \lambda]$  (Tail Line) such that  $\text{TL}[q] = \langle \alpha_q, \beta_q \rangle$  are the coefficients of the segment associated to the tail of  $Z[q]$ . By construction, such segment provides a linear  $\varepsilon$ -approximation for the subarray  $A[\text{HE}[q], \text{PE}[q]]$ .
- A vector of arrays  $\text{TC}[1, \lambda]$  (Tail Corrections) such that  $\text{TC}[q]$  is an array of length  $\text{PE}[q] - \text{HE}[q] + 1$  providing the corrections for the elements in the subarray  $A[\text{HE}[q], \text{PE}[q]]$  covered by  $Z[q]$ 's tail. By construction, such corrections are smaller than  $\varepsilon$  in modulus.

Using the values in  $\text{TL}$  and  $\text{TC}$  we can recover the subarrays  $A[j, h]$  corresponding to the phrases' tails. We show that using all the above vectors we can recover the whole array  $A$ .

► **Lemma 3.** *Let  $S[i+1, j]$  denote the head of phrase  $Z[q]$ , and let  $r = \text{HS}[q]$  and  $e = \text{PE}[r]$ . Then, for  $t = i+1, \dots, j$ , it holds*

$$A[t] = A[t - (j - e)] + (A[j] - A[e]), \quad (1)$$

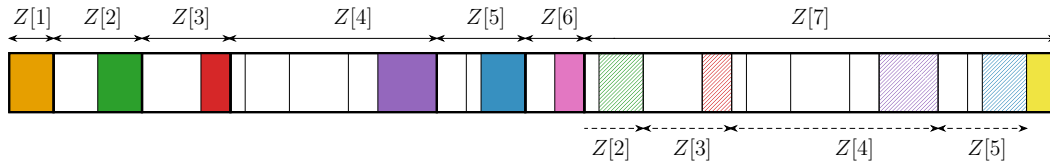
where  $A[j]$  (resp.  $A[e]$ ) can be retrieved in constant time from  $\text{TL}[q]$  and  $\text{TC}[q]$  (resp.  $\text{TL}[r]$  and  $\text{TC}[r]$ ).

**Proof.** By construction,  $S[i+1, j]$  is identical to a suffix of  $Z[1] \cdots Z[r]$ . Since such a suffix ends at position  $e = \text{PE}[r]$ , it holds  $S[i+1, j] \equiv S[e - j + i + 1, e]$  and

$$\begin{aligned} A[t] &= A[j] - (S[j] + S[j-1] + \cdots + S[t+1]) \\ &= (A[j] - A[e]) + A[e] - (S[e] + S[e-1] + \cdots + S[t+1 - (j-e)]) \\ &= (A[j] - A[e]) + A[t - (j-e)]. \end{aligned}$$

For the second part of the lemma, we notice that  $A[j]$  is the first value covered by  $Z[q]$ 's tail, while  $A[e]$  is the last value covered by  $Z[r]$ 's tail. ◀

Using the above lemma, we can show by induction that given a position  $t \in [1, n]$  we can retrieve  $A[t]$ . The main idea is to use a binary search on  $\text{PE}$  to retrieve the phrase  $Z[q]$  containing  $t$ . Then, if  $t \geq \text{HE}[q]$ , we get  $A[t]$  from  $\text{TL}[q]$  and  $\text{TC}[q]$ ; otherwise, we use Lemma 3 and get  $A[t]$  by retrieving  $A[t - (j - e)]$  using recursion. In the following, we will formalise this intuition in a complete algorithm, but before doing so, we need to introduce some additional notation.



■ **Figure 3** The  $LZ_\epsilon$  parsing with the definition of meta-characters. Cells represent meta-characters, and the coloured cells are also tails.  $Z[7]$ 's head consists of a copy of a substring that starts inside  $Z[2]$  and ends at the end of  $Z[5]$  (notice the diagonal patterns in  $Z[7]$ 's head with the same colours of the tails in  $Z[2] \cdots [5]$ ). Meta-characters in  $Z[7]$ 's head are defined from the meta-characters in the copy. Note that  $Z[7]$ 's first meta-character is a suffix of  $Z[2]$ 's first meta-character.

Using the  $LZ_\epsilon$  parsing, we partition the string  $S$  into *meta-characters* as follows. The first phrase in the parsing  $Z[1] = S[1, PE[1]]$  is our first meta-character (note  $Z[1]$  has an empty head, so  $HE[1] = 0$  and the pair  $\langle TL[1], TC[1] \rangle$  encodes the subarray  $A[0, PE[1]]$ ). Now, assuming we have already parsed  $Z[1] \cdots Z[q - 1]$  and partitioned  $S[1, PE[q - 1]]$  into meta-characters, we partition the next phrase  $Z[q]$  into meta-characters as follows:  $Z[q]$ 's tail will form a meta-character by itself, while  $Z[q]$ 's head “inherits” the partition into meta-characters from its source. Indeed, recall that  $Z[q]$ 's head is a copy of a suffix of  $Z[1] \cdots Z[r]$ , with  $r = HS[q]$ . Such a suffix, say  $S[a, b]$ , belongs to the portion of  $S$  already partitioned into meta-characters. Since by construction  $Z[r]$ 's tail is a meta-character  $X_r$ , we know that  $X_r$  is a suffix of  $S[a, b]$ . Working backwards from  $X_r$  we obtain the sequence  $X_0 \cdots X_r$  of meta-characters covering  $S[a, b]$ . Note that it is possible that  $X_0$ , the meta-character containing  $S[a]$ , starts before  $S[a]$ . We thus define  $X'_0$  as the suffix of  $X_0$  starting at  $S[a]$  and define the meta-character partition of  $Z[q]$ 's head as  $X'_0 X_1 \cdots X_r$ . This process is depicted in Figure 3. Note that each meta-character is either the tail of some phrase or it is the suffix of a tail. We do not really compute the meta-characters but only use them in our analysis, as in the following result.

► **Lemma 4.** *Algorithm 1 computes  $select(t) = A[t]$  in  $\mathcal{O}(\log \lambda + M_{\max})$  time, where  $\lambda$  is the number of phrases in the  $LZ_\epsilon$  parsing and  $M_{\max}$  is the maximum number of meta-characters in a single phrase.*

**Proof.** The correctness of the algorithm follows by Lemma 3. To prove the time bound, observe that Line 2 clearly takes  $\mathcal{O}(\log \lambda)$  time. Let  $\ell$  denote the number of meta-characters between the one containing position  $t$  up to the end of  $Z[q]$ . We show by induction on  $\ell$  that  $SELECT-AUX(t, q)$  takes  $\mathcal{O}(\ell)$  time. If  $\ell = 1$ , then  $t$  belongs to  $Z[q]$ 's tail, and the value  $A[t]$  is retrieved in  $\mathcal{O}(1)$  time from  $TL[q]$  and  $TC[q]$ .

If  $\ell > 1$ , the algorithm retrieves the value  $A[t']$  from a previous phrase  $Z[q']$ , with  $q' = r - k$ , where  $k$  is the number of times Line 13 is executed. Since  $Z[q]$  meta-characters are induced by those in its source, we get that the number of meta-characters between the one containing  $t'$  and the end of  $Z[r]$  is  $\ell - 1$ , and the number of meta-characters between the one containing  $t'$  and the end of  $Z[q']$  is  $\ell' \leq \ell - 1 - k$ . By the inductive hypothesis, the call to  $SELECT-AUX(t', q')$  takes  $\mathcal{O}(\ell')$ , and the overall cost of  $SELECT-AUX(t, q)$  is  $\mathcal{O}(k) + \mathcal{O}(\ell') = \mathcal{O}(\ell)$ , as claimed. ◀

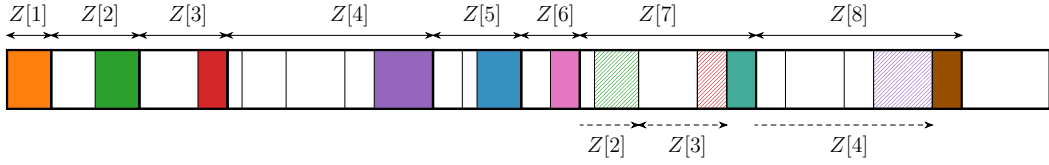
It is easy to see that for some input  $t$  Algorithm 1 takes  $\Theta(M_{\max})$  time. To reduce the complexity, we now show how to modify the parsing so that  $M_{\max}$  is upper bounded by a user-defined parameter  $M > 1$ . The resulting parsing could contain some repeated phrases, but note that Lemma 4 does not require the phrases to be different: repeated phrases will only affect the space usage.

**Algorithm 1** Recursive select procedure.

---

1:	<b>procedure</b> SELECT( $t$ )	
2:	$q \leftarrow$ the smallest $i$ such that $\text{PE}[i] \geq t$ , found via a binary search on PE	
3:	<b>return</b> SELECT-AUX( $t, q$ )	
4:	<b>procedure</b> SELECT-AUX( $t, q$ )	$\triangleright$ Invariant: $\text{PE}[q - 1] < t \leq \text{PE}[q]$
5:	<b>if</b> $t > \text{HE}[q]$ <b>then</b>	$\triangleright$ If $t$ belongs to the tail of $Z[q]$
6:	<b>return</b> $A[t]$	$\triangleright A[t]$ is computed from $\text{TL}[q], \text{TC}[q]$
7:	$r \leftarrow q' \leftarrow \text{HS}[q]$	$\triangleright$ The head of $Z[q]$ is a suffix of $Z[1] \cdots Z[r]$
8:	$j \leftarrow \text{HE}[q]$	$\triangleright j$ is the last position of the head of $Z[q]$
9:	$e \leftarrow \text{PE}[r]$	$\triangleright e$ is the last position of $Z[r]$
10:	$\Delta \leftarrow A[j] - A[e]$	$\triangleright \Delta$ can be computed in $\mathcal{O}(1)$ time by Lemma 3
11:	$t' \leftarrow t - (j - e)$ ;	$\triangleright A[t] = A[t'] + \Delta$ by Lemma 3
12:	<b>while</b> $t' > \text{PE}[q']$ <b>do</b>	$\triangleright$ Find the phrase $Z[\cdot]$ containing $t'$
13:	$q' \leftarrow q' - 1$	$\triangleright$ Go back one word
14:	<b>return</b> SELECT-AUX( $t', q'$ ) + $\Delta$	$\triangleright$ The returned value is $A[t]$ by Lemma 3

---



**Figure 4** The  $LZ_\epsilon$  parsing of the same string of Figure 3 with  $M = 5$ . The phrase  $Z[7]$  from Figure 3 is invalid since it has 13 meta-characters.  $Z[7]$  head can have at most 4 meta-characters, so we define  $Z[7]$  by setting  $\text{HS}[7] = 3$  (Step 2b). Next, we define  $Z[8]$  by setting  $\text{HS}[8] = 4$  (Step 2c).

To build a  $LZ_\epsilon$  parsing in which each phrase contains at most  $M$  meta-characters, we proceed as follows. Assuming  $S[1, i]$  has already been parsed as  $Z[1], \dots, Z[q - 1]$ , we first compute the longest prefix  $S[i + 1, j]$  which is a suffix of  $Z[1] \cdots Z[r]$  for some  $r < q$ . Let  $m$  denote the number of meta-characters in  $S[i + 1, j]$ . Then (see Figure 4):

1. If  $m < M$ , then  $Z[q]$  is defined as usual with  $\text{HS}[q] = r$ . Since  $Z[q]$ 's tails constitute an additional meta-character,  $Z[q]$  has  $m + 1 \leq M$  meta-characters, as required.
2. Otherwise, if  $m \geq M$ , we do the following.
  - a. We scan  $S[i + 1, j]$  backward dropping copies of  $Z[r], Z[r - 1], \dots$  until we are left with a prefix  $S[i + 1, k_s]$  containing less than  $M$  meta-characters. By construction,  $S[i + 1, k_s]$  is a suffix of  $Z[1] \cdots Z[s]$  for some  $s < r$  and since each phrase contains at most  $M$  meta-characters,  $S[i + 1, k_s]$  is non-empty.
  - b. We define  $Z[q]$  by setting  $S[i + 1, k_s]$  as its head,  $\text{HS}[q] = s$ , and by defining  $Z[q]$ 's tail as usual.
  - c. Next, we consider  $Z[s + 1] \equiv S[k_s, k_{s+1}]$ . By construction,  $Z[s + 1]$  contains at most  $M$  meta-characters while  $S[i + 1, k_{s+1}]$  contains more than  $M$  meta-characters. If  $Z[q]$  ends before position  $k_{s+1}$  (i.e.  $\text{PE}[q] < k_{s+1}$ ), we define an additional phrase  $Z[q + 1]$  with heads equal to  $S[\text{PE}[q] + 1, k_{s+1}]$ ,  $\text{HS}[q + 1] = s + 1$  and with a tail defined as usual. This ensures that  $Z[q]$  alone or  $Z[q]Z[q + 1]$  contains at least  $M$  meta-characters.

**Lemma 5.** *The  $LZ_\epsilon$  parsing with limit  $M$  contains at most  $2n/M$  repeated phrases.*

**Proof.** In the algorithm described above, repeated phrases are created only at Steps 2b and 2c. Indeed, both  $Z[q]$  defined in Step 2b and  $Z[q+1]$  defined in Step 2c could be identical to a previous phrase. However, the concatenation  $Z[q]Z[q+1]$  covers at least  $S[i+1, k_{s+1}]$  so by construction contains *at least*  $M$  meta-characters. Hence, Steps 2b and 2c can be executed at most  $n/M$  times. ◀

In the following, let  $\sigma$  denote the number of distinct gaps in  $S$  (i.e., the alphabet size of  $S$ ), for any  $\rho > 0$ , we denote by  $LZ_\epsilon^\rho$  the parsing computed with the above algorithm with  $M = \log^{1+\rho} n$ . The following lemma shows that the space to represent the parsing can be bounded in terms of the  $k$ th order entropy of the gap-string  $S$  plus  $o(n \log \sigma)$  bits.

► **Lemma 6.** *Let  $\sigma$  denote the number of distinct gaps in  $S$ . The arrays PE, HE, and HS produced by the  $LZ_\epsilon^\rho$  parsing can be stored in  $nH_k(S) + \mathcal{O}(n/\log^\rho n) + o(n \log \sigma)$  bits for any positive  $k = o(\log_\sigma n)$ , and still support constant-time access to their elements.*

**Proof.** Let  $\lambda$  denote the number of phrases in the parsing. We write  $\lambda = \lambda_r + \lambda_d$ , where  $\lambda_r$  is the number of repeated phrases, and  $\lambda_d$  is the number of distinct phrases. By Lemma 5 it is  $\lambda_r \leq n/(2 \log^{1+\rho} n)$ , while for the number  $\lambda_d$  of distinct phrases it is [27, Lemmas 3.9 and 3.10]

$$\lambda_d = \mathcal{O}\left(\frac{n}{\log_\sigma n}\right) \quad \text{and} \quad \lambda_d \log \lambda_d \leq nH_k(S) + \lambda_d \log \frac{n}{\lambda_d} + \mathcal{O}(\lambda_d(1 + k \log \sigma)) \quad (2)$$

for any constant  $k \geq 0$ . The vectors PE and HE contain  $\lambda$  increasing values in the range  $[1, n]$ . We encode each of them in  $\lambda \log \frac{n}{\lambda} + \mathcal{O}(\lambda)$  bits using Lemma 2. Since  $f(x) = x \log(n/x)$  is increasing for  $x \leq n/e$  and  $\lambda = \mathcal{O}(n/\log_\sigma n)$ , it is  $\lambda \log \frac{n}{\lambda} + \mathcal{O}(\lambda) = \mathcal{O}(n(\log \sigma)(\log \log n)/\log n) = o(n \log \sigma)$ .

We encode HS using  $\lambda$  cells of size  $\lceil \log \lambda \rceil = \log \lambda + \mathcal{O}(1)$  bits for a total of

$$\lambda_r \log(\lambda_r + \lambda_d) + \lambda_d \log(\lambda_r + \lambda_d) + \mathcal{O}(\lambda) \quad \text{bits.}$$

Since  $\lambda_d = \mathcal{O}(n/\log_\sigma n)$  and  $\lambda_r = \mathcal{O}(n/\log^{1+\rho} n)$ , it is  $\lambda_d + \lambda_r = \mathcal{O}(n/\log_\sigma n)$  and the first term is  $\mathcal{O}(n/\log^\rho n)$ . The second term can be bounded by noticing that, if  $\lambda_d \leq \lambda_r$ , the second term is smaller than the first. Otherwise, from (2) we have

$$\lambda_d \log(\lambda_r + \lambda_d) \leq \lambda_d \log(2\lambda_d) \leq nH_k(S) + \lambda_d \log \frac{n}{\lambda_d} + \mathcal{O}(\lambda_d(1 + k \log \sigma)).$$

By the same reasoning as above, we have  $\lambda_d \log \frac{n}{\lambda_d} = o(n \log \sigma)$  and  $\lambda_d(1 + k \log \sigma) = \mathcal{O}((nk \log \sigma)/\log_\sigma n) = o(n \log \sigma)$  for  $k = o(\log_\sigma n)$ . ◀

Combining Lemma 6 with 4 and recalling that  $\log \lambda = \mathcal{O}(\log^{1+\rho} n)$ , we get

► **Theorem 7.** *Let  $\sigma$  denote the number of distinct gaps in  $S$ . Using the  $LZ_\epsilon^\rho$  parsing we can compute  $\text{select}(t)$  in  $\mathcal{O}(\log^{1+\rho} n)$  time using  $nH_k(S) + \mathcal{O}(n/\log^\rho n) + o(n \log \sigma)$  bits of space plus the space used for the  $\lambda$  segments (array TL) and for the corrections of the elements in  $A$  covered by the tails in the parsing (array TC), for any positive  $k = o(\log_\sigma n)$ .*

In the proof of Lemma 6 one can see the interplay between the term  $\mathcal{O}(n/\log^\rho n)$  coming from the repeated phrases and the term  $o(n \log \sigma)$  coming from the distinct phrases in  $LZ_\epsilon^\rho$ . In particular, if  $\sigma$  is small (i.e., there are few distinct gaps), then  $o(n \log \sigma)$  becomes  $\mathcal{O}(n \log \log n/\log n)$  and the space bound turns out to be  $nH_k(S) + \mathcal{O}(n/\log^\rho n)$  bits. Also, note that the number of segments  $\lambda$  in  $LZ_\epsilon^\rho$  is always smaller than the number of segments in a plain LA-vector. Also, the total length of the  $LZ_\epsilon^\rho$  tails is always smaller than  $n$ . Hence, our approach is no worse than the LA-vector in space.



We now show that the  $LZ_\varepsilon^\rho$  parsing support efficient rank queries. The starting point is the following lemma, whose proof is analogous to the one of Lemma 3.

► **Lemma 8.** *Let  $S[i+1, j]$  denote the head of phrase  $Z[q]$ , and let  $r = \text{HS}[q]$  and  $e = \text{PE}[r]$ . Then, for any  $v$  such that  $A[i] < v \leq A[j]$ , it holds  $\text{rank}(v) = \text{rank}(v - (A[j] - A[e])) + (j - e)$ .*

► **Theorem 9.** *Using the  $LZ_\varepsilon^\rho$  parsing we can compute  $\text{rank}(v)$  in  $\mathcal{O}(\log^{1+\rho} n + \log \varepsilon)$  time within the space stated in Theorem 7.*

**Proof.** We answer  $\text{rank}(v)$  with an algorithm similar to Algorithm 1. First, we compute the index  $q$  of the phrase  $Z[q]$  such that  $A[\text{PE}[q-1]] < v \leq A[\text{PE}[q]]$  with a binary search on the values  $A[\text{PE}[i]]$ . If the parsing has  $\lambda$  phrases this takes  $\mathcal{O}(\log \lambda)$  time, since we can retrieve  $A[\text{PE}[i]]$  in constant time using  $\text{PE}[i]$ ,  $\text{TL}[i]$  and  $\text{TC}[i]$ .

Next, we set  $j = \text{HE}[q]$  and compare  $v$  with  $A[j]$  (which again we can retrieve in constant time since it is the first value covered by  $Z[q]$ 's tail). If  $v \geq A[j]$ , we return  $j$  plus the rank of  $v$  in  $A[j, \text{PE}[q]]$ , which we can compute in  $\mathcal{O}(\log \varepsilon)$  time from  $\text{TL}[q]$  and  $\text{TC}[q]$  using the algorithm in [7, §4]. If  $v < A[j]$ , we set  $e = \text{PE}[\text{HS}[q]]$  and compute  $\text{rank}(v)$  recursively using Lemma 8. Before the recursive call, we need to compute the index  $q'$  of the phrase such that  $A[\text{PE}[q'-1]] < v' \leq A[\text{PE}[q']]$ , for  $v' = v - (A[j] - A[e])$ . To this end, we execute the same while loop as the one in Lines 12–13 of Algorithm 1 with the test  $t' > \text{PE}[q']$  replaced by  $v' > A[\text{PE}[q']]$ . Reasoning as in the proof of Lemma 4, we get that the overall time complexity is  $\mathcal{O}(\log \lambda + M_{\max} + \log \varepsilon) = \mathcal{O}(\log^{1+\rho} n + \log \varepsilon)$ . ◀

#### 4 The block- $\varepsilon$ tree

In this section, we design a repetition aware version of the LA-vector by following an approach that focuses on query efficiency and uses space bounded in terms of the complexity measure  $\delta$  reviewed in Section 2. We do so by building a variant of the block tree [5] on a combination of the gap-string  $S$  and the piecewise linear  $\varepsilon$ -approximation. We name this variant block- $\varepsilon$  tree, and show that it achieves time-space bounds which are competitive with the ones achieved by block trees and LA-vectors [7] because it combines both forms of compressibility discussed in this paper: repetitiveness and approximate linearity.

The main idea of the block- $\varepsilon$  tree consists in first building a traditional block tree structure over the gap-string  $S[1, n]$  of  $A$ . Recall that every node of the block tree represents a substring of  $S$ , and thus it implicitly represents the corresponding subarray of  $A$ . Then, we prune the tree by dropping the subtrees whose corresponding subarray of  $A$  can be covered more succinctly by segments and corrections (i.e. whose LA-vector representation wins over the block-tree representation). Note that, compared to LA-vector, we do not encode segments and corrections corresponding to substrings of  $S$  that have been encountered earlier, that is, we exploit the repetitiveness of  $S$  to compress the piecewise linear  $\varepsilon$ -approximation at the core of LA-vector. On the other hand, compared to block trees, we drop subtrees whose substrings can be encoded more efficiently by segments and corrections, that is, we exploit the approximate linearity of subarrays of  $A$ . Below we detail how to orchestrate this interplay to achieve efficient queries and compressed space occupancy in the block- $\varepsilon$  tree.

For simplicity of exposition, assume that  $n = \delta 2^h$  for some integer  $h$ , where  $\delta$  is the string complexity of  $S$ . The block- $\varepsilon$  tree is organised into  $h' \leq h$  levels. The first level (level zero) logically divides the string  $S$  into  $\delta$  blocks of size  $s_0 = n/\delta$ . In general, blocks at level  $\ell$  have size  $s_\ell = n/(\delta 2^\ell)$ , because they are recursively halved until possibly reaching the last level  $h = \log \frac{n}{\delta}$ , where blocks have size  $s_h = 1$ .

At any level, if two blocks  $S_q$  and  $S_{q+1}$  are consecutive in  $S$  and they form the leftmost occurrence in  $S$  of their content, then we say that both  $S_q$  and  $S_{q+1}$  are marked. A marked block  $S_q$  that is not in the last level becomes an internal node of the tree. Such an internal node has two children corresponding to the two equal-size sub-blocks in which  $S_q$  is split into. On the other hand, an unmarked block  $S_r$  becomes a leaf of the tree because, by construction, its content occurs earlier in  $S$  and thus we can encode it by storing (i) a leftward pointer  $q$  to the marked blocks  $S_q, S_{q+1}$  at the same level  $\ell$  containing its leftmost occurrence, taking  $\log \frac{n}{s_\ell}$  bits; (ii) the offset  $o$  of  $S_r$  into the substring  $S_q \cdot S_{q+1}$ , taking  $\log s_\ell$  bits. Furthermore, to recover the values of  $A$  corresponding to  $S_r$ , we store (iii) the difference  $\Delta$  between the value of  $A$  corresponding to the beginning of  $S_r$  and the value of  $A$  at the pointed occurrence of  $S_r$ , taking  $\log u$  bits. Overall, each unmarked block needs  $\log n + \log u$  bits of space.

To describe the pruning process, we first define a cost function  $c$  on the nodes of the block- $\varepsilon$  tree. For an unmarked block  $S_r$ , we define the cost  $c(S_r) = \log n + \log u$ , which accounts for the space in bits taken by  $q$ ,  $o$  and  $\Delta$ . For a marked block  $S_q$  at the last level  $h$ , we define the cost  $c(S_q) = \log u$ , which accounts for the space in bits taken by its single corresponding element of  $A$ . Instead, consider a marked block  $S_q$  at level  $\ell < h$  for which there exists a segment approximating with error  $\varepsilon_q \leq \varepsilon$  the corresponding elements of  $A$ . Suppose  $\varepsilon_q$  is minimal, that is, there is no  $\varepsilon' < \varepsilon_q$  such that there exists a segment  $\varepsilon'$ -approximating those same elements of  $A$ . Let  $\kappa$  be the space in bits taken by the parameters  $\langle \alpha, \beta \rangle$  of the segment, e.g.  $\kappa = 2 \log u + \log n$  if we encode  $\beta$  in  $\log u$  bits and  $\alpha$  as a rational number with a  $\log u$ -bit numerator and a  $\log n$ -bit denominator [7, §2]. We assign to such  $S_q$  a cost  $c(S_q)$  defined recursively as

$$c(S_q) = \min \begin{cases} \kappa + s_\ell \log \varepsilon_q + \log \log u \\ 2 \log n + \sum_{S_x \in \text{child}(S_q)} c(S_x) \end{cases} \quad (3)$$

The first branch of Equation (3) accounts for an encoding of the subarray of  $A$  corresponding to  $S_q$  via an  $\varepsilon_q$ -approximate segment, the corrections of  $\log \varepsilon_q$  bits for each of the  $s_\ell$  elements in  $S_q$ , and the exponent  $y$  of  $\varepsilon_q = 2^y - 1$  to keep track of its value, respectively. The second branch of Equation (3) accounts for an encoding that recursively splits  $S_q$  into two children, i.e. an encoding via two  $\log n$ -bit pointers plus the optimal cost of the children. Finally, if there is no linear  $\varepsilon$ -approximation (and thus no  $\varepsilon_q$ -approximation with  $\varepsilon_q \leq \varepsilon$ ) for  $S_q$ , we assign to such  $S_q$  the cost indicated in the second branch of Equation (3).

A postorder traversal of the block- $\varepsilon$  tree is sufficient to assign a cost to its nodes and possibly prune some of its subtrees. Specifically, after recursing on the two children of a marked block  $S_q$  at level  $\ell$ , we check if the first branch of Equation (3) gives the minimum. In that case, we prune the subtree rooted at  $S_q$  and store instead the encoding of the block via the parameters  $\langle \alpha, \beta \rangle$  and the  $s_\ell$  corrections in an array  $C_q$ . As a technical remark, this pruning requires fixing the destination of any leftward pointer that starts from an unmarked block  $S_r$  and ends to a (pruned) descendant of  $S_q$ . For this purpose, we first make  $S_r$  pointing to  $S_q$ . Then, since any leftward pointer points to a *pair* of marked blocks (unless the offset is zero), both or just one of them belongs to the pruned subtree. In the second case, we require an additional pointer from  $S_r$  to the block that does not belong to the pruned subtree. This additional pointer does not change the asymptotic complexity of the structure. Overall, this pruning process yields a tree with  $h' \leq h$  levels.

In the full paper, we show how to support **rank** and **select** queries on the block- $\varepsilon$  tree in worst-case  $\mathcal{O}(\log \log_w \frac{u}{\delta} + h' + \log \varepsilon)$  time and  $\mathcal{O}(h')$  time, respectively.

We observe that the block- $\varepsilon$  tree achieves space-time complexities no worse than a standard block tree construction on  $S$ . This is due to the pruning of subtrees guided by the space-conscious cost function  $c(\cdot)$  and by the resulting reduction in the number of levels,

which positively impact the query time. Compared to LA-vector, the block- $\varepsilon$  tree can take advantage of repetitions and avoid the encoding of subarrays of  $A$  corresponding to repeated substrings of  $S$ . Furthermore, since the block- $\varepsilon$  tree allocates the most succinct encoding for a subarray of  $A$  by considering the smallest  $\varepsilon_q \leq \varepsilon$  giving a linear  $\varepsilon_q$ -approximation, it could be regarded as the repetition-aware analogous of the space-optimised LA-vector [7, §6], in which all values of  $\varepsilon = 0, 2^0, 2^1, \dots, 2^{\log u}$  are considered. The block- $\varepsilon$  tree has the advantage of potentially storing fewer corrections at the cost of storing the tree topology. Using the straightforward pointer-based encoding we discussed above, the tree topology takes  $\mathcal{O}(\delta \log \frac{n}{\delta} \log n)$  bits in the worst case, but for the next section we implement a more succinct pointerless encoding (details in the full paper). We notice, nonetheless, that the more repetitive is the string  $S$ , the smaller is  $\delta$ , thus the overhead of the tree topology gets negligible.

Finally, we mention that the block- $\varepsilon$  tree could employ other compressed rank-select dictionaries in its nodes, yielding a hybrid compression approach [39] that can benefit from the orchestration of bicriteria optimisation and proper pruning of its topology to achieve the best space occupancy, given a bound on the query time, or vice versa (à la [12, 17, 39]).

## 5 Experiments

We implemented the block- $\varepsilon$  tree, as it is the simplest and most practical contribution of this paper.

We compare our block- $\varepsilon$  tree with the block tree of [5], built on the characteristic bitvector  $\text{bv}(A)$ , and with the space-optimised LA-vector of [7]. All these implementations are written in C++ and build on the `sds1` library [19]. For both the block tree and the block- $\varepsilon$  tree, we use a branching factor of two and vary the length  $b$  of the last-level blocks as  $b \in \{2^3, 2^4, \dots, 2^9\}$ . Due to space limitations, we do not show the full space-time trade-off of each structure but report only the most space-efficient configurations. A comparison with other rank/select dictionaries is beyond the scope of our work, and it was already investigated in the literature for the individual LA-vector and the block tree [5, 7]. On the other hand, we note that our experimental study is the first to compare LA-vectors and block trees.

As datasets, we use (i) three postings lists with different densities  $n/u$  from the GOV2 inverted index [39]; (ii) six integers lists obtained by enumerating the positions of the first, second and third most frequent character in each of the Burrows-Wheeler transform of two text files: URL and 5GRAM [7]; (iii) three integers lists obtained by enumerating, respectively, the positions of both Ts and Gs, of Ts, and of Gs in the Burrows-Wheeler transform of the first gigabyte of the human reference genome GRCh38.p13.

For each tested structure, query operation, and dataset, we generate a batch of  $10^5$  random queries and measure the average query time in nanoseconds and the space occupancy of the structure in bits per integer on a machine with 202 GB of RAM and a 2.30 GHz Intel Xeon Gold 5118 CPU.

Table 1 shows the results. First and foremost, we note that LA-vector is  $10.51\times$  faster in select and  $4.69\times$  faster in rank than the block tree on average, while for space there is no clear winner over all the datasets. This comparison, which was not known in the literature, illustrates that the combination of approximate linearity and repetitiveness is interesting not only from a theoretical point of view, as commented in the introduction, but also from a practical point of view.

Let us now compare the performance of our block- $\varepsilon$  tree against the block tree and the LA-vector. The block- $\varepsilon$  tree is  $2.19\times$  faster in select than the block tree, and it is either faster ( $1.32\times$ ) or slower ( $1.27\times$ ) in rank. With respect to LA-vector, the block- $\varepsilon$  tree is always

■ **Table 1** The performance of LA-vector, the block tree over the characteristic bitvector  $\text{bv}(A)$  and the block- $\varepsilon$  tree over twelve datasets of different size  $n$  and universe size  $u$ . The **select** and **rank** columns show the average query time of the operations in nanoseconds. The space of each structure is shown in Bits Per Integer (BPI). For the block tree and the block- $\varepsilon$  tree, the value  $b$  denotes the length of the last-level block that gave the most space-efficient configuration.

Dataset			LA-vector			Block tree on $\text{bv}(A)$					Block- $\varepsilon$ tree on $S$				
Name ( $n/u$ )	$n/10^6$	$u/10^6$	select	rank	BPI	$b$	select	rank	BPI	Depth	$b$	select	rank	BPI	Depth (Avg)
GOV2 (76.6%)	18.85	24.62	69	130	1.85	64	668	519	<b>0.69</b>	12	16	451	825	1.89	14 (9.98)
GOV2 (40.6%)	9.85	24.62	60	129	3.48	128	686	531	<b>1.56</b>	11	256	367	638	3.26	10 (8.73)
GOV2 (4.1%)	1.00	24.62	33	96	3.01	32	645	573	4.62	13	128	407	465	<b>2.92</b>	10 (9.73)
URL (5.6%)	57.98	1039.92	124	144	2.83	32	1017	733	<b>2.58</b>	18	16	762	909	3.41	16 (12.94)
URL (1.3%)	13.56	1039.91	98	123	<b>6.34</b>	32	987	753	8.57	18	32	463	664	7.32	10 (8.39)
URL (0.4%)	3.73	1039.86	34	87	<b>1.28</b>	32	831	783	1.84	19	16	400	553	1.51	11 (7.92)
5GRAM (9.8%)	145.40	1476.73	171	249	4.40	32	1176	876	<b>3.64</b>	18	32	621	999	5.01	12 (10.27)
5GRAM (2.0%)	29.20	1476.73	132	177	<b>6.37</b>	32	1143	863	8.80	18	64	483	733	6.96	9 (7.81)
5GRAM (0.8%)	11.22	1476.69	95	125	<b>7.56</b>	32	1017	826	11.25	19	64	421	592	8.34	9 (7.61)
DNA (49.0%)	490.10	1000.00	250	446	5.27	512	1158	922	<b>2.09</b>	14	512	535	1070	3.65	3 (2.98)
DNA (29.5%)	294.68	1000.00	218	416	6.20	512	1227	989	<b>3.46</b>	14	512	368	718	4.57	2 (1.96)
DNA (19.6%)	195.42	1000.00	195	384	6.69	512	1206	972	5.21	14	512	335	654	<b>5.01</b>	2 (1.94)

slower. But, for what concerns the space, the block- $\varepsilon$  tree improves both the LA-vector and the block tree in the sparsest GOV2 and DNA, and in the vast majority of the remaining datasets it is the second-best structure for space occupancy (except for the densest GOV2, URL and 5GRAM). This shows that space-wise, the block- $\varepsilon$  tree can be a robust data structure in that it often achieves a good compromise by exploiting both kinds of regularities: repetitiveness (block trees) and approximate linearity (LA-vectors).

For future work, we believe the block- $\varepsilon$  tree can be improved along at least two avenues. First, the block- $\varepsilon$  tree at a certain level is constrained to use fixed-length blocks (and thus segments), whilst the LA-vector minimises its space occupancy using segments whose start/end positions do not have to coincide with a subdivision in blocks. Removing this limitation, inherited from the block tree, would help to better capture approximate linearity and improve the space occupancy of the block- $\varepsilon$  tree. Second, the block- $\varepsilon$  tree captures the repetitiveness of the gap string  $S$ , while for the densest datasets of Table 1 it appears worthwhile to consider the repetitiveness in  $\text{bv}(A)$ , as done by the block tree. Therefore, adapting our pruning strategy to  $\text{bv}(A)$  is likely to improve the space occupancy in these densest datasets (though, the space-time bounds will then depend on  $u$  instead of  $n$ ).

## 6 Conclusions

We introduced novel compressed rank/select dictionaries by exploiting two sources of regularity arising in real data: repetitiveness and approximate linearity. Our first contribution, the  $LZ_\varepsilon^p$  parsing, combines backward copies with linear  $\varepsilon$ -approximation thus supporting efficient queries within a space complexity bounded by the  $k$ th order entropy of the gaps in the input data. Our second contribution, the block- $\varepsilon$  tree, is a structure that adapts smoothly to both sources of regularities by offering an improved query-time efficiency compared to  $LZ_\varepsilon^p$ . We experimented with a preliminary implementation of the block- $\varepsilon$  tree showing that it effectively exploits both repetitiveness and approximate linearity.

Our study opens up a plethora of opportunities for future research. Firstly, we notice that the PGM-index [17] is also based on a variant of the piecewise linear  $\varepsilon$ -approximation, and thus it can still benefit from the ideas presented in this paper to make its space occupancy repetition aware. Secondly, the compression of segments and corrections in both  $LZ_\varepsilon^p$  and

the block- $\varepsilon$  tree is an orthogonal problem for which one can devise further compression mechanisms (see e.g. [17, Theorem 3]). Thirdly, the construction of the  $LZ_\varepsilon^p$  phrases and the block- $\varepsilon$  tree could be investigated inside a bicriteria framework, which seeks to optimise the query time and space usage under some given constraints [11]. Finally, inspired by our preliminary results, we plan to engineer a more query-efficient implementation of the block- $\varepsilon$  tree that computes an optimal node pruning using a family of compressed data structures in addition to  $\varepsilon$ -approximate segments.

---

## References

- 1 Rachit Agarwal, Anurag Khandelwal, and Ion Stoica. Succinct: enabling queries on compressed data. In *Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.
- 2 Diego Arroyuelo and Rajeev Raman. Adaptive succinctness. In *Proc. 26th International Symposium on String Processing and Information Retrieval (SPIRE)*, 2019.
- 3 Jeremy Barbay and Gonzalo Navarro. Compressed representations of permutations, and applications. In *Proc. 26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2009.
- 4 Djamel Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Theory and practice of monotone minimal perfect hashing. *ACM Journal of Experimental Algorithmics*, 16, 2008.
- 5 Djamel Belazzougui, Manuel Cáceres, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez, Simon J. Puglisi, and Yasuo Tabei. Block trees. *Journal of Computer and System Sciences*, 117:1–22, 2021.
- 6 Djamel Belazzougui, Patrick Hagge Cording, Simon J. Puglisi, and Yasuo Tabei. Access, rank, and select in grammar-compressed strings. In *Proc. 23rd Annual European Symposium on Algorithms (ESA)*, pages 142–154, 2015.
- 7 Antonio Boffa, Paolo Ferragina, and Giorgio Vinciguerra. A “learned” approach to quicken and compress rank/select dictionaries. In *Proc. 23rd SIAM Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 46–59, 2021.
- 8 David Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, Canada, 1996.
- 9 Peter Elias. Efficient storage and retrieval by content and address of static files. *Journal of the ACM*, 21(2):246–260, 1974.
- 10 Robert Mario Fano. *On the number of bits required to implement an associative memory. Memo 61*. Massachusetts Institute of Technology, Project MAC, 1971.
- 11 Andrea Farruggia, Paolo Ferragina, Antonio Frangioni, and Rossano Venturini. Bicriteria data compression. *SIAM Journal on Computing*, 48(5):1603–1642, 2019.
- 12 Paolo Ferragina, Raffaele Giancarlo, and Giovanni Manzini. The myriad virtues of wavelet trees. *Information and Computation*, 207(8):849–866, 2009.
- 13 Paolo Ferragina, Stefan Kurtz, Stefano Lonardi, and Giovanni Manzini. Computational biology. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 59. CRC Press, 2nd edition, 2018.
- 14 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. On the performance of learned data structures. *Theoretical Computer Science*, 871:107–120, 2021.
- 15 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.
- 16 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theoretical Computer Science*, 372(1):115–121, 2007.
- 17 Paolo Ferragina and Giorgio Vinciguerra. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *PVLDB*, 13(8):1162–1175, 2020.
- 18 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67(1), 2020.

- 19 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: plug and play with succinct data structures. In *Proc. 13th International Symposium on Experimental Algorithms (SEA)*, pages 326–337, 2014.
- 20 Simon Gog, Juha Kärkkäinen, Dominik Kempa, Matthias Petri, and Simon J. Puglisi. Fixed block compression boosting in FM-indexes: Theory and practice. *Algorithmica*, 81(4):1370–1391, 2019.
- 21 Alexander Golynski, Alessio Orlandi, Rajeev Raman, and S. Srinivasa Rao. Optimal indexes for sparse bit vectors. *Algorithmica*, 69(4):906–924, 2014.
- 22 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.
- 23 John L. Hennessy and David A. Patterson. A new golden age for computer architecture. *Communications of the ACM*, 62(2):48–60, 2019.
- 24 Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Hybrid compression of bitvectors for the FM-index. In *Proc. 24th Data Compression Conference (DCC)*, pages 302–311, 2014.
- 25 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In *Proc. 14th Latin American Symposium on Theoretical Informatics (LATIN)*, 2020.
- 26 Sambasiva Rao Kosaraju and Giovanni Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 1999.
- 27 Sebastian Kreft and Gonzalo Navarro. On compressing and indexing repetitive sequences. *Theoretical Computer Science*, 483:115–133, 2013.
- 28 Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.
- 29 Veli Mäkinen, Djamel Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design*. Cambridge University Press, 2015.
- 30 Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332–347, 2007.
- 31 J. Ian Munro. Tables. In *Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 1996.
- 32 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. 38th Annual Symposium on Foundations of Computer Science (FOCS)*, 1997.
- 33 Gonzalo Navarro. Spaces, trees, and colors: the algorithmic landscape of document retrieval on sequences. *ACM Computing Surveys*, 46(4), 2014.
- 34 Gonzalo Navarro. *Compact data structures: a practical approach*. Cambridge University Press, 2016.
- 35 Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Computing Surveys*, 54(2), 2020.
- 36 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007.
- 37 Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proc. 9th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2007.
- 38 Joseph O’Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 24(9):574–578, 1981.
- 39 Giuseppe Ottaviano, Nicola Tonellotto, and Rossano Venturini. Optimal space-time tradeoffs for inverted indexes. In *Proc. 8th ACM International Conference on Web Search and Data Mining (WSDM)*, 2015.
- 40 Mihai Pătraşcu. Succincter. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008.



## 64:16 Repetition- and Linearity-Aware Rank/Select Dictionaries

- 41 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding  $k$ -ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.
- 42 Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam Smith. Sublinear algorithms for approximating string compressibility. *Algorithmica*, 65(3):685–709, 2013.
- 43 Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1230–1239, 2006.
- 44 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.



# Pattern Masking for Dictionary Matching

Panagiotis Charalampopoulos ✉ 

The Interdisciplinary Center Herzliya, Israel

Huiping Chen ✉ 

King's College London, UK

Peter Christen ✉ 

Australian National University, Canberra, Australia

Grigorios Loukides ✉ 

King's College London, UK

Nadia Pisanti ✉ 

University of Pisa, Italy

Solon P. Pissis ✉ 

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski ✉ 

University of Warsaw, Poland

---

## Abstract

Data masking is a common technique for sanitizing sensitive data maintained in database systems, and it is also becoming increasingly important in various application areas, such as in record linkage of personal data. This work formalizes the Pattern Masking for Dictionary Matching (PMDM) problem. In PMDM, we are given a dictionary  $\mathcal{D}$  of  $d$  strings, each of length  $\ell$ , a query string  $q$  of length  $\ell$ , and a positive integer  $z$ , and we are asked to compute a smallest set  $K \subseteq \{1, \dots, \ell\}$ , so that if  $q[i]$  is replaced by a wildcard for all  $i \in K$ , then  $q$  matches at least  $z$  strings from  $\mathcal{D}$ . Solving PMDM allows providing data utility guarantees as opposed to existing approaches.

We first show, through a reduction from the well-known  $k$ -Clique problem, that a decision version of the PMDM problem is NP-complete, even for strings over a binary alphabet. We thus approach the problem from a more practical perspective. We show a combinatorial  $\mathcal{O}((d\ell)^{|K|/3} + d\ell)$ -time and  $\mathcal{O}(d\ell)$ -space algorithm for PMDM for  $|K| = \mathcal{O}(1)$ . In fact, we show that we cannot hope for a faster combinatorial algorithm, unless the combinatorial  $k$ -Clique hypothesis fails [Abboud et al., SIAM J. Comput. 2018; Lincoln et al., SODA 2018]. We also generalize this algorithm for the problem of masking multiple query strings simultaneously so that every string has at least  $z$  matches in  $\mathcal{D}$ .

Note that PMDM can be viewed as a generalization of the decision version of the dictionary matching with mismatches problem: by querying a PMDM data structure with string  $q$  and  $z = 1$ , one obtains the minimal number of mismatches of  $q$  with any string from  $\mathcal{D}$ . The query time or space of all known data structures for the *more restricted* problem of dictionary matching with at most  $k$  mismatches incurs some exponential factor with respect to  $k$ . A simple exact algorithm for PMDM runs in time  $\mathcal{O}(2^\ell d)$ . We present a data structure for PMDM that answers queries over  $\mathcal{D}$  in time  $\mathcal{O}(2^{\ell/2}(2^{\ell/2} + \tau)\ell)$  and requires space  $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2}d)$ , for any parameter  $\tau \in [1, d]$ .

We complement our results by showing a two-way polynomial-time reduction between PMDM and the Minimum Union problem [Chlamtáč et al., SODA 2017]. This gives a polynomial-time  $\mathcal{O}(d^{1/4+\epsilon})$ -approximation algorithm for PMDM, which is tight under a plausible complexity conjecture.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Pattern matching

**Keywords and phrases** string algorithms, dictionary matching, wildcards, record linkage, query term dropping

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.65

**Related Version** *Extended Version*: <https://arxiv.org/abs/2006.16137>



© Panagiotis Charalampopoulos, Huiping Chen, Peter Christen, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, and Jakub Radoszewski;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 65; pp. 65:1–65:19

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



**Funding** This paper is part of the PANGAIA project that has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 872539. This paper is also part of the ALPACA project that has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 956229.

*Panagiotis Charalampopoulos*: Supported by the Israel Science Foundation grant 592/17.

*Huiping Chen*: Supported by a CSC Scholarship.

*Grigorios Loukides*: Supported in part by the Leverhulme Trust RPG-2019-399 project.

*Nadia Pisanti*: Supported by the University of Pisa under the “PRA – Progetti di Ricerca di Ateneo” (Institutional Research Grants) – Project no. PRA\_2020-2021\_26.

*Jakub Radoszewski*: Supported by the Polish National Science Center, grant number 2018/31/D/ST6/03991.

## 1 Introduction

Let us start with a true incident to illustrate the essence of the computational problem formalized in this work. In the Netherlands, water companies bill the non-drinking and drinking water separately. The 6th author of this paper had direct debit for the former but not for the latter. When he tried to set up the direct debit for the latter, he received the following masked message by the company:

```
Is this you?
Initial: S. Name: P***** E-mail address: s*****13@g***1.com
Bank account number: NL10RABO*****11.
```

The rationale of the data masking is: the client should be able to identify themselves, to help the companies *link* the client’s profiles, but not infer the identity of any other client, so that clients’ privacy is preserved. Thus, the masked version of the data is required to conceal as few symbols as possible but correspond to a sufficient number of other clients.

This requirement can be formalized as the Pattern Masking for Dictionary Matching (PMDM) problem: Given a dictionary  $\mathcal{D}$  of  $d$  strings, each of length  $\ell$ , a query string  $q$  of length  $\ell$ , and a positive integer  $z$ , PMDM asks to compute a smallest set  $K \subseteq \{1, \dots, \ell\}$ , so that if  $q[i]$ , for all  $i \in K$ , is replaced by a wildcard,  $q$  matches at least  $z$  strings from  $\mathcal{D}$ . The PMDM problem applies data masking, a common operation to sanitize personal data maintained in database systems [1, 26, 56]. In particular, PMDM lies at the heart of record linkage of databases containing personal data [23, 39, 40, 52, 59, 61], which is the main application we consider in this work.

*Record linkage* is the task of identifying records that refer to the same entities across databases, in situations where no entity identifiers are available in these databases [22, 32, 47]. This task is of high importance in various application domains featuring personal data, ranging from the health sector and social science research, to national statistics and crime and fraud detection [23, 36]. In a typical setting, the task is to link two databases that contain names or other attributes, known collectively as quasi-identifiers (QIDs) [60]. The similarity between each pair of records (a record from one of the databases and a record from the other) is calculated with respect to their values in QIDs, and then all compared record pairs are classified into matches (the pair is assumed to refer to the same person), non-matches (the two records in the pair are assumed to refer to different people), and potential matches (no decision about whether the pair is a match or non-match can be made) [22, 32]. Unfortunately, potential matches happen quite often [9]. A common approach [52, 59] to deal with potential matches is to conduct a manual clerical review, where a domain expert

looks at the attribute values in record pairs and then makes a manual match or non-match decision. At the same time, to comply with policies and legislation, one needs to prevent domain experts from inferring the identity of the people represented in the manually assessed record pairs [52]. The challenge is to achieve desired data protection/utility guarantees; i.e. enabling a domain expert to make good decisions without inferring peoples' identities.

To address this challenge, we can solve PMDM twice, for a potential match  $(q_1, q_2)$ . The first time we use as input the query string  $q_1$  and a reference dictionary (database)  $\mathcal{D}$  containing personal records from a sufficiently large population (typically, much larger than the databases to be linked). The second time, we use as input  $q_2$  instead of  $q_1$ . Since each masked  $q$  derived by solving PMDM matches at least  $z$  records in  $\mathcal{D}$ , the domain expert would need to distinguish between at least  $z$  individuals in  $\mathcal{D}$  to be able to infer the identity of the individual corresponding to the masked string. The underlying assumption is that  $\mathcal{D}$  contains one record per individual. Also, some wildcards from one masked string can be superimposed on another to ensure that the expert does not gain more knowledge from combining the two strings, and the resulting strings would still match at least  $z$  records in  $\mathcal{D}$ . Thus, by solving PMDM in this setting, we provide privacy guarantees alike  $z$ -map [57]; a variant of the well-studied  $z$ -anonymity [54] privacy model.<sup>1</sup> In  $z$ -map, each record of a dataset must match at least  $z$  records in a reference dataset, from which the dataset is derived. In our setting, we consider a pattern that is not necessarily contained in the reference dataset. Offering such privacy is desirable in real record linkage systems where databases containing personal data are being linked [23, 40, 61]. On the other hand, since each masked  $q$  contains the minimum number of wildcards, the domain expert is still able to use the masked  $q$  to meaningfully classify a record pair as a match or as a non-match. Offering such utility is again desirable in record linkage systems [52]. Record linkage is an important application for our techniques, because no existing approach can provide privacy and utility guarantees when releasing linkage results to domain experts [41]. In particular, existing approaches [40, 41] recognize the need to offer privacy by preventing the domain expert from distinguishing between a small number of individuals, but they provide *no algorithm* for offering such privacy, let alone an algorithm offering utility guarantees as we do.

A secondary application where PMDM is of importance is *query term dropping*, an information retrieval task that seeks to drop keywords (terms) from a query, so that the remaining keywords retrieve a sufficiently large number of documents. This task is performed by search engines, such as Google [8], and by e-commerce platforms such as e-Bay [42], to improve users' experience [29, 58] by making sufficiently many search results available to users. For example, e-Bay applies query term dropping, removing one term, in our test query:

```
Query: vacuum database cleaner
Query results: 0 results found for vacuum database cleaner
               42 results found for vacuum cleaner
```

We could perform query term dropping by solving PMDM in a setting where strings in a dictionary correspond to document terms and a query string corresponds to a user's query. Then, we provide the user with the masked query, after removing all wildcards, and with its matching strings from the dictionary. Query term dropping is a relevant application for our techniques, because existing techniques [58] do not minimize the number of dropped terms. Rather, they drop keywords randomly, which may unnecessarily shorten the query,

<sup>1</sup> The notation used for such privacy models is generally  $k$  instead of  $z$ , e.g.  $k$ -anonymity [55, 57].

or drop keywords based on custom rules, which is not sufficiently generic to deal with all queries. More generally, our techniques can be applied to drop terms from any top- $z$  database query [33] to ensure there are  $z$  results in the query answer.

**Related Algorithmic Work.** Let us denote the wildcard symbol by  $\star$  and provide a brief overview of works related to PMDM, the main problem considered in this paper.

- *Partial Match:* Given a dictionary  $\mathcal{D}$  of  $d$  strings over an alphabet  $\Sigma = \{0, 1\}$ , each of length  $\ell$ , and a string  $q$  over  $\Sigma \sqcup \{\star\}$  of length  $\ell$ , the problem asks whether  $q$  matches any string from  $\mathcal{D}$ . This is a well-studied problem [14, 18, 35, 46, 50, 51, 53]. Patrascu [50] showed that any data structure for the Partial Match problem with cell-probe complexity  $t$  must use space  $2^{\Omega(\ell/t)}$ , assuming the word size is  $\mathcal{O}(d^{1-\epsilon}/t)$ , for any constant  $\epsilon > 0$ . The key difference to PMDM is that the wildcard positions in the query strings are fixed.
- *Dictionary Matching with  $k$ -errors:* A similar line of research to that of Partial Match has been conducted under the Hamming and edit distances, where, in this case,  $k$  is the maximum allowed distance between the query string and a dictionary string [11, 12, 15, 17, 25, 64]. The structure of Dictionary Matching with  $k$ -errors is very similar to Partial Match as each wildcard in the query string gives  $|\Sigma|$  possibilities for the corresponding symbol in the dictionary strings. On the other hand, in Partial Match the wildcard positions are fixed. The PMDM problem is a generalization of the decision version of the Dictionary Matching with  $k$ -errors problem (under Hamming distance): by querying a data structure for PMDM with string  $q$  and  $z = 1$ , one obtains the minimum number of mismatches of  $q$  with any string from  $\mathcal{D}$ , which suffices to answer the decision version of the Dictionary Matching with  $k$ -errors problem. The query time or space of all known data structures for Dictionary Matching with  $k$ -mismatches incurs some exponential factor with respect to  $k$ . In [24], Cohen-Addad et al. showed that, in the pointer machine model, for the reporting version of the problem, one cannot avoid exponential dependency on  $k$  either in the space or in the query time. In the word-RAM model, Rubinfeld showed that, conditional on the Strong Exponential Time Hypothesis [16], any data structure that can be constructed in time polynomial in the total size  $|\mathcal{D}|$  of the strings in the dictionary cannot answer queries in time strongly sublinear in  $|\mathcal{D}|$ .

We next provide a brief overview of other algorithmic works related to PMDM.

- *Dictionary Matching with  $k$ -wildcards:* Given a dictionary  $\mathcal{D}$  of total size  $N$  over an alphabet  $\Sigma$  and a query string  $q$  of length  $\ell$  over  $\Sigma \sqcup \{\star\}$  with up to  $k$  wildcards, the problem asks for the set of matches of  $q$  in  $\mathcal{D}$ . This is essentially a parameterized variant of the Partial Match problem. The seminal paper of Cole et al. [25] proposed a data structure occupying  $\mathcal{O}(N \log^k N)$  space allowing for  $\mathcal{O}(\ell + 2^k \log \log N + |\text{output}|)$ -time querying. This data structure is based on recursively computing a heavy-light decomposition of the suffix tree and copying the subtrees hanging off light children. Generalizations and slight improvements have been proposed in [13], [43], and [28]. In [13] the authors also proposed an alternative data structure that instead of a  $\log^k N$  factor in the space complexity has a multiplicative  $|\Sigma|^{k^2}$  factor. Nearly-linear-sized data structures that essentially try all different combinations of letters in the place of wildcards and hence incur a  $|\Sigma|^k$  factor in the query time have been proposed in [13, 44]. On the lower bound side, Afshani and Nielsen [3] showed that, in the pointer machine model, essentially any data structure for the problem in scope must have exponential dependency on  $k$  in either the space or the query time, explaining the barriers hit by the existing approaches.
- *Enumerating Motifs with  $k$ -wildcards:* Given an input string  $s$  of length  $n$  over an alphabet  $\Sigma$  and positive integers  $k$  and  $z$ , this problem asks to enumerate all motifs over  $\Sigma \sqcup \{\star\}$  with up to  $k$  wildcards that occur at least  $z$  times in  $s$ . As the size of the output is exponential in  $k$ , the enumeration problem has such a lower bound. Several approaches

exist for efficient motif enumeration, all aimed at reducing the impact of the output's size: efficient indexing to minimise the output delay [7, 30]; exploiting a hierarchy of wildcards positions according to the number of occurrences [10]; or defining a subset of motifs of fixed-parameter tractable size (in  $k$  or  $z$ ) that can generate all the others [5, 48, 49].

**Our Contributions.** We consider the word-RAM model of computations with  $w$ -bit machine words, where  $w = \Omega(\log(d\ell))$ , for stating our results. We make the following contributions:

1. (Section 3) A reduction from the  $k$ -Clique problem to a decision version of the PMDM problem, which implies that PMDM is NP-hard, even for strings over a binary alphabet.
2. (Section 4) A combinatorial  $\mathcal{O}((d\ell)^{k/3} + d\ell)$ -time and  $\mathcal{O}(d\ell)$ -space algorithm for PMDM if  $k = |K| = \mathcal{O}(1)$ , which is optimal if the combinatorial  $k$ -Clique hypothesis is true.
3. (Section 5) We consider a generalized version of PMDM, referred to as MPMDM: we are given a collection  $\mathcal{M}$  of  $m$  query strings (instead of one query string) and we are asked to compute a smallest set  $K$  so that, for every  $q$  from  $\mathcal{M}$ , if  $q[i]$ , for all  $i \in K$ , is replaced by a wildcard, then  $q$  matches at least  $z$  strings from dictionary  $\mathcal{D}$ . We show an  $\mathcal{O}((d\ell)^{k/3} z^{m-1} + d\ell)$ -time algorithm for MPMDM, for  $k = |K| = \mathcal{O}(1)$  and  $m = \mathcal{O}(1)$ .
4. (Section 6) A data structure for PMDM that answers queries over  $\mathcal{D}$  in  $\mathcal{O}(2^{\ell/2}(2^{\ell/2} + \tau)\ell)$  time and requires space  $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2}d)$ , for any parameter  $\tau \in [1, d]$ .
5. (Section 7) A polynomial-time  $\mathcal{O}(d^{1/4+\epsilon})$ -approximation algorithm for PMDM, which we show to be tight under a plausible complexity conjecture.

Let us now discuss why our data structure results (Section 6) cannot be directly obtained using data structures for Dictionary Matching with  $k$ -wildcards. Conceivably, one could construct such a data structure, and then iterate over all subsets of  $\{1, \dots, \ell\}$ , querying for the masked string. Existing data structures for dictionary matching with wildcards (cf. [13, Table 1], [44], and [28]), that allow querying a pattern with at most  $\ell$  wildcards, have

- (a) either  $\Omega(\min\{\sigma^\ell, d\})$  query time, thus yielding  $\Omega(2^\ell \cdot \min\{\sigma^\ell, d\})$  query time for our problem, and space  $\Omega(d\ell)$ , a trade-off dominated by the SMALL- $\ell$  algorithm (cf. our Table 1);
- (b) or  $\Omega(\ell)$  query time, thus yielding  $\Omega(2^\ell \ell)$  query time for our problem, and  $\Omega(d\ell \log^\ell \log(d\ell))$  space, a trade-off dominated by the DS SIMPLE (cf. our Table 1).

## 2 Definitions and Notation

**Strings.** An *alphabet*  $\Sigma$  is a finite nonempty set whose elements are called *letters*. We assume throughout an integer alphabet  $\Sigma = [1, |\Sigma|]$ . Let  $x = x[1] \cdots x[n]$  be a *string* of length  $|x| = n$  over  $\Sigma$ . For two indices  $1 \leq i \leq j \leq n$ ,  $x[i..j] = x[i] \cdots x[j]$  is the *substring* of  $x$  that starts at position  $i$  and ends at position  $j$  of  $x$ . By  $\varepsilon$  we denote the *empty string* of length 0. A *prefix* of  $x$  is a substring of  $x$  of the form  $x[1..j]$ , and a *suffix* of  $x$  is a substring of  $x$  of the form  $x[i..n]$ . A *dictionary* is a collection of strings. We also consider alphabet  $\Sigma_\star = \Sigma \sqcup \{\star\}$ , where  $\star$  is a *wildcard* letter that is not in  $\Sigma$  and *matches* all letters from  $\Sigma_\star$ . Then, given a string  $x$  over  $\Sigma_\star$  and a string  $y$  over  $\Sigma$  with  $|x| = |y|$ , we say that  $x$  *matches*  $y$  if and only if  $x[i] = y[i]$  or  $x[i] = \star$ , for all  $1 \leq i \leq |x|$ . Given a string  $x$  of length  $n$  and a set  $S \subseteq \{1, \dots, n\}$ , we denote by  $x_S = x \otimes S$  the string obtained by first setting  $x_S = x$  and then  $x_S[i] = \star$ , for all  $i \in S$ . We then say that  $x$  is *masked* by  $S$ .

The main problem considered in this paper is the following.

## PATTERN MASKING FOR DICTIONARY MATCHING (PMDM)

**Input:** A dictionary  $\mathcal{D}$  of  $d$  strings, each of length  $\ell$ , a string  $q$  of length  $\ell$ , and a positive integer  $z$ .

**Output:** A smallest set  $K \subseteq \{1, \dots, \ell\}$  such that  $q_K = q \otimes K$  matches at least  $z$  strings from  $\mathcal{D}$ .

We refer to the problem of computing only the size  $k$  of a smallest set  $K$  as PMDM-SIZE. We also consider the data structure variant of the PMDM problem in which  $\mathcal{D}$  is given for preprocessing, and  $q, z$  queries are to be answered on-line. Throughout, we assume that  $k \geq 1$  as the case  $k = 0$  corresponds to the well-studied dictionary matching problem for which there exists a classic optimal solution [4]. We further assume  $z \leq d$ ; otherwise the PMDM has trivially no solution. In what follows, we use  $N$  to denote  $d\ell$ .

**Tries.** Let  $\mathcal{M}$  be a finite set containing  $m > 0$  strings over  $\Sigma$ . The *trie* of  $\mathcal{M}$ , denoted by  $\mathcal{R}(\mathcal{M})$ , contains a node for every distinct prefix of a string in  $\mathcal{M}$ ; the root node is  $\varepsilon$ ; the set of leaf nodes is  $\mathcal{M}$ ; and edges are of the form  $(u, \alpha, u\alpha)$ , where  $u$  and  $u\alpha$  are nodes and  $\alpha \in \Sigma$  is the label. The *compact trie* of  $\mathcal{M}$ , denoted by  $\mathcal{T}(\mathcal{M})$ , contains the root, the branching nodes, and the leaf nodes of  $\mathcal{R}(\mathcal{M})$ . Each maximal branchless path segment from  $\mathcal{R}(\mathcal{M})$  is replaced by a single edge, and a fragment of a string  $M \in \mathcal{M}$  is used to represent the label of this edge in  $\mathcal{O}(1)$  space. The size of  $\mathcal{T}(\mathcal{M})$  is thus  $\mathcal{O}(m)$ . The most well-known example of a compacted trie is the suffix tree of a string: the compacted trie of all the suffixes of the string [62]. To access the children of a trie node by the first letter of their edge label in  $\mathcal{O}(1)$  time we use perfect hashing [27]. In this case, the claimed complexities hold *with high probability* (w.h.p., for short), that is, with probability at least  $1 - N^{-c}$  (recall that  $N = d\ell$ ), where  $c > 0$  is a constant fixed at construction time. Assuming that the children of every trie node are sorted by the first letters of their edge labels, randomization can be avoided at the expense of a  $\log |\Sigma|$  factor incurred by binary searching for the appropriate child.

### 3 PMDM-Size is NP-hard

We show that the following decision version of PMDM-SIZE is NP-complete.

#### $k$ -PMDM

**Input:** A dictionary  $\mathcal{D}$  of  $d$  strings, each of length  $\ell$ , a string  $q$  of length  $\ell$ , and positive integers  $z \leq d$  and  $k \leq \ell$ .

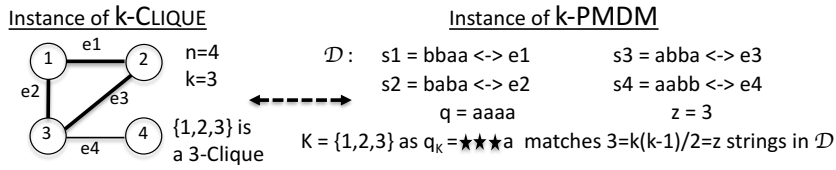
**Output:** Is there a set  $K \subseteq \{1, \dots, \ell\}$  of size  $k$ , such that  $q_K = q \otimes K$  matches at least  $z$  strings from  $\mathcal{D}$ ?

Our reduction is from the well-known NP-complete  $k$ -CLIQUE problem [37]: Given an undirected graph  $G$  on  $n$  nodes and a positive integer  $k$ , decide whether  $G$  contains a clique of size  $k$  (a *clique* is a subset of the nodes of  $G$  that are pairwise adjacent).

► **Theorem 1.** *Any instance of the  $k$ -CLIQUE problem for a graph with  $n$  nodes and  $m$  edges can be reduced in  $\mathcal{O}(nm)$  time to a  $k$ -PMDM instance with  $\ell = n$ ,  $d = m$  and  $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ .*

**Proof.** Let  $G = (V, E)$  be an undirected graph on  $n = |V|$  nodes numbered 1 through  $n$ , in which we are looking for a clique of size  $k$ . We reduce  $k$ -CLIQUE to  $k$ -PMDM as follows. Consider the alphabet  $\{\mathbf{a}, \mathbf{b}\}$ . Set  $q = \mathbf{a}^n$ , and for every edge  $(u, v) \in E$  such that  $u < v$ , add string  $\mathbf{a}^{u-1}\mathbf{b}\mathbf{a}^{v-u-1}\mathbf{b}\mathbf{a}^{n-v}$  to  $\mathcal{D}$ . Set  $z = k(k-1)/2$ . Then  $G$  contains a clique of size  $k$ , if and only if  $k$ -PMDM returns a positive answer. This can be seen by the fact that cliques of





**Figure 1** An example of the reduction from  $k$ -CLIQUE to  $k$ -PMDM. The solution for both is  $\{1, 2, 3\}$  as shown. Note that, for  $k = 4$ , the instance of 4-PMDM would need  $z = 6$  matches; neither this many matches can be found in  $\mathcal{D}$  nor a 4-clique can be found in the graph.

size  $k$  in  $G$  are in one-to-one correspondence with subsets  $K \subseteq \{1, \dots, n\}$  of size  $k$  for which  $q_K$  matches  $z$  strings from  $\mathcal{D}$ : the elements of  $K$  correspond to the nodes of a clique and the  $z$  strings correspond to its edges.  $k$ -PMDM is clearly in NP and the result follows. ◀

An example of the reduction from  $k$ -CLIQUE to  $k$ -PMDM is shown in Figure 1.

► **Corollary 2.**  $k$ -PMDM is NP-complete for strings over a binary alphabet.

Our reduction (Theorem 1) shows that solving  $k$ -PMDM efficiently even for strings over a binary alphabet would imply a breakthrough for the  $k$ -CLIQUE problem for which it is known that, in general, no fixed-parameter tractable algorithm with respect to parameter  $k$  exists unless the Exponential Time Hypothesis (ETH) fails [19, 34]. That is,  $k$ -CLIQUE has no  $f(k)n^{o(k)}$  time algorithm, and is thus W[1]-complete (again, under the ETH hypothesis). On the upper bound side,  $k$ -CLIQUE can be trivially solved in  $\mathcal{O}(n^k)$  time (enumerating all subsets of nodes of size  $k$ ), and this can be improved to  $\mathcal{O}(n^{\omega k/3})$  time for  $k$  divisible by 3 using square matrices multiplication ( $\omega$  is the exponent of square matrix multiplication). However, for general  $k \geq 3$  and any constant  $\epsilon > 0$ , the  $k$ -CLIQUE hypothesis states that there is no  $\mathcal{O}(n^{(\omega/3-\epsilon)k})$ -time algorithm and no combinatorial  $\mathcal{O}(n^{(1-\epsilon)k})$ -time algorithm [2, 45, 63].

In particular, assuming that the  $k$ -CLIQUE hypothesis is true, due to Theorem 1, we cannot hope to devise a combinatorial algorithm for  $k$ -PMDM requiring  $\mathcal{O}((d\ell)^{(1-\epsilon)k/3})$  time, for any  $k \geq 3$  and  $\epsilon > 0$ , since  $d\ell = nm$ . In Section 4, we show a combinatorial  $\mathcal{O}((d\ell)^{k/3})$ -time algorithm, for constant  $k \geq 3$ , for the optimization version of  $k$ -PMDM (seeking to maximize the matches), which can then be trivially applied to solve  $k$ -PMDM in the same time complexity, thus matching the above conditional lower bound.

Given an undirected graph  $G$ , an *independent set* is a subset of nodes of  $G$  such that no two distinct nodes of the subset are adjacent. Let us note that the problem of computing a maximum clique in a graph  $G$ , which is equivalent to that of computing the maximum independent set in the complement of  $G$ , cannot be  $n^{1-\epsilon}$ -approximated in polynomial time, for any  $\epsilon > 0$ , unless  $P = NP$  [31, 65].

Any algorithm solving PMDM-SIZE can be trivially applied to solve  $k$ -PMDM.

► **Corollary 3.** PMDM-SIZE is NP-hard for strings over a binary alphabet.

## 4 Exact Algorithms for a Bounded Number $k$ of Wildcards

We consider the following problem, which we solve by exact algorithms.

HEAVIEST  $k$ -PMDM  
**Input:** A dictionary  $\mathcal{D}$  of  $d$  strings, each of length  $\ell$ , a string  $q$  of length  $\ell$ , and a positive integer  $k \leq \ell$ .  
**Output:** A set  $K \subseteq \{1, \dots, \ell\}$  of size  $k$  such that  $q_K = q \otimes K$  matches the maximum number of strings in  $\mathcal{D}$ .



We will show the following result, which we will employ to solve the PMDM problem.

► **Theorem 4.** *HEAVIEST  $k$ -PMDM for  $k = \mathcal{O}(1)$  can be solved in  $\mathcal{O}(N + \min\{N^{k/3}, \ell^k\})$  time, where  $N = d\ell$ .*

A *hypergraph*  $H$  is a pair  $(V, E)$ , where  $V$  is the set of nodes of  $H$  and  $E$  is a set of non-empty subsets of  $V$ , called *hyperedges* – in order to simplify terminology we will simply call them edges. Hypergraphs are a generalization of graphs in the sense that an edge can connect more than two nodes. Recall that the size of an edge is the number of nodes it contains. The *rank* of  $H$ , denoted by  $r(H)$ , is the maximum size of an edge of  $H$ .

We refer to a hypergraph  $H \times K = (K, \{e : e \in E, e \subseteq K\})$ , where  $K$  is a subset of  $V$ , as a  $|K|$ -*section*.  $H \times K$  is the hypergraph induced by  $H$  on the nodes of  $K$ , and it contains all edges of  $H$  whose elements are all in  $K$ . A hypergraph is *weighted* when each of its edges is associated with a weight. We define *the weight* of a weighted hypergraph as the sum of the weights of all of its edges. In what follows, we also refer to weights of nodes for conceptual clarity; this is equivalent to having a singleton edge of equal weight consisting of that node.

We define the following auxiliary problem on hypergraphs (see also [20]).

HEAVIEST  $k$ -SECTION

**Input:** A weighted hypergraph  $H = (V, E)$ , with  $E$  given as a list, and an integer  $k > 0$ .

**Output:** A subset  $K$  of size  $k$  of  $V$  such that  $H \times K$  has maximum weight.

When  $k = \mathcal{O}(1)$ , we preprocess the edges of  $H$  as follows in order to have  $\mathcal{O}(1)$ -time access to any queried edge. We represent each edge as a string, whose letters correspond to its elements in increasing order. Then, we sort all such strings lexicographically using radix sort in  $\mathcal{O}(|E|)$  time and construct a trie over them. An edge can then be accessed in  $\mathcal{O}(k \log k) = \mathcal{O}(1)$  time by a forward search starting from the root node of the trie.

A polynomial-time  $\mathcal{O}(n^{0.697831+\epsilon})$ -approximation for HEAVIEST  $k$ -SECTION, for any  $\epsilon > 0$ , for the case when all hyperedges of  $H$  have size at most 3 was shown in [20] (see also [6]).

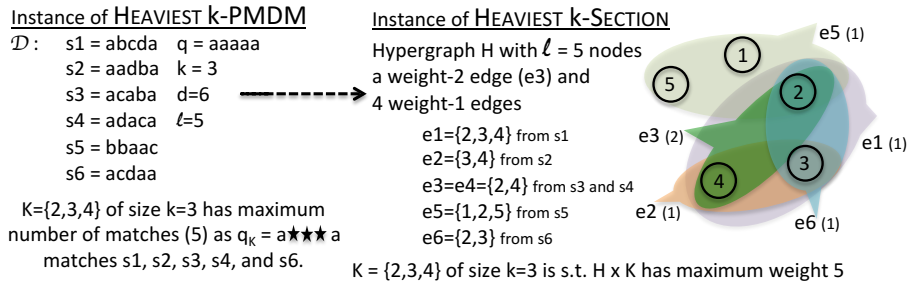
Two remarks are in place. First, we can focus on edges of size up to  $k$  as larger edges cannot, by definition, exist in any  $k$ -section. Second, HEAVIEST  $k$ -SECTION is a generalization of the problem of deciding whether a  $(c, k)$ -hyperclique (i.e. a set of  $k$  nodes whose subsets of size  $c$  are all in  $E$ ) exists in a graph, which in turn is a generalization of  $k$ -CLIQUE. Unlike  $k$ -CLIQUE, the  $(c, k)$ -hyperclique problem is not known to benefit from fast matrix multiplication in general; see [45] for a discussion on its hardness.

► **Lemma 5.** *HEAVIEST  $k$ -PMDM can be reduced to HEAVIEST  $k$ -SECTION for a hypergraph with  $\ell$  nodes and  $d$  edges in  $\mathcal{O}(N)$  time, where  $N = d\ell$ .*

**Proof.** We first compute the set  $M_s$  of positions of mismatches of  $q$  with each string  $s \in \mathcal{D}$ . We ignore strings from  $\mathcal{D}$  that match  $q$  exactly, as they will match  $q$  after changing any set of letters of  $q$  to wildcards. This requires  $\mathcal{O}(d\ell) = \mathcal{O}(N)$  time in total.

Let us consider an empty hypergraph (i.e. with no edges)  $H$  on  $\ell$  nodes, numbered 1 through  $\ell$ . Then, for each string  $s \in \mathcal{D}$ , we add  $M_s$  to the edge-set of  $H$  if  $|M_s| \leq k$ ; if this edge already exists, we simply increment its weight by 1.

We set the parameter  $k$  of HEAVIEST  $k$ -SECTION to the parameter  $k$  of HEAVIEST  $k$ -PMDM. We now observe that for  $K \subseteq V$  with  $|K| = k$ , the weight of  $H \times K$  is equal to the number of strings that would match  $q$  after replacing with wildcards the  $k$  letters of  $q$  at the positions corresponding to elements of  $K$ . The statement follows. ◀



■ **Figure 2** An example of the reduction from HEAVIEST  $k$ -PMDM to HEAVIEST  $k$ -SECTION. The solutions are at the bottom. Each edge has its weight in brackets and the total weight is  $d = 6$ .

An example of the reduction in Lemma 5 is shown in Figure 2.

The next lemma gives a straightforward solution to HEAVIEST  $k$ -SECTION. It is analogous to algorithm SMALL- $\ell$ , presented in Section 6, but without the optimization in computing sums of weights over subsets. It implies a linear-time algorithm for HEAVIEST 1-SECTION.

► **Lemma 6.** HEAVIEST  $k$ -SECTION, for any constant  $k$ , can be solved in  $\mathcal{O}(|V|^k + |E|)$  time and  $\mathcal{O}(|V| + |E|)$  space.

**Proof.** For every subset  $K \subseteq V$  of size at most  $k$ , we sum the weights of all edges corresponding to its subsets. There are  $\binom{|V|}{k} = \mathcal{O}(|V|^k)$  choices for  $|K|$ , each having  $2^k - 1$  non-empty subsets: for every subset, we can access the corresponding edge (if it exists) in  $\mathcal{O}(1)$  time. ◀

We next show that for the cases  $k = 2$  and  $k = 3$ , there exist more efficient solutions. In particular, we provide a linear-time algorithm for HEAVIEST 2-SECTION.

► **Lemma 7.** HEAVIEST 2-SECTION can be solved in  $\mathcal{O}(|V| + |E|)$  time.

**Proof.** Let  $K$  be a set of nodes of size 2 such that  $H \times K$  has maximum weight. We decompose the problem in two cases. For each of the cases, we give an algorithm that considers several 2-sections such that the heaviest of them has weight equal to that of  $H \times K$ .

*Case 1.* There is an edge  $e = K$  in  $E$ . For each edge  $e \in E$  of size 2, i.e. edge in the classic sense, we compute the sum of its weight and the weights of the nodes that it is incident to. This step requires  $\mathcal{O}(|E|)$  time.

*Case 2.* There is no edge equal to  $K$  in  $E$ . We compute  $H \times \{v_1, v_2\}$ , where  $v_1, v_2$  are the two nodes with maximum weight, i.e. max and second-max. This step takes  $\mathcal{O}(|V|)$  time.

In the end, we return the heaviest 2-section among those returned by the algorithms for the two cases, breaking ties arbitrarily. ◀

We next show that for  $k = 3$  the result of Lemma 6 can be improved when  $|E| = o(|V|^2)$ .

► **Lemma 8.** HEAVIEST 3-SECTION can be solved in time  $\mathcal{O}(|V| \cdot |E|)$  using  $\mathcal{O}(|V| + |E|)$  space.

**Proof.** Let  $K$  be a set of nodes of size 3 such that  $H \times K$  has maximum weight. We decompose the problem into the following three cases.

*Case 1.* There is an edge  $e = K$  in  $E$ . We go through each edge  $e \in E$  of size 3 and compute the weight of  $H \times e$  in  $\mathcal{O}(1)$  time. This takes  $\mathcal{O}(|E|)$  time in total. Let the edge yielding the maximum weight be  $e_{\max}$ .

*Case 2.* There is no edge of size larger than one in  $H \times K$ . We compute  $H \times \{v_1, v_2, v_3\}$ , where  $v_1, v_2, v_3$  are the three nodes with maximum weight, i.e. max, second-max and third-max. This step takes  $\mathcal{O}(|V|)$  time.

## 65:10 Pattern Masking for Dictionary Matching

*Case 3.* There is an edge of size 2 in  $H \times K$ . We can pick an edge  $e$  of size 2 from  $E$  in  $\mathcal{O}(|E|)$  ways and a node  $v$  from  $V$  in  $\mathcal{O}(|V|)$  ways. We compute the weight of  $H \times (e \cup \{v\})$  for all such pairs. Let the pair yielding maximum weight be  $(e', u')$ .

Finally, the maximum weight of  $H \times K'$  for  $K' \in \{e_{\max}, \{v_1, v_2, v_3\}, e' \cup \{u'\}\}$  is equal to the weight of  $H \times K$ , breaking ties arbitrarily.  $\blacktriangleleft$

► **Lemma 9.** *HEAVIEST  $k$ -SECTION for an arbitrarily large constant  $k \geq 4$  can be solved in time  $\mathcal{O}((|V| \cdot |E|)^{k/3})$  using  $\mathcal{O}(|V| + |E|)$  space.*

**Proof.** If  $|E| > |V|^2$ , then the simple algorithm of Lemma 6 solves the problem in time

$$\mathcal{O}(|V|^k + |E|) = \mathcal{O}(|V|^{k/3}(|V|^2)^{k/3} + |E|) = \mathcal{O}((|V| \cdot |E|)^{k/3})$$

and linear space. We can thus henceforth assume that  $|E| \leq |V|^2$ .

Let  $K$  be a set of nodes of size at most  $k$  such that  $H \times K$  has maximum weight. If  $H \times K$  contains isolated nodes (i.e. nodes not contained in any edge), they can be safely deleted without altering the result. We can thus assume that  $H \times K$  does not contain isolated nodes, and that  $|V| \leq k|E|$  since otherwise the hypergraph  $H$  would contain isolated nodes.

We first consider the case that the rank  $r(H \times K) > 1$ , i.e. there is an edge of  $H \times K$  of size at least 2. We design a branching algorithm that constructs several candidate sets; the ones with maximum weight will have weight equal to that of  $H \times K$ . We will construct a set of nodes  $X$ , starting with  $X := \emptyset$ . For each set  $X$  that we process, let  $Z_X$  be the superset of  $X$  of size at most  $k$  such that  $H \times Z_X$  has maximum weight. We have the following two cases:

*Case 1.* There is an edge  $e$  in  $H \times Z_X$  that contains at least two nodes from  $Z_X \setminus X$ . To account for this case, we select every possible such edge  $e$ , set  $X := X \cup e$ , and continue the branching algorithm.

*Case 2.* Each edge in  $H \times Z_X$  contains at most one node from  $Z_X \setminus X$ . In this case we conclude the branching algorithm as follows. For every node  $v \in V \setminus X$  we compute its weight as the total weight of edges  $Y \cup \{v\} \in E$  for  $Y \subseteq X$  in  $\mathcal{O}(2^k) = \mathcal{O}(1)$  time. Finally, in  $\mathcal{O}(|V|k) = \mathcal{O}(|V|)$  time we select  $k - |X|$  nodes with largest weights and insert them into  $X$ . The total time complexity of this step is  $\mathcal{O}(|V|)$ . This case also works if  $|X| = k$  and then its time complexity is only  $\mathcal{O}(1)$ .

The correctness of this branching algorithm follows from an easy induction, showing that at every level of the branching tree there is a subset of  $K$ .

Let us now analyze the time complexity of this branching algorithm. Each branching in Case 1 takes  $\mathcal{O}(|E|)$  time and increases the size of  $|X|$  by at least 2. At every node of the branching tree we call the procedure of Case 2. It takes  $\mathcal{O}(|V|)$  time if  $|X| < k$ .

If the procedure of Case 2 is called in a non-leaf node of the branching tree, then its  $\mathcal{O}(|V|)$  running time is dominated by the  $\mathcal{O}(|E|)$  time that is required for further branching since we have assumed that  $|V| \leq k|E|$ . Hence, it suffices to bound (a) the total time complexity of calls to the algorithm for Case 2 in leaves that correspond to sets  $X$  such that  $|X| < k$  and (b) the total number of leaves that correspond to sets  $X$  such that  $|X| = k$ .

If  $k$  is even, (a) is bounded by  $\mathcal{O}(|E|^{(k-2)/2}|V|)$  and (b) is bounded by  $\mathcal{O}(|E|^{k/2})$ . Hence, (b) dominates (a) and we have

$$\mathcal{O}(|E|^{k/2}) = \mathcal{O}(|E|^{k/3}|E|^{k/6}) = \mathcal{O}(|E|^{k/3}|V|^{k/3}). \quad (1)$$

If  $k$  is odd, (a) is bounded by  $\mathcal{O}(|E|^{(k-1)/2}|V|)$  and (b) is bounded by  $\mathcal{O}(|E|^{(k-1)/2})$ , which is dominated by (a). By using (1) for  $k - 3$  we also have:

$$\mathcal{O}(|E|^{(k-1)/2} \cdot |V|) = \mathcal{O}(|E|^{(k-3)/2} \cdot |E| \cdot |V|) = \mathcal{O}((|E| \cdot |V|)^{(k-3)/3} \cdot |E| \cdot |V|) = \mathcal{O}((|E| \cdot |V|)^{k/3}).$$

We now consider the case that  $r(H \times K) = 1$ . We use the algorithm for Case 2 above that works in  $\mathcal{O}(|V|)$  time, which is  $\mathcal{O}(|V| \cdot |E|)$ . ◀

Lemmas 5-9 imply Theorem 4, which we employ iteratively to obtain the following result.

► **Theorem 10.** *PMDM can be solved in time  $\mathcal{O}(N + \min\{N^{k/3}, \ell^k\})$  using space  $\mathcal{O}(N)$  if  $k = \mathcal{O}(1)$ , where  $N = d\ell$ .*

**Proof.** We apply Lemma 5 to obtain a hypergraph with  $|V| = \ell$  and  $|E| = d$ . Starting with  $k = 1$  and for growing values of  $k$ , we solve HEAVIEST  $k$ -SECTION until we obtain a solution of weight at least  $z$ , employing either only Lemma 6, or Lemmas 6, 7, 8, 9 for  $k = 1, 2, 3$  and  $k \geq 4$ , respectively. We obtain  $\mathcal{O}(N + \min\{N^{k/3}, \ell^k\})$  time and  $\mathcal{O}(N)$  space. ◀

## 5 Exact Algorithms for a Bounded Number $m$ of Query Strings

Recall that masking a potential match  $(q_1, q_2)$  in record linkage can be performed by solving PMDM twice and superimposing the wildcards (see Section 1). In this section, we consider the following generalized version of PMDM to perform the masking simultaneously. The advantage of this approach is that it minimizes the final number of wildcards in  $q_1$  and  $q_2$ .

MULTIPLE PATTERN MASKING FOR DICTIONARY MATCHING (MPMDM)

**Input:** A dictionary  $\mathcal{D}$  of  $d$  strings, each of length  $\ell$ , a collection  $\mathcal{M}$  of  $m$  strings, each of length  $\ell$ , and a positive integer  $z$ .

**Output:** A smallest set  $K \subseteq \{1, \dots, \ell\}$  such that, for every  $q$  from  $\mathcal{M}$ ,  $q_K = q \otimes K$  matches at least  $z$  strings from  $\mathcal{D}$ .

Let  $N = d\ell$ . We show the following theorem.

► **Theorem 11.** *MPMDM can be solved in time  $\mathcal{O}(N + \min\{N^{k/3}z^{m-1}, \ell^k\})$  if  $k = \mathcal{O}(1)$  and  $m = \mathcal{O}(1)$ , where  $N = d\ell$ .*

We use a generalization of HEAVIEST  $k$ -SECTION in which the weights are  $m$ -tuples that are added and compared component-wise, and we aim to find a subset  $K$  such that the weight of  $H \times K$  is at least  $(z, \dots, z)$ . An analogue of Lemma 6 holds without any alterations, which accounts for the  $\mathcal{O}(N + \ell^k)$ -time algorithm. We adapt the proof of Lemma 9 as follows. The branching remains the same, but we have to tweak the final step, that is, what happens when we are in Case 2. For  $m = 1$  we could simply select a number of largest weights, but for  $m > 1$  multiple criteria need to be taken into consideration. All in all, the problem reduces to a variation of the classic Multiple-Choice Knapsack problem [38], which we solve using dynamic programming. Overall, we pay an additional  $\mathcal{O}(z^{m-1})$  factor in the complexity of handling of Case 2, which yields the complexity of Theorem 11.

## 6 A Data Structure for PMDM Queries

We next show algorithms and data structures for the PMDM problem under the assumption that  $2^\ell$  is reasonably small. We measure space in terms of  $w$ -bit machine words, where  $w = \Omega(\log(d\ell))$ , and focus on showing space vs. query-time trade-offs for answering  $q, z$  PMDM queries over  $\mathcal{D}$ . A summary of the complexities of the data structures is shown in Table 1. Specifically, algorithm SMALL- $\ell$  and data structure SIMPLE are used as building blocks in the more involved data structure SPLIT underlying the following theorem.

► **Theorem 12.** *There exists a data structure that answers  $q, z$  PMDM queries over  $\mathcal{D}$  in time  $\mathcal{O}(2^{\ell/2}(2^{\ell/2} + \tau)\ell)$  w.h.p. and requires space  $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2}d)$ , for any  $\tau \in [1, d]$ .*

## 65:12 Pattern Masking for Dictionary Matching

■ **Table 1** Basic complexities of the data structures from Section 6.

Data structure	Space	Query time
Algorithm SMALL- $\ell$	$\mathcal{O}(d\ell)$	$\mathcal{O}(2^\ell \ell + d\ell)$
DS SIMPLE	$\mathcal{O}(2^\ell d)$	$\mathcal{O}(2^\ell \ell)$
DS SPLIT, any $\tau$	$\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2}d)$	$\mathcal{O}(2^{\ell/2} \cdot (2^{\ell/2} + \tau)\ell)$
DS SPLIT for $\tau = 2^{\ell/4}\sqrt{d}$	$\mathcal{O}(2^{\ell/2}d)$	$\mathcal{O}(2^\ell \ell + 2^{3\ell/4}\sqrt{d}\ell)$

**Algorithm SMALL- $\ell$ :  $\mathcal{O}(d\ell)$  Space,  $\mathcal{O}(2^\ell \ell + d\ell)$  Query Time.** No data structure on top of the dictionary  $\mathcal{D}$  is stored. In the query algorithm, we initialize an array  $A$  of size  $2^\ell$  with zeros. For an  $\ell$ -bit vector  $m$ , by  $K_m \subseteq \{1, \dots, \ell\}$  let us denote the set of the positions of set bits of  $m$ . Now for every possible  $\ell$ -bit vector  $m$  we want to compute the number of strings in  $\mathcal{D}$  that match  $q_{K_m} = q \otimes K_m$ .

To this end, for every string  $s \in \mathcal{D}$ , we compute the set  $K$  of positions in which  $s$  and  $q$  differ. For  $m$  that satisfies  $K = K_m$ , we increment  $A[m]$ . This computation takes  $\mathcal{O}(d\ell)$  time and  $\mathcal{O}(1)$  extra space. Then we apply a folklore dynamic-programming-based approach to compute array  $B$ , which is defined as follows:

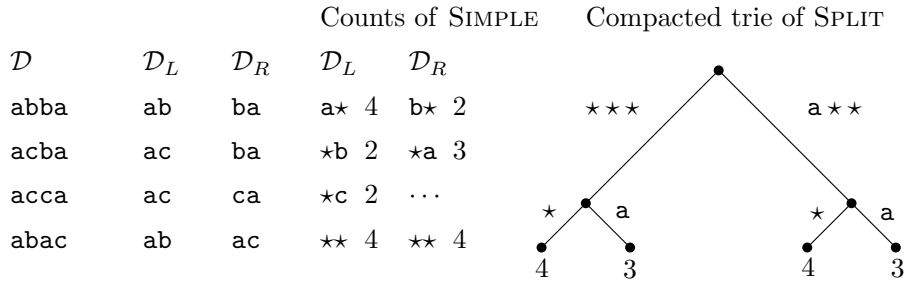
$$B[m] = \sum_{j \in S(m)} A[j], \text{ where } S(m) = \{j \in [1, 2^\ell] : K_j \subseteq K_m\}.$$

In other words,  $B[m]$  stores the number of strings from  $\mathcal{D}$  that match  $q_{K_m}$ . It takes  $\mathcal{O}(\ell 2^\ell)$  time and  $\mathcal{O}(2^\ell)$  extra space. Thus, overall, the (query) time required by algorithm SMALL- $\ell$  is  $\mathcal{O}(\ell 2^\ell + d\ell)$ , the data structure space is  $\mathcal{O}(d\ell)$ , and the extra space is  $\mathcal{O}(2^\ell)$ .

We first present SIMPLE, an auxiliary data structure, which we will apply later on to construct DS SPLIT, a data structure with the space/query-time trade-off of Theorem 12.

**DS SIMPLE:  $\mathcal{O}(2^\ell d)$  Space,  $\mathcal{O}(2^\ell \ell)$  Query Time.** We initialize an empty set  $\mathcal{Q}$ . For each possible subset of  $\{1, \dots, \ell\}$  we do the following. We mask the corresponding positions in all strings from  $\mathcal{D}$  and then sort the masked strings lexicographically. By iterating over the lexicographically sorted list of the masked strings, we count how many copies of each distinct (masked) string we have in our list. We insert each such (masked) string to  $\mathcal{Q}$  along with its count. After processing all  $2^\ell$  subsets, we construct a compacted trie for the strings in  $\mathcal{Q}$ ; each leaf corresponds to a unique element of  $\mathcal{Q}$ , and stores this element's count. The total space occupied by this compacted trie is thus  $\mathcal{O}(2^\ell d)$ . Upon an on-line query  $q$  (of length  $\ell$ ) and  $z$ , we apply all possible  $2^\ell$  masks to  $q$  and read the count for each of them from the compacted trie in  $\mathcal{O}(\ell)$  time per mask. Next, we show how to decrease the exponential dependency on  $\ell$  in the space complexity when  $2^\ell = o(d)$ , incurring extra time in the query.

**DS SPLIT:  $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2}d)$  Space,  $\mathcal{O}(2^{\ell/2} \cdot (2^{\ell/2} + \tau)\ell)$  Query Time, for any  $\tau$ .** This trade-off is relevant when  $\tau = \omega(\sqrt{d})$ ; otherwise the DS SIMPLE is better. We split each string  $p \in \mathcal{D}$  roughly in the middle, to prefix  $p_L$  and suffix  $p_R$ ; specifically,  $p = p_L p_R$  and  $|p_L| = \lceil \ell/2 \rceil$ . We create dictionaries  $\mathcal{D}_L = \{p_L : p \in \mathcal{D}\}$  and  $\mathcal{D}_R = \{p_R : p \in \mathcal{D}\}$ . Let us now explain how to process  $\mathcal{D}_L$ ; we process  $\mathcal{D}_R$  analogously. Let  $\lambda = \lceil \ell/2 \rceil$ . We construct DS SIMPLE over  $\mathcal{D}_L$ . This requires space  $\mathcal{O}(2^{\ell/2}d)$ . Let  $\tau$  be an input parameter, intuitively used as the minimum frequency threshold. For each of the possible  $2^\lambda$  masks, we can have at most  $\lfloor d/\tau \rfloor$  (masked) strings with frequency at least  $\tau$ . Over all masks, we thus have at most  $2^\lambda \lfloor d/\tau \rfloor$  such strings, which we call  $\tau$ -frequent. For every pair of  $\tau$ -frequent strings, one from  $\mathcal{D}_L$  and one from  $\mathcal{D}_R$ , we store the number of occurrences of their concatenation in  $\mathcal{D}$  using a compacted trie as in DS SIMPLE. This requires space  $\mathcal{O}(2^\ell d^2/\tau^2)$ .



■ **Figure 3** Let  $\tau = 3$ . If both  $q'_L$  and  $q'_R$  are 3-frequent (we check this using the counts of DS SIMPLE), we read the count for  $q'_L q'_R$  from the compacted trie of DS SPLIT. If  $q'_L$  is 3-infrequent, we apply SMALL- $\ell$  on  $q_R$  and the dictionary consisting of at most  $\tau = 3$  strings from  $\mathcal{D}_R$  corresponding to the right halves of strings in  $\mathcal{D}_L$  that match  $q'_L$ .

Consider  $\mathcal{D}_L$ . For each mask  $i$  and each string  $p_L \in \mathcal{D}_L$ , we can afford to store the list of all strings in  $\mathcal{D}_L$  that match  $p_L \otimes i$ . Note that we have computed this information when sorting for constructing DS SIMPLE over  $\mathcal{D}_L$ . This information requires space  $\mathcal{O}(2^{\ell/2}d)$ . Thus, DS SPLIT requires  $\mathcal{O}(2^\ell d^2/\tau^2 + 2^{\ell/2}d)$  space overall.

Let us now show how to answer an on-line  $q, z$  query. Let  $q = q_L q_R$  with  $|q_L| = \lceil \ell/2 \rceil$ . We iterate over all possible  $2^\ell$  masks.

For a mask  $i$ , let  $q' = q \otimes i$ . We split  $q'$  into two halves,  $q'_L$  and  $q'_R$  with  $q' = q'_L q'_R$  and  $|q'_L| = \lceil \ell/2 \rceil$ . First, we check whether each of  $q'_L$  and  $q'_R$  is  $\tau$ -infrequent using the DS SIMPLE we have constructed for  $\mathcal{D}_L$  and  $\mathcal{D}_R$ , respectively, in time  $\mathcal{O}(\ell)$ . We have the following two cases (inspect also Figure 3).

- If both halves are  $\tau$ -frequent, we can read the frequency of their concatenation using the stored compacted trie in time  $\mathcal{O}(\ell)$ .
- Else, at least one of the two halves is  $\tau$ -infrequent. Assume without loss of generality that  $q'_L$  is  $\tau$ -infrequent. Let  $\mathcal{F}$  be the dictionary consisting of at most  $\tau$  strings from  $\mathcal{D}_R$  that correspond to the right halves of strings in  $\mathcal{D}_L$  that match  $q'_L$ . Naïvely counting how many elements of  $\mathcal{F}$  match  $q'_R$  could require  $\Omega(\tau\ell)$  time, and thus  $\Omega(2^\ell\tau\ell)$  overall. Instead, we apply algorithm SMALL- $\ell$  on  $q_R$  and  $\mathcal{F}$ . The crucial point is that if we ever come across  $q'_L$  again (for a different mask on  $q$ ), we will not need to do anything. We can maintain whether  $q'_L$  has been processed by temporarily marking the leaf corresponding to it in DS SIMPLE for  $\mathcal{D}_L$ . Thus, overall, we perform the SMALL- $\ell$  algorithm  $\mathcal{O}(2^{\ell/2})$  times, each time in  $\mathcal{O}((2^{\ell/2} + \tau)\ell)$  time. This completes the proof of Theorem 12.

**Efficient Construction.** For completeness, we next show how to construct DS SPLIT in  $\mathcal{O}(d\ell \log(d\ell) + 2^\ell d\ell + 2^\ell \ell d^2/\tau^2)$  time. We preprocess  $\mathcal{D}$  by sorting its letters in  $\mathcal{O}(d\ell \log(d\ell))$  time. The DS SIMPLE for  $\mathcal{D}_L$  and  $\mathcal{D}_R$  can then be constructed in  $\mathcal{O}(2^{\ell/2}d\ell)$  time. We then create the compacted trie for pairs of  $\tau$ -frequent strings. For each of the  $2^\ell$  possible masks, say  $i$ , and each string  $p \in \mathcal{D}$ , we split  $p' = p \otimes i$  in the middle to obtain  $p'_L$  and  $p'_R$ . If both  $p'_L$  and  $p'_R$  are  $\tau$ -frequent then  $p'$  will be in the set of strings for which we will construct the compacted trie for pairs of  $\tau$ -frequent strings. The counts for each of those strings can be read in  $\mathcal{O}(\ell)$  time from a DS SIMPLE over  $\mathcal{D}$ , which we can construct in time  $\mathcal{O}(2^\ell d\ell)$  – this data structure is then discarded. The compacted trie construction requires time  $\mathcal{O}(2^\ell \ell d^2/\tau^2)$ .

**Comparison of the Data Structures.** DS SIMPLE has lower query time than algorithm SMALL- $\ell$ . However, its space complexity can be much higher. DS SPLIT can be viewed as an intermediate option. For  $\tau$  as in Table 1, it has lower query time than algorithm SMALL- $\ell$  for



## 65:14 Pattern Masking for Dictionary Matching

$d = \omega(2^{3\ell/2})$ , while keeping moderate space complexity. DS SPLIT always has higher query time than DS SIMPLE, but its space complexity is lower by a factor of  $2^{\ell/2}$ . For example, for  $d = 2^{2\ell}$  we get the complexities shown in Table 2.

■ **Table 2** Basic complexities of the data structures from Section 6 for  $d = 2^{2\ell}$ .

Data structure	Space	Query time
Algorithm SMALL- $\ell$	$\mathcal{O}(2^{2\ell}\ell)$	$\mathcal{O}(2^{2\ell}\ell)$
DS SIMPLE	$\mathcal{O}(2^{3\ell})$	$\mathcal{O}(2^{\ell}\ell)$
DS SPLIT for $\tau = 2^{5\ell/4}$	$\mathcal{O}(2^{5\ell/2})$	$\mathcal{O}(2^{7\ell/4}\ell)$

## 7 Approximation Algorithm for PMDM

Clearly, PMDM is at least as hard as PMDM-SIZE because it also outputs the positions of the wildcards (set  $K$ ). Thus, PMDM is also NP-hard. In what follows, we show existence of a polynomial-time approximation algorithm for PMDM whose approximation factor is given with respect to  $d$ . Specifically, we show the following approximation result for PMDM.

► **Theorem 13.** *For any constant  $\epsilon > 0$ , there is an  $\mathcal{O}(d^{1/4+\epsilon})$ -approximation algorithm for PMDM, whose running time is polynomial in  $N$ , where  $N = d\ell$ .*

Our result is based on the Minimum Union (MU) problem [21], which we define next.

MINIMUM UNION (MU)

**Input:** A collection  $\mathcal{S}$  of  $d$  sets over a universe  $U$  and a positive integer  $z \leq d$ .

**Output:** A collection  $\mathcal{T} \subseteq \mathcal{S}$  with  $|\mathcal{T}| = z$  such that the size of  $\cup_{S \in \mathcal{T}} S$  is minimized.

To illustrate the MU problem, consider an instance of it where  $U = \{1, 2, 3, 4, 5\}$ ,  $\mathcal{S} = \{\{1\}, \{1, 2, 3\}, \{1, 3, 5\}, \{3\}, \{3, 4, 5\}, \{4\}, \{4, 5\}, \{5\}\}$ , with  $d = |\mathcal{S}| = 8$ , and  $z = 4$ . Then  $\mathcal{T} = \{\{3\}, \{3, 4, 5\}, \{4\}, \{4, 5\}\}$  is a solution because  $|\mathcal{T}| = z = 4$  and  $|\cup_{S \in \mathcal{T}} S| = 3$  is minimum. The MU problem is NP-hard and the following approximation result is known.

► **Theorem 14** ([21]). *For any constant  $\epsilon > 0$ , there is an  $\mathcal{O}(d^{1/4+\epsilon})$ -approximation algorithm for MU, whose running time is polynomial in the size of  $\mathcal{S}$ .*

► **Theorem 15.** *PMDM can be reduced to MU in time polynomial in  $N$ .*

**Proof.** We reduce the PMDM problem to MU in polynomial time as follows. Given any instance  $\mathcal{I}_{\text{PMDM}}$  of PMDM, we construct an instance  $\mathcal{I}_{\text{MU}}$  of MU in time  $\mathcal{O}(d\ell)$  by performing the following steps:

1. The universe  $U$  is set to  $\{1, \dots, \ell\}$ .
2. We start with an empty collection  $\mathcal{S}$ . Then, for each string  $s_i$  in  $\mathcal{D}$ , we add member  $S_i$  to  $\mathcal{S}$ , where  $S_i$  is the set of positions where string  $q$  and string  $s_i$  have a mismatch. This can be done trivially in time  $\mathcal{O}(d\ell)$  for all strings in  $\mathcal{D}$ .
3. Set the  $z$  of the MU problem to the  $z$  of the PMDM problem.

Thus, the total time  $\mathcal{O}(d\ell)$  needed for Steps 1 to 3 above is clearly polynomial in the size of  $\mathcal{I}_{\text{PMDM}}$ .

▷ **Claim 16.** For any solution  $\mathcal{T}$  to  $\mathcal{I}_{\text{MU}}$  and any solution  $K$  to  $\mathcal{I}_{\text{PMDM}}$ ,  $|K| = |\cup_{S \in \mathcal{T}} S|$ .



Proof. Let  $\mathcal{F} \subseteq \mathcal{D}$  consist of  $z$  strings that match  $q_K$ . Further, let the set  $\mathcal{F}^*$  consist of the elements of  $\mathcal{S}$  corresponding to strings in  $\mathcal{F}$ . We have  $|\cup_{S \in \mathcal{T}} S| \leq |\cup_{S \in \mathcal{F}^*} S| \leq |K|$ .

Now, let  $C = \cup_{S \in \mathcal{T}} S$ . Then,  $q_C = q \otimes C$  matches at least  $z$  strings from  $\mathcal{D}$  and hence  $|K| \leq |C| = |\cup_{S \in \mathcal{T}} S|$ .  $\triangleleft$

To conclude the proof, it remains to show that given a solution  $\mathcal{T}$  to  $\mathcal{I}_{\text{MU}}$  we can obtain a solution  $K$  to  $\mathcal{I}_{\text{PMDM}}$  in time polynomial in the size of  $\mathcal{I}_{\text{MU}}$ . This readily follows from the proof of the above claim: it suffices to set  $K = \cup_{S \in \mathcal{T}} S$ .  $\blacktriangleleft$

**Proof of Theorem 13.** The reduction in Theorem 15 implies that there is a polynomial-time approximation algorithm for PMDM. In particular, Theorem 14 provides an approximation guarantee for MU that depends on the number of sets of the input  $\mathcal{S}$ . In Step 2 of the reduction of Theorem 15, we construct *one* set for the MU instance per *one* string of the dictionary  $\mathcal{D}$  of the PMDM instance. Also, from the constructed solution  $\mathcal{T}$  to the MU instance, we obtain a solution  $K$  to the PMDM instance by simply substituting the positions of  $q$  corresponding to the elements of the sets of  $\mathcal{T}$  with wildcards. This construction implies the approximation result of Theorem 13 that depends on the size of  $\mathcal{D}$ .  $\blacktriangleleft$

**Sanity Check.** Theorem 1 (reduction from  $k$ -CLIQUE to  $k$ -PMDM) and Theorem 13 (approximation algorithm for PMDM) do not contradict the inapproximability results for the maximum clique problem (see Section 3), since our reduction from  $k$ -CLIQUE to  $k$ -PMDM cannot be adapted to a reduction from maximum clique to PMDM-SIZE.

**Two-Way Reduction.** Chlamtáč et al. [21] also show that their polynomial-time  $\mathcal{O}(d^{1/4+\epsilon})$ -approximation algorithm for MU is tight under a plausible conjecture for the so-called Hypergraph Dense vs Random problem. In what follows, we also show that approximating the MU problem can be reduced to approximating PMDM in polynomial time and hence the same tightness result applies to PMDM.

► **Theorem 17.** *MU can be reduced to PMDM in time polynomial in the size of  $\mathcal{S}$ .*

**Proof.** Let  $||\mathcal{S}||$  denote the total number of elements in the  $d$  members of  $\mathcal{S}$ . We reduce the MU problem to the PMDM problem in polynomial time as follows. Given any instance  $\mathcal{I}_{\text{MU}}$  of MU, we construct an instance  $\mathcal{I}_{\text{PMDM}}$  of PMDM by performing the following steps:

1. Sort the union of all elements of members of  $\mathcal{S}$ , and assign to each element  $j$  a unique rank  $\text{rank}(j) \in \{1, \dots, |U|\}$ . Set  $\ell = |U|$ . This can be done in  $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$  time.
2. Set the query string  $q$  equal to the string  $\mathbf{a}^\ell$  of length  $\ell$ . For each set  $S_i$  in  $\mathcal{S}$ , construct a string  $s_i = \mathbf{a}^\ell$ , set  $s_i[\text{rank}(j)] := \mathbf{b}$  if and only if  $j \in S_i$ , and add  $s_i$  to dictionary  $\mathcal{D}$ . This can be done in  $\mathcal{O}(d\ell)$  time.
3. Set the  $z$  of the PMDM problem equal to the  $z$  of the MU problem. This can be done in  $\mathcal{O}(1)$  time.

Thus, the total time  $\mathcal{O}(d\ell \log(d\ell))$  needed for Steps 1 to 3 above is clearly polynomial in the size of  $\mathcal{I}_{\text{MU}}$  as  $\ell \leq ||\mathcal{S}||$ .

A proof of the following claim is analogous to that of Claim 16.

▷ **Claim 18.** For any solution  $\mathcal{T}$  to  $\mathcal{I}_{\text{MU}}$  and any solution  $K$  to  $\mathcal{I}_{\text{PMDM}}$ ,  $|K| = |\cup_{S \in \mathcal{T}} S|$ .

To conclude the proof, it remains to show that, given a solution  $K$  to  $\mathcal{I}_{\text{PMDM}}$ , we can obtain a solution  $\mathcal{T}$  to  $\mathcal{I}_{\text{MU}}$  in time polynomial in the size of  $\mathcal{I}_{\text{PMDM}}$ . It suffices to pick  $z$  sets in  $\mathcal{S}$  that are subsets of  $K$ . Their existence is guaranteed by construction, because such

sets correspond to the at least  $z$  strings in  $\mathcal{D}$  that have  $\mathbf{b}$  in a subset of the positions in  $K$ . This selection can be done naïvely in  $\mathcal{O}(|\mathcal{S}|)$  time. Finally, the above claim guarantees that they indeed form a solution to  $\mathcal{I}_{\text{MU}}$ . ◀

---

## References

- 1 Secure critical data with Oracle Data Safe (white paper). <https://www.oracle.com/a/tech/docs/dbsec/data-safe/wp-security-data-safe.pdf>, September 2020.
- 2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant’s parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.
- 3 Peyman Afshani and Jesper Sindahl Nielsen. Data structure lower bounds for document indexing problems. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, volume 55 of *LIPICs*, pages 93:1–93:15, 2016. doi:10.4230/LIPICs.ICALP.2016.93.
- 4 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
- 5 Alberto Apostolico and Laxmi Parida. Incremental paradigms of motif discovery. *Journal of Computational Biology*, 11(1):15–25, 2004. doi:10.1089/106652704773416867.
- 6 Benny Applebaum. Pseudorandom generators with long stretch and low locality from random local one-way functions. *SIAM Journal on Computing*, 42(5):2008–2037, 2013. doi:10.1137/120884857.
- 7 Hiroki Arimura and Takeaki Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *Journal of Combinatorial Optimization*, 13(3):243–262, 2007. doi:10.1007/s10878-006-9029-1.
- 8 David R. Bailey, Alexis J. Battle, Benedict A. Gomes, and P. Pandurang Nayak. Estimating confidence for query revision models, U.S. Patent US7617205B2 (granted to Google), 2009.
- 9 Martha Bailey, Connor Cole, Morgan Henderson, and Catherine Massey. How well do automated linking methods perform? Lessons from U.S. historical data. NBER Working Papers 24019, National Bureau of Economic Research, Inc, 2017. doi:10.3386/w24019.
- 10 Giovanni Battaglia, Davide Cangelosi, Roberto Grossi, and Nadia Pisanti. Masking patterns in sequences: A new class of motif discovery with don’t cares. *Theoretical Computer Science*, 410(43):4327–4340, 2009. doi:10.1016/j.tcs.2009.07.014.
- 11 Djamal Belazzougui. Faster and space-optimal edit distance "1" dictionary. In *20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009)*, volume 5577 of *Lecture Notes in Computer Science*, pages 154–167. Springer, 2009. doi:10.1007/978-3-642-02441-2\_14.
- 12 Djamal Belazzougui and Rossano Venturini. Compressed string dictionary search with edit distance one. *Algorithmica*, 74(3):1099–1122, 2016. doi:10.1007/s00453-015-9990-0.
- 13 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. String indexing for patterns with wildcards. *Theory of Computing Systems*, 55(1):41–60, 2014. doi:10.1007/s00224-013-9498-4.
- 14 Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *31st ACM Symposium on Theory of Computing (STOC 1999)*, pages 312–321, 1999. doi:10.1145/301250.301330.
- 15 Gerth Stølting Brodal and Srinivasan Venkatesh. Improved bounds for dictionary look-up with one error. *Information Processing Letters*, 75(1-2):57–59, 2000. doi:10.1016/S0020-0190(00)00079-X.
- 16 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation, 4th International Workshop (IWPEC 2009)*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0\_6.

- 17 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. Compressed indexes for approximate string matching. *Algorithmica*, 58(2):263–281, 2010. doi:10.1007/s00453-008-9263-2.
- 18 Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *29th International Colloquium on Automata, Languages and Programming (ICALP 2002)*, pages 451–462, 2002. doi:10.1007/3-540-45465-9\_39.
- 19 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.
- 20 Eden Chlamtáč, Michael Dinitz, Christian Konrad, Guy Kortsarz, and George Rabanca. The densest k-subhypergraph problem. *SIAM Journal on Discrete Mathematics*, 32(2):1458–1477, 2018. doi:10.1137/16M1096402.
- 21 Eden Chlamtáč, Michael Dinitz, and Yury Makarychev. Minimizing the union: Tight approximations for small set bipartite vertex expansion. In *28th ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 881–899, 2017. doi:10.1137/1.9781611974782.56.
- 22 Peter Christen. *Data Matching – Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-31164-2.
- 23 Peter Christen, Thilina Ranbaduge, and Rainer Schnell. *Linking Sensitive Data*. Springer, Heidelberg, 2020. doi:10.1007/978-3-030-59706-1.
- 24 Vincent Cohen-Addad, Laurent Feuilloley, and Tatiana Starikovskaya. Lower bounds for text indexing with mismatches and differences. In *30th ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1146–1164, 2019. doi:10.1137/1.9781611975482.70.
- 25 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don’t cares. In *36th ACM Symposium on Theory of Computing (STOC 2004)*, pages 91–100, 2004. doi:10.1145/1007352.1007374.
- 26 Alfredo Cuzzocrea and Hossain Shahriar. Data masking techniques for nosql database security: A systematic review. In *2017 IEEE International Conference on Big Data (BigData 2017)*, pages 4467–4473, 2017. doi:10.1109/BigData.2017.8258486.
- 27 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with  $O(1)$  worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
- 28 Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In *Algorithms - 22th Annual European Symposium (ESA 2014)*, volume 8737 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2\_38.
- 29 Sreenivas Gollapudi, Samuel Ieong, Alexandros Ntoulas, and Stelios Pappas. Efficient query rewrite for structured web queries. In *20th ACM International Conference on Information and Knowledge Management (CIKM 2011)*, pages 2417–2420, 2011. doi:10.1145/2063576.2063981.
- 30 Roberto Grossi, Giulia Menconi, Nadia Pisanti, Roberto Trani, and Søren Vind. Motif trie: An efficient text index for pattern discovery with don’t cares. *Theoretical Computer Science*, 710:74–87, 2018. doi:10.1016/j.tcs.2017.04.012.
- 31 Johan Hastad. Clique is hard to approximate within  $n^{1-\epsilon}$ . *Acta Mathematica*, 182:105–142, 1999. doi:10.1007/BF02392825.
- 32 Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data quality and record linkage techniques*. Springer, 2007.
- 33 Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4), 2008. doi:10.1145/1391729.1391730.
- 34 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

- 35 T. S. Jayram, Subhash Khot, Ravi Kumar, and Yuval Rabani. Cell-probe lower bounds for the partial match problem. *Journal of Computer and System Sciences*, 69(3):435–447, 2004. doi:10.1016/j.jcss.2004.04.006.
- 36 Dimitrios Karapiperis, Aris Gkoulalas-Divanis, and Vassilios S. Verykios. Summarizing and linking electronic health records. *Distributed and Parallel Databases*, pages 1–40, 2019. doi:10.1007/s10619-019-07263-0.
- 37 Richard M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2010. doi:10.1007/978-3-540-68279-0\_8.
- 38 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *The Multiple-Choice Knapsack Problem*, pages 317–347. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24777-7\_11.
- 39 Pradap Konda, Sanjib Das, Paul Suganthan G.C., Philip Martinkus, Adel Ardalan, Jeffrey R. Ballard, Yash Govind, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. Technical perspective: Toward building entity matching management systems. *SIGMOD Record*, 47(1):33–40, 2018. doi:10.1145/3277006.3277015.
- 40 Hye-Chung Kum, Ashok Krishnamurthy, Ashwin Machanavajjhala, Michael K. Reiter, and Stanley Ahalt. Privacy preserving interactive record linkage (PIRL). *Journal of the American Medical Informatics Association*, 21(2):212–220, 2014. doi:10.1136/amiajnl-2013-002165.
- 41 Hye-Chung Kum, Eric D. Ragan, Gurudev Ilangovan, Mahin Ramezani, Qinbo Li, and Cason Schmit. Enhancing privacy through an interactive on-demand incremental information disclosure interface: Applying privacy-by-design to record linkage. In *Fifteenth USENIX Conference on Usable Privacy and Security*, pages 175–189, 2019. doi:10.5555/3361476.3361489.
- 42 Prathyusha Senthil Kumar, Praveen Arasada, and Ravi Chandra Jammalamadaka. Systems and methods for generating search query rewrites, U.S. Patent US10108712B2 (granted to ebay), 2018.
- 43 Moshe Lewenstein, J. Ian Munro, Venkatesh Raman, and Sharma V. Thankachan. Less space: Indexing for queries with wildcards. *Theoretical Computer Science*, 557:120–127, 2014. doi:10.1016/j.tcs.2014.09.003.
- 44 Moshe Lewenstein, Yakov Nekrich, and Jeffrey Scott Vitter. Space-efficient string indexing for wildcard pattern matching. In *31st Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, pages 506–517, 2014. doi:10.4230/LIPIcs.STACS.2014.506.
- 45 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *29th ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.
- 46 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998. doi:10.1006/jcss.1998.1577.
- 47 George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys*, 53(2), 2020. doi:10.1145/3377455.
- 48 Laxmi Parida, Isidore Rigoutsos, Aris Floratos, Daniel E. Platt, and Yuan Gao. Pattern discovery on character sets and real-valued data: Linear bound on irredundant motifs and an efficient polynomial time algorithm. In *11th ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 297–308, 2000. doi:10.1145/338219.338266.
- 49 Nadia Pisanti, Maxime Crochemore, Roberto Grossi, and Marie-France Sagot. Bases of motifs for generating repeated patterns with wild cards. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):40–50, 2005. doi:10.1109/TCBB.2005.5.
- 50 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. doi:10.1137/09075336X.

- 51 Mihai Pătraşcu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. *SIAM Journal on Computing*, 39(2):730–741, 2009. doi:10.1137/070684859.
- 52 Eric D. Ragan, Hye-Chung Kum, Gurudev Ilangovan, and Han Wang. Balancing privacy and information disclosure in interactive record linkage with visual masking. In *ACM Conference on Human Factors in Computing Systems (CHI 2018)*, 2018. doi:10.1145/3173574.3173900.
- 53 Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976. doi:10.1137/0205003.
- 54 Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1998)*, page 188. Association for Computing Machinery, 1998. doi:10.1145/275487.275508.
- 55 Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information:  $k$ -anonymity and its enforcement through generalization and suppression. Technical report, Computer Science Laboratory, SRI International, 1998.
- 56 Ricardo Jorge Santos, Jorge Bernardino, and Marco Vieira. A data masking technique for data warehouses. In *15th International Database Engineering and Applications Symposium (IDEAS 2011)*, pages 61–69, 2011. doi:10.1145/2076623.2076632.
- 57 Latanya Sweeney. *Computational disclosure control: a primer on data privacy protection*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2001. URL: <http://hdl.handle.net/1721.1/8589>.
- 58 Zehong Tan, Canran Xu, Mengjie Jiang, Hua Yang, and Xiaoyuan Wu. Query rewrite for null and low search results in ecommerce. In *SIGIR Workshop On eCommerce*, volume 2311 of *CEUR Workshop Proceedings*, 2017.
- 59 Yufei Tao. Entity matching with active monotone classification. In *37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS 2018)*, pages 49–62, 2018. doi:10.1145/3196959.3196984.
- 60 Dinusha Vatsalan and Peter Christen. Scalable privacy-preserving record linkage for multiple databases. In *23rd ACM International Conference on Information and Knowledge Management (CIKM 2014)*, pages 1795–1798, 2014. doi:10.1145/2661829.2661875.
- 61 Dinusha Vatsalan, Ziad Sehili, Peter Christen, and Erhard Rahm. Privacy-preserving record linkage for Big Data: Current approaches and research challenges. In *Handbook of Big Data Technologies*, pages 851–895. Springer, 2017. doi:10.1007/978-3-319-49340-4.
- 62 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (SWAT 1973)*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.
- 63 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *2018 International Congress of Mathematicians (ICM)*, pages 3447–3487, 2019. doi:10.1142/9789813272880\_0188.
- 64 Andrew Chi-Chih Yao and Frances F. Yao. Dictionary look-up with one error. *Journal of Algorithms*, 25(1):194–202, 1997. doi:10.1006/jagm.1997.0875.
- 65 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.



# Resilient Level Ancestor, Bottleneck, and Lowest Common Ancestor Queries in Dynamic Trees

Luciano Gualà  

Department of Enterprise Engineering, University of Rome “Tor Vergata”, Italy

Stefano Leucci   

Dept. of Information Engineering, Computer Science, and Mathematics, University of L’Aquila, Italy

Isabella Ziccardi  

Dept. of Information Engineering, Computer Science, and Mathematics, University of L’Aquila, Italy

---

## Abstract

We study the problem of designing a *resilient* data structure maintaining a tree under the Faulty-RAM model [Finocchi and Italiano, STOC’04] in which up to  $\delta$  memory words can be corrupted by an adversary. Our data structure stores a rooted dynamic tree that can be updated via the addition of new leaves, requires linear size, and supports *resilient* (weighted) level ancestor queries, lowest common ancestor queries, and bottleneck vertex queries in  $O(\delta)$  worst-case time per operation.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis

**Keywords and phrases** level ancestor queries, lowest common ancestor queries, bottleneck vertex queries, resilient data structures, faulty-RAM model, dynamic trees

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.66

**Related Version** *Full Version*: <https://arxiv.org/abs/2109.09588>

**Funding** *Luciano Gualà*: This work was partially supported by the project E89C20000620005 “ALgorithmic aspects of BLOckchain TECHnology” (ALBLOTECH).

## 1 Introduction

Due to a diverse spectrum of reasons, ranging from manufacturing defects to charge collection, the data stored in modern memories can sometimes face corruptions, a problem that is exacerbated by the recent growth in the amount of stored data. To make matters worse, even a single memory corruption can cause classical algorithms and data structures to fail catastrophically. One mitigation approach relies on low-level error-correcting schemes that transparently detect and correct such errors. These schemes however either require expensive hardware or employ space-consuming replication strategies. Another approach, which has recently received considerable attention [8, 17, 19, 23, 24, 25, 27], aims to design *resilient* algorithms and data structures that are able to remain operational even in the presence of memory faults, at least with respect to the set of uncorrupted values.

In this paper we consider the problem of designing resilient data structures that store a *dynamic* rooted tree  $T$  while answering several types of queries. More formally, we focus on maintaining a tree that initially consists of a single vertex (the root of the tree) and can be dynamically augmented via the  $\text{AddLeaf}(v)$  operation that appends a new leaf as a child of an existing vertex  $v$ .<sup>1</sup> It is possible to *query*  $T$  in order to obtain information about its current topology. We mainly concerned on the following well-known query types:

---

<sup>1</sup> In the literature this setting is also called *incremental* or *semi-dynamic* to emphasize that arbitrary insertions and deletions of tree vertices/edges are not supported. In this paper, unless otherwise specified, we follow the terminology of [14] by considering *dynamic* trees that only support insertion of leaves.





**(Weighted) Level Ancestor Queries:** Given a vertex  $v$  and an integer  $k$ , the query  $\text{LA}(v, k)$  returns the  $k$ -parent of  $v$ , i.e., the vertex at distance  $k$  from  $v$  among the ancestors of  $v$ . In the weighted version of the problem each vertex of the tree  $T$  is associated with a small (polylogarithmic) positive integer weight, and a query needs to report the closest ancestor  $u$  of  $v$  such that the total weight of the path from  $v$  to  $u$  in  $T$  is at least  $k$ .

**Lowest Common Ancestor Queries:** Given two vertices  $u, v$ , the query  $\text{LCA}(u, v)$  returns the vertex at maximum depth in  $T$  that is simultaneously and ancestor of both  $u$  and  $v$ .

**Bottleneck Queries:** In this problem, each vertex has an associated integer weight and, given two vertices  $u, v$ , a  $\text{BVQ}(u, v)$  query reports the minimum/maximum-weight vertex in the path between  $u$  and  $v$  in  $T$ .<sup>2</sup> It is worth noticing that, when  $T$  is a path, the above problem can be seen as a dynamic version of the classical *range minimum query* problem. In the range minimum query problem, a query  $\text{RMQ}(i, j)$  reports the minimum element between the  $i$ -th and the  $j$ -th element of a (static) input sequence [4].

For all of the above problems, *linear-size non-resilient* data structures supporting both the `AddLeaf` and the query operations in constant worst-case time are known [2, 11]. It is then natural to investigate what can be achieved for the above problem when the sought data structures are required to withstand memory faults.

To precisely capture the behaviour of resilient algorithms, one needs to employ a model of computation that takes into account potential memory corruptions. To this aim, we adopt the *Faulty-RAM* model introduced by Finocchi and Italiano in [19]. This model is similar to the classical RAM model except that all but  $O(1)$  memory words can be subject to corruptions that alter their contents to an arbitrary value. The overall number of corruptions is upper bounded by a parameter  $\delta$  and such corruptions are chosen in a *worst-case* fashion by a computationally unbounded *adversary*.

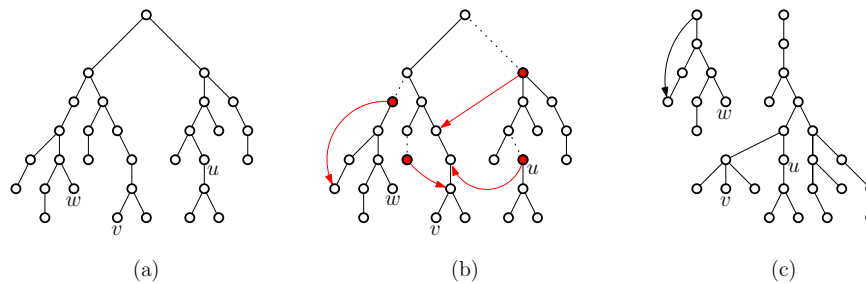
A simple error-correcting strategy based on replication provides a general scheme for obtaining resilient versions of any classical *non-resilient* data structure at a cost of a  $\Theta(\delta)$  blowup in both the time needed for each operation and the size of the data structure. This space overhead is undesirable, especially when  $\delta$  can be large. For the above reason, the main goal in the area is obtaining compact solutions with a particular focus on linear-size data structures [8, 16, 17, 18, 19, 23, 24, 27]. However, for linear-size data structures, even  $\delta = \omega(1)$  corruptions can be already sufficient for the adversary to irreversibly corrupt some of the stored elements [16]. The solution adopted in the literature is that of suitable relaxing the notion of correctness by only requiring queries to answer correctly with respect to the portions of the data structure that are uncorrupted. Notice that this is not easy to obtain since corruptions in unrelated parts of the data structure can still misguide the execution of a query (see [16] for a discussion).<sup>3</sup>

## 1.1 Our results

We design a data structure maintaining a dynamic tree that can be updated via the addition of new leaves, and supports *resilient* (weighted) LA, LCA, and BVQ queries.

<sup>2</sup> It is easy to see that this also captures the well-known *bottleneck edge query* variant [12], in which weights are placed on edges instead of vertices.

<sup>3</sup> For example, the authors of [16] consider the problem of designing linear-size resilient dictionaries adopt a notion of *resilient search* that requires the search procedure to answer correctly w.r.t. all uncorrupted keys (see Section 1.2 for a more precise definition). Notice how the classical solutions based on search trees do not meet this requirement since a single unrelated corruption can destroy the tree path leading to the sought key.



■ **Figure 1** Illustration of resilient LA queries. The current tree  $T$  logically maintained by the data structure is depicted in (a). In this example, each vertex maintains a reference to its parent in  $T$ . In (b) some of the parent-child relationships have been altered by the adversary by corrupting the nodes highlighted in red. Since the algorithm cannot distinguish corrupted memory words from uncorrupted ones, its (defective) view of  $T$  is shown in (c). Nevertheless, a resilient data structure must still be able to correctly answer queries involving uncorrupted paths. For example, the query  $\text{LA}(v, k)$  is required to answer correctly for all (meaningful) values of  $k$  since the path from  $v$  to the root is uncorrupted, while query  $\text{LA}(w, k)$  is required to answer correctly for  $k \leq 2$ . Since  $u$  is corrupted, the query  $\text{LA}(u, k)$  is allowed to answer incorrectly for every value of  $k$ .

Our data structure stores each vertex of the current tree  $T$  in a single memory word of  $\Theta(\log n)$  bits. We will say that a vertex  $v$  is corrupted if the memory word associated with  $v$  has been modified by the adversary. A resilient query is required to correctly report the answer when no vertex in the tree path between the two vertices explicitly or implicitly defined by the query is corrupted. For example, a  $\text{LA}(v, k)$  query correctly reports the  $k$ -parent  $u$  of  $v$  whenever every vertex in the unique path from  $u$  to  $v$  in  $T$  is uncorrupted.

We deem our notion of resilient query to be quite natural since in any reasonable representation of  $T$  the adversary can locally corrupt the parent-children relationship and hence change the observed topology of  $T$ . See Figure 1 for an example.

Our data structure occupies linear (w.r.t. the current number of nodes) space, and supports the **AddLeaf** operation and **LA**, **BVQ**, and **LCA** queries in  $O(\delta)$  worst-case time. For weighted **LA** queries, the above bound on the query time holds as long as  $\delta = O(\text{polylog} n)$ .

We point out that our solution is obtained through a general vertex-coloring scheme which is, in turn, used to “shrink”  $T$  down to a compact tree  $Q$  of size  $O(n/\delta)$  that can be made resilient via replication. Each edge of  $Q$  represents a path of length  $\delta$  between two consecutive colored nodes in  $T$ . If no corruption occurs, this coloring scheme is regular and will color all vertices having a depth that is a multiple of  $\delta$ . While it is possible for corruptions to *locally* destroy the above pattern, our coloring is able to automatically recover as soon as we move away from the corrupted portions of the tree. We feel that such a scheme can be of independent interest as a useful tool to design other resilient data structures involving dynamic trees.

We leave the problem of understanding whether, similarly to other resilient data structures [16, 24], one can prove a lower bound of  $\Omega(\delta)$  on the time needed to perform **AddLeaf** operation and/or to answer our queries.

## 1.2 Related work

**Non-resilient data structures.** Before discussing the known result in our faulty memory model, we first give an overview of the closest related results in the fault-free case. Since the landscape of data structures that answer queries on dynamic trees is vast and diverse, we will focus only on the best-known data-structures capable of answering **LA**, **BVQ**, or **LCA** queries.

As far as LA queries are concerned, the problem has been first formalized in [5] and in [14]. Both papers consider the case in which the tree  $T$  is *static* and show how to build, in linear-time, a data structure that requires linear space and that answers queries in constant worst-case time (albeit the hidden constant in [5] is quite large). A simple and elegant construction achieving the same (optimal) asymptotic bounds is given in [3]. In [14], the *dynamic* version of the problem was also considered: the authors provide a data structure supporting both LA queries and the `AddLeaf` operation in constant *amortized* time. The best known *dynamic* data structure is the one of [2], which implements the above operations in constant *worst-case* time. This data structure also supports constant-time BVQ queries and constant-time weighted LA queries when the vertex weights are polylogarithmically bounded integers. Moreover, the solution of [2] also provides amortized bounds for the problem of maintaining a forest of  $n$  nodes under *link* operations (i.e., edge additions that connect two vertices in different trees of the forest) and LA queries. In this case, a sequence of  $m$  operations requires  $O(m\alpha(m, n))$  time, where  $\alpha$  is the inverse Ackermann function.

Regarding BVQ queries with integer weights, in addition to the solution discussed above (which supports leaf additions and queries in constant-time), [7] shows how to also support leaf deletions using  $O(1)$  amortized time per update and constant worst-case query time.

The problem of answering LCA queries is a fundamental problem which has been introduced in [1]. In [22], Harel and Tarjan show how to preprocess in linear time any *static* tree in order to build a linear-space data structure that is able to answer LCA queries in  $O(1)$  time. The case of *dynamic* trees is also well-understood: it is possible to simultaneously support (i) insertions of leaves and internal nodes, (ii) deletion of leaves and internal nodes with a single child, and (iii) LA queries, in constant worst-case time per operation [11].

**Resilient data structures.** As already mentioned, the Faulty-RAM model has been introduced in [19] and used in the context of resilient data structures in [17] where the authors focused on designing resilient dictionaries, i.e., dynamic sets that support insertions, deletions, and lookup of elements. Here the lookup operation is only required to answer correctly if either (i) the searched key  $k$  is in the dictionary and is uncorrupted, or (ii)  $k$  is not in the dictionary and no corrupted key equals  $k$ . The best-known (linear-size) resilient dictionary is provided in [8] and supports each operation in the optimal  $O(\log n + \delta)$  worst-case time, where  $n$  is the number of stored elements. The Faulty-RAM model has also been adopted in [24], where the authors design a (linear-size) resilient priority queue, i.e., a priority queue supporting two operations: *insert* (which adds a new element in the queue) and *deletemin*. Here *deletemin* deletes and returns either the minimum uncorrupted value or one of the corrupted values. Each operation requires  $O(\log n + \delta)$  amortized time, while  $\Omega(\log n + \delta)$  time is needed to answer an *insert* followed by a *deletemin*.

The Faulty-RAM model has also been adopted in the context of designing resilient algorithms. We refer the reader to [23] for a survey on this topic.

A resilient dictionary for a variant of the Faulty-RAM model in which the set of corruptible memory words is random (but still unknown to the algorithm) has been designed in [25].

In a broader sense, problems that involve non-reliable computation have received considerable attention in the literature, especially in the context of sorting and searching. See for example [9, 10, 13, 15, 20, 21, 26].

### 1.3 Structure of the paper

The paper is organized as follows. Section 2 introduces the used notation and formally defines the Faulty-RAM model. It also briefly describes the error-correcting replication strategy mentioned in the introduction. For technical convenience, in Section 3 and 4 we describe our

data structure for LA queries only. This allows us to introduce all the ideas behind the more general coloring scheme discussed above. As a warm up, we first consider the simpler case in which the tree  $T$  is *static* and is already known at construction time (Section 3), and we then tackle the dynamic version of the problem (Section 4) for which we give our main result. Due to space limitation, the description of how to modify our data structure to handle the other types of queries is omitted and can be found in the full version of the paper.

## 2 Preliminaries

**Notation.** Let  $T$  be a rooted tree. For each node  $v \in T$ , we denote with  $\text{parent}(v)$  the parent of  $v$ . If  $\pi$  is a path, we denote by  $|\pi|$  its length, i.e., the number of its edges. Given any two nodes  $u, v$ , we denote by  $d_T(u, v)$  the length of the (unique) path between  $u$  and  $v$  in  $T$ . Moreover, if  $\pi$  traverses  $u$  and  $v$ , we denote by  $\pi[u : v]$  the subpath of  $\pi$  between  $u$  and  $v$ , endpoints included. We will use round brackets instead of square brackets to denote that the corresponding endpoint is excluded (so that, e.g.,  $\pi(u : v]$  denotes the subpath of  $\pi$  between  $u$  and  $v$  where  $u$  is excluded and  $v$  is included).

**Faulty memory model.** We now formally describe the *Faulty-RAM* model introduced by Finocchi and Italiano in [19]. In this model the memory is divided in two regions: a *safe region* with  $O(1)$  memory locations, whose locations are known to the algorithm designer, and the (unreliable) *main memory*. An adaptive adversary can perform up to  $\delta$  corruptions, where a corruption consists in instantly modifying the content of a word from the main memory. The adversary knows the algorithm and the current contents of the memory, has an unbounded computational power, and can simultaneously perform one or more corruptions at any point in time. The safe region cannot be corrupted by the adversary and there is no error-detection mechanism that allows the algorithm to distinguish the corrupted memory locations from the uncorrupted ones.

Without assuming the existence of  $O(1)$  words of safe memory, no reliable computation is possible: in particular, the safe memory can store the code of the algorithm itself, which otherwise could be corrupted by the adversary.

As observed in [16] (and already mentioned in the introduction), there is a simple strategy that allows any non-fault tolerant data structure on the RAM model to also work on the Faulty-RAM model, albeit with multiplicative  $\Theta(\delta)$  blow-up in its time and space complexities. Essentially, such a solution implements a trivial error-correcting mechanism by simulating each memory word  $w$  in the RAM model with a set  $W$  of  $2\delta + 1$  memory words in the Faulty-RAM model: writing a value  $x$  to  $w$  means writing  $x$  to all words in  $W$ , and reading  $w$  means computing the majority value of the words in  $W$  (which can be done in  $O(\delta)$  time, and  $O(1)$  space using the safe memory region and the Boyer-Moore majority vote algorithm [6]). We refer to such technique as the *replication strategy*.

## 3 Warming Up: LA queries in Static Trees

In order to introduce our ideas, in this section we will show how to build a simplified version of our resilient data structure when the tree  $T$  cannot be dynamically modified. Our simplified data structure requires linear space and answers level-ancestor queries in  $O(\delta)$  time. As opposed to our *dynamic* data structure, in this special case the tree  $T$  must be known in advance and hence we need to initialize our data structure from an input tree  $T$ . For simplicity, we assume that no corruptions occur while our data structure is being built. Notice that this assumption can be removed by carefully using the replication strategy described in Section 2.

■ **Algorithm 1** Answers a level ancestor query  $\text{LA}(v, k)$  in the special case of static trees.

---

```

1 if  $k \leq 2\delta$  then return  $\text{climb}(v, k)$  ;
2 Climb up the tree  $T$  from  $v$  for  $2\delta$  nodes searching a black node;
3 if the previous procedure did not find a black node then return error;
4  $v' \leftarrow$  a black node found in the previous procedure;
5  $d \leftarrow$  distance between  $v'$  and  $v$ ;  $k' \leftarrow k - d$ ;
6  $u' \leftarrow \text{LA}_Q(q_{v'}, \lfloor k'/\delta \rfloor)$ ;
7  $k_{\text{rest}} \leftarrow k' - \lfloor k'/\delta \rfloor \cdot \delta$ ;
8 return  $\text{climb}(u', k_{\text{rest}})$ ;

```

---

**Description of the Data Structure.** Let  $T$  be a rooted tree with  $n$  nodes. To define the data structure for  $T$ , we need to divide the nodes of  $T$  into two sets: the *black* nodes and the *white* nodes. We define the set of black nodes to ensure that its cardinality is  $O(n/\delta)$ : a node  $v$  in  $T$  is *black* if we simultaneously have that (i) its depth in  $T$  is a multiple of  $\delta$ , and (ii) the subtree of  $T$  rooted in  $v$  has height at least  $\delta - 1$ . A node  $v$  in  $T$  is *white* if it is not black. We notice that for each black  $v$  node in  $T$  there are at least  $\delta$  distinct nodes (i.e., all the vertices in the path from  $v$  to any vertex having depth  $\delta - 1$  in the subtree of  $T$  rooted at  $v$ ), thus implying that the total number of black nodes in  $T$  is at most  $n/\delta$ .

If we define a relation of parenthood for the black nodes of  $T$ , we can define a new *black tree*  $Q$  in which each vertex  $\bar{v}$  is associated with black vertex  $v$  of  $T$ . The parent of  $\bar{v}$  in  $Q$  is the vertex  $\bar{u}$  corresponding to the lowest black proper ancestor  $u$  of  $v$  in  $T$ . See Figure 2 for an example.

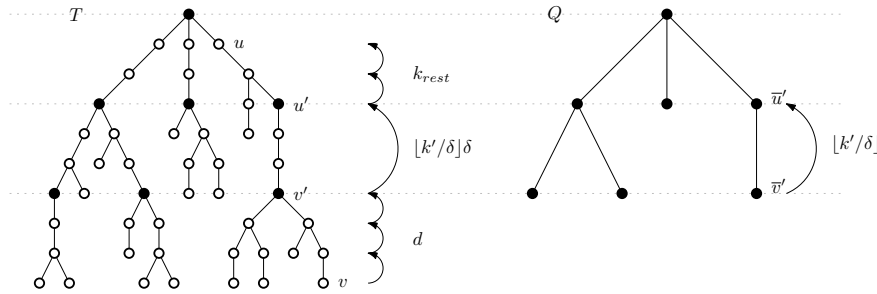
Our data structure stores the (colored) tree  $T$ , as described in the following, along with an additional data structure  $D_Q$  that is able to answer LA queries on  $Q$ . The tree  $T$  is stored as an array of records, where each record is associated with a vertex of  $T$ , occupies  $\Theta(\log n)$  bits, and is stored in a single memory word. The memory word associated with a node  $v$  stores:

- a pointer  $p_v$  to  $\text{parent}(v)$ , if any. If  $v$  is the root of  $T$  then  $p_v = \text{null}$ ;
- a pointer  $q_v$  to the corresponding node  $\bar{v}$  in  $Q$ , if any. If no such node exists, i.e. if  $v$  is white, then  $q_v = \text{null}$ .

Moreover we maintain, for each vertex  $\bar{v}$  of  $Q$ , a pointer to the corresponding vertex  $v$  of  $T$  as satellite data. The data structure  $D_Q$  is the resilient version of any (non-resilient) data structure that is capable of answering LA queries on static trees in constant time and requires linear space (see, e.g., the data structure in [3]).

As we observed before, any data structure can be made resilient with a multiplicative  $\Theta(\delta)$  blow-up in its time and space complexities. In our case, since the number of vertices in  $Q$  is  $O(n/\delta)$ , the final space required to store  $D_Q$  is  $O(n)$  and the query time becomes  $O(\delta)$ . Notice that, in spite of the (at most  $\delta$ ) memory corruptions performed by the adversary, the data structure  $D_Q$  always returns the correct answer to all possible LA queries on  $Q$ . We will denote by  $\text{LA}_Q(\bar{v}, k)$  the level-ancestor query on  $Q$ , which returns the vertex of  $T$  corresponding to the  $k$ -parent of  $v$  in  $Q$ .

**The resilient level-ancestor query.** In this section we show how to implement our resilient LA query. We start by defining a routine that will be useful in the sequel: if  $v$  is a node of  $T$  and  $i$  is a non-negative integer, we denote by  $\text{climb}(v, i)$  a procedure that returns the vertex reached by a walk on  $T$  that starts from  $v$  and iteratively moves to the parent of the current vertex  $i$  times. When the procedure encounters a vertex  $u$  with pointer  $p_u = \text{null}$  that has



**Figure 2** Left: A static tree  $T$  that has been colored according to the scheme in Section 3 for  $\delta = 3$ . Right: the corresponding black tree  $Q$ . We also show the path climbed while answering the query  $\text{LA}(v, k)$  with  $k = 8$ . In this case  $d = 3$ ,  $\lfloor k'/\delta \rfloor = 1$  and  $k_{\text{rest}} = 2$ . Notice how  $Q$  is used to quickly reach  $u'$  from  $v'$ .

to be followed,  $\text{climb}(v, i)$  reports that the root has been reached. Notice that  $\text{climb}(v, i)$  requires  $O(i)$  time and, whenever no corrupted vertices are encountered during the walk, it correctly returns the  $i$ -parent of  $v$ . Although the  $\text{climb}(\cdot, \cdot)$  procedure could immediately be used to answer an LA query, doing so require  $\Omega(n)$  time in the worst case. To improve the query time we use the data structure  $D_Q$  described above and we distinguish between *short* and *long* LA queries depending on the value of  $k$ .

Short queries, i.e., queries  $\text{LA}(v, k)$  with  $k \leq 2\delta$ , are handled by simply invoking  $\text{climb}(v, k)$  and, from the above observation, it follows that this is a resilient query. For longer queries the idea is that of locating a nearby black ancestor of  $v$ , performing an  $\text{LA}_Q$  query on  $Q$  to quickly reach a black descendant  $u'$  of the  $k$ -parent  $w$  of  $v$  such that  $d(u', w) \leq \delta$ , and finally using the  $\text{climb}$  procedure once more to reach  $w$  from  $u'$ . See Algorithm 1.

During the execution of our resilient query algorithm we always ensure that all followed pointers are valid. Since we are dealing with a static tree  $T$ , we can handle invalid pointers simply by halting the whole query procedure and reporting an error. A slightly more sophisticated handling of invalid pointers will be used to tackle the dynamic case. An example LA query is given in Figure 2.

The correctness of the above algorithm immediately follows from the fact that, when no vertex between  $v$  and the  $k$ -th ancestor  $v$  is corrupted,  $v$  must have a black ancestor at distance at most  $2\delta$  and from the fact that the replication strategy ensures that all queries on  $Q$  are always answered correctly.<sup>4</sup>

To show that Algorithm 1 answers an LA query in  $O(\delta)$  time, we notice that the  $\text{climb}$  operations in lines 1 and 8 require time  $O(\delta)$ , and so does line 2. Moreover, the query to  $D_Q$  (line 6) can also be performed in  $O(\delta)$  time as discussed above.

#### 4 LA queries in dynamic trees

In this section we provide our main result for LA queries. In the full version of the paper, we show how our ideas can be extended to also handle weighted LA, BVQ, and LCA queries.

<sup>4</sup> Here the distance of  $2\delta$  is essentially tight as it can be seen, e.g., by considering a tree  $T$  consisting of a path of length  $2\delta - 2$  rooted in one of its endpoints. The only black vertex of  $T$  is the root. Notice how the vertex  $u$  at depth  $\delta$  is white since the subtree of  $T$  rooted in  $u$  has height  $\delta - 2$ .

## 4.1 Description of the Data Structure

Some of the key ideas behind our data structure for LA queries in dynamic trees are extensions of the ones used for static case. Namely, the nodes of  $T$  are colored with either *black* or *white*, the set of *black* nodes will have size  $O(n/\delta)$ , and it will correspond to the vertex set of an auxiliary *black forest*  $Q$ . Ideally, in absence of corruptions,  $Q$  is exactly the black tree as defined in the static case, namely the tree in which the parent of each (black) node  $\bar{v}$  in  $Q$  is the vertex  $\bar{u}$  associated with the lowest black proper ancestor  $u$  of  $v$  in  $T$ .

Moreover, we would still like the vertices of  $T$  having a depth that is a multiple of  $\delta$  to be colored black, similarly to the static case. However, we can no longer afford to maintain such a rigid coloring scheme since the tree is now being dynamically constructed via successive `AddLeaf` operations, and the adversary's corruptions might cause vertices to become miscolored. We will however ensure that such a regular coloring pattern will be followed by the portions of  $T$  that are sufficiently distant from the adversary's corruptions. This will allow us to answer LA queries using a strategy similar to the one employed for the static case.

Our data structure stores the following information. The record of a node  $v$  maintains, in addition to the pointer  $p_v$  to its parent and to the pointer  $q_v$  to the corresponding node  $\bar{v}$  in  $Q$  (if any), an additional field  $\text{flag}_v$ . Intuitively,  $\text{flag}_v$  can be thought of as a Boolean value in  $\{\perp, \top\}$ . The initial value of a flag is  $\perp$  and we say that the flag is *unspent*. Spending a flag means setting it to  $\top$ . We will spend these flags to “pay” for the creation of new black nodes. Spent flags will also signal the presence of a nearby black ancestor.

For technical reasons, we also allow an unspent flag  $\text{flag}_v$  to be additionally *annotated* with a pair  $(x, i)$  where  $x$  is (the name of) a node and  $i$  is an integer. In practice this amounts to setting  $\text{flag}_v$  to  $(x, i)$ , which is logically interpreted as  $\perp$ . Such an annotated flag is still unspent. This provides an additional safeguard against corruptions that may occur during the execution of our leaf insertion algorithm (see Section 4.2).

The node records are stored into a dynamic array  $\mathcal{A}$ , whose current size  $n$  is kept in the safe region of memory. This array supports both elements insertions and random accesses in constant worst-case time.<sup>5</sup>

The pointer  $p_v$  is then the index (in  $\{0, \dots, n-1\}$ ) of the record corresponding to the parent of  $v$  in  $\mathcal{A}$ . Initially,  $\mathcal{A}$  only contains the root  $r$  of  $T$  at index 0. Moreover, we will always store new leaves at the end of  $\mathcal{A}$  so that, in absence of corruptions, the index of a vertex  $v$  in  $\mathcal{A}$  is always smaller than the index of any of its descendants. As a consequence, whenever we observe the index stored in pointer  $p_v$  is greater than or equal to the index of  $v$  itself, we know that  $v$  must have been corrupted by the adversary. We find convenient to use the above fact to simplify the handling of corrupted vertices: whenever we encounter an invalid pointer  $p_v \geq v$  we treat it as being equal to 0, i.e., an invalid pointer  $p_v$  always refers to the root  $r$  of  $T$ . This rule also applies to any read pointer, including those accessed by the `climb( $\cdot$ ,  $\cdot$ )` procedure already defined in Section 3.

Then the (possibly corrupted) contents of  $\mathcal{A}$ , at any point in time, induce a *noisy tree*  $\mathcal{T}$  whose root is  $r$ , and the parent of each vertex  $v \neq r$  is the vertex pointed by  $p_v$  according to the above rule. Clearly, if no corruptions occur  $T$  and  $\mathcal{T}$  coincide.

<sup>5</sup> The standard textbook technique which handles insertions into already full array by moving the current elements into a new array of double capacity already achieves  $O(1)$  amortized time per insertion. With some additional technical care, the above bound also holds in the worst case. The idea is to distribute the work needed to move elements into the new array over the insertions operations that would cause the current array to become full (it suffices to move 2 elements per insertion). Using this scheme, at any point in time, each element is stored into a single memory word.



Moreover, we store a resilient data structure  $D_Q$  that, in addition to the already-defined  $\text{LA}_Q(\bar{v}, k)$  query, also supports the following additional operations in  $O(\delta)$  time.

$\text{NewTree}_Q(v)$ : Given a vertex  $v$  of  $T$ , it creates a new tree in the forest  $Q$  consisting of a single vertex  $\bar{v}$  associated to  $v$ , and it returns a pointer to  $\bar{v}$ .

$\text{AddLeaf}_Q(\bar{u}, v)$ : Given a vertex  $\bar{u}$  of  $Q$ , and a vertex  $v$  of  $T$ , it creates a new vertex  $\bar{v}$  associated to  $v$  as a children of  $\bar{u}$  in  $Q$ . Finally, it returns a pointer to the newly added vertex  $\bar{v}$ .

This data structure  $D_Q$  is the resilient version, obtained using the replication strategy, of the linear-size data structure that supports both the  $\text{AddLeaf}$  operation and  $\text{LA}$  queries in constant time [2]. Notice that  $D_Q$  always returns the correct answer to all possible  $\text{LA}$  queries on  $Q$ . Moreover, once we ensure that the number of vertices that become black (and hence the size of  $Q$ ) is always  $O(n/\delta)$ , we have that the (resilient) data structure  $D_Q$  requires  $O(n)$  space (this will be shown formally in the proof Theorem 6).

## 4.2 The AddLeaf operation

Before describing our implementation of the  $\text{AddLeaf}$  operation, it is useful to give some additional definitions. We say that  $v$  is *near-a-black* in a tree  $\tilde{T}$  if there exists some  $k \in \{1, 2, \dots, \delta\}$  such that the  $k$ -parent of  $v$  in  $\tilde{T}$  is black. Moreover, we say that  $v$  is *black-free* in  $\tilde{T}$  if no  $k$ -parent of  $v$  in  $\tilde{T}$  for  $k \in \{1, 2, \dots, 2\delta - 1\}$  is black.

The procedure  $\text{AddLeaf}(x_{par})$  takes a vertex  $x_{par}$  of  $T$  as input and adds a new child  $x$  of  $x_{par}$  to  $T$  (see Algorithm 2). The record corresponding to new vertex  $x$  is appended at the end of the dynamic array  $\mathcal{A}$ . For simplicity we will assume that, during the execution of  $\text{AddLeaf}(x_{par})$ , the record of vertex  $x$  is never corrupted by the adversary. This can be guaranteed without loss of generality since a (temporary) record for  $x$  can be kept in safe memory and copied back to  $\mathcal{A}$  (which is stored in the unreliable main memory) at the end of the procedure.

Our algorithm consists of a first *discovery* phase and possibly of a second additional *execution* phase. The aim of the discovery phase is that of exploring the current tree by climbing up to  $3\delta - 1$  levels of  $\mathcal{T}$  from  $x$  while gathering information for the second phase. In order to do so, Algorithm 2 climbs  $\delta$  levels of  $\mathcal{T}$  from the newly inserted node  $x$ , reaching a vertex  $y$ , and checks during the process that all the flags associated with the traversed nodes are unspent. If any of these flags is spent, we immediately return from the  $\text{AddLeaf}(x_{par})$  procedure without performing the execution phase. Otherwise, the algorithm climbs  $2\delta - 1$  further levels from  $y$  to determine whether  $y$  appears to be black-free or near-a-black. In the latter case, it keeps track of the distance  $\ell$  from  $y$  to the closest black proper ancestor  $y'$  of  $y$  that is encountered. If  $y$  is neither black-free nor near-a-black, we return from the  $\text{AddLeaf}(x_{par})$  procedure (without performing the execution phase), otherwise we move on to the execution phase. A technical detail of the discovery phase is the following: while climbing from  $x_{par}$  to  $y$ , the generic  $i$ -th unspent flag is annotated with  $(x, i)$  (possibly overwriting any existing previous annotation) and will be checked by the execution phase. Recall that these flags remain unspent.

The execution phase once again climbs  $\delta$  levels of  $\mathcal{T}$  starting from  $x$ , with the goal of changing the color of an existing white vertex to black (hence creating a corresponding black node in  $Q$ ). This is guaranteed to happen unless the annotations of the unspent flags set during the discovery phase reveal that one such vertex has been corrupted in the meantime. The creation of a new black vertex in  $Q$  is “paid for” by *spending* these  $\delta$  unspent flags (i.e., setting them to  $\top$ ). The position of the new black vertex depends on whether  $y$  was

---

**Algorithm 2**  $\text{AddLeaf}(x_{par})$ .

---

```

1 Add a new record  $x$  at the end of  $\mathcal{A}$ ;  $p_x \leftarrow x_{par}$ ;  $\text{flag}_x \leftarrow \perp$ ;  $q_x \leftarrow \text{null}$ ;
   // Discovery Phase
2  $y \leftarrow x$ ; // Check the flags of the lowest  $\delta$  proper ancestors of  $x$ 
3 for  $i = 1, \dots, \delta$  do
4   if  $y = r$  then return  $x$ ;
5    $y \leftarrow p_y$ ;
6   if  $\text{flag}_y = \top$  then return  $x$ ; // Return immediately if a spent flag is found
7    $\text{flag}_y \leftarrow (x, i)$ ; // Annotate  $\text{flag}_y$ 

   // Check whether  $y$  is near-a-black.
   //  $\ell$  will be the distance to the closest proper black ancestor  $y'$  of  $y$ , if any
8  $y' \leftarrow y$ ;  $\ell \leftarrow 0$ ; near_black  $\leftarrow$  false;
9 while  $\ell < \delta$  and  $y' \neq r$  and near_black = false do
10   $y' \leftarrow p_{y'}$ ;  $\ell \leftarrow \ell + 1$ ;
11  if  $y'$  is black then near_black  $\leftarrow$  true;

   // If  $y$  is not near-a-black, check whether it is black-free
12 if near_black = false then
13   $z \leftarrow y'$ ;
14  for  $\delta - 1$  times do
15    if  $z = r$  then break;
16     $z \leftarrow p_z$ ;
17    if  $z$  is black then return  $x$ ;

   // Execution Phase. (Node  $y$  was either near-a-black or black-free)
   // Acquire the flags of the lowest  $\delta$  proper ancestors of  $x$ 
18  $z \leftarrow x$ ;
19 for  $i = 1, \dots, \delta$  do
20  if  $z = r$  then return  $x$ ;
21   $z \leftarrow p_z$ ;
22  if  $\text{flag}_z \neq (x, i)$  then return  $x$ ; // Check the annotation of  $\text{flag}_z$ 
23  if near_black = true and  $i = \ell$  then  $x' \leftarrow z$ ; //  $y'$  is the  $\delta$ -parent of  $x'$ 
24   $\text{flag}_z \leftarrow \top$ ; // Spend  $\text{flag}_z$ 

25 if near_black = true then
26   $q_{x'} \leftarrow \text{AddLeaf}_Q(q_{y'}, x')$ ;
27 else
28   $q_y \leftarrow \text{NewTree}_Q(y)$ ;
29 return  $x$ ;

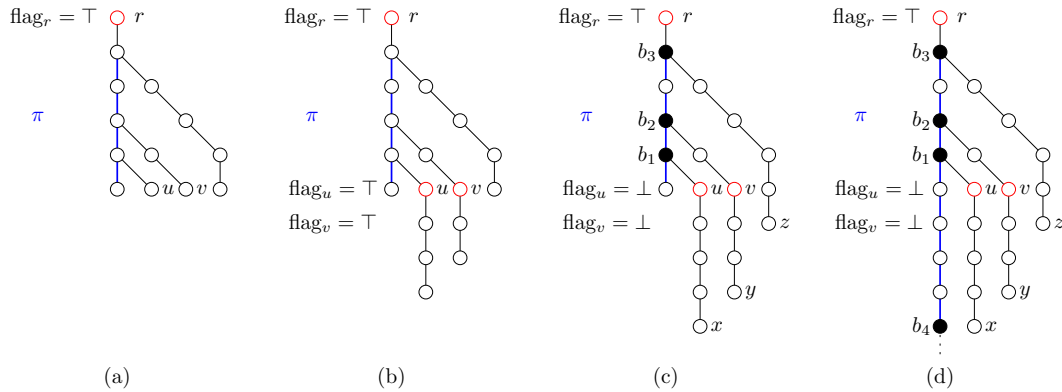
```

---

near-a-black or black-free. In the former case the vertex  $y'$  discovered in the first phase will be the  $\delta$ -parent of the new black vertex  $x'$ , and a new leaf  $\bar{x}'$  is appended to  $\bar{y}'$  in  $Q$ . In the latter case,  $y$  will become black and a new tree containing a single vertex  $\bar{y}$  is added to  $Q$ . Notice that, if a vertex  $b$  is colored black during the  $\text{AddLeaf}$  operation, the execution phase always spends  $\text{flag}_b$ .

### 4.3 Analysis of the data structure

In this section we analyze our data structure. The core of the analysis is to show that the  $\text{AddLeaf}$  operation in Algorithm 2 guarantees that in  $\mathcal{T}$ , if we are sufficiently distant from all the corrupted vertices, the black nodes are regularly distributed. The formal property



■ **Figure 3** An example showing that an uncorrupted path  $\pi$  (depicted in blue) can exhibit an irregular pattern of black vertices (d). Situation (a) can be reached when the adversary corrupts  $r$  by setting  $\text{flag}_r = \top$  before the insertions of the other nodes take place. To obtain (b), the adversary can set  $\text{flag}_u = \text{flag}_v = \top$ , thus corrupting  $u$  and  $v$  before  $u$  and  $v$ 's descendants are inserted. If the adversary sets  $\text{flag}_u$  and  $\text{flag}_v$  back to  $\perp$  before  $x, y$ , and  $z$  are inserted (in this order), we arrive in configuration (c) in which  $b_1, b_2$ , and  $b_3$  have been colored black. Inserting the remaining vertices yields (d).

is stated in Lemma 5. We first need to prove auxiliary properties. In Figure 3 we give an example that shows that, even in an uncorrupted path, if we are not sufficiently distant from corruptions, the black nodes can form irregular patterns in the path.

The following lemma shows that if the flag of a vertex  $w$  appears to be spent, then either there must be a nearby black ancestor of  $w$ , unless a nearby corruption occurred. See Figure 4 (a).

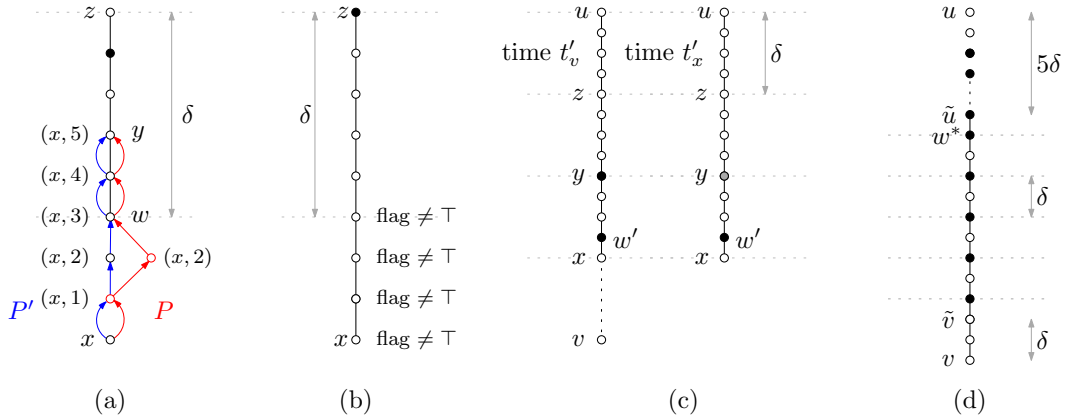
► **Lemma 1.** *Let  $w$  and  $z$  be two nodes such that  $z$  is the  $\delta$ -parent of  $w$  in  $T$  and such that no node in the path  $\pi$  from  $z$  to  $w$  in  $T$  has been corrupted. If  $\text{flag}_w = \top$ , then there exists a black node in  $\pi(z : w)$ .*

**Proof.** Let  $x$  be the node whose insertion in  $T$  caused  $\text{flag}_w$  to be set to  $\top$ . Moreover, let  $P$  be the path of length  $\delta$  from  $x$  to  $y$  traversed in the discovery phase of Algorithm 2 in lines 2–7. Similarly, let  $P'$  be the path from  $x$  to  $y$  traversed in the execution phase of Algorithm 2 in lines 18–24.

Clearly,  $P'$  contains  $w$ . Moreover, if  $w$  is the  $i$ -th node traversed in  $P'$ , then  $\text{flag}_w = (x, i)$  in the execution phase and (since  $w$  is uncorrupted),  $\text{flag}_w$  was set to  $(x, i)$  in the discovery phase. As a consequence,  $w$  is also the  $i$ -th node in  $P$  and  $P[y : w] = P'[y : w]$ . Hence,  $y$  is at distance at most  $\delta - 1$  from  $w$  in  $P$  (and in  $T$ ) showing that  $z$  is a proper ancestor of  $y$ . Therefore all nodes in  $P'[y : w]$  are uncorrupted, and the loop in lines 18–24 of Algorithm 2 is executed to completion. This ensures that the execution phase will color a node  $b$  black. We distinguish two cases depending on whether  $y$  was observed to be near-a-black or black-free in the discovery phase.

If  $y$  is black-free, then  $b$  is exactly  $y$  and the claim follows. Otherwise,  $y$  is near-a-black and the discovery phase computed the distance  $\ell$  between  $x$  and its closest black proper ancestor. If  $\ell \geq i$ , then Algorithm 2 colors a vertex in  $P(z : w] = \pi(z : w]$  black. Otherwise, if  $\ell < i$ , the discovery phase observed that the  $\ell$ -parent  $y'$  of  $y$  was black. Since  $\ell < \delta$ ,  $y'$  lies in  $\pi(z : y]$ . ◀

Next lemma shows that an uncorrupted path of length at least  $3\delta$  must contain a black vertex.



■ **Figure 4** (a) Graphical representation of the proof of Lemma 1, for  $\delta = 5$ . (b), (c), (d): Representations of the statements of Lemma 3, Lemma 4, and Lemma 5, respectively.

► **Lemma 2.** *Let  $x$  and  $z$  be nodes in  $T$  such that  $z$  is the  $3\delta$ -parent of  $x$  in  $T$  and such that no node in the path  $\pi$  from  $x$  to  $z$  in  $T$  has been corrupted. Then, there exists a black node  $w$  in  $\pi[z : x]$ .*

**Proof.** Since no vertex in  $\pi$  has been corrupted, the path  $\pi$  must also belong to the noisy tree  $\mathcal{T}$ . In the rest of the proof we assume that  $\pi[z : x]$  contains no black nodes and show that this leads to a contradiction.

Let  $y$  the  $\delta$ -parent of  $x$  in  $\pi$  and let  $t_x$  be the time at which the  $\text{AddLeaf}(\cdot)$  operation that adds  $x$  to  $T$  is invoked. We know that, at time  $t_x$ , there exists no node  $w$  in  $\pi[y : x]$  such that  $\text{flag}_w = \top$  since otherwise Lemma 1 would immediately imply the existence of a black node in  $\pi[z : w]$  contradicting the initial assumption. Then, the invocation of Algorithm 2 that inserts  $x$  also performs its execution phase.

Moreover,  $y$  must be black-free at time  $t_x$ , and hence it is colored black during such a phase (refer to the pseudocode of Algorithm 2, and recall that a black-free node is not near-a-black). Since  $y$  is not corrupted it must still be black, leading to a contradiction. ◀

Recall that we would like each uncorrupted path to contain a black vertex every  $\delta$  levels. Consider an uncorrupted path  $\pi$  of length between  $\delta$  and  $2\delta$  with a single black vertex  $z$  on top. Then, the vertex at distance  $\delta$  from  $z$  is “overdue” to become black. Next lemma shows that all flags associated with descendants of the overdue vertex in  $\pi$  must be unspent. In some sense, the data structure is preparing to recolor the missing black vertex. This will happen once  $\delta$  unspent flags are available. See Figure 4 (b).

► **Lemma 3.** *Let  $x$  and  $z$  be two nodes in  $T$  such that:  $z$  is an ancestor of  $x$  in  $T$ , no node in the path  $\pi$  from  $z$  to  $x$  in  $T$  has been corrupted, and  $\delta \leq |\pi| < 2\delta$ . We have that, immediately after vertex  $x$  is inserted, if the only black vertex in  $\pi$  is  $z$  then all the nodes  $w$  in  $\pi$  at distance at least  $\delta$  from  $z$  in  $T$  are such that  $\text{flag}_w \neq \top$ .*

**Proof.** Since no vertex in  $\pi$  has been corrupted, the path  $\pi$  must also belong to the noisy tree  $\mathcal{T}$ . In what follows, we prove that, immediately after vertex  $x$  is inserted, the existence of a node  $w$  between  $x$  and  $z$  in  $\pi$  such that  $d_{\mathcal{T}}(w, z) \geq \delta$  and  $\text{flag}_w = \top$  leads to a contradiction. Indeed, since  $\text{flag}_w = \top$ , Lemma 1 implies the existence of a black node in  $\pi[z : w]$ , and this contradicts the fact that  $z$  is the only black node in  $\pi[z : x]$ . ◀

The next technical lemma is about the timing at which vertices of a long uncorrupted path become black. This will be instrumental to prove Lemma 5. See Figure 4 (c).

► **Lemma 4.** *Let  $u$  and  $v$  be two nodes in  $T$  such that  $u$  is an ancestor of  $v$ ,  $d_T(u, v) \geq 3\delta$  and no node in the path  $\pi$  from  $v$  to  $u$  in  $T$  has been corrupted. Let  $y$  (resp.  $x$ ) be the node in  $\pi$  at distance  $2\delta$  (resp.  $3\delta$ ) from  $u$  in  $\pi$ . Let  $t'_v$  (resp.  $t'_x$ ) be the time immediately after the vertex  $v$  (resp.  $x$ ) is inserted. If the node  $y$  is black at time  $t'_v$ , then there exists a node  $w'$  in  $\pi[y : x]$  that is black at time  $t'_x$ .*

**Proof.** Since no vertex in  $\pi$  has been corrupted, the path  $\pi$  must also belong to the noisy tree  $\mathcal{T}$ . In the rest of the proof we assume towards a contradiction that  $y$  is black at time  $t'_v$ , yet there are no black nodes in  $\pi[y : x]$  at time  $t'_x$ .

Let  $z$  be the  $\delta$ -parent of  $y$  in  $\pi$ . Let  $\bar{t}_y$  be the time immediately before  $y$  is colored black. At time  $\bar{t}_y$  there are only two possible scenarios:

**Scenario 1:** At time  $\bar{t}_y$ , the node  $y$  is black-free;

**Scenario 2:** At time  $\bar{t}_y$ , the node  $z$  is the only black node in  $T$  in  $\pi[z : y]$ .

We denote with  $t_x$  the time immediately before vertex  $x$  is inserted in  $T$  and we consider the two scenarios separately (notice that  $\bar{t}_y$  refers to a later time than  $t_x$ ). We split scenario 1 into two additional subcases:

**Subcase 1.1:** at time  $t_x$  all the nodes  $w$  in  $\pi[y : x]$  are such that  $\text{flag}_w \neq \top$ ;

**Subcase 1.2:** at time  $t_x$  there is a node  $w$  in  $\pi[y : x]$  such that  $\text{flag}_w = \top$ .

We start considering subcase 1.1. Since  $\bar{t}_y$  follows  $t_x$ , and  $y$  is black-free at time  $\bar{t}_y$ , vertex  $y$  must also be black-free at time  $t_x$ . Then, during the insertion of  $x$ , Algorithm 2 colors  $y$  black yielding a contradiction.

We now analyze subcase 1.2. Since  $\text{flag}_w = \top$ , Lemma 1 implies the existence of a black node  $b$  in  $\pi[w, z]$  and, since we assume that there are no black nodes in  $\pi[y : x]$ ,  $b$  is in  $\pi(z : y)$ . This shows that  $y$  cannot be black-free at time  $\bar{t}_y$  and contradicts the hypothesis of scenario 1.1.

We now consider Scenario 2, which we subdivide into three subcases:

**Subcase 2.1.** at time  $t_x$  all the nodes  $w$  in  $\pi[y : x]$  are such that  $\text{flag}_w \neq \top$  and  $z$  is white;

**Subcase 2.2.** at time  $t_x$  all the nodes  $w$  in  $\pi[y : x]$  are such that  $\text{flag}_w \neq \top$  and  $z$  is black;

**Subcase 2.3.** at time  $t_x$  there is a node  $w$  in  $\pi[y : x]$  such that  $\text{flag}_w = \top$ .

We start by handling subcase 2.1. For the initial assumption, and for definition of this case, we have that there are no black nodes in  $\pi[z : x]$  at time  $t_x$ . Since  $z$  is colored black at some time  $\bar{t}_z$  following  $t_x$ , we know that the  $\delta - 1$  nodes ancestor of  $z$  are not black at time  $t_x$ , since this is incompatible with the fact that  $z$  will become black. Since  $\pi$  is not corrupted, we know that  $y$  is black-free in  $T$  at time  $t_x$ . This implies that  $y$  is colored black during the insertion of  $x$  in  $T$ , and hence  $y$  is black at time  $t'_x$  contradicting our hypotheses.

We proceed by analyzing subcase 2.2. At time  $\bar{t}_y$  all nodes in  $\pi[z : y]$ , except for  $z$ , are white and hence the same is true at time  $t_x$ . Since  $z$  is black at time  $t_x$  and  $\text{flag}_w \neq \top$  for all nodes  $w$  in  $\pi[y : x]$ , the **AddLeaf** procedure adding  $x$  will color  $y$  black. Hence  $y$  is black at time  $t'_x$ . This is a contradiction.

We now consider subcase 2.3. Together with Lemma 1,  $\text{flag}_w = \top$  implies the existence of a black node  $b$  in  $\pi(z : w)$ . Since we assume all the nodes in  $\pi[y : x]$  to be white, the black node  $b$  is in  $\pi(z, y)$ , contradicting the hypothesis of scenario 2. ◀

Now, we are ready to prove our main property about the pattern of black vertices discussed at the beginning of this section. See Figure 4 (d).

► **Lemma 5.** *Let  $u$  and  $v$  be two nodes in  $T$  such that  $u$  is an ancestor of  $v$ , the distance between  $u$  and  $v$  is at least  $7\delta$ , and no node in the path  $\pi$  from  $u$  to  $v$  has been corrupted. Let  $\tilde{u}$  be the node at distance  $5\delta$  from  $u$  in  $\pi$  and let  $\tilde{v}$  be the node at distance  $\delta$  from  $v$  in  $\pi$ . Then there is a black node  $w^*$  in  $\pi[\tilde{u} : v]$  such that:*

- *The distance between  $w^*$  and  $\tilde{u}$  is at most  $\delta$ .*
- *A generic node in  $\pi[w^* : \tilde{v}]$  at distance  $d$  from  $w^*$  is black iff  $d$  is a multiple of  $\delta$ . Moreover, if  $w$  is a black vertex in  $\pi(w^*, \tilde{v})$  and  $\bar{w}$  is the associated black vertex in  $Q$ , the parent of  $\bar{w}$  in  $Q$  corresponds to the  $\delta$ -parent of  $w$  in  $\pi$ .*

**Proof.** Since no vertex in  $\pi$  has been corrupted, the path  $\pi$  must also belong to the noisy tree  $\mathcal{T}$ . Then, Lemma 2 ensures that, at any time following the insertion of  $\tilde{u}$  in  $T$ , there exists a black ancestor  $y$  of  $\tilde{u}$  such that  $d_\pi(y, \tilde{u}) \leq 3\delta$ . Such a vertex  $y$  is the  $\delta$ -parent of some vertex  $x$  in  $\pi$ . We denote by  $u'$  the  $2\delta$ -parent of  $y$  in  $\pi$  and by  $t'_x$  the time immediately after  $x$  is inserted. Since the length of  $\pi[u' : v]$  is at least  $3\delta$  and  $y$  must be black when  $v$  is inserted, we can invoke Lemma 4 to conclude that there exists a node in  $\pi[y : x]$  that is black at time  $t'_x$ . We choose  $w_0$  as the closest ancestor of  $x$  that is black at time  $t'_x$ . Moreover, for  $i = 1, \dots, \lfloor \pi[w_0 : v] / \delta \rfloor$  we let  $w_i$  be the unique vertex at distance  $\delta i$  from  $w_0$  in  $\pi[w_0, v]$ . Finally, let  $t'_i$  be the time immediately after the insertion of  $w_i$  into  $T$ .

We will prove by induction on  $i \geq 1$  that (i) at time  $t'_i$ , all vertices  $w_0, w_1, \dots, w_{i-1}$  are black; (ii) from time  $t'_i$  onward, all vertices in  $\pi[w_0, w_i]$  that do not belong to  $\{w_0, w_1, \dots, w_i\}$  are white.

We start by considering the base case  $i = 1$ . Regarding (i), we know that  $w_0$  is black at time  $t'_x$ , and hence  $w_0$  is also black at time  $t'_1$  (which cannot precede  $t'_x$ ). Regarding (ii), by our choice of  $w_0$  we know that at time  $t'_x$ , the only black vertex in  $\pi[w_0, x]$  is  $w_0$ . Moreover, Algorithm 2 can only color a node  $b$  black if none of the  $\delta - 1$  lowest proper ancestors of  $b$  is black. This implies that no vertex in  $\pi(w_0, w_1)$  will be colored black.

We now assume that the claim is true up to  $i \geq 1$  and prove it for  $i + 1$ . We first argue that the following property holds: (\*) at time  $t'_{i+1}$  all vertices in  $\pi(w_i : w_{i+1})$  are white. Indeed, suppose towards a contradiction that there exists some black vertex  $b$  in  $\pi(w_i : w_{i+1})$  at time  $t'_{i+1}$ . When  $b$  was colored black, either its  $\delta$ -parent  $b'$  was black or  $b$  was black-free. In the former case we immediately have a contradiction since  $b'$  must be a vertex of  $\pi(w_{i-1}, w_i)$  but all such vertices are white by the induction hypothesis. In the latter case  $b$  must have been colored black after the insertion of  $w_i$  but, by the induction hypothesis, we know that from time  $t'_i$  onwards  $w_{i-1}$  is black. This contradicts the hypothesis that  $b$  was black-free.

Next, we prove (i). Suppose towards a contradiction that  $w_i$  is white at time  $t'_{i+1}$ . Then, using (\*) and the induction hypothesis, we can invoke Lemma 3 on the subpath of  $\pi$  between  $w_{i-1}$  and the parent of  $w_{i+1}$  to conclude that all nodes  $w$  in  $\pi[w_i : w_{i+1})$  are such that  $\text{flag}_w \neq \top$ . Hence, during the insertion of  $w_{i+1}$ , Algorithm 2 reaches line 8 and checks whether  $w_i$  is near-a-black. Since this is indeed the case, a new black vertex is created in  $\pi[w_i : w_{i+1})$ , providing the sought contradiction. Let  $\bar{w}_i$  (resp.  $\bar{w}_{i-1}$ ) be the vertex in  $Q$  associated with  $w_i$  (resp.  $w_{i-1}$ ). Notice that this argument also shows that, at time  $t'_{i+1}$ ,  $\bar{w}_i$  is a child of  $\bar{w}_{i-1}$  in  $Q$  since  $w_i$  becomes black after time  $t'_i$  and not later than time  $t'_{i+1}$ , when  $w_{i-1}$  was already black.

To prove (ii) it suffices to notice that, by inductive hypothesis, we only need to argue about the nodes in  $\pi(w_i : w_{i+1})$ . From (\*) we know that these nodes are white at time  $t'_{i+1}$ , while (i) ensures that  $w_i$  is black at time  $t'_{i+1}$ . Then, a similar argument to the one used in the base case shows that Algorithm 2 will never color any node in  $\pi(w_i : w_{i+1})$  black (as long as the nodes in  $\pi$  remain uncorrupted). This concludes the proof by induction.

Let  $w'$  be the node at distance  $\delta$  from  $\tilde{u}$  in  $\pi[\tilde{u} : v]$ . Notice that  $w_0$  belongs to  $\pi[u : w']$ . If  $w_0$  lies in  $\pi(\tilde{u} : w')$ , we can choose  $w^* = w_0$ . Otherwise,  $w_0$  is an ancestor of  $\tilde{u}$  and, from (i) and (ii), there is exactly one black vertex  $b$  in  $\pi(\tilde{u} : w')$  and we choose  $w^* = b$ . ◀

■ **Algorithm 3** Answers a level ancestor query  $\text{LA}(v, k)$  in dynamic trees.

---

```

1 if  $k \leq 7\delta$  then
2   return  $\text{climb}(v, k)$ ;
3  $\bar{v} \leftarrow \text{climb}(v, \delta)$ ;
4 Climb up the tree  $\mathcal{T}$  from  $\bar{v}$  for up to  $\delta$  nodes searching a black node;
5 if the previous procedure did not find a black node then
6   return error;
7  $v' \leftarrow$  a black node found in the previous procedure;
8  $d \leftarrow$  distance between  $v'$  and  $v$ ;
9  $k' \leftarrow k - d - 5\delta$ ;
10  $u' \leftarrow \text{LA}_Q(q_{v'}, \lfloor k'/\delta \rfloor)$ ;
11  $k_{\text{rest}} \leftarrow k' - \lfloor k'/\delta \rfloor \cdot \delta + 5\delta$ ;
12 return  $\text{climb}(u', k_{\text{rest}})$ ;

```

---

The above lemma suggests a natural query algorithm. The query procedure is similar to the one for static case. When  $k \leq 7\delta$  we climb in  $\mathcal{T}$  the nodes of the path from  $v$  to the  $k$ -parent of  $v$  in a trivial way. Otherwise, Lemma 5 ensures that if no vertex in the path  $P$  from  $v$  to its level ancestor in  $T$  was corrupted by the adversary, then every other  $\delta$ -th vertex of  $P$  is colored black except, possibly, for an initial subpath of length  $\delta$  and for a trailing subpath of length  $5\delta$ . The query procedure explicitly “climbs” these portions of  $P$  and queries  $D_Q$  to quickly skip over its remaining “middle” part. The pseudo-code is given in Algorithm 3.

We are now ready to prove the main theorem of this section.

► **Theorem 6.** *Our data structure requires linear space, supports the **AddLeaf** operation in  $O(\delta)$  worst-case time, and can answer resilient LA queries in  $O(\delta)$  worst-case time.*

**Proof.** The correctness of the query immediately follows from Lemma 5. Moreover, the time required to perform an **AddLeaf** or an LA operation is  $O(\delta)$  since in both cases  $O(\delta)$  vertices of  $\mathcal{T}$  are visited and a single  $O(\delta)$ -time operation involving  $D_Q$  is performed.

We now discuss the size of our data structure. Clearly, the space used to store the vector  $\mathcal{A}$  of all records is  $O(n)$ . We only need to argue about the size of  $D_Q$ . Recall that  $D_Q$  is the resilient version, obtained using the replication strategy, of the data structure that requires linear space, takes constant time to answer each LA query and to perform each **AddLeaf** operation [2]. In order to show that  $D_Q$  requires  $O(n)$  space we will argue that the number of black vertices is  $O(\frac{n}{\delta})$ . As consequence we have that the size of  $D_Q$  is  $O(n)$ .

To bound the number of vertices in  $Q$ , notice that in order to add a new vertex to  $Q$  we need to spend  $\delta$  flags that were previously unspent. Moreover, a spent flag never becomes unspent unless the adversary corrupts the record of the corresponding node (by using one of its  $\delta$  corruptions). As a consequence the nodes in  $Q$  are at most  $(n + \delta)/\delta = O(n/\delta)$ . ◀

---

## References

- 1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. On finding lowest common ancestors in trees. *SIAM J. Comput.*, 5(1):115–132, 1976. doi:10.1137/0205011.
- 2 Stephen Alstrup and Jacob Holm. Improved algorithms for finding level ancestors in dynamic trees. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, ICALP '00, pages 73–84, Berlin, Heidelberg, 2000. Springer-Verlag.
- 3 Michael Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321:5–12, January 2004.



- 4 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839\_9.
- 5 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, April 1994. doi:10.1016/S0022-0000(05)80002-9.
- 6 Robert S. Boyer and J. Strother Moore. MJRTY: A fast majority vote algorithm. In Robert S. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Automated Reasoning Series, pages 105–118. Kluwer Academic Publishers, 1991.
- 7 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. Path minima queries in dynamic weighted trees. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 290–301, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- 8 Gerth Stølting Brodal, Rolf Fagerberg, Irene Finocchi, Fabrizio Grandoni, Giuseppe F. Italiano, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Optimal resilient dynamic dictionaries. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms – ESA 2007*, pages 347–358, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 9 Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. Competitive analysis of the top- $K$  ranking problem. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1245–1264. SIAM, 2017. doi:10.1137/1.9781611974782.81.
- 10 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms – Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2013. doi:10.1007/978-3-642-17327-1.
- 11 Richard Cole and Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM J. Comput.*, 34(4):894–923, 2005. doi:10.1137/S0097539700370539.
- 12 Erik D. Demaine, Gad M. Landau, and Oren Weimann. On cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. doi:10.1007/s00453-012-9683-x.
- 13 Dariusz Dereniowski, Aleksander Łukasiewicz, and Przemysław Uznański. An efficient noisy binary search in graphs via median approximation. In Paola Flocchini and Lucia Moura, editors, *Combinatorial Algorithms*, pages 265–281, Cham, 2021. Springer International Publishing.
- 14 Paul F. Dietz. Finding level-ancestors in dynamic trees. In Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, pages 32–40, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.
- 15 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM J. Comput.*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.
- 16 Irene Finocchi, Fabrizio Grandoni, and Giuseppe Italiano. Resilient dictionaries. *ACM Transactions on Algorithms*, 6, December 2009. doi:10.1145/1644015.1644016.
- 17 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Resilient search trees. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07*, pages 547–553, USA, 2007. Society for Industrial and Applied Mathematics.
- 18 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci.*, 410(44):4457–4470, 2009. doi:10.1016/j.tcs.2009.07.026.
- 19 Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC '04*, pages 101–110, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007375.
- 20 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal sorting with persistent comparison errors. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 49:1–49:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.49.

- 21 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, Paolo Penna, and Guido Proietti. Dual-mode greedy algorithms can save energy. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPICs*, pages 64:1–64:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ISAAC.2019.64.
- 22 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.
- 23 Giuseppe F. Italiano. Resilient algorithms and data structures. In Tiziana Calamoneri and Josep Díaz, editors, *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, volume 6078 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2010. doi:10.1007/978-3-642-13073-1\_3.
- 24 Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Priority queues resilient to memory faults. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Proceedings of the 10th International Workshop on Algorithms and Data Structures (WADS'07)*, volume 4619 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2007. doi:10.1007/978-3-540-73951-7\_12.
- 25 Stefano Leucci, Chih-Hung Liu, and Simon Meierhans. Resilient dictionaries for randomly unreliable memory. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPICs*, pages 70:1–70:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.70.
- 26 Andrzej Pelc. Searching games with errors – fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002. doi:10.1016/S0304-3975(01)00303-6.
- 27 Umberto Ferraro Petrillo, Fabrizio Grandoni, and Giuseppe F. Italiano. Data structures resilient to memory faults: An experimental study of dictionaries. *ACM J. Exp. Algorithmics*, 18, 2013. doi:10.1145/2444016.2444022.



# Computing Shapley Values for Mean Width in 3-D

Shuhao Tan   

Department of Computer Science, University of Maryland, College Park, MD, USA

---

## Abstract

The Shapley value is a classical concept from game theory, which is used to evaluate the importance of a player in a cooperative setting. Assuming that players are inserted in a uniformly random order, the Shapley value of a player  $p$  is the expected increase in the value of the characteristic function when  $p$  is inserted. Cabello and Chan (SoCG 2019) recently showed how to adapt this to a geometric context on planar point sets. For example, when the characteristic function is the area of the convex hull, the Shapley value of a point is the average amount by which the convex-hull area increases when this point is added to the set. Shapley values can be viewed as an indication of the relative importance/impact of a point on the function of interest.

In this paper, we present an efficient algorithm for computing Shapley values in 3-dimensional space, where the function of interest is the mean width of the point set. Our algorithm runs in  $O(n^3 \log^2 n)$  time and  $O(n)$  space. This result can be generalized to any point set in  $d$ -dimensional space ( $d \geq 3$ ) to compute the Shapley values for the mean volume of the convex hull projected onto a uniformly random  $(d-2)$ -subspace in  $O(n^d \log^2 n)$  time and  $O(n)$  space. These results are based on a new data structure for a dynamic variant of the convolution problem, which is of independent interest. Our data structure supports incremental modifications to  $n$ -element vectors (including cyclical rotation by one position). We show that  $n$  operations can be executed in  $O(n \log^2 n)$  time and  $O(n)$  space.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry; Theory of computation  $\rightarrow$  Data structures design and analysis

**Keywords and phrases** Shapley value, mean width, dynamic convolution

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.67

**Related Version** *Full Version*: <https://arxiv.org/abs/2002.05252>

**Acknowledgements** The author is very grateful to David Mount for discussions and advice for the paper. The author is also thankful to Geng Lin for proofreading and helping improving the readability.

## 1 Introduction

Given a point set  $P$  in  $d$ -dimensional space, many different functions can be applied to extract information about the set's geometric structure. Often, this involves properties of the convex hull of the set, such as its surface area and mean width. In the context of approximation, a natural question involves the “impact” that any given point of  $P$  has on the measure of interest. One method for modeling the notion of impact arises from the concept of the Shapley value (defined below) in the context of cooperative games in game theory. In this paper, we will present an efficient algorithm to compute the Shapley values for points in 3-D where the payoff of a point set is defined as the mean width of its convex hull.

Formally, a *coalition game* consists of a set of players  $N$  and a characteristic function  $v : 2^N \rightarrow \mathbb{R}$ , where  $v(\emptyset) = 0$ . In our setting, we take the players to be a point set  $N \subset \mathbb{R}^3$  and consider the characteristic function  $v(Q) = w(\text{conv}(Q))$  for  $Q \subset N$ , where  $\text{conv}(Q)$  denotes the convex hull of  $Q$ , and  $w(\text{conv}(Q))$  denotes the mean width of  $\text{conv}(Q)$ . (Formal definitions will be given in Section 2.)



© Shuhao Tan;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 67; pp. 67:1–67:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To define Shapley values, let  $\pi$  be a uniformly random permutation of  $N$ , and  $P_N(\pi, i)$  be the set of players in  $N$  that appear before player  $i$  in the permutation  $\pi$ . The *Shapley value* of player  $i \in N$  is defined to be the expected increase in  $v$  induced by the addition of player  $i$ , that is,

$$\phi(i) = \mathbb{E}_\pi[v(P_N(\pi, i) \cup \{i\}) - v(P_N(\pi, i))]. \quad (1)$$

Thus, the Shapley value represents the expected marginal contribution of  $i$  to the objective function over all permutations of  $N$ .

There are wide applications of Shapley values. A survey by Winter [27] and a book dedicated to this topic [24] provide insights on how the concept can be interpreted and applied in multiple ways, such as utility of players, allocation of resources of the grand coalition and measure of power in a voting system. Moreover, the values can be characterized axiomatically, making it the only natural quantity that satisfies certain properties. More details can be found in standard game theory textbooks ([14], [8, Chapter 5], [22, Section 9.4]).

In convex geometry and measure theory, *intrinsic volumes* are a key concept to characterize the “size” and “shape” of a convex body regardless of the translation and rotation in its underlying space. For example, Steiner’s formula [12] relates intrinsic volumes to the volume of the Minkowski-sum of a convex body and a ball. In general, one can define *valuation* to be a class of measure-like maps on open sets in a topological space. A formal definition of this concept can be found in [4, 16]. Functions such as volume and surface area fall into this class. Hadwiger’s Theorem [13, 17] asserts that every valuation that is continuous and invariant under rigid motion in  $\mathbb{R}^d$  is a linear combination of intrinsic volumes. In  $d$ -dimensional space, the  $d$ -th,  $(d - 1)$ -st and first intrinsic volumes are proportional to the usual Lebesgue measure, the surface area and the mean width, respectively [25, Chapter 4]. Cabello and Chan [7] presented efficient algorithms to compute Shapley values for area and perimeter for a point set in 2-D, which can be naturally extended to volume and surface area in 3-D. An algorithm that efficiently computes Shapley values for mean width in 3-D will then imply an algorithm that efficiently computes Shapley values for any continuous valuation in 3-D.

### Related work

The problem of computing Shapley values for area functions on a point set was introduced by Cabello and Chan [7]. They provided algorithms to compute Shapley values for area of convex hull, area of minimum enclosing disk, area of anchored rectangle, area of bounding box and area of anchored bounding box for a point set in 2-D. They also gave algorithms for the hull perimeter by slightly modifying these algorithms. All the quantities they considered are defined in 2-D space. Although their algorithms naturally extend to higher dimension, there are interesting applications of Shapley values in higher dimension that have yet been explored. The mean width considered in this paper is one such example.

Cabello and Chan also drew connections between computing Shapley values and stochastic computational geometry models. There have been many studies on the behavior of the convex hull under unipoint model where each point has an existential probability, for example see [1, 10, 15, 19, 26, 28]. In particular, Xue et al. [28] discussed the expected diameter and width of the convex hull. Huang et al. [15] presented a way to construct  $\epsilon$ -coreset for directional width under the model.

Mean width is often considered in the context of random polytopes in stochastic geometry. Müller [21] showed the asymptotic behavior of the mean width for a random polytope generated by sampling points on a convex body. Böröczyky [6] refined the result under the assumption that the convex body is smooth. Alonso-Gutiérrez and Prochno [3] considered

the case when the points are sampled inside an isotropic convex body. However, these results only consider the statistics when the number of points is large and the results are asymptotic. This paper views mean width on a computational perspective instead.

### Our contributions

We show that the Shapley values for mean width of the convex hull for a point set in  $\mathbb{R}^3$  can be computed in  $O(n^3 \log^2 n)$  time and  $O(n)$  space. Our approach is similar to the one used by Cabello and Chan for the case of the convex-hull area, in the sense that we look at the incremental formation of convex hull and consider the contribution of individual geometric objects. In our case, we break down the mean width and express that in terms of quantities related only to edges and then apply the linearity of Shapley values.

A major difference is that the expression for mean width contains angle at the edge, making the calculation for the probability term for Shapley values depend on the intersection of two halfspaces as opposed to only one halfspace. This prompts an expression that looks like a convolution. This convolution only changes slightly when evaluated from one point to another. The setup can be captured as an instance of dynamic convolution, where one can change the convolution kernel at any position, and query any single position in the vector involved in the convolution. Algebraic computations of this form were explored by Reif and Tate [23]. Frandsen et al. [11] gave a worst-case lower bound of  $\Omega(\sqrt{n})$  time per operation for this problem. By exploiting the structure of sweeping by polar angle, we obtain a variant of dynamic convolution, where we have a pointer to the convolution kernel and another pointer to the convolution result. We are only allowed to query at the pointer, update the convolution kernel at the pointer and move the pointers by one position. We present an online data structure for this variant that has  $O(n \log^2 n)$  overall time for  $n$  operations. There are some occurrences of algebraic computation in computational geometry, but many works [2, 5, 7, 18] do not have such dynamic setting and rely mostly on computing a single convolution or multi-point evaluation of polynomials. Only a few (see, e.g., [9]) employed a dynamic data structure. We believe our data structure is of independent interest for algorithms based on sweeping.

## 2 Preliminaries

**Mean width.** The mean width of a compact convex body  $X$  can be seen as the mean 1-volume of  $X$  projected on a uniformly random 1-subspace. More formally, let  $X \subset \mathbb{R}^d$  be a compact convex body. For  $1 \leq s \leq d$ , the *mean  $s$ -projection* of  $X$  is defined as:

$$M_s(X) = \int_{Q^s} |X_u| f_s(u) du, \quad (2)$$

where  $Q^s$  is the set of  $s$ -subspace,  $X_u$  is the projection of  $X$  on  $u$ ,  $|\cdot|$  is the volume or the canonical measure in the underlying space,  $f_s$  is the probability density function for uniformly sampling  $s$ -subspace from  $Q^s$ .

In this manner, the mean width of a point set  $P$  can be defined as  $M_1(\text{conv}(P))$  where  $\text{conv}(P)$  is the convex hull of  $P$ . It turns out that the mean  $s$ -projection of a convex polytope ( $1 \leq s \leq d - 2$ ) can be decomposed into values only related to its  $s$ -faces.

For a vertex of a polygon in 2-D, its interior angle can be naturally defined as the angle containing points within the polygon. And its exterior angle is the complement of the interior angle. This notion can be similarly generalized to higher dimensional faces in higher dimension. More formally, let  $X \subset \mathbb{R}^d$  be a convex polytope. Let  $V$  be a  $s$ -face of  $X$ . Let  $Q(V)$  be

## 67:4 Computing Shapley Values for Mean Width in 3-D

the set of halfspaces such that  $V$  is on the hyperplanes defining the halfspace. The *exterior angle* of  $V$  is defined as  $\psi(V) = \frac{|\{q \in Q(V) : X \subset q\}|}{|Q(V)|}$ . Let  $n(q)$  be the normal vector contained by the halfspace  $q$ . The *interior angle* of  $V$  is defined as  $\chi(V) = \frac{|\{q \in Q(V) : \exists p \in V, \exists \epsilon > 0, p + \epsilon n(q) \in X\}|}{|Q(V)|}$ . Moreover, when  $s = d - 2$ , and  $V \neq X$ , we have  $\chi(V) + \psi(V) = 1/2$  for all  $s$ -face  $V$  of  $X$ .

► **Lemma 1** (Miles [20]). *Let  $X \subset \mathbb{R}^d$  be a convex polytope, and  $1 \leq s \leq d - 2$ . Let  $V_s(X)$  be the set of  $s$ -face of  $X$ . We have:*

$$M_s(X) = C_{s,d} \sum_{V \in V_s(X)} |V| \psi(V). \quad (3)$$

where  $C_{s,d}$  is some constant related to  $s$  and  $d$ , in particular,  $C_{1,3} = 1/2$ .

**Random permutations.** When considering random permutations, it is common to compute probabilities where constraints on the order of appearance of disjoint sets are imposed. The following lemma follows from simple counting where a proof can be found in [7].

► **Lemma 2.** *Let  $N$  be a set with  $n$  elements. Let  $\{x\}$ ,  $A$  and  $B$  be disjoint sets. The probability that all of  $A$  appears before  $x$  and all of  $B$  appears after  $x$  in a uniformly random permutation  $\pi$  is  $\frac{|A|!|B|!}{(|A|+|B|+1)!}$ .*

**Assumptions.** We assume points in 3-D are of general positions where no three points are co-linear and no four points are co-planar. All points are assumed distinct. Throughout the paper, we often look at the projection of a 3-D point  $p$  on a plane, we will still use the same symbol  $p$  when looking on the plane for simplicity. We also assume a unit-cost real-RAM model of computation, where any arithmetic operation between two real numbers costs unit time, and a real number costs unit space to store.

### 3 Dynamic Convolution with Local Updates and Queries

Before presenting the algorithm to compute mean width, we first present a data structure that is central to the algorithm.

We consider a variant of dynamic convolution where only local updates and queries are permitted. More formally, let  $g : \mathbb{Z} \rightarrow \mathbb{R}$  be a fixed function where we can evaluate  $g(x)$  in constant time for any  $x$ . Let  $f : \mathbb{Z} \rightarrow \mathbb{R}$  initially be a zero function where we can change its values later. Let  $p, c \in \mathbb{Z}$  be two variables initially 0. We want to design a data structure that supports the following operations:

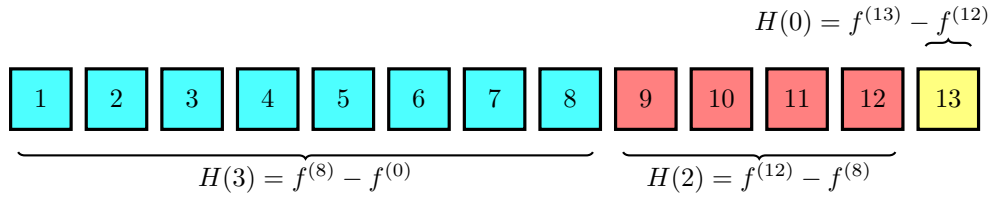
QUERY	UPDATE( $x$ )	INCP	DECP
Output $\sum_{i \in \mathbb{Z}} f(i+c)g(i)$	$f(p) \leftarrow f(p) + x$	$p \leftarrow p + 1$	$p \leftarrow p - 1$
ROTATELEFT	ROTATERIGHT		
$c \leftarrow c + 1$	$c \leftarrow c - 1$		

Let  $f^{(i)}$  be the  $f$  after the  $i$ -th operation, so  $f^{(0)} = 0$ . And similarly for  $p^{(i)}$ , and  $c^{(i)}$ . We can make the following observations:

► **Lemma 3.** *For any  $0 \leq i < j$ ,  $\max \text{supp}(f^{(j)} - f^{(i)}) - \min \text{supp}(f^{(j)} - f^{(i)}) \leq j - i$ , where  $\text{supp}(f)$  is the support of  $f$ .*

**Proof.** There are at most  $j - i$  INCP or DECP operations between the  $i$ -th and the  $j$ -th operation, so  $\max_{i \leq k \leq j} p^{(k)} - \min_{i \leq k \leq j} p^{(k)} \leq j - i$ . And we have  $\max \text{supp}(f^{(j)} - f^{(i)}) \leq \max_{i \leq k \leq j} p^{(k)}$  and  $\min \text{supp}(f^{(j)} - f^{(i)}) \geq \min_{i \leq k \leq j} p^{(k)}$ . ◀





■ **Figure 1** An example for  $n = 13 = 2^3 + 2^2 + 2^0$ . Each box represents an operation.

► **Corollary 4.** For any  $0 \leq i < j$ ,  $|\text{supp}(f^{(j)} - f^{(i)})| \leq j - i$ .

► **Lemma 5.** For any  $0 \leq i < j$ ,  $\max_{i \leq k \leq j} c^{(k)} - \min_{i \leq k \leq j} c^{(k)} \leq j - i$ .

**Proof.** There are at most  $j - i$  ROTATELEFT or ROTATERIGHT operations between the  $i$ -th and the  $j$ -th operation. ◀

► **Lemma 6.** For any  $0 \leq i < j$ ,  $\sum_k f^{(j)}(k + c)g(k) = \sum_k f^{(i)}(k + c)g(k) + \sum_k (f^{(j)} - f^{(i)})(k + c)g(k)$ .

From Lemma 6, we can see that it is possible to break the operations into chunks, and only consider the changes on  $f$  for each chunk. Lemma 3 ensures that the actual differences are proportional to the size of a chunk. So the idea is: after a chunk of size  $m$  is finished, we pre-compute all queries for the next  $m$  operations by considering all possible changes in  $c$ . We incrementally build different levels of chunks so that after finishing a chunk, we merge it with smaller chunks to build a larger chunk. Lemma 5 and Corollary 4 ensure that the prediction only depends on a small set of values of  $g$ . More specifically, we will use chunks sizes with power of two based on the binary form of the number of operations so far and merge chunks when needed. A binary-counter-like argument can be applied to achieve a amortized time of  $O(\log^2 n)$  per operation.

We maintain chunks of changes based on the binary representation of the number of operations we have seen so far. Assume we've just performed the  $k$ -th operation. Let  $b(k, i)$  be the  $i$ -th bit from right in the binary representation of  $k$ . We can write  $k = \sum_{i: b(k, i)=1} 2^i$  and we are going to use chunks of size  $2^i$  for  $b(k, i) = 1$ , where larger chunks are closer to the start of operations. Let  $i_1 < i_2 < \dots < i_q$  be the  $i$ 's such that  $b(k, i) = 1$ , and  $d(l) = \sum_{j=1}^l 2^{i_j}$ . Let  $s(j) = k - d(j)$  and  $e(j) = k - d(j - 1)$ .  $(s(j), e(j)]$  represents the range of the  $j$ -th chunk. In other words, we split the operations into chunks of size  $2^{i_q}, 2^{i_{q-1}}, \dots, 2^{i_1}$ . We maintain chunks of changes as  $H(i_j) = f^{(e(j))} - f^{(s(j))}$ . See Figure 1 for an example. Moreover, for each chunk, we maintain an array of predicted queries  $A(i_j)$  on the chunk  $H(i_j)$ , so that  $A(i_j, 2^{i_j} + c) = \sum_p H(i_j)(p + c^{(e(j))} + c)g(p)$  for  $-2^{i_j} \leq c \leq 2^{i_j}$ . We also maintain an array of  $I$  so that  $I(i_j) = c^{(k)} - c^{(e(j))}$ .

Assume we have  $A$ ,  $H$ , and  $I$  after  $n$  operations, the  $(n+1)$ -st operation can be performed easily: QUERY should output  $\sum_p f^{(n)}(p + c^{(n)})g(p)$ , and we have:

$$\begin{aligned} \sum_p f^{(n)}(p + c^{(n)})g(p) &= \sum_p \left( \sum_{i_j: b(n, i_j)=1} f^{(e(j))} - f^{(s(j))} \right) (p + c^{(n)})g(p) \\ &= \sum_{i_j: b(k, i_j)=1} \sum_p (f^{(e(j))} - f^{(s(j))})(p + c^{(e(j))} + I(i_j))g(p) \\ &= \sum_{i_j: b(k, i_j)=1} \sum_p H(i_j)(p + c^{(e(j))} + I(i_j))g(p) \\ &= \sum_{i_j: b(k, i_j)=1} A(i_j, 2^{i_j} + I(i_j)). \end{aligned}$$

So we can answer query by performing a summation over  $A$ . UPDATE is handled by simply documenting the change. INCP and DECP are handled by incrementing and decrementing  $p$ , respectively. ROTATELEFT and ROTATERIGHT are handled by incrementing and decrementing all entries of  $I$ , respectively.

After each operation, an additional maintaining step is performed to ensure  $A$ ,  $H$ ,  $I$  contain the correct information for the following operations. Assume that we want to maintain the data structure after the  $n$ -th operation, we record the change in  $f$  as  $\Delta f = f^{(n)} - f^{(n-1)}$ . Observe that  $A(i)$  doesn't change if  $b(n, i) = b(n-1, i)$ . Let  $i^* = \min\{i : b(n, i) = 1\}$ , then only  $A(i)$ 's such that  $i \leq i^*$  change. Moreover,  $A(i)$  becomes empty if  $i < i^*$ . Similar phenomenon can be seen in the increment of a binary counter. We expect:

$$\begin{aligned} A(i^*, 2^{i^*} + c) &= \sum_p H(i^*)(p + c^{(n)} + c)g(p) \\ &= \sum_p (f^{(n)} - f^{(n-2^{i^*})})(p + c^{(n)} + c)g(p) \\ &= \sum_p \left( \Delta f + \sum_{j=0}^{i^*-1} f^{(n-2^j)} - f^{(n-2^{j+1})} \right) (p + c^{(n)} + c)g(p) \\ &= \sum_p \left( \Delta f + \sum_{j=0}^{i^*-1} H(j) \right) (p + c^{(n)} + c)g(p). \end{aligned}$$

So we can merge  $\Delta f$  and  $\{H(j) : j < i^*\}$  to get  $H(j^*) = f^{(n)} - f^{(n-2^{i^*})}$ , and compute  $A(i^*, 2^{i^*} + c) = \sum_p H(i^*)(p + c^{(n)} + c)g(p)$  for all  $-2^{i^*} \leq c \leq 2^{i^*}$ . This is almost a convolution. To see it clearer, let  $L = \min \text{supp } H(i^*)$  and  $R = \max \text{supp } H(i^*)$ , we can rewrite the expression as  $A(i^*, 2^{i^*} + c) = \sum_{p=L}^R H(i^*)(p)g(p - c^{(n)} - c)$ . We now want to shift the origin so that  $\text{supp } H$  can start at 0 to conform to the definition of a convolution. Let  $H'(p) = H(i^*)(L + p)$  for  $0 \leq p \leq R - L$ , and  $H'(p) = 0$  otherwise. Let  $g'(p) = g(-p + L - c^{(n)} + 2^{i^*+1})$ , for  $0 \leq p \leq 3 \cdot 2^{i^*}$ , and  $g'(p) = 0$  otherwise. we have:

$$\begin{aligned} A(i^*, 2^{i^*} + c) &= \sum_{p=L}^R H(i^*)(p)g(p - c^{(n)} - c) \\ &= \sum_{p=0}^{R-L} H(i^*)(L + p)g(p - c + L - c^{(n)}) \\ &= \sum_{p=0}^{R-L} H'(p)g'(2^{i^*+1} + c - p) \\ &= H' * g'[2^{i^*+1} + c] \end{aligned}$$

Where  $H' * g'$  is the discrete convolution of  $H'$  and  $g'$ .

By Lemma 3,  $R - L \leq 2^{i^*}$ , so  $0 \leq 2^{i^*+1} + c - p \leq 3 \cdot 2^{i^*}$ . We can treat both  $H'$  and  $g'$  as circular vectors of size  $3 \cdot 2^{i^*}$ , and compute  $H' * g'[2^{i^*} + c]$  for all  $-2^{i^*} \leq c \leq 2^{i^*}$  by the Fast Fourier Transform. We then use the result to construct  $A(i^*)$ . We set  $I(i^*) = 0$ , and clear all  $A(i)$ ,  $H(i)$ , and  $I(i)$  where  $i < i^*$ .

Now that we have a way to maintain  $A$ ,  $H$ , and  $I$ , the following Lemma shows that the QUERY operation always succeed, i.e.,  $-2^i \leq I(i) \leq 2^i$ .

► **Lemma 7.** *After the  $n$ -th operation, for any  $i_j$  such that  $b(n, i_j) = 1$ ,  $|c^{(n)} - c^{(e(j))}| \leq 2^{i_j}$ .*

**Proof.**  $|c^{(n)} - c^{(e(j))}| \leq \max_{e(j) \leq k \leq n} c^{(k)} - \min_{e(j) \leq k \leq n} c^{(k)} \leq n - e(j) = d(j-1) \leq \sum_{k=0}^{i_j-1} 2^k = 2^{i_j}$ .  $\blacktriangleleft$

We now analyze the complexity of the data structure:

► **Theorem 8.** *There is a data structure that supports  $n$  operations for Dynamic Convolution with Local Updates and Queries in  $O(\log^2 n)$  amortized time and  $O(n)$  space.*

**Proof.** Without considering the additional maintaining step after each operation, it is easy to see a single INCP, DECP, or UPDATE takes  $O(1)$  time. Likewise a single ROTATELEFT, ROTATERIGHT, or QUERY takes  $O(\log n)$  time since the number of bits in the binary representation of  $n$  is  $O(\log n)$ .

In the maintaining step, whenever the  $i$ -th bit in the binary representation of  $n$  changes from 0 to 1, we need to merge changes of total size  $O(2^i)$ , run FFT on two arrays of size  $3 \cdot 2^i$ , and clear histories of total size  $O(2^i)$ . These in all take  $O(2^i \log(2^i))$  time where the bottleneck is FFT. Like the analysis in binary counter,  $i$ -th bit flips from 0 to 1 in every  $2^i$  operations, so the total time spent on maintaining step is:  $\sum_{i=0}^{\log n} \frac{n}{2^i} O(2^i \log(2^i)) = O\left(n \sum_{i=0}^{\log n} i \log(2)\right) = O(n \log^2 n)$ .

Adding two parts, the total time for  $n$  operation is  $O(n \log^2 n)$ , yielding an amortized time of  $O(\log^2 n)$  per operation.

At any point, the space used by the data structure is linear to the size of  $A, H, I$ , even during the maintaining step. So the space complexity is  $O\left(\sum_{i=0}^{\log n} 2^i\right) = O(n)$ .  $\blacktriangleleft$

► **Corollary 9.** *Let  $g : \mathbb{Z} \rightarrow \mathbb{R}$  be a fixed function where we can evaluate  $g(x)$  in constant time. Let  $Q$  be a queue with elements in  $\mathbb{R}$ . There is a data structure that supports  $n$  operations in  $O(\log^2 n)$  amortized time and  $O(n)$  space, where each operation is either pushing to the tail of  $Q$ , popping from the head of  $Q$  or querying  $\sum_{i=1}^{|Q|} Q(i)g(|Q| - i)$ .*

**Proof.** We use two instances of data structure  $I_1 = (f_1, g_1, p_1, c_1)$  and  $I_2 = (f_2, g_2, p_2, c_2)$  from Theorem 8. We make  $g_1(x) = g_2(x) = g(-x)$ . Whenever we want to perform a push  $x$ , we perform UPDATE( $x$ ), ROTATELEFT and INCP on  $I_1$ , and perform ROTATELEFT on  $I_2$ . Whenever we want to perform a pop  $x$ , we perform UPDATE( $-x$ ) and INCP on  $I_2$ . Whenever we want to query, we query both  $I_1$  and  $I_2$  and return the sum. Each operation to the queue expands to constant number of operations on  $I_1$  and  $I_2$ , so the running time is still  $O(\log^2 n)$  amortized and the space is  $O(n)$ .  $\blacktriangleleft$

## 4 Computing Shapley Value of Mean Width in 3-D

### 4.1 Classification of Cases

Let  $X$  be a convex polyhedron in 3-D, Lemma 1 becomes:

$$M_1(X) = \frac{1}{2} \sum_{e \in E(X)} l(e)\psi(e), \quad (4)$$

where  $E(X)$  is the set of edges of  $X$  and  $l(e)$  is the length of edge  $e$ .

Let  $N \subset \mathbb{R}^3$  be a point set. Let  $n$  be the number of points in the point set. The mean width we are considering is  $M_1(\text{conv}(N))$ . Given a permutation  $\pi$ , and a point  $p \in N$ , for convenience we define  $C(\pi, p) = \text{conv}(P_N(\pi, p) \cup \{p\})$  and  $C'(\pi, p) = \text{conv}(P_N(\pi, p))$ . We then define the contribution of  $p$  under permutation  $\pi$  to be:

$$\Delta(\pi, p) = M_1(C(\pi, p)) - M_1(C'(\pi, p)). \quad (5)$$

## 67:8 Computing Shapley Values for Mean Width in 3-D

When there are at least three points in  $N$ , we have  $\psi(e) = 1/2 - \chi(e)$  for all  $e \in E(N)$ . In the case where  $N$  is an edge  $e$ , we have  $\psi(e) = 1$ . It is also clear that a single point has width 0. So for  $p \in N$ , when considering the Shapley value  $\phi(p)$ , we can classify the permutations into three cases:

- CASE 1: There is one point before  $p$  in the permutation. i.e., after inserting  $p$ , the point set forms a line segment.
- CASE 2: There are two points before  $p$  in the permutation. i.e., after inserting  $p$ , the point set forms a triangle.
- CASE 3: There are three or more points before  $p$  in the permutation.

In other words, we can write  $\phi(p)$  as:

$$\phi(p) = \sum_{\pi} \frac{\Delta(\pi, p)}{n!} = \sum_{\pi: \text{Case 1}} \frac{\Delta(\pi, p)}{n!} + \sum_{\pi: \text{Case 2}} \frac{\Delta(\pi, p)}{n!} + \sum_{\pi: \text{Case 3}} \frac{\Delta(\pi, p)}{n!}. \quad (6)$$

For Case 1, we can rewrite the summation as:

$$\begin{aligned} \sum_{\pi: \text{Case 1}} \frac{\Delta(\pi, p)}{n!} &= \sum_{\substack{q \in P \\ q \neq p}} \sum_{\pi: \pi = (q, p, \dots)} \frac{\Delta(\pi, p)}{n!} = \sum_{\substack{q \in P \\ q \neq p}} \sum_{\pi: \pi = (q, p, \dots)} \frac{d(p, q)/2}{n!} \\ &= \sum_{\substack{q \in P \\ q \neq p}} \frac{d(p, q)}{2} \Pr(\pi = (q, p, \dots)) = \sum_{\substack{q \in P \\ q \neq p}} \frac{d(p, q)}{2} \frac{1}{n(n-1)}. \end{aligned} \quad (7)$$

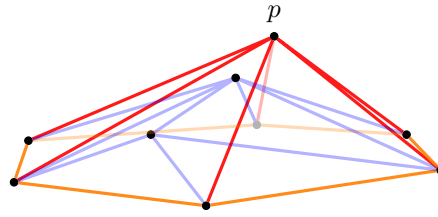
Here  $d(\cdot, \cdot)$  is the Euclidean distance function. So for any  $p$ , we can compute the summation in  $O(n)$  time by enumerating  $q$ . And it takes  $O(n^2)$  in total to compute for every  $p$ .

For Case 2, we can rewrite the summation as:

$$\begin{aligned} \sum_{\pi: \text{Case 2}} \frac{\Delta(\pi, p)}{n!} &= \sum_{\substack{q, r \in P \\ p, q, r \text{ are distinct}}} \sum_{\pi: \pi = (q, r, p, \dots)} \frac{\Delta(\pi, p)}{n!} \\ &= \sum_{\substack{q, r \in P \\ p, q, r \text{ are distinct}}} \sum_{\pi: \pi = (q, r, p, \dots)} \frac{\frac{d(p, q) + d(r, q) + d(p, r)}{4} - \frac{d(r, q)}{2}}{n!} \\ &= \sum_{\substack{q, r \in P \\ p, q, r \text{ are distinct}}} \frac{d(p, q) + d(p, r) - d(r, q)}{4} \Pr(\pi = (q, r, p, \dots)) \\ &= \sum_{\substack{q, r \in P \\ p, q, r \text{ are distinct}}} \frac{d(p, q) + d(p, r) - d(r, q)}{4} \frac{1}{n(n-1)(n-2)}. \end{aligned} \quad (8)$$

Like Case 1, we can compute the summation in  $O(n^2)$  time by enumerating  $q$  and  $r$ . And it takes  $O(n^3)$  in total to compute for every  $p$ .

In Case 3, each edge we are considering has two faces attached to it. In other words, each edge can be characterized by the common edge shared by two triangles formed by four points. We can denote an edge by  $(q, r, t_1, t_2)$ , where  $(q, r)$  is the edge and is the common edge of  $\triangle qrt_1$  and  $\triangle qrt_2$ . Without loss of generality, we assume unordered tuples when writing  $(q, r)$  and  $(t_1, t_2)$  to avoid double-counting. The exterior angle is completely determined by the quadruple. In case the convex hull has only 3 points, we allow  $t_1 = t_2$ . We can then apply Lemma 1 and express  $M_1(\text{conv}(N'))$  for a point set  $N'$  as:



■ **Figure 2** Change of edges when  $p$  is inserted, showing only the part visible to  $p$ . Red edges: edges added to the convex hull. Blue edges: edges removed from the convex hull. Orange edges: edges with angle changed.

$$\begin{aligned}
 M_1(\text{conv}(N')) &= \frac{1}{2} \sum_{e \in E(\text{conv}(N'))} l(e)\psi(e) = \frac{1}{2} \sum_{\substack{e=(q,r,t_1,t_2) \\ q,r,t_1,t_2 \in N' \\ \Delta qrt_1, \Delta qrt_2 \text{ are faces of } \text{conv}(N')}} l(e)\psi(e) \\
 &= \frac{1}{2} \sum_{\substack{e=(q,r,t_1,t_2) \\ q,r,t_1,t_2 \in N'}} d(q,r)\psi(e) I_{\text{conv}(N')}(q,r,t_1,t_2)
 \end{aligned}$$

where we use an indicator variable:

$$I_X(q,r,t_1,t_2) = \begin{cases} 1 & \Delta qrt_1, \Delta qrt_2 \text{ are faces of convex polyhedron } X \\ 0 & \text{otherwise} \end{cases}$$

Conditioning on  $\pi$  being Case 3, we can write  $\Delta(\pi, p)$  as:

$$\Delta(\pi, p) = \frac{1}{2} \sum_{\substack{e=(q,r,t_1,t_2) \\ q,r,t_1,t_2 \in N}} l(e)\psi(e) (I_{C(\pi,p)}(q,r,t_1,t_2) - I_{C'(\pi,p)}(q,r,t_1,t_2))$$

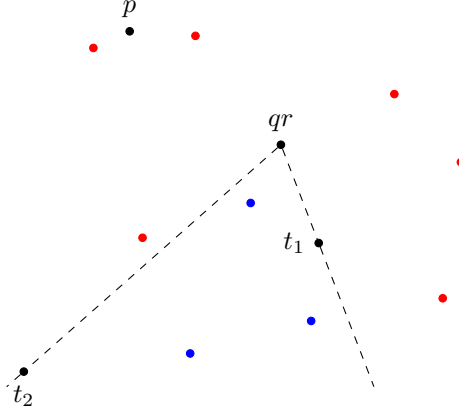
We can write in the form of expectation as:

$$\begin{aligned}
 \sum_{\pi: \text{Case 3}} \frac{\Delta(\pi, p)}{n!} &= \mathbb{E}_\pi[\Delta(\pi, p) | \pi : \text{Case 3}] \Pr(\pi : \text{Case 3}) \\
 &= \frac{1}{2} \sum_{\substack{e=(q,r,t_1,t_2) \\ q,r,t_1,t_2 \in N}} l(e)\psi(e) \mathbb{E}_\pi [I_{C(\pi,p)}(q,r,t_1,t_2) - I_{C'(\pi,p)}(q,r,t_1,t_2) | \pi : \text{Case 3}] \Pr(\pi : \text{Case 3})
 \end{aligned}$$

Observe that  $I_{C(\pi,p)}(q,r,t_1,t_2) = 1$  implies  $\pi$  being Case 3. The summation reduces to:

$$\frac{1}{2} \sum_{q,r \in N} \sum_{\substack{t_1, t_2 \in N \\ e=(q,r,t_1,t_2)}} l(e)\psi(e) \mathbb{E}_\pi [I_{C(\pi,p)}(q,r,t_1,t_2) - I_{C'(\pi,p)}(q,r,t_1,t_2)] \tag{9}$$

Now consider the impact when inserting  $p$  to  $P_N(\pi, p)$ . The convex hull does not change if  $p$  is inside  $C'(\pi, p)$ . Otherwise if we treat  $C'(\pi, p)$  as an opaque object, all the faces visible to  $p$  will be removed in  $C(\pi, p)$  and a pyramid-like cone with apex at  $p$  will be added to  $C(\pi, p)$ . In terms of edges, there are three types of edges: edges removed, edges added and edges with angle changed. See Figure 2 for an illustration.



■ **Figure 3** Projection of  $N$  along  $qr$ . Edge  $qr$  gets removed when forming  $\text{conv}(P_N(\pi, p) \cup \{p\})$  if and only if (a)  $p$  is not in the cone formed by  $qr$ ,  $t_1$  and  $t_2$ , (b)  $q$ ,  $r$ ,  $t_1$  and  $t_2$  appear before  $p$  in  $\pi$  and (c) No point outside the cone (red point) appears before  $p$  in  $\pi$ .

As we are using a 4-tuple to represent an edge in the summation, edges with angles changed can be seen as removal and addition with different  $(t_1, t_2)$ . We have:

$$\begin{aligned}
 & I_{C(\pi, p)}(q, r, t_1, t_2) - I_{C'(\pi, p)}(q, r, t_1, t_2) \\
 = & \begin{cases} -1 & (q, r, t_1, t_2) \text{ is an edge of } C'(\pi, p) \text{ and visible to } p \\ 1 & (q, r, t_1, t_2) \text{ is an edge of } C(\pi, p) \text{ and } p \in \{t_1, t_2\} \\ 1 & (q, r, t_1, t_2) \text{ is an edge of } C(\pi, p) \text{ and } p \in \{q, r\} \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

The first three cases are correspondent to blue+orange, orange and red edges in Figure 2 respectively. So we can write:

$$\begin{aligned}
 & \mathbb{E}_\pi [I_{C(\pi, p)}(q, r, t_1, t_2) - I_{C'(\pi, p)}(q, r, t_1, t_2)] \\
 = & \Pr((q, r, t_1, t_2) \text{ is an edge of } C(\pi, p) \text{ and } p \in \{t_1, t_2\}) \\
 & + \Pr((q, r, t_1, t_2) \text{ is an edge of } C(\pi, p) \text{ and } p \in \{q, r\}) \\
 & - \Pr((q, r, t_1, t_2) \text{ is an edge of } C'(\pi, p) \text{ and visible to } p)
 \end{aligned} \tag{10}$$

This gives us a way to split the final summation in Equation 9 further into 3 summations. We will show how to compute them efficiently in the following subsection.

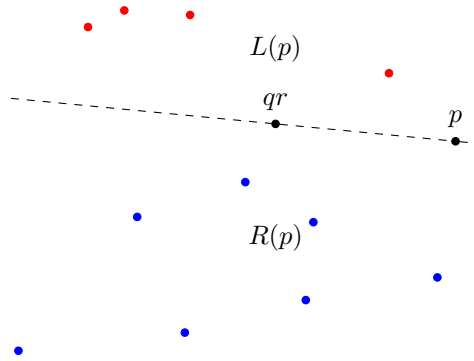
## 4.2 Handling Case 3

The idea is to enumerate edges  $(q, r)$  and compute  $\sum_{\substack{t_1, t_2 \in N \\ e=(q, r, t_1, t_2)}} l(e)\psi(e) \Pr(\cdot)$  for all  $p$  where  $\Pr(\cdot)$  is one of the three probabilities in Equation 10.

**$(q, r, t_1, t_2)$  is an edge of  $C'(\pi, p)$  and visible to  $p$**

Given a pair of points  $(q, r)$ , we look at the projection of  $N$  along the direction of  $qr$ .

- **Lemma 10.**  *$(q, r, t_1, t_2)$  is an edge of  $C'(\pi, p)$  and visible to  $p$  if and only if*
- (a)  $p$  is not in the cone formed by  $qr$ ,  $t_1$  and  $t_2$ .
  - (b)  $q$ ,  $r$ ,  $t_1$  and  $t_2$  appear before  $p$  in  $\pi$ .
  - (c) No point outside the cone appears before  $p$  in  $\pi$ .



■ **Figure 4** Partition of points based on whether a point is left to  $p - qr$  or right to  $p - qr$ . Red points are left, and blue points are right.

**Proof.** (b) is immediate as we need  $q, r, t_1$  and  $t_2$  to be in  $P_N(\pi, p)$  so that the edge can be in  $C'(\pi, p)$ . Given (b),  $(q, r, t_1, t_2)$  is an edge of  $C'(\pi, p)$  if and only if  $\Delta qrt_1$  and  $\Delta qrt_2$  form two supporting planes of  $P_N(\pi, p)$  if and only if no point outside the cone appears before  $p$  in  $\pi$ . Finally,  $(q, r, t_1, t_2)$  is visible to  $p$  if and only if  $pq$  and  $pr$  are completely outside  $C'(\pi, p)$  if and only if  $p$  is not in the cone formed by  $qr, t_1$  and  $t_2$ . See Figure 3 for an example. ◀

Treat  $qr$  as the origin and let  $p_1, p_2, \dots, p_{n-2}$  be the rest of the points in  $N$  sorted by polar angles relative to  $qr$ . In other words,  $p_1, p_2, \dots, p_{n-2}$  is the order of points when we sweep a ray starting from  $qr$  around counterclockwise, initially to the direction of positive  $x$ -axis. Let  $\theta_i$  be the polar angle of  $p_i$ . For convenience, we treat the sequence  $p_1, p_2, \dots, p_{n-2}$  as a cyclic array, in the sense that  $p_{n-1} = p_1$ . Moreover, when we iterate through the sequence,  $\theta_i$  is non-decreasing. In other words, when we iterate  $p_1, p_2, \dots, p_{n-1}, p_n, \dots$ , although  $p_{n-1}$  and  $p_1$  are the same point, we treat  $\theta_{n-1} = \theta_1 + 2\pi$ .

Let  $W_{qr}(p_i, p_j)$  be the number of points in the cone formed by  $qr, p_i$  and  $p_j$ . Let  $g(i) = 4!(n - 5 - i)! / (n - i)!$ . For a point  $p$  outside the cone, Lemma 2 gives us:

$$\Pr((q, r, p_i, p_j) \text{ is an edge of } C'(\pi, p) \text{ and visible to } p) = g(W_{qr}(p_i, p_j)) \tag{11}$$

Let  $S(i)$  be the set of pairs  $(p_j, p_k)$  such that  $p_i$  is not in the cone formed by  $qr, p_j$  and  $p_k$ . We assume  $j \leq k$  and the cone is formed by sweeping from  $p_j$  to  $p_k$  counterclockwise. For a given point  $p_i$ , Lemma 10 and Equation 11 gives:

$$\begin{aligned} & \sum_{\substack{t_1, t_2 \in N \\ e=(q, r, t_1, t_2)}} l(e)\psi(e) \Pr((q, r, t_1, t_2) \text{ is an edge of } C'(\pi, p) \text{ and visible to } p_i) \\ &= \sum_{(p_j, p_k) \in S(i)} d(q, r) \left( \frac{1}{2} - \frac{\theta_k - \theta_j}{2\pi} \right) g(W_{qr}(p_j, p_k)) \end{aligned}$$

Splitting the summation we get:

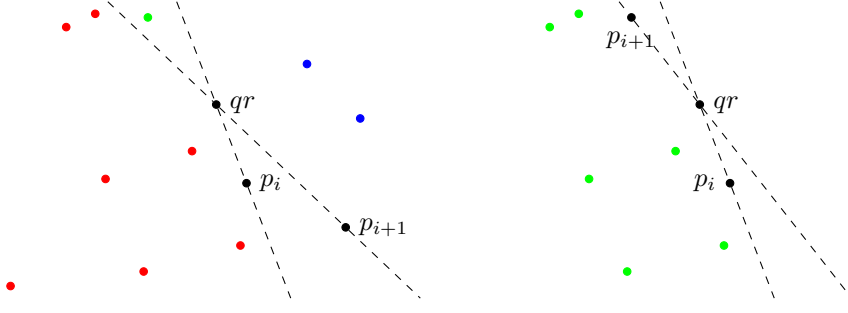
$$d(q, r) \left( \sum_{(p_j, p_k) \in S(i)} \frac{1}{2} g(W_{qr}(p_j, p_k)) - \sum_{(p_j, p_k) \in S(i)} \frac{\theta_k - \theta_j}{2\pi} g(W_{qr}(p_j, p_k)) \right) \tag{12}$$

We now show how to compute  $\sum_{(p_j, p_k) \in S(i)} \frac{\theta_k - \theta_j}{2\pi} g(W_{qr}(p_j, p_k))$  for all  $i$ . For each point  $p$ , we partition the point set by whether a point is left to  $p - qr$  or right to  $p - qr$ . As shown in Figure 4, we define:

$$L(p) = \{p' \in N : (p - qr) \times (p' - qr) > 0\} \quad R(p) = \{p' \in N : (p - qr) \times (p' - qr) < 0\} \tag{13}$$



## 67:12 Computing Shapley Values for Mean Width in 3-D



■ **Figure 5** When changing from  $S(i)$  to  $S(i+1)$ , pairs formed between  $p_{i+1}$  and  $L(p_{i+1}) \cup \{p_{i+1}\}$  are removed and pairs formed between  $p_i$  and  $R(p_i) \cup \{p_i\}$  are added. Left:  $p_{i+1} \in L(p_i)$ . Right:  $p_{i+1} \in R(p_i)$ .

Consider the difference between  $S(i)$  and  $S(i+1)$ . It is easy to see:

$$\begin{aligned} \sum_{(p_j, p_k) \in S(i+1)} \frac{\theta_k - \theta_j}{2\pi} g(W_{qr}(p_j, p_k)) &= \sum_{(p_j, p_k) \in S(i)} \frac{\theta_k - \theta_j}{2\pi} g(W_{qr}(p_j, p_k)) \\ &+ \sum_{p_j \in R(p_i) \cup \{p_i\}} \frac{\theta_i - \theta_j}{2\pi} g(W_{qr}(p_i, p_j)) \\ &- \sum_{p_j \in L(p_{i+1}) \cup \{p_{i+1}\}} \frac{\theta_j - \theta_{i+1}}{2\pi} g(W_{qr}(p_j, p_{i+1})) \end{aligned} \quad (14)$$

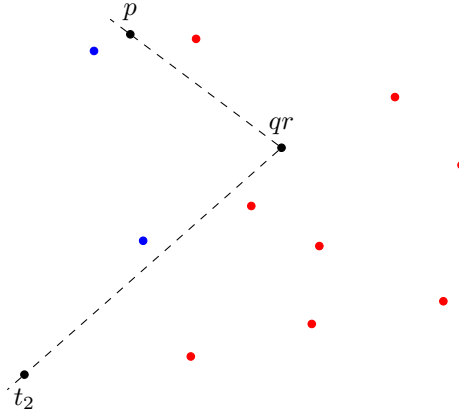
as demonstrated in Figure 5. We can further write:

$$\begin{aligned} &\sum_{p_j \in R(p_i) \cup \{p_i\}} \frac{\theta_i - \theta_j}{2\pi} g(W_{qr}(p_i, p_j)) \\ &= \theta_i \sum_{p_j \in R(p_i) \cup \{p_i\}} \frac{1}{2\pi} g(W_{qr}(p_i, p_j)) - \sum_{p_j \in R(p_i) \cup \{p_i\}} \frac{\theta_j}{2\pi} g(W_{qr}(p_i, p_j)) \end{aligned} \quad (15)$$

We will show how to compute  $\sum_{p_j \in R(p_i) \cup \{p_i\}} f(j)g(W_{qr}(p_i, p_j))$  for an arbitrary function  $f$  and for all  $i$  at the same time. Equation 15 can then be computed by using  $f(j) = 1/2\pi$  and  $f(j) = \theta_j/2\pi$ , respectively.

If we sort the points in  $R(p_i) \cup \{p_i\}$  by polar angles as  $q_1, q_2, \dots, q_l$ , where  $l = |R(p_i) \cup \{p_i\}|$ , it is easy to see  $W_{qr}(p_i, q_j) = l - j$  for  $1 \leq j \leq l$ . In other words, the number of points within the cone formed by  $qr$ ,  $p_i$ , and  $q_j$  is equal to the distance between  $q_j$  and  $p_i$  in the sequence  $p_1, p_2, \dots$ .

Now we consider an instance of the data structure from Corollary 9. We use  $g(i) = 4!(n-5-i)/(n-i)!$  as the function used by the data structure. We start by choosing an arbitrary point  $p_i$  and consider a ray opposite to  $p_i - qr$ . We sweep this ray counterclockwise until hitting  $p_i$ , for each point  $p_j$  hit by the ray, we perform a push  $f(j)$  to the data structure. After we hit  $p_i$ , we perform a query and the result is exactly  $\sum_{p_j \in R(p_i) \cup \{p_i\}} f(j)g(W_{qr}(p_i, p_j))$ . Next we sweep the ray to  $p_{i+1}$  and pop all the points that are left to  $p_{i+1}$ . They should all appear at the head of the data structure. We then perform a query and the result is exactly  $\sum_{p_j \in R(p_{i+1}) \cup \{p_{i+1}\}} f(j)g(W_{qr}(p_i, p_j))$ . We keep sweeping, popping and querying until coming back to  $p_{i+n-3}$  which is  $p_{i-1}$ . During this process, each point gets pushed and popped at most twice and we perform  $n-2$  queries. So the running time is  $O(n \log^2 n)$  and the space complexity is  $O(n)$  according to Corollary 9.



■ **Figure 6** Projection of  $N$  along  $qr$ . Edge  $qr$  gets added when forming  $\text{conv}(P_N(\pi, p) \cup \{p\})$  if and only if (a)  $q, r$  and  $t_2$  appear before  $p$  in  $\pi$  (b) No point outside the cone formed by  $qr, p$  and  $t_2$  (red point) appears before  $p$  in  $\pi$ .

Hence we can compute  $\sum_{p_j \in R(p_i) \cup \{p_i\}} \frac{\theta_i - \theta_j}{2\pi} g(W_{qr}(p_i, p_j))$  for all  $i$  in  $O(n \log^2 n)$  time and  $O(n)$  space. With the same idea but sweeping the other way around, we can also compute  $\sum_{p_j \in L(p_{i+1}) \cup \{p_{i+1}\}} \frac{\theta_j - \theta_{i+1}}{2\pi} g(W_{qr}(p_j, p_{i+1}))$  for all  $i$  in the same time and space complexity.

Next we only need to compute  $\sum_{(p_j, p_k) \in S(i)} \frac{\theta_k - \theta_j}{2\pi} g(W_{qr}(p_j, p_k))$  for a single  $i$  and then used the pre-computed result to update to next  $\sum_{(p_j, p_k) \in S(i+1)} \frac{\theta_k - \theta_j}{2\pi} g(W_{qr}(p_j, p_k))$  in constant time. Like in the previous case, we need to compute summations of the form  $\sum_{(p_j, p_k) \in S(i)} f(j)g(W_{qr}(p_j, p_k))$ . We again use an instance of data structure from Corollary 9 and use the same  $g$  as above. We start by pushing  $p_{i+1}$  and sweep counterclockwise. For each point hit, we first pop all the points that are left to the point, push the point to the data structure and finally perform a query. We stop after hitting  $p_{i+n-3}$  which is  $p_{i-1}$ . The sum of all the queries will then be  $\sum_{(p_j, p_k) \in S(i)} f(j)g(W_{qr}(p_j, p_k))$ . In this procedure, each point is pushed and popped at most once, and  $n - 3$  queries are made. So it takes  $O(n \log^2 n)$  time and  $O(n)$  space to compute for a single  $i$ . After that it takes  $O(n)$  time to compute for all  $i$  by transitioning from  $i$  to  $i + 1$  in constant time.

Using the same idea, we can compute the other part of Equation 12 in  $O(n \log^2 n)$  time and  $O(n)$  space as well.

For a fixed pair  $(q, r)$ , it takes  $O(n \log n)$  to sort other points by polar angles on the projected plane. And it takes  $O(n)$  to pre-compute  $g$ . Hence, it takes  $O(n \log^2 n)$  time and  $O(n)$  space to compute  $\sum_{e=(q,r,t_1,t_2)} \sum_{t_1, t_2 \in N} l(e)\psi(e) \Pr(\cdot)$  for all  $p$  for the case where  $(q, r, t_1, t_2)$  is edge of  $C'(\pi, p)$  and visible to  $p$ . And it takes  $O(n^3 \log^2 n)$  time and  $O(n)$  space overall by enumerating all pairs of  $(q, r)$ .

### $(q, r, t_1, t_2)$ is an edge of $C(\pi, p)$ and $p \in \{t_1, t_2\}$

Again we fix a pair  $(q, r)$ , and look at the projection of  $N$  along the direction of  $qr$ . Without loss of generality, assume  $p = t_1$  in this case.

► **Lemma 11.**  $(q, r, p, t_2)$  is edge of  $C(\pi, p)$  if and only if

- (a)  $q, r$  and  $t_2$  appear before  $p$  in  $\pi$ .
- (b) No point outside the cone formed by  $qr, p$  and  $t_2$  appears before  $p$  in  $\pi$ .

## 67:14 Computing Shapley Values for Mean Width in 3-D

The proof will be almost the same as the proof for Lemma 10. See Figure 6 for an illustration.

In this case

$$\Pr((q, r, p, t_2) \text{ is an edge of } C(\pi, p)) = \frac{3!(n-4-W_{qr}(p, t_2))!}{(n-W_{qr}(p, t_2))!} \quad (16)$$

We have a slightly different  $g(i) = 3!(n-4-i)/(n-i)!$ . Any point except  $p$  can form a cone with  $p$ . So we can write

$$\begin{aligned} & \sum_{\substack{t_2 \in N \\ e=(q,r,p_i,t_2)}} l(e)\psi(e) \Pr((q, r, p, t_2) \text{ is an edge of } C(\pi, p)) \\ &= \sum_{p_j \in R(p_i)} d(q, r) \left( \frac{1}{2} - \frac{\theta_i - \theta_j}{2\pi} \right) g(W_{qr}(p_i, p_j)) \\ &+ \sum_{p_j \in L(p_i)} d(q, r) \left( \frac{1}{2} - \frac{\theta_j - \theta_i}{2\pi} \right) g(W_{qr}(p_j, p_i)) \end{aligned} \quad (17)$$

In Subsection 4.2, we've shown how to compute summations with very similar form as summations in Equation 17. The only differences are that we take  $p_j \in R(p_i)$  instead of  $p_j \in R(p_i) \cup \{p_i\}$  and we have a slightly different  $g$  here. But clearly these don't change the asymptotic running time. Hence it takes  $O(n^3 \log^2 n)$  time in total and  $O(n)$  space by using the same method.

### **$(q, r, t_1, t_2)$ is an edge of $C(\pi, p)$ and $p \in \{q, r\}$**

Without loss of generality, assume  $p = q$ . We look at the projection of  $N$  along the direction of  $pr$ .

► **Lemma 12.**  *$(p, r, t_1, t_2)$  is edge of  $C(\pi, p)$  if and only if*

- (a)  $r, t_1$  and  $t_2$  appear before  $p$  in  $\pi$ .
- (b) No point outside the cone formed by  $pr, t_1$  and  $t_2$  appears before  $p$  in  $\pi$ .

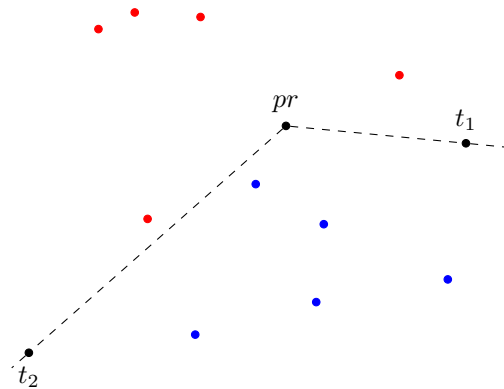
The proof will be almost the same as the proof for Lemma 10. See Figure 7 for an illustration.

In this case

$$\Pr((p, r, t_1, t_2) \text{ is an edge of } C(\pi, p)) = \frac{3!(n-4-W_{qr}(t_1, t_2))!}{(n-W_{qr}(t_1, t_2))!} \quad (18)$$

And we use  $g(i) = 3!(n-4-i)/(n-i)!$ . In this case, any pair  $(t_1, t_2)$  with  $t_1 \neq t_2$  will contribute to result. So we can write

$$\begin{aligned} & \sum_{\substack{t_1, t_2 \in N \\ e=(p,r,t_1,t_2)}} l(e)\psi(e) \Pr((p, r, t_1, t_2) \text{ is an edge of } C(\pi, p)) \\ &= \frac{1}{2} \sum_{p_k} \sum_{p_j \in R(p_k)} d(p, r) \left( \frac{1}{2} - \frac{\theta_k - \theta_j}{2\pi} \right) g(W_{qr}(p_k, p_j)) \\ &+ \frac{1}{2} \sum_{p_k} \sum_{p_j \in L(p_k)} d(p, r) \left( \frac{1}{2} - \frac{\theta_j - \theta_k}{2\pi} \right) g(W_{qr}(p_j, p_k)) \end{aligned} \quad (19)$$



■ **Figure 7** Projection of  $N$  along  $pr$ . Edge  $pr$  gets added when forming  $\text{conv}(P_N(\pi, p) \cup \{p\})$  if and only if (a)  $r$ ,  $t_1$  and  $t_2$  appear before  $p$  in  $\pi$  (b) No point outside the cone formed by  $pr$ ,  $t_1$  and  $t_2$  (red point) appears before  $p$  in  $\pi$ .

We have a factor of  $\frac{1}{2}$  because each pair is counted twice. In the previous case, all the inner summations have been pre-computed. So for a fixed  $(p, r)$ , the summation can be computed in  $O(n)$  time given previous computation. Hence in total it takes  $O(n^3)$  time and no additional space to compute this case.

Now that all the cases take no more than  $O(n^3 \log^2 n)$  time and  $O(n)$  space to compute, we present our main theorem:

► **Theorem 13.** *Shapley values for mean width for a point set in 3-D can be computed in  $O(n^3 \log^2 n)$  time and  $O(n)$  space.*

## 5 Discussion

We have presented an algorithm to compute Shapley values of a point set in 3-D with respect to the mean width of its convex hull. We provided an efficient algorithm based on a data structure for a variant of dynamic convolution. We believe the data structure may be of independent interest.

Our algorithm naturally extends to higher dimension to compute Shapley values for  $M_{d-2}(\text{conv}(P))$  for a  $d$ -dimensional point set  $P$ . This relies on the fact that the orthogonal space of a  $(d-2)$ -face is a plane. In general, it takes  $O(n^d \log^2 n)$  time to compute the Shapley values. It is also known that for a convex polytope  $X$ ,  $M_{d-1}(X)$  is equivalent to the  $(d-1)$ -volume of the boundary of  $X$  up to some constant [20]. Hence the algorithm by Cabello and Chan [7] with natural extension can be used to compute Shapley values for  $M_{d-1}(\text{conv}(P))$  in  $O(n^d)$  time. It would be natural to ask whether there are efficient algorithms to compute  $M_i(\text{conv}(P))$  in general.

We can also consider  $\epsilon$ -coreset of Shapley values for geometric objects. Let  $P$  be a set of geometric objects and  $v$  be a characteristic function on  $P$ . We can define the  $\epsilon$ -coreset  $\tilde{P}$  to be a weighted set of geometric objects such that, for any geometric objects  $x$ ,  $(1 - \epsilon)\phi_{P \cup \{x\}}(x) \leq \phi_{\tilde{P} \cup \{x\}}(x) \leq (1 + \epsilon)\phi_{P \cup \{x\}}(x)$  where  $\phi_N$  means that the underlying player set for the Shapley value is  $N$ . Intuitively,  $\phi_{P \cup \{x\}}(x)$  means the contribution  $x$  makes when  $x$  is added as an additional player. Does  $\tilde{P}$  exist? If so, what is the upper and lower bound of its size? How fast can we find a coreset?

## References

- 1 Pankaj K. Agarwal, Sarel Har-Peled, Subhash Suri, Hakan Yıldız, and Wuzhou Zhang. Convex hulls under uncertainty. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms – ESA 2014*, pages 37–48, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-44777-2\_4.
- 2 Deepak Ajwani, Saurabh Ray, Raimund Seidel, and Hans Raj Tiwary. On computing the centroid of the vertices of an arrangement and related problems. In Frank Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures*, pages 519–528, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-73951-7\_45.
- 3 David Alonso-Gutiérrez and Joscha Prochno. On the Gaussian behavior of marginals and the mean width of random polytopes. *Proceedings of the American Mathematical Society*, 143(2):821–832, 2015. doi:10.1090/S0002-9939-2014-12401-4.
- 4 Mauricio Alvarez-Manilla, Abbas Edalat, and Nasser Saheb-Djahromi. An extension result for continuous valuations. *Journal of the London Mathematical Society*, 61(2):629–640, 2000. doi:10.1112/S0024610700008681.
- 5 Boris Aronov and Matthew J. Katz. Batched point location in SINR diagrams via algebraic tools. *ACM Trans. Algorithms*, 14(4):41:1–41:29, August 2018. doi:10.1145/3209678.
- 6 Karoly J. Böröczky, Ferenc Fodor, Matthias Reitzner, and Viktor Vigh. Mean width of random polytopes in a reasonably smooth convex body. *Journal of Multivariate Analysis*, 100(10):2287–2295, 2009. doi:10.1016/j.jmva.2009.07.003.
- 7 Sergio Cabello and Timothy M. Chan. Computing Shapley Values in the Plane. In Gill Barequet and Yusu Wang, editors, *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SoCG.2019.20.
- 8 Satya R. Chakravarty, Manipushpak Mitra, and Palash Sarkar. *A Course on Cooperative Game Theory*. Cambridge University Press, 2015. doi:10.1017/CB09781107415997.
- 9 Timothy M. Chan. A (slightly) faster algorithm for Klee’s measure problem. *Computational Geometry*, 43(3):243–250, 2010. Special Issue on 24th Annual Symposium on Computational Geometry (SoCG’08). doi:10.1016/j.comgeo.2009.01.007.
- 10 Martin Fink, John Hershberger, Nirman Kumar, and Subhash Suri. Hyperplane Separability and Convexity of Probabilistic Point Sets. In Sándor Fekete and Anna Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SoCG.2016.38.
- 11 Gudmund Skovbjerg Frandsen, Johan P. Hansen, and Peter Bro Miltersen. Lower bounds for dynamic algebraic problems. *Information and Computation*, 171(2):333–349, 2001. doi:10.1006/inco.2001.3046.
- 12 Alfred Gray. *Steiner’s Formula*, pages 209–229. Birkhäuser Basel, Basel, 2004. doi:10.1007/978-3-0348-7966-8\_10.
- 13 Hugo Hadwiger. *Vorlesungen über Inhalt, Oberfläche und Isoperimetrie*, volume 93. Springer-Verlag, 2013. doi:10.1007/978-3-642-94702-5.
- 14 Sergiu Hart. Shapley value. In John Eatwell, Murray Milgate, and Peter Newman, editors, *Game Theory*, pages 210–216. Palgrave Macmillan UK, London, 1989. doi:10.1007/978-1-349-20181-5\_25.
- 15 Lingxiao Huang, Jian Li, Jeff M. Phillips, and Haitao Wang. epsilon-kernel coresets for stochastic points. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:18, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2016.50.
- 16 Roland Huber. Continuous valuations. *Mathematische Zeitschrift*, 212(1):455–477, January 1993. doi:10.1007/BF02571668.

- 17 Daniel A. Klain. A short proof of Hadwiger's characterization theorem. *Mathematika*, 42(2):329–339, 1995. doi:10.1112/S0025579300014625.
- 18 Stefan Langerman. On the complexity of halfspace area queries. *Discrete & Computational Geometry*, 30(4):639–648, October 2003. doi:10.1007/s00454-003-2856-2.
- 19 Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235, March 2008. doi:10.1007/s00453-008-9174-2.
- 20 Roger E. Miles. Poisson flats in Euclidean spaces. part I: A finite number of random uniform flats. *Advances in Applied Probability*, 1(2):211–237, 1969. doi:10.2307/1426218.
- 21 Josef S. Müller. On the mean width of random polytopes. *Probability Theory and Related Fields*, 82(1):33–37, June 1989. doi:10.1007/BF00340011.
- 22 Roger B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991. doi:10.2307/j.ctvjjsf522.
- 23 John H. Reif and Stephen R. Tate. On dynamic algorithms for algebraic problems. *Journal of Algorithms*, 22(2):347–371, 1997. doi:10.1006/jagm.1995.0807.
- 24 Alvin E. Roth, editor. *The Shapley Value: Essays in Honor of Lloyd S. Shapley*. Cambridge University Press, 1988. doi:10.1017/CB09780511528446.
- 25 Rolf Schneider. *Convex Bodies: The Brunn–Minkowski Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2 edition, 2013. doi:10.1017/CB09781139003858.
- 26 Subhash Suri, Kevin Verbeek, and Hakan Yıldız. On the most likely convex hull of uncertain points. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013*, pages 791–802, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40450-4\_67.
- 27 Eyal Winter. The Shapley value. In *Handbook of Game Theory with Economic Applications*, volume 3, chapter 53, pages 2025–2054. Elsevier, 2002. doi:10.1016/S1574-0005(02)03016-3.
- 28 Jie Xue, Yuan Li, and Ravi Janardan. On the expected diameter, width, and complexity of a stochastic convex-hull. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 581–592, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-62127-2\_49.





# Simple Envy-Free and Truthful Mechanisms for Cake Cutting with a Small Number of Cuts

Takao Asano 

Chuo University, Tokyo, Japan

---

## Abstract

For the cake-cutting problem, Alijani, et al. [2, 25] and Asano and Umeda [4, 5] gave envy-free and truthful mechanisms with a small number of cuts, where the desired part of each player's valuation function is a single interval on a given cake. In this paper, we give envy-free and truthful mechanisms with a small number of cuts, which are much simpler than those proposed by Alijani, et al. [2, 25] and Asano and Umeda [4, 5]. Furthermore, we show that this approach can be applied to the envy-free and truthful mechanism proposed by Chen, et al. [13], where the valuation function of each player is more general and piecewise uniform. Thus, we can obtain an envy-free and truthful mechanism with a small number of cuts even if the valuation function of each player is piecewise uniform, which solves the future problem posed by Alijani, et al. [2, 25].

**2012 ACM Subject Classification** Theory of computation → Algorithmic game theory and mechanism design

**Keywords and phrases** cake-cutting problem, envy-freeness, fairness, truthfulness, mechanism design

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.68

**Acknowledgements** The author would like to thank Professor Shigeo Tsujii of Chuo University.

## 1 Introduction

The problem of dividing a cake among players in a fair manner has attracted the attention of mathematicians, economists, political scientists and computer scientists [6, 7, 11, 13, 14, 15, 16, 22, 23, 24] since it was first considered by Banach and Knaster [11] and Steinhaus [27, 28]. The cake-cutting problem is often used as a metaphor for prominent real-world problems that involve the division of a heterogeneous divisible good [10, 14, 26, 32].

Formally, the cake-cutting problem is stated as follows [13]: Given a divisible heterogeneous cake  $C$  represented by an interval  $[0, 1)$  and  $n$  players  $N = \{1, 2, \dots, n\}$  where each player  $i \in N$  has a valuation function  $v_i$  over the cake  $C$ , divide the cake  $C$  and find an allocation of the cake  $C$  to the players that satisfies one or several fairness criteria. In the cake cutting literature, one of the most important criteria is *envy-freeness* [7]. In an envy-free allocation, each player considers her/his own allocation at least as good as any other player's allocation.

A *piece*  $A$  of cake  $C$  is a finite union of disjoint subintervals  $X$  of  $C$ . A piece  $A$  can also be viewed as a set of disjoint subintervals  $X$  of  $C$ . For a general valuation function  $v_i$  of player  $i \in N$  which is integrable or piecewise continuous, the value  $V_i(A)$  of a piece  $A$  of cake  $C$  for player  $i$  can be written by  $\int_{x \in A} v_i(x) dx$ . Thus, the value  $V_i(A)$  of the piece  $A$  of disjoint subintervals  $X$  of  $C$  for player  $i$  is  $V_i(A) = \sum_{X \in A} V_i(X)$ .

Since general valuation functions may not have a finite discrete representation as an input to the cake-cutting problem, most algorithms and computational complexity analyses are based on oracle computation models. Among them a most popular computation model for general integrable valuation functions is the Robertson-Webb model based on two types of queries: evaluation and cut [24]. For envy-freeness, Stromquist [23, 30] showed that there is no finite envy-free cake cutting algorithm that outputs a contiguous allocation to each player for any  $n \geq 3$ , although an envy-free allocation with a contiguous interval allocation to each player is guaranteed to exist [29, 31]. Note that any cake cutting algorithm that outputs a



© Takao Asano;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 68; pp. 68:1–68:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

contiguous allocation to each player uses  $n - 1$  cuts on the cake  $C$ . If a contiguous allocation to each player is not required, Aziz and Mackenzie [6] showed that there is an envy-free cake cutting algorithm with  $O(n^{n^{n^{n^n}}})$  queries. Procaccia showed that any envy-free cake cutting algorithm requires  $\Omega(n^2)$  queries [21]. Furthermore, Deng, Qi and Saberi [14] showed that finding an envy-free allocation using  $n - 1$  cuts on cake  $C$  is PPAD-complete when valuation functions are given explicitly by polynomial-time algorithms, although their result requires very general (e.g., non-additive, non monotone) valuation functions [18].

In recent papers, some restricted classes of valuation functions have been studied [7, 8, 10, 12, 13, 20]. Piecewise uniform and piecewise constant valuation functions are two special classes of valuation functions [2, 7, 13, 25]. For a nonnegative valuation function  $v$  on cake  $C$ , let  $D(v) = \{x \in C \mid v(x) > 0\}$ . Thus, we can consider that  $D(v)$  consists of several disjoint maximal contiguous intervals. Then  $v$  is called *piecewise uniform* if  $v(x) = v(y)$  holds for all  $x, y \in D(v)$ . Similarly,  $v$  is called *piecewise constant* if, for each contiguous interval  $I$  in  $D(v)$ ,  $v(x') = v(x'')$  holds for all  $x', x'' \in I$ . Note that  $v(x) \neq v(y)$  may hold for  $x \in I$  and  $y \in J$  when  $I, J$  are two distinct maximal contiguous intervals in  $D(v)$  of piecewise constant valuation  $v$ . Thus, a piecewise uniform valuation is always piecewise constant. One of the most important properties of these valuation functions is that they can be described concisely. Kurokawa, Lai, and Procaccia [19] proved that finding an envy-free allocation in the Robertson-Webb model when the valuation functions are piecewise uniform is as hard as solving the problem without any restriction on the valuation functions.

The cake-cutting problem has been studied not only from the viewpoint of computational complexity but also from the game theoretical point of view [2, 7, 8, 9, 13, 20, 25]. Chen, Lai, Parkes, and Procaccia [13] considered a strong notion of truthfulness (denoted by strategy-proofness), in which the players' dominant strategies are to reveal their true valuations over the cake. They presented an envy-free and truthful mechanism for the cake-cutting problem based on maximum flow and minimum cut techniques [34] when the valuation functions are piecewise uniform. Aziz and Ye [7] considered the problem when valuation functions are piecewise constant and piecewise uniform. They designed three algorithms CCEA, MEA, and CSD for piecewise constant valuations, which partially solve an open problem for piecewise constant valuations posed by Chen et al. in [13]. They showed that CCEA runs in  $O(n^5 M^2 \log(\frac{n^2}{M}))$  time, where  $M$  is the number of subintervals defined by the union of discontinuity points of the players' piecewise constant valuations ( $M \leq 2 \sum_{i \in N} m_i$  where  $m_i$  is the number of maximal contiguous intervals in  $D(v_i) = \{x \in C \mid v_i(x) > 0\}$  of piecewise constant valuation  $v_i$ ). They also showed that, when CCEA and MEA are restricted to piecewise uniform valuations, CCEA and MEA become essentially the same as the mechanism in [13]. However, note that CCEA, MEA and the mechanism in [13] for dividing the cake use  $\Omega(nM)$  cuts [2, 25].

Alijani, Farhadi, Ghodsi, Seddighin, and Tajik [2, 25] considered that the number of cuts is important and considered the following cake-cutting problem by requiring  $D(v_i) = \{x \in C \mid v_i(x) > 0\}$  of piecewise uniform valuation  $v_i$  of each player  $i \in N$  to be a single contiguous interval  $C_i$  in cake  $C$ : Given a divisible heterogeneous cake  $C$ ,  $n$  strategic players  $N = \{1, 2, \dots, n\}$  with valuation interval  $C_i \subseteq C$  of each player  $i \in N$ , find a mechanism for dividing  $C$  into pieces and allocating pieces of  $C$  to  $n$  players  $N$  to meet the following conditions: (i) the mechanism is envy-free; (ii) the mechanism is truthful; and (iii) the number of cuts made on cake  $C$  is small. And they gave an envy-free and truthful mechanism with at most  $2n - 2$  cuts [2, 25], although their original mechanism is not actually envy free [5] and corrected later by themselves. Asano and Umeda [4, 5] also gave an alternative envy-free and truthful mechanism with at most  $2n - 2$  cuts.

In this paper, we give envy-free and truthful mechanisms with a small number of cuts, which lead to a much simpler mechanism than those proposed by Alijani, et al. [2, 25] and Asano and Umeda [4, 5]. Thus, we can obtain a much simpler envy-free and truthful mechanism with at most  $2n - 2$  cuts which runs in  $O(n^3)$  time for the above cake-cutting problem. Furthermore, we show that this approach can be applied to the envy-free and truthful mechanism proposed by Chen, et al. [13] for the more general cake-cutting problem where the valuation function  $v_i$  of each player  $i \in N$  is piecewise uniform. Thus, this approach can make their envy-free and truthful mechanism use at most  $2M - 2$  cuts and we solve the open problem posed by Alijani, et al. [2, 25], where  $M \leq 2 \sum_{i \in N} m_i$  and  $m_i$  is the number of maximal contiguous intervals in  $D(v_i) = \{x \in C \mid v_i(x) > 0\}$  of  $v_i$  as mentioned above.

## 2 Preliminaries

We are given a divisible heterogeneous cake  $C = [0, 1) = \{x \mid 0 \leq x < 1\}$ <sup>1</sup>,  $n$  strategic players  $N = \{1, 2, \dots, n\}$  with valuation interval  $C_i = [\alpha_i, \beta_i) = \{x \mid 0 \leq \alpha_i \leq x < \beta_i \leq 1\} \subseteq C$  of each player  $i \in N$ . We denote by  $\mathcal{C}_N$  the (multi-) set of valuation intervals of all the players  $N$ , i.e.,  $\mathcal{C}_N = (C_1, C_2, \dots, C_n)$ . We also write  $\mathcal{C}_N = (C_i : i \in N)$ . Valuation intervals  $\mathcal{C}_N$  is called *solid*, if, for every  $x \in C$ , there is a player  $i \in N$  whose valuation interval  $C_i \in \mathcal{C}_N$  contains  $x$ . As in [2, 4, 7, 25], we will assume that  $\mathcal{C}_N$  is solid, i.e.,  $\bigcup_{C_i \in \mathcal{C}_N} C_i = C$ , throughout this paper.

A union  $X$  of mutual disjoint sets  $X_1, X_2, \dots, X_k$  is denoted by  $X = X_1 + X_2 + \dots + X_k = \sum_{\ell=1}^k X_\ell$ . A *piece*  $A_i$  of cake  $C$  is a union of mutually disjoint subintervals  $A_{i_1}, A_{i_2}, \dots, A_{i_{k_i}}$  of  $C$ . Thus,  $A_i = A_{i_1} + A_{i_2} + \dots + A_{i_{k_i}} = \sum_{\ell=1}^{k_i} A_{i_\ell}$ . A partition  $A_N = (A_1, A_2, \dots, A_n)$  of cake  $C$  into  $n$  disjoint pieces  $A_1, A_2, \dots, A_n$  is called an *allocation* of  $C$  to  $n$  players  $N$  if each piece  $A_i = \sum_{\ell=1}^{k_i} A_{i_\ell}$  is allocated to player  $i \in N$ . We also write  $A_N = (A_i : i \in N)$ . Thus, in allocation  $A_N = (A_i : i \in N)$  of  $C$  to  $n$  players  $N$ ,  $\sum_{i \in N} A_i = C$  holds and  $A_i = \sum_{\ell=1}^{k_i} A_{i_\ell}$  is called an *allocated piece* of  $C$  to player  $i \in N$ .

For an interval  $X = [x', x'')$ , we denote by  $cl(X)$  the *closure* of  $X$  and thus  $cl(X) = [x', x''] = \{x \mid x' \leq x \leq x''\}$ . For an interval  $X = [x', x'')$  of  $C$ , the *length* of  $X$ , denoted by  $len(X)$ , is defined by  $x'' - x'$ . For a piece  $A = \sum_{\ell=1}^k X_\ell$  of cake  $C$ , the *length* of  $A$ , denoted by  $len(A)$ , is defined by the total sum of  $len(X_\ell)$ , i.e.,  $len(A) = \sum_{\ell=1}^k len(X_\ell)$ . For each  $i \in N$  and valuation interval  $C_i$  of player  $i$ , the *value* of piece  $A = \sum_{\ell=1}^k X_\ell$  for player  $i$ , denoted by  $V_i(A)$ , is the total sum of  $len(X_\ell \cap C_i)$ , i.e.,  $V_i(A) = \sum_{\ell=1}^k len(X_\ell \cap C_i)$ . For an allocation  $A_N = (A_i : i \in N)$  of cake  $C$  to  $n$  players  $N$ , if  $V_i(A_i) \geq V_i(A_j)$  for all  $j \in N$ , then the allocated piece  $A_i$  to player  $i$  is called *envy-free* for player  $i$ . If, for every player  $i \in N$ , the allocated piece  $A_i$  to player  $i$  is envy-free for player  $i$ , then the allocation  $A_N = (A_i : i \in N)$  to  $n$  players  $N$  is called *envy-free*.

Let  $\mathcal{M}$  be a mechanism (i.e., a polynomial-time algorithm in this paper) for the cake-cutting problem. Let  $\mathcal{C}_N = (C_i : i \in N)$  be an arbitrary input to  $\mathcal{M}$  and  $A_N = (A_i : i \in N)$  be an allocation of cake  $C$  to  $n$  players  $N$  obtained by  $\mathcal{M}$ . If  $A_N = (A_i : i \in N)$  for every input  $\mathcal{C}_N = (C_i : i \in N)$  to  $\mathcal{M}$  is envy-free then  $\mathcal{M}$  is called *envy-free*.

Now, assume that only player  $i \in N$  gives a false valuation interval  $C'_i$  and let  $C'_N(i) = (C'_j : j \in N)$  (all the other players  $j \neq i$  give true valuation intervals  $C_j$  and thus  $C'_j = C_j$  for each  $j \neq i$ ) be an input to  $\mathcal{M}$  and let an allocation of cake  $C$  to  $n$  players  $N$  obtained

<sup>1</sup> We assume  $C = [0, 1) = \{x \mid 0 \leq x < 1\}$ . We also assume, if an interval  $X = [x', x'') = \{x \mid x' \leq x < x''\}$  of  $C = [0, 1)$  is cut at  $y \in X$  with  $x' < y < x''$  then  $X$  is divided into two subintervals  $X' = [x', y)$  and  $X'' = [y, x'')$ .

by  $\mathcal{M}$  be  $A'_N(i) = (A'_j : j \in N)$ . The values of  $A_i = \sum_{\ell=1}^{k_i} A_{i_\ell}$  and  $A'_i = \sum_{\ell=1}^{k'_i} A'_{i_\ell}$  for player  $i$  are  $V_i(A_i) = \sum_{\ell=1}^{k_i} \text{len}(A_{i_\ell} \cap C_i)$  and  $V_i(A'_i) = \sum_{\ell=1}^{k'_i} \text{len}(A'_{i_\ell} \cap C_i)$  (note that  $V_i(A'_i) \neq \sum_{\ell=1}^{k'_i} \text{len}(A'_{i_\ell} \cap C'_i)$ ). If  $V_i(A_i) \geq V_i(A'_i)$ , then there is no merit for player  $i$  to give false  $C'_i$  and player  $i$  will report true valuation interval  $C_i$  to  $\mathcal{M}$ . For each player  $i \in N$ , if this holds for all such  $C'_i$ 's, then  $\mathcal{M}$  is called *truthful* (allocation  $A_N = (A_i : i \in N)$  obtained by  $\mathcal{M}$  is also called *truthful*).

For valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  and an interval  $X = [x', x'']$  of cake  $C$ , let  $N(X)$  be the set of players  $i$  in  $N$  whose valuation interval  $C_i$  is contained in  $X$  and let  $\mathcal{C}_{N(X)}$  be the (multi-) set of valuation intervals in  $\mathcal{C}_N$  which are contained in  $X$ . Thus,  $N(X) = \{i \in N \mid C_i \subseteq X, C_i \in \mathcal{C}_N\}$  and  $\mathcal{C}_{N(X)} = (C_i \in \mathcal{C}_N : i \in N(X))$ . The *density* of interval  $X = [x', x'']$  of  $C$ , denoted by  $\rho(X)$ , is defined by  $\rho(X) = \frac{\text{len}(X)}{|N(X)|} = \frac{x''-x'}{|N(X)|}$ . The density  $\rho(X)$  is the average length of pieces of the players in  $N(X)$  when the part  $X$  of cake  $C$  is divided among the players in  $N(X)$ . Let  $\mathcal{X}$  be the set of all nonempty intervals in  $C$ . Let  $\rho_{\min}$  be the minimum density among the densities of all nonempty intervals in  $C$ , i.e.,  $\rho_{\min} = \min_{X \in \mathcal{X}} \rho(X)$ . Let  $\mathcal{X}_{\min} = \{X \in \mathcal{X} \mid \rho(X) = \rho_{\min}\}$ . Thus,  $\mathcal{X}_{\min}$  is the set of all intervals of minimum density in  $C$ . Note that, for each interval  $X = [x', x''] \in \mathcal{X}_{\min}$ , there are valuation intervals  $C_i = [\alpha_i, \beta_i], C_j = [\alpha_j, \beta_j] \in \mathcal{C}_N$  with  $x' = \alpha_i$  and  $x'' = \beta_j$ . Thus, the set of all intervals of minimum density in  $C$  can be computed in  $O(n^2)$  time. An interval  $X \in \mathcal{X}_{\min}$  is called a *maximal interval of minimum density* if no other interval of  $\mathcal{X}_{\min}$  contains  $X$  properly. A *minimal interval of minimum density* is similarly defined.

### 3 Core Mechanism $\mathcal{M}_1$

For cake  $C = [0, 1)$ ,  $n$  strategic players  $N = \{1, 2, \dots, n\}$ , and solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with  $C_i = [\alpha_i, \beta_i] \subseteq C$  of each player  $i \in N$ , each mechanism  $\mathcal{M}$  in [2, 25] and [4, 5] uses a small number of cuts and finds an allocation  $A_N = (A_i : i \in N)$  to players  $N$  satisfying the following: (a)  $\mathcal{M}$  is envy-free; (b)  $\mathcal{M}$  is truthful; (c)  $A_i \subseteq C_i$  for each  $i \in N$ ; and (d)  $\sum_{i \in N} A_i = C$ . However, their mechanisms were quite complicated.

In this paper, we give a much simpler envy-free and truthful mechanism with a small number of cuts. For this purpose, we first give a core mechanism  $\mathcal{M}_1$  which assumes that cake  $C = [0, 1)$  is an interval of minimum density  $\rho_{\min}$  in  $C = [0, 1)$  (thus,  $\rho_{\min} = \frac{1}{n}$ ).

#### Algorithm 1 Core Mechanism $\mathcal{M}_1$ .

---

**Input:** Cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$  and solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with valuation interval  $C_i = [\alpha_i, \beta_i] \subseteq C$  of each player  $i \in N$  and  $\bigcup_{C_i \in \mathcal{C}_N} C_i = C$ , where  $C = [0, 1)$  is an interval of minimum density  $\rho_{\min} = \frac{1}{n}$  in cake  $C = [0, 1)$ .

**Output:** Allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$ ,  $\text{len}(A_i) = \rho_{\min}$  for each  $i \in N$  and  $\sum_{i \in N} A_i = C$ .

sort  $\mathcal{C}_N = (C_i : i \in N)$  in a lexicographic order with respect to  $(\beta_i, \alpha_i)$  and assume

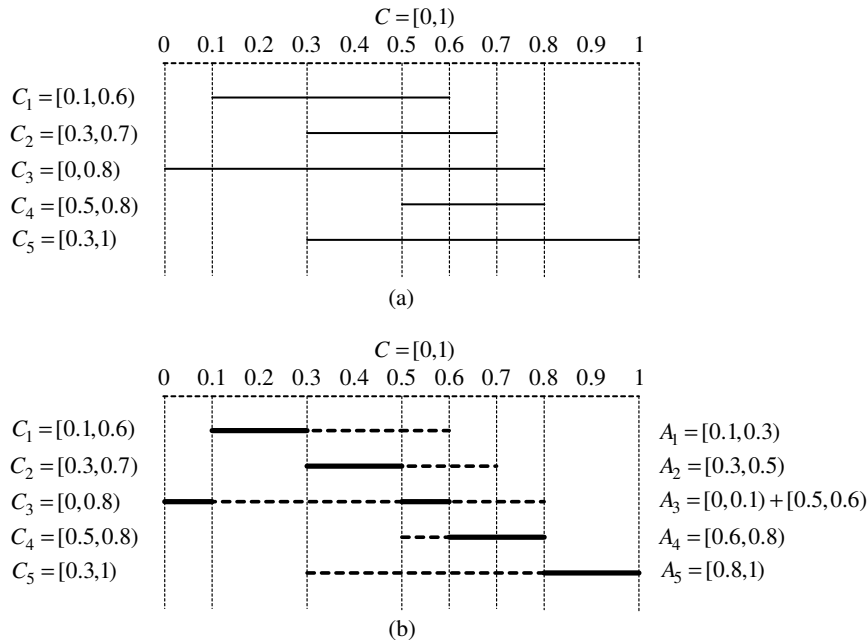
$C_1 \leq C_2 \leq \dots \leq C_n$  in this lexicographic order;

set  $A_1 = [a_1, b_1] \subseteq C_1$  with length  $\rho_{\min}$  such that  $a_1 = \alpha_1$  and  $b_1 = \alpha_1 + \rho_{\min}$ ;

**for**  $i = 2$  **to**  $n$  **do**

set  $A_i = [a_i, b_i] \setminus \sum_{\ell=1}^{i-1} A_\ell$  with length  $\rho_{\min}$  such that  $[a_i, b_i] \subseteq C_i$  and  $a_i$  is the leftmost endpoint in  $C_i \setminus \sum_{\ell=1}^{i-1} A_\ell$ ;

---



■ **Figure 1** (a) Solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with  $\rho(C) = \rho_{\min} = 0.2$ . (b) Allocation  $A_N = (A_i : i \in N)$  obtained by  $\mathcal{M}_1$ .

Figure 1 shows an example of solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with  $\rho(C) = \rho_{\min} = 0.2$  and an allocation  $A_N = (A_i : i \in N)$  obtained by  $\mathcal{M}_1$ .

► **Theorem 1.** For cake  $C = [0, 1]$ ,  $n$  players  $N = \{1, 2, \dots, n\}$ , and solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with valuation interval  $C_i = [\alpha_i, \beta_i]$  of each player  $i \in N$  and  $\bigcup_{C_i \in \mathcal{C}_N} C_i = C$ , let  $C = [0, 1]$  be an interval of minimum density  $\rho_{\min} = \frac{1}{n}$  in cake  $C = [0, 1]$ . Then,  $\mathcal{M}_1$  finds an allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = \rho_{\min}$  for each  $i \in N$  and  $\sum_{i \in N} A_i = C$  in  $O(n \log n)$  time. Furthermore, the number of cuts made by  $\mathcal{M}_1$  on cake  $C$  is at most  $2n - 2$ .

**Proof.** The number of cuts made on cake  $C$  is clearly at most  $2n - 2$ , since  $\mathcal{M}_1$  uses at most two cuts at  $a_i$  and  $b_i$  for each  $i \in N$  to obtain  $A_i$  and no cut is required at  $0, 1$  of cake  $C = [0, 1]$ . Similarly, it can be easily shown that  $\mathcal{M}_1$  runs in  $O(n \log n)$  time, since lexicographical sorting of  $\mathcal{C}_N = (C_i : i \in N)$  requires  $O(n \log n)$  time and  $a_i, b_i$  for each  $i \in N$  can be found in  $O(\log n)$  time based on appropriate data structures.

We next prove the proposition that  $\mathcal{M}_1$  correctly finds an allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = \rho_{\min}$  for each  $i \in N$  and  $\sum_{i \in N} A_i = C$  by induction on  $n$ .

If  $n = 1$  then the proposition is clearly true.

Now we assume that the proposition is true for  $n - 1$  players and consider  $n \geq 2$  players. Of course,  $A_1 = [a_1, b_1] \subseteq C_1 = [\alpha_1, \beta_1]$  with  $a_1 = \alpha_1$  and  $b_1 = \alpha_1 + \rho_{\min}$  (thus of length  $\rho_{\min}$ ) is allocated to player 1, since  $\rho(C_1) \geq \rho_{\min}$  and thus the length of  $C_1$  is  $\beta_1 - \alpha_1 \geq \rho_{\min}$ . Then we delete  $A_1 = [a_1, b_1]$  and virtually consider  $a_1 = b_1$  (we call this as *virtual shrinking* of hollow interval  $A_1$  after deletion of  $A_1$ ). Note that, since we performed virtual shrinking of hollow interval  $A_1 = [a_1, b_1]$  and virtually considered  $a_1 = b_1$ , the remaining cake  $C' = C \setminus A_1$  may be considered as a single interval and each  $C'_k = C_k \setminus A_1$  ( $k \in N \setminus \{1\}$ ) may also be considered as a single interval. Thus, the resulting cake-cutting problem may be considered to be the same as the original cake-cutting problem, that is, it consists of cake  $C' = C \setminus A_1$

which is a single interval, players  $N' = N \setminus \{1\}$ , and valuation intervals  $\mathcal{C}'_{N'}$  with a single interval  $C'_k = C_k \setminus A_1 \subseteq C'$  of each remaining player  $k \in N'$ . The solidness of  $\mathcal{C}'_{N'}$  (i.e.,  $\bigcup_{C'_i \in \mathcal{C}'_{N'}} C'_i = C'$ ) also holds, which can be obtained as follows.

If  $\alpha_1 > 0$  then there is a valuation interval  $C_j = [\alpha_j, \beta_j] \in \mathcal{C}_N$  with  $\alpha_j = 0$  and  $\beta_j \geq \beta_1$  by the solidness of  $\mathcal{C}_N$  (i.e.,  $\bigcup_{C_i \in \mathcal{C}_N} C_i = C$ ) and the fact that  $\mathcal{C}_N$  was sorted in the lexicographic order, and thus,  $\bigcup_{C'_i \in \mathcal{C}'_{N'}} C'_i = C'$ .

If  $\alpha_1 = 0$  then let  $\alpha = \min\{\alpha_i \mid C_i = [\alpha_i, \beta_i] \in \mathcal{C}_N \setminus \{C_1\}\} < 1$ . Then we have  $\alpha \leq \rho_{\min}$ , since if  $\alpha > \rho_{\min}$  then, for interval  $X = [\alpha, 1)$ , we would have  $N(X) = N' = N \setminus \{1\}$  and  $\rho(X) = \frac{\text{len}(X)}{|N(X)|} = \frac{1-\alpha}{n-1} < \rho_{\min}$  by  $\rho_{\min} = \frac{1}{n}$ , a contradiction since  $\rho_{\min}$  is the minimum density of all intervals of  $C$  and thus  $\rho(X) \geq \rho_{\min}$ . This implies that  $\bigcup_{C'_i \in \mathcal{C}'_{N'}} C'_i = C'$  even if  $\alpha_1 = 0$ .

The density  $\rho'$  of intervals in this resulting cake-cutting problem can be easily shown to satisfy  $\rho'(X') \geq \rho_{\min}$  for each nonempty interval  $X'$  of  $C'$  and  $\rho'(C') = \rho_{\min}$ . Actually,  $\rho'(C') = \frac{\text{len}(C')}{|N'|} = \frac{1-\rho_{\min}}{n-1} = \frac{1-\frac{1}{n}}{n-1} = \frac{1}{n} = \rho_{\min}$ . Each nonempty interval  $X' \subseteq C'$  can be written by  $X' = X \setminus A_1$  for some interval  $X = [x', x''] \subseteq C$ . Let  $Y = X' \cup A_1 = X \cup A_1$  (it is possible that there are many  $X$ , but  $Y$  is uniquely determined). Of course,  $X' = Y \setminus A_1$ .

If  $Y$  is not a single interval of  $C$ , then  $\text{cl}(X) \cap \text{cl}(A_1) = [x', x''] \cap [\alpha_1, \alpha_1 + \rho_{\min}] = \emptyset$  (i.e.,  $x' < x'' < \alpha_1 < \alpha_1 + \rho_{\min}$  or  $\alpha_1 < \alpha_1 + \rho_{\min} < x' < x''$ ), which implies that there is unique  $X \subseteq C$  with  $X' = X \setminus A_1 = Y \setminus A_1 = X$  and  $\rho'(X') = \rho(X) \geq \rho_{\min}$ .

Thus, we can assume that  $Y$  is a single interval  $Y = [y', y'']$  of  $C$  with  $y' \leq \alpha_1 < \alpha_1 + \rho_{\min} \leq y''$  by  $A_1 = [\alpha_1, \alpha_1 + \rho_{\min}] \subseteq Y = X \cup A_1$ . For each  $k \in N' = N \setminus \{1\}$  and  $C_k = [\alpha_k, \beta_k] \in \mathcal{C}_N$ , if  $C_k \subseteq Y$ , then we have  $\emptyset \neq C'_k = C_k \setminus A_1 \in \mathcal{C}'_{N'}$  and  $C'_k \subseteq X' = Y \setminus A_1$  (since if  $C_k \subseteq A_1$  then  $\beta_k \leq \alpha_1 + \rho_{\min} \leq \beta_1$  by  $A_1 = [\alpha_1, \alpha_1 + \rho_{\min}] \subseteq C_1 = [\alpha_1, \beta_1]$  and we would have  $\beta_k = \alpha_1 + \rho_{\min} = \beta_1$  and  $A_1 = C_1$  by the fact that  $\mathcal{C}_N = (C_i : i \in N)$  was sorted in a lexicographic order with respect to  $(\beta_i, \alpha_i)$ , and thus  $\rho(A_1) \leq \frac{1}{2}\rho_{\min} < \rho_{\min}$ , a contradiction). Similarly, if  $C_k \not\subseteq Y$ , then it is clear that  $\emptyset \neq C'_k = C_k \setminus A_1 \in \mathcal{C}'_{N'}$  and  $C'_k \not\subseteq X' = Y \setminus A_1$ . Thus,  $N'(X') = \{k \in N' \mid C'_k \in \mathcal{C}'_{N'}, C'_k \subseteq X'\} = \{k \in N \setminus \{1\} \mid C_k \in \mathcal{C}_N, C_k \subseteq Y\} = N(Y) \setminus \{1\}$ . This implies  $|N'(X')| = |N(Y)| - 1$  or  $|N'(X')| = |N(Y)|$ . If  $1 \notin N(Y)$  (i.e., if  $C_1 = [\alpha_1, \beta_1] \not\subseteq Y = [y', y'']$ ) then  $y'' < \beta_1$  by  $y' \leq \alpha_1$  and we have  $N(Y) = \emptyset$  since  $\mathcal{C}_N = (C_i : i \in N)$  was sorted in a lexicographic order with respect to  $(\beta_i, \alpha_i)$ . Thus, if  $1 \notin N(Y)$  then  $|N'(X')| = |N(Y)| = 0$  and  $\rho'(X') = \frac{\text{len}(X')}{|N'(X')|} = \infty \geq \rho_{\min}$ . Now, assume  $1 \in N(Y)$ . Then,  $|N'(X')| = |N(Y)| - 1$  and, by  $\text{len}(Y) = |N(Y)|\rho(Y)$  and  $\rho(Y) \geq \rho_{\min}$ , we have  $\rho'(X') = \frac{\text{len}(X')}{|N'(X')|} = \frac{\text{len}(Y) - \text{len}(A_1)}{|N(Y)| - 1} = \frac{|N(Y)|\rho(Y) - \rho_{\min}}{|N(Y)| - 1} \geq \frac{|N(Y)|\rho_{\min} - \rho_{\min}}{|N(Y)| - 1} = \rho_{\min}$ .

Thus, since we performed virtual shrinking of hollow interval  $A_1 = [a_1, b_1)$  and virtually considered  $a_1 = b_1$ , the resulting cake-cutting problem with density  $\rho'$  may be considered to be the same as the original cake-cutting problem, that is, it consists of cake  $C' = C \setminus A_1$  which is a single interval of length  $1 - \frac{1}{n}$  and  $\rho'(C') = \rho'_{\min} = \rho_{\min} = \frac{1}{n}$ , players  $N' = N \setminus \{1\}$ , and solid valuation intervals  $\mathcal{C}'_{N'}$  with a single interval  $C'_k = C_k \setminus A_1 \subseteq C'$  of each remaining player  $k \in N'$  and  $\bigcup_{C'_i \in \mathcal{C}'_{N'}} C'_i = C'$ . Note that the lexicographic order of  $\mathcal{C}'_{N'}$  is the same as that of  $\mathcal{C}_{N'}$  and  $C'_2 \leq \dots \leq C'_n$  holds. Note also that, if we consider the cake  $C'$  of length  $1 - \frac{1}{n}$  as being of length 1 by virtually multiplying  $\frac{n}{n-1}$  then the minimum density  $\rho'_{\min} = \rho_{\min} = \frac{1}{n}$  will become  $\rho'_{\min} = \frac{1}{n-1}$  and we can use induction hypothesis as usual.

By induction hypothesis, the proposition is true in the resulting cake-cutting problem and we can obtain an allocation  $A'_{N'} = (A'_i : i \in N')$  with  $A'_i \subseteq C'_i$  and  $\text{len}(A'_i) = \rho_{\min}$  for each  $i \in N'$  and  $\sum_{i \in N'} A'_i = C'$ . From  $A'_{N'} = (A'_i : i \in N')$ , we can obtain an allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = \rho_{\min}$  for each  $i \in N'$  as follows: if  $A'_i = [a'_i, b'_i]$  contains the virtually shrunken interval  $A_1 = [a_1, b_1)$  then let  $A_i = [a_i, a_1) + [b_1, b_i]$  by considering  $a_1 \neq b_1$ ; otherwise, let  $A_i = A'_i$ . This is called *inverse virtual shrinking* of  $A_1$ . Thus, the proposition is true for  $n$  players.  $\blacktriangleleft$



#### 4 Application to Mechanism of Asano and Umeda [4]

For a given input of cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$ , and solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with valuation interval  $C_i = [\alpha_i, \beta_i)$  of each player  $i \in N$ , the mechanism of Asano and Umeda [4] first finds all the maximal intervals of minimum density  $\rho_{\min}$ . Let  $H_1 = [h'_1, h''_1), \dots, H_L = [h'_L, h''_L)$  be all the maximal intervals of minimum density  $\rho_{\min}$  in cake  $C = [0, 1)$ . Their mechanism then cuts cake  $C = [0, 1)$  at both endpoints of each  $H_\ell$  ( $\ell = 1, \dots, L$ ). As shown in [4], the closures of two distinct maximal intervals of minimum density are disjoint and these cuts at both endpoints of each maximal interval of minimum density can be done independently. By these cuts, the original cake-cutting problem is reduced into two types of cake-cutting subproblems of type (i) and type (ii) as follows:

- (i) the cake-cutting problem within each maximal interval  $H_\ell = [h'_\ell, h''_\ell)$  ( $\ell = 1, \dots, L$ ) of minimum density (which consists of cake  $H_\ell$ , players  $N(H_\ell)$  whose valuation intervals are in  $H_\ell$  and valuations  $\mathcal{C}_{N(H_\ell)}$  with density  $\rho$ ); and
- (ii) the cake-cutting problem obtained by deleting all  $H_\ell = [h'_\ell, h''_\ell)$  ( $\ell = 1, \dots, L$ ), i.e., the cake-cutting problem for cake  $C' = C \setminus \sum_{\ell=1}^L H_\ell$ , players  $N' = N \setminus \sum_{\ell=1}^L N(H_\ell)$  and valuations  $\mathcal{C}'_{N'}$  (which consists of valuations  $C'_k = C_k \setminus \sum_{\ell=1}^L H_\ell \neq \emptyset$  for all  $k \in N'$ ) with density  $\rho'$  and  $\bigcup_{C'_k \in \mathcal{C}'_{N'}} C'_k = C'$ .

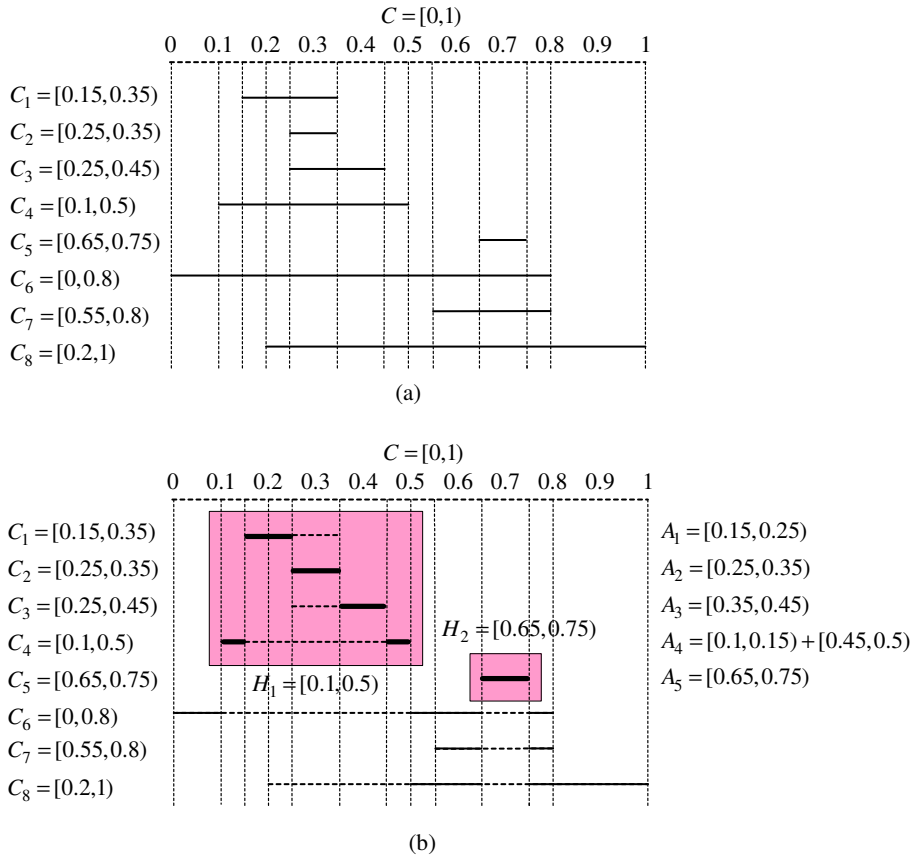
Note that the cake-cutting problem of type (i) is almost the same as the original cake-cutting problem, since cake  $H_\ell$  is a single interval, each valuation  $C_k \in \mathcal{C}_{N(H_\ell)}$  is also a single interval, and the valuation intervals  $\mathcal{C}_{N(H_\ell)}$  is solid as shown in [4]. Thus, based on the core mechanism  $\mathcal{M}_1$  (Algorithm 1), for each  $\ell = 1, \dots, L$ , we can find an allocation  $A_{N(H_\ell)} = (A_i : i \in N(H_\ell))$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = \rho_{\min}$  for each  $i \in N(H_\ell)$  and  $\sum_{i \in N(H_\ell)} A_i = H_\ell$ .

On the other hand, the cake-cutting problem of type (ii) is different from the original cake-cutting problem, because the resulting cake  $C' = C \setminus \sum_{\ell=1}^L H_\ell$  may become a set of two or more disjoint intervals and each remaining valuation  $C'_k = C_k \setminus \sum_{\ell=1}^L H_\ell \neq \emptyset$  may also become a set of two or more disjoint intervals. However, the cake-cutting problem of type (ii) can be solved in almost the same way by *virtually shrinking* all  $H_\ell$ . That is, we virtually shrink each *hollow interval*  $H_\ell = [h'_\ell, h''_\ell)$  (since  $H_\ell$  was already deleted) and virtually consider  $h'_\ell = h''_\ell$ . By virtually shrinking of all  $H_\ell = [h'_\ell, h''_\ell)$ , cake  $C' = C \setminus \sum_{\ell=1}^L H_\ell$  becomes a single interval  $C'^{(S)}$ , players  $N' = N \setminus \sum_{\ell=1}^L N(H_\ell)$  remains the same, each valuation  $C'_k \in \mathcal{C}'_{N'}$  becomes a single interval  $C'^{(S)}_k$  of  $C'^{(S)}$ , and the valuation intervals  $\mathcal{C}'^{(S)}_{N'} = (C'^{(S)}_k : k \in N')$  becomes solid (i.e.,  $\bigcup_{k \in N'} C'^{(S)}_k = C'^{(S)}$ ). Thus, by virtually shrinking of all  $H_\ell$ , the cake-cutting problem of type (ii) above can be reduced to the cake-cutting problem of type (i) for cake  $C'^{(S)}$ , players  $N' = N \setminus \sum_{\ell=1}^L N(H_\ell)$ , solid valuation intervals  $\mathcal{C}'^{(S)}_{N'} = (C'^{(S)}_k : C'_k \in \mathcal{C}'_{N'})$  with  $\bigcup_{k \in N'} C'^{(S)}_k = C'^{(S)}$  and the same density  $\rho'^{(S)} = \rho'$ , which can be solved recursively. Note that, if  $\rho(C) > \rho_{\min}$  then the minimum density  $\rho'_{\min}$  of intervals in the cake-cutting problem of type (ii) satisfies  $\rho'_{\min} > \rho_{\min}$  as shown in [4, 5].

From an allocation  $A'_{N'} = (A'^{(S)}_k : k \in N')$  to players  $N'$  where  $A'^{(S)}_k$  is the allocated piece of cake  $C'^{(S)}$  to player  $k \in N'$  with  $A'^{(S)}_k \subseteq C'^{(S)}_k$  and  $\sum_{i \in N'} A'^{(S)}_i = C'^{(S)}$ , we obtain an allocation  $A'_{N'} = (A'_k : k \in N')$  to players  $N'$  where  $A'_k$  is the allocated piece of cake  $C'$  to player  $k$  with  $A'_k \subseteq C'_k$  and  $\sum_{i \in N'} A'_i = C'$  as follows: if  $A'^{(S)}_k$  contains a shrunken interval  $H_\ell^{(S)}$  of hollow interval  $H_\ell$ , then let  $A'_k$  be the set of disjoint intervals obtained from  $A'^{(S)}_k$  by restoring each shrunken interval  $H_\ell^{(S)}$  in  $A'^{(S)}_k$  to the original hollow interval  $H_\ell = [h'_\ell, h''_\ell)$ ; otherwise, let  $A'_k = A'^{(S)}_k$ . They called this *inverse shrinking* of all  $H_\ell$  ( $\ell = 1, \dots, L$ ) in [4].

We call the method based on the core mechanism  $\mathcal{M}_1$  (Algorithm 1) described above the *modified mechanism of Asano and Umeda*. The details are in Section 5. Note that all the maximal intervals of minimum density  $\rho_{\min}$  can be obtained in  $O(n^2)$  time, since there are





■ **Figure 2** (a) Example of  $C_N = (C_i : i \in N)$ . (b) The maximal intervals  $H_1, H_2$  of minimum density  $\rho_{\min} = 0.1$ , with  $N(H_1) = \{1, 2, 3, 4\}$ ,  $A_{N(H_1)} = (A_i : i \in N(H_1))$ ,  $N(H_2) = \{5\}$ ,  $A_{N(H_2)} = (A_5)$  in the 1st iteration.

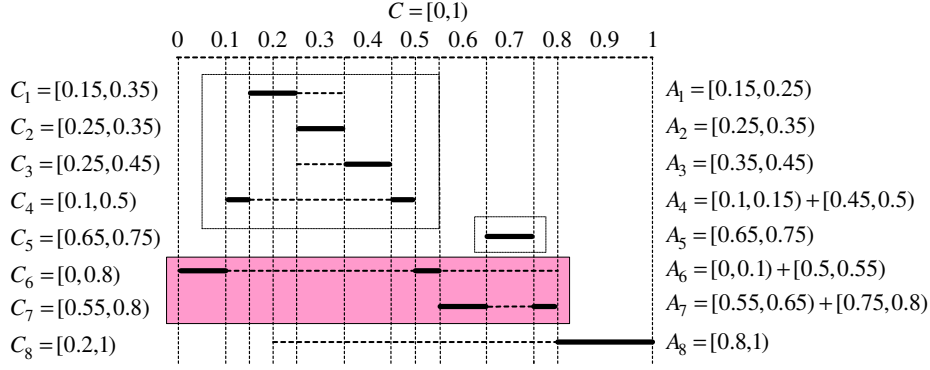
at most  $2n$  endpoints of the valuation intervals in  $C_N$  and the endpoints of each interval of minimum density are endpoints of some valuation intervals. The number of cuts made over  $C$  is also at most  $2n - 2$ . Envy-freeness and truthfulness will be given in Section 5, although they were given in [5] (also given in [2, 13, 25]). Thus, the following theorem holds as in [4].

► **Theorem 2.** *The modified mechanism of Asano and Umeda correctly finds, in  $O(n^3)$  time, an envy-free and truthful allocation  $A_N = (A_i : i \in N)$  of cake  $C$  to  $n$  players  $N$  with  $A_i \subseteq C_i$  for each player  $i \in N$  and  $\sum_{i \in N} A_i = C$ . Furthermore, the number of cuts made over  $C$  by the mechanism is at most  $2n - 2$ .*

For an input example in Figure 2(a), the modified mechanism of Asano and Umeda works as shown in Figure 2(b) and Figure 3.

## 5 Details of Modified Mechanism of Asano and Umeda

As we described in Section 4, Mechanism of Asano and Umeda [4] can be significantly simplified based on  $\mathcal{M}_1$  (Algorithm 1). Actually,  $\mathcal{M}_1$  can be slightly modified and used as Procedure  $\text{CutMaxInterval}(\cdot, \cdot, \cdot)$  in Mechanism of Asano and Umeda [4] as follows.



■ **Figure 3** The second and third iterations for the example in Figure 2. In the second iteration, the minimum density is  $\rho_{\min} = 0.15$  and  $N(H_1) = \{6, 7\}$ ,  $A_6 = [0, 0.1) + [0.5, 0.55)$  and  $A_7 = [0.55, 0.65) + [0.75, 0.8)$ . In the third (last) iteration, the minimum density is  $\rho_{\min} = 0.2$  and  $N(H_1) = \{8\}$  and  $A_8 = [0.8, 1)$ .

■ **Procedure**  $\text{CutMaxInterval}(R, H, \mathcal{D}_R)$ .

**Input:** Cake  $H = [h', h'')$ , players  $R$  and solid valuation intervals  $\mathcal{D}_R = (D_i : i \in R)$  with valuation interval  $D_i = [\alpha'_i, \beta'_i) \subseteq H$  of each player  $i \in R$  and

$\bigcup_{D_i \in \mathcal{D}_R} D_i = H$ , where  $H$  is an interval of minimum density  $\rho_{\min}$  in  $H$ .

**Output:** Allocation  $A_R = (A_i : i \in R)$  with  $A_i \subseteq D_i$  and  $\text{len}(A_i) = \rho_{\min}$  for each  $i \in R$  and  $\sum_{i \in R} A_i = H$ .

sort  $\mathcal{D}_R = (D_i = [\alpha'_i, \beta'_i) : i \in R)$  in a lexicographic order with respect to  $(\beta'_i, \alpha'_i)$  and assume  $R = \{r_1, r_2, \dots, r_{|R|}\}$  and  $D_{r_1} \leq D_{r_2} \leq \dots \leq D_{r_{|R|}}$  in this order;

set  $A_{r_1} = [a_{r_1}, b_{r_1}) \subseteq D_{r_1}$  with length  $\rho_{\min}$  such that  $a_{r_1} = \alpha'_{r_1}$  and  $b_{r_1} = \alpha'_{r_1} + \rho_{\min}$ ;

**for**  $i = 2$  **to**  $|R|$  **do**

set  $A_{r_i} = [a_{r_i}, b_{r_i}) \setminus \sum_{\ell=1}^{i-1} A_{r_\ell}$  with length  $\rho_{\min}$  such that  $[a_{r_i}, b_{r_i}) \subseteq D_{r_i}$  and

$a_{r_i}$  is the leftmost endpoint in  $D_{r_i} \setminus \sum_{\ell=1}^{i-1} A_{r_\ell}$ ;

Thus, our modified mechanism of Asano and Umeda can be written as follows (we omit inverse virtual shrinking).

■ **Algorithm 2** Modified Mechanism of Asano and Umeda.

**Input:** Cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$  and solid valuation intervals

$\mathcal{C}_N = (C_i : i \in N)$  with valuation interval  $C_i = [\alpha_i, \beta_i) \subseteq C$  of each player  $i \in N$  and  $\bigcup_{C_i \in \mathcal{C}_N} C_i = C$ .

**Output:** Allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  for  $i \in N$  and  $\sum_{i \in N} A_i = C$ .  
 $\text{CutCake}(N, C, \mathcal{C}_N)$ ;

Now we will give a proof of envy-freeness and truthfulness of Modified Mechanism of Asano and Umeda (Algorithm 2) described in Theorem 2, which is almost the same as given in [13, 25].

Let  $T$  be the number of recursive calls  $\text{CutCake}(\cdot, \cdot, \cdot)$  in Modified Mechanism of Asano and Umeda (Algorithm 2). Note that, although we use  $D^{(S)}$ ,  $D_k^{(S)} \in \mathcal{D}_{P'}^{(S)}$ ,  $\mathcal{D}_{P'}^{(S)}$  in  $\text{CutCake}(P', D^{(S)}, \mathcal{D}_{P'}^{(S)})$  which are obtained from  $D'$ ,  $D'_k \in \mathcal{D}'_{P'}$ ,  $\mathcal{D}'_{P'}$  by virtual shrinking of all  $H_1, \dots, H_L$ , we will not distinguish them from now on, since we just performed *virtual*

■ **Procedure** CutCake( $P, D, \mathcal{D}_P$ ).

---

**Input:** Cake  $D$  which can be considered to be a single interval, players  $P$ , and solid valuation intervals  $\mathcal{D}_P = (D_i : i \in P)$  with valuation interval  $D_i = [\alpha'_i, \beta'_i] \subseteq D$  of each player  $i \in P$  and  $\bigcup_{D_i \in \mathcal{D}_P} D_i = D$  (the density of each interval  $X$  of  $D$  is denoted by  $\rho(X)$ ).

**Output:** Allocation  $A_P = (A_i : i \in P)$  with  $A_i \subseteq D_i$  for  $i \in P$  and  $\sum_{i \in P} A_i = D$ . Find all the maximal intervals of minimum density  $\rho_{\min}$  in the cake-cutting problem with cake  $D$ , players  $P$  and solid valuation intervals  $\mathcal{D}_P$ ;

Let  $H_1 = [h'_1, h''_1], \dots, H_L = [h'_L, h''_L]$  be all the maximal intervals of minimum density  $\rho_{\min}$ ;

**for**  $\ell = 1$  **to**  $L$  **do**

cut cake  $D$  at both endpoints  $h'_\ell, h''_\ell$  of  $H_\ell$ ;

$R_\ell = \{k \in P \mid D_k \subseteq H_\ell, D_k \in \mathcal{D}_P\}$ ;  $\mathcal{D}_{R_\ell} = (D_k \in \mathcal{D}_P : k \in R_\ell)$ ;

CutMaxInterval( $R_\ell, H_\ell, \mathcal{D}_{R_\ell}$ );

$P' = P$ ;  $D' = D$ ;

**for**  $\ell = 1$  **to**  $L$  **do**  $P' = P' \setminus R_\ell$ ;  $D' = D' \setminus H_\ell$ ;

//  $P' = P \setminus \sum_{\ell=1}^L R_\ell$  and  $D' = D \setminus \sum_{\ell=1}^L H_\ell$

**if**  $P' \neq \emptyset$  **then**

$\mathcal{D}'_{P'} = \emptyset$ ;

**for** each  $D_k \in \mathcal{D}_P$  with  $k \in P'$  **do**  $D'_k = D_k \setminus \sum_{\ell=1}^L H_\ell$ ;  $\mathcal{D}'_{P'} = \mathcal{D}'_{P'} + \{D'_k\}$ ;

Let  $D^{(S)}, D_k^{(S)} \in \mathcal{D}'_{P'}, \mathcal{D}_{P'}^{(S)}$  be obtained from  $D', D'_k \in \mathcal{D}'_{P'}, \mathcal{D}'_{P'}$  by virtual shrinking of all  $H_1, \dots, H_L$ ;

CutCake( $P', D^{(S)}, \mathcal{D}_{P'}^{(S)}$ );

---

*shrinking*. Thus, we consider  $D^{(S)} = D', (D_k^{(S)} \in \mathcal{D}_{P'}^{(S)}) = (D'_k \in \mathcal{D}'_{P'})$  and  $\mathcal{D}_{P'}^{(S)} = \mathcal{D}'_{P'}$ . We denote by CutCake( $P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}}$ ) the  $t$ -th recursive call of CutCake( $\cdot, \cdot, \cdot$ ). Note that the first call of CutCake( $\cdot, \cdot, \cdot$ ) is CutCake( $N, C, \mathcal{C}_N$ ). Let  $\rho_{\min}^{(t)}$  be the minimum density of the cake cutting problem with cake  $D^{(t)}$ , players  $P^{(t)}$  and the solid valuation intervals  $\mathcal{D}_{P^{(t)}}$ . Clearly,  $C = D^{(1)} \supset D^{(2)} \supset \dots \supset D^{(T)}$  and  $N = P^{(1)} \supset P^{(2)} \supset \dots \supset P^{(T)}$ . Furthermore, as shown in [4, 5], the inequality

$$\rho_{\min}^{(1)} < \rho_{\min}^{(2)} < \dots < \rho_{\min}^{(T)}$$

holds. We denote by CutMaxInterval( $R_\ell^{(t)}, H_\ell^{(t)}, \mathcal{D}_{R_\ell^{(t)}}$ ) the CutMaxInterval( $\cdot, \cdot, \cdot$ ) called in CutCake( $P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}}$ ), where  $H_\ell^{(t)}$  is a maximal interval of minimum density  $\rho_{\min}^{(t)}$  in cake  $D^{(t)}$ , players  $P^{(t)}$  and solid valuation intervals  $\mathcal{D}_{P^{(t)}}$  by virtually shrinking of hollow pieces in  $C \setminus D^{(t)}$ . For each player  $i \in N$ , there is  $t \in \{1, \dots, T\}$  such that allocation  $A_i \subseteq C_i$  to player  $i$  is determined in CutMaxInterval( $R_\ell^{(t)}, H_\ell^{(t)}, \mathcal{D}_{R_\ell^{(t)}}$ ). Thus, it is clear that Modified Mechanism of Asano and Umeda (Algorithm 2) finds an allocation  $A_N = (A_i : i \in N)$  of cake  $C$  to  $n$  players  $N$  with  $A_i \subseteq C_i$  for each player  $i \in N$  and  $\sum_{i \in N} A_i = C$ .

Envy-freeness can be proved as follows (which is almost the same as given in [13, 25]). Let  $A_i \subseteq C_i$  to player  $i$  be determined in CutMaxInterval( $R_\ell^{(t)}, H_\ell^{(t)}, \mathcal{D}_{R_\ell^{(t)}}$ ) called in the  $t$ -th recursive call CutCake( $P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}}$ ). Thus,  $V_i(A_i) = \text{len}(A_i) = \rho_{\min}^{(t)}$ . Similarly, let  $A_j \subseteq C_j$  to player  $j$  be determined in CutMaxInterval( $R_{\ell'}^{(t')}, H_{\ell'}^{(t')}, \mathcal{D}_{R_{\ell'}^{(t)'}}$ ) called in the  $t'$ -th recursive call CutCake( $P^{(t')}, D^{(t')}, \mathcal{D}_{P^{(t)'}}$ ). If  $t' \leq t$  then  $V_i(A_j) = \text{len}(A_j \cap C_i) \leq \text{len}(A_j) = \rho_{\min}^{(t')} \leq \rho_{\min}^{(t)} = \text{len}(A_i) = V_i(A_i)$ . Otherwise (i.e., if  $t' > t$ ), although  $V_j(A_j) = \text{len}(A_j) = \rho_{\min}^{(t')} > \rho_{\min}^{(t)} = \text{len}(A_i) = V_i(A_i)$ , we have  $A_j \cap C_i = \emptyset$  and  $V_i(A_j) = \text{len}(A_j \cap C_i) = 0 \leq \rho_{\min}^{(t)} = \text{len}(A_i) = V_i(A_i)$ . Thus, envy-freeness is clear.

Truthfulness can be proved as follows. Assume that only player  $i \in N$  gives a false valuation interval  $C'_i$  and let  $\mathcal{C}'_N(i) = (C'_j : j \in N)$  be an input to Modified Mechanism of Asano and Umeda (Algorithm 2) and  $A'_N(i) = (A'_j : j \in N)$  be the obtained allocation of cake  $C$  to  $n$  players  $N$  (in the argument below, we assume that  $\mathcal{C}'_N(i) = (C'_j : j \in N)$  is solid, although this restriction can be removed by a little more complicated argument). Let  $A_i \subseteq C_i$  to player  $i$  be determined in  $\text{CutMaxInterval}(R_\ell^{(t)}, H_\ell^{(t)}, \mathcal{D}_{R_\ell^{(t)}})$  called in the  $t$ -th recursive call  $\text{CutCake}(P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}})$ . Similarly, let  $A'_i \subseteq C'_i$  with  $A'_i = \sum_{\ell'=1}^{k'_i} A'_{i,\ell'}$  to player  $i$  be determined in  $\text{CutMaxInterval}(R_{\ell'}^{(t')}, H_{\ell'}^{(t')}, \mathcal{D}'_{R_{\ell'}^{(t')}})$  called in the  $t'$ -th recursive call  $\text{CutCake}(P^{(t')}, D^{(t')}, \mathcal{D}'_{P^{(t')}})$ . Thus,  $V_i(A_i) = \text{len}(A_i) = \rho_{\min}^{(t)}$  and  $V_i(A'_i) = \sum_{\ell'=1}^{k'_i} \text{len}(A'_{i,\ell'} \cap C_i) \leq \sum_{\ell'=1}^{k'_i} \text{len}(A'_{i,\ell'}) = \rho'_{\min}^{(t')}$ , where  $\rho'_{\min}^{(t')}$  is the minimum density of the intervals in cake  $D^{(t')}$  with  $P^{(t')}$  and  $\mathcal{D}'_{P^{(t')}})$ . We divide the case into two cases: (i)  $\rho'_{\min}^{(t')} \leq \rho_{\min}^{(t)}$ ; and (ii)  $\rho'_{\min}^{(t')} > \rho_{\min}^{(t)}$ .

(i)  $\rho'_{\min}^{(t')} \leq \rho_{\min}^{(t)}$ . In this case, it is clear that  $V_i(A_i) = \text{len}(A_i) = \rho_{\min}^{(t)} \geq \rho'_{\min}^{(t')} = \sum_{\ell'=1}^{k'_i} \text{len}(A'_{i,\ell'}) \geq \sum_{\ell'=1}^{k'_i} \text{len}(A'_{i,\ell'} \cap C_i) = V_i(A'_i)$ .

(ii)  $\rho'_{\min}^{(t')} > \rho_{\min}^{(t)}$ . In this case,  $t' \geq t$  holds, which can be shown as follows.

Suppose contrarily that  $t' < t$ . Then for two inputs  $\mathcal{C}_N = (C_j : j \in N)$  and  $\mathcal{C}'_N(i) = (C'_j : j \in N)$ , Modified Mechanism of Asano and Umeda (Algorithm 2) makes the same behavior before the  $t'$ -th recursive calls  $\text{CutCake}(P^{(t')}, D^{(t')}, \mathcal{D}_{P^{(t')}})$  and  $\text{CutCake}(P^{(t')}, D^{(t')}, \mathcal{D}'_{P^{(t')}})$ . Thus,  $P^{(t'')} = P^{(t'')}$ ,  $D^{(t'')} = D^{(t'')}$  and  $\mathcal{D}'_{P^{(t'')}} \setminus \{D_i^{(t'')}\} = \mathcal{D}_{P^{(t'')}} \setminus \{D_i^{(t'')}\}$  for each  $t'' = 1, \dots, t'$ , where  $D_i^{(t'')} = C_i \cap D^{(t'')}$  and  $D'_i{}^{(t'')} = C'_i \cap D^{(t'')}$ . Let  $X = [x', x'']$  be a maximal interval of minimum density  $\rho_{\min}^{(t')}$  in  $D^{(t')} = D^{(t')}$ . Thus,  $\rho^{(t')}(X) = \rho_{\min}^{(t')}$ . Furthermore,  $D_i^{(t')} = C_i \cap D^{(t')} \not\subseteq X$ , since otherwise (i.e., if  $D_i^{(t')} \subseteq X$ )  $A_i$  to player  $i$  would be determined in the  $t'$ -th recursive call  $\text{CutCake}(P^{(t')}, D^{(t')}, \mathcal{D}_{P^{(t')}})$ , which is a contradiction (since  $t > t'$  and  $A_i$  is determined in the  $t$ -th call  $\text{CutCake}(P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}})$ ). Thus, if  $D'_i{}^{(t')} = C'_i \cap D^{(t')} \not\subseteq X$  then  $\rho^{(t')}(X) = \rho^{(t')}(X)$  holds by  $D_i^{(t')} \not\subseteq X$ , and otherwise (i.e., if  $D'_i{}^{(t')} \subseteq X$ )  $\rho^{(t')}(X) < \rho^{(t')}(X)$  holds by  $D_i^{(t')} \not\subseteq X$ . This implies that  $\rho^{(t')}(X) \leq \rho^{(t')}(X)$  and  $\rho'_{\min}^{(t')} \leq \rho^{(t')}(X) \leq \rho^{(t')}(X) = \rho_{\min}^{(t')} < \rho'_{\min}^{(t')}$  by  $t' < t$ . However, this is a contradiction, since  $\rho'_{\min}^{(t')} > \rho_{\min}^{(t)}$  in this case.

Thus, we have  $t' \geq t$  in this case of  $\rho'_{\min}^{(t')} > \rho_{\min}^{(t)}$ . As we mentioned above, for two inputs  $\mathcal{C}_N = (C_j : j \in N)$  and  $\mathcal{C}'_N(i) = (C'_j : j \in N)$ , Modified Mechanism of Asano and Umeda (Algorithm 2) makes the same behavior before the  $t$ -th recursive calls  $\text{CutCake}(P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}})$  and  $\text{CutCake}(P^{(t)}, D^{(t)}, \mathcal{D}'_{P^{(t)}})$ . Thus,  $P^{(t)} = P^{(t)}$ ,  $D^{(t)} = D^{(t)}$  and  $\mathcal{D}'_{P^{(t)}} \setminus \{D_i^{(t)}\} = \mathcal{D}_{P^{(t)}} \setminus \{D_i^{(t)}\}$ , where  $D_i^{(t)} = C_i \cap D^{(t)}$  and  $D'_i{}^{(t)} = C'_i \cap D^{(t)}$ . For each player  $j \in N$  with  $A_j$  determined in  $\text{CutMaxInterval}(R_\ell^{(t)}, H_\ell^{(t)}, \mathcal{D}_{R_\ell^{(t)}})$  in the  $t$ -th call  $\text{CutCake}(P^{(t)}, D^{(t)}, \mathcal{D}_{P^{(t)}})$ , let  $A'_j$  be determined in the  $t'_j$ -th call  $\text{CutCake}(P^{(t'_j)}, D^{(t'_j)}, \mathcal{D}'_{P^{(t'_j)}})$ . Thus  $t'_j \geq t$ .

We will show that  $\rho'_{\min}^{(t)} \geq \rho_{\min}^{(t)}$ . If  $t' = t$  then this is true since  $\rho'_{\min}^{(t)} = \rho'_{\min}^{(t')} > \rho_{\min}^{(t)}$ . Now we assume  $t' > t$ . Let  $X = [x', x'']$  be a maximal interval of minimum density  $\rho_{\min}^{(t)}$  in  $D^{(t)} = D^{(t)}$ . Thus,  $\rho^{(t)}(X) = \rho_{\min}^{(t)}$ . Furthermore,  $D'_i{}^{(t)} = C'_i \cap D^{(t)} \not\subseteq X$  holds, since otherwise (i.e., if  $D'_i{}^{(t)} \subseteq X$ )  $A'_i$  to player  $i$  would be determined in the  $t$ -th recursive call  $\text{CutCake}(P^{(t)}, D^{(t)}, \mathcal{D}'_{P^{(t)}})$  and  $t' = t$  (which contradicts  $t' > t$ ). Thus, if  $D_i^{(t)} = C_i \cap D^{(t)} \not\subseteq X$ , then  $\rho^{(t)}(X) = \rho^{(t)}(X)$  by  $D_i^{(t)} \not\subseteq X$  and  $\rho'_{\min}^{(t)} = \rho^{(t)}(X) = \rho^{(t)}(X) \geq \rho_{\min}^{(t)}$ . If  $D'_i{}^{(t)} \subseteq X$ , then  $\rho^{(t)}(X) > \rho^{(t)}(X)$  by  $D_i^{(t)} \not\subseteq X$  and  $\rho'_{\min}^{(t)} = \rho^{(t)}(X) > \rho^{(t)}(X) \geq \rho_{\min}^{(t)}$ . By the argument above, we have  $\rho'_{\min}^{(t)} \geq \rho_{\min}^{(t)}$ .

Thus, for each  $j \in R_\ell^{(t)} \setminus \{i\}$ , we have  $\text{len}(A'_j) = \rho_{\min}^{(t'_j)} \geq \rho_{\min}^{(t)} \geq \rho_{\min}^{(t)}$  and  $A'_j \subseteq C_j \cap D^{(t)} = C_j \cap D^{(t)} \subseteq H_\ell^{(t)}$ . By  $\sum_{j \in N} A'_j = C$  and  $\sum_{j \in R_\ell^{(t)}} A'_j = (\cup_{j \in R_\ell^{(t)}} C_j) \cap D^{(t)} = H_\ell^{(t)}$ , we have

$$\begin{aligned} V_i(A'_i) &= \text{len}(A'_i \cap C_i) = \text{len}(A'_i \cap C_i \cap D^{(t)}) \\ &\leq \text{len}(H_\ell^{(t)}) - \sum_{j \in R_\ell^{(t)} \setminus \{i\}} \text{len}(A'_j \cap C_j \cap D^{(t)}) = \text{len}(H_\ell^{(t)}) - \sum_{j \in R_\ell^{(t)} \setminus \{i\}} \text{len}(A'_j) \\ &\leq \text{len}(H_\ell^{(t)}) - \rho_{\min}^{(t)} (|R_\ell^{(t)}| - 1) = |R_\ell^{(t)}| \rho_{\min}^{(t)} - \rho_{\min}^{(t)} (|R_\ell^{(t)}| - 1) \\ &= \rho_{\min}^{(t)} = V_i(A_i). \end{aligned}$$

Thus, truthfulness of Modified Mechanism of Asano and Umeda (Algorithm 2) is proved.

## 6 Second Mechanism $\mathcal{M}_2$

In this section, we give the second version  $\mathcal{M}_2$  which can be applied to the envy-free and truthful mechanism proposed by Chen, et al. [13] where the valuation function of each player is more general and piecewise uniform. We are given a cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$ , and solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with valuation interval  $C_i = [\alpha_i, \beta_i) \subseteq C$  of each player  $i \in N$  as before. We are also given  $(s_i : i \in N)$  such that there is an allocation  $A'_N = (A'_i : i \in N)$  to players  $N$  with  $A'_i \subseteq C_i$  and  $s_i = \text{len}(A'_i) > 0$  for each  $i \in N$  and  $\sum_{i \in N} A'_i = C$  (thus  $\sum_{i \in N} s_i = 1$ ). Note that there is no need to have such an allocation  $A'_N = (A'_i : i \in N)$  in hand.

Then  $\mathcal{M}_2$  is almost the same as  $\mathcal{M}_1$  and can be written as follows.

### Algorithm 3 Second Mechanism $\mathcal{M}_2$ .

---

**Input:** Cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$  and solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  with valuation interval  $C_i = [\alpha_i, \beta_i)$  of each player  $i \in N$  and  $\cup_{C_i \in \mathcal{C}_N} C_i = C$  and  $(s_i : i \in N)$  for players  $N$  such that there is an allocation  $A'_N = (A'_i : i \in N)$  to players  $N$  with  $A'_i \subseteq C_i$  and  $\text{len}(A'_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A'_i = C$  (thus  $\sum_{i \in N} s_i = 1$ ).

**Output:** Allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A_i = C$ .

sort  $\mathcal{C}_N = (C_i : i \in N)$  in a lexicographic order with respect to  $(\beta_i, \alpha_i)$  and assume  $C_1 \leq C_2 \leq \dots \leq C_n$  in this lexicographic order;

set  $A_1 = [a_1, b_1) \subseteq C_1$  with length  $s_1$  such that  $a_1 = \alpha_1$  and  $b_1 = \alpha_1 + s_1$ ;

**for**  $i = 2$  **to**  $n$  **do**

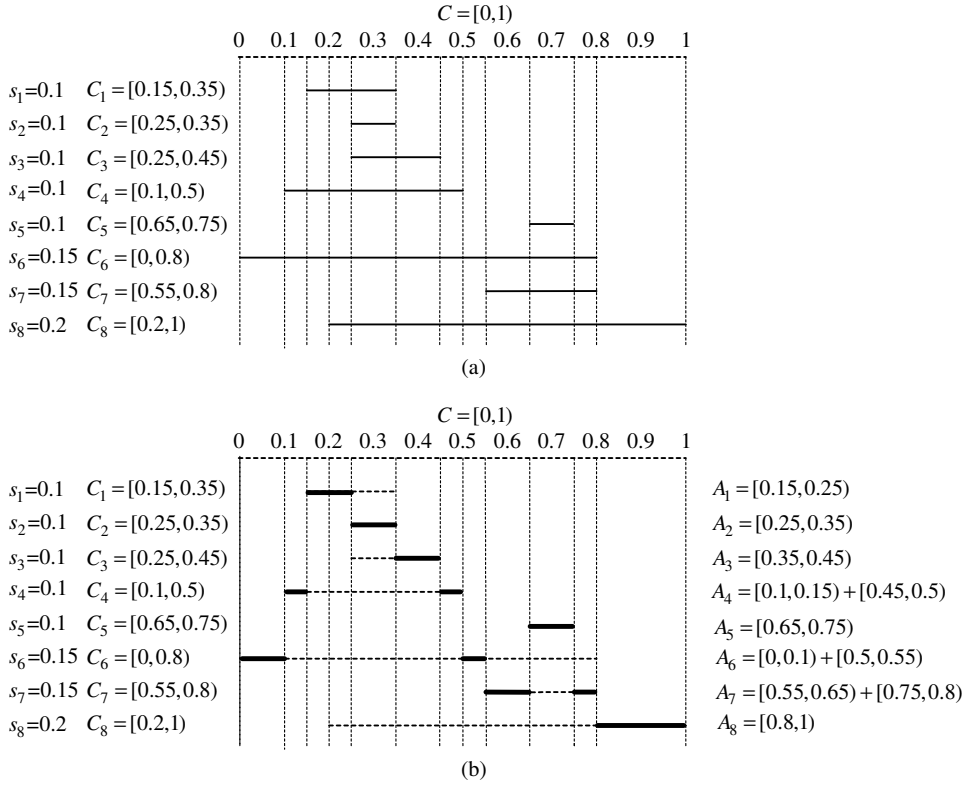
set  $A_i = [a_i, b_i) \setminus \sum_{\ell=1}^{i-1} A_\ell$  with length  $s_i$  such that  $[a_i, b_i) \subseteq C_i$  and  $a_i$  is the leftmost endpoint in  $C_i \setminus \sum_{\ell=1}^{i-1} A_\ell$ ;

---

Figure 4 shows an example of solid valuation intervals  $\mathcal{C}_N = (C_i : i \in N)$  and  $(s_i : i \in N)$  with  $\sum_{i \in N} s_i = 1$  and an allocation  $A_N = (A_i : i \in N)$  obtained by  $\mathcal{M}_2$ . By an argument similar to one in Proof of Theorem 1 we have the following theorem.

► **Theorem 3.**  $\mathcal{M}_2$  correctly finds an allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A_i = C$  in  $O(n \log n)$  time. Furthermore, the number of cuts made by  $\mathcal{M}_2$  on cake  $C$  is at most  $2n - 2$ .

**Proof.** Since the time complexity  $O(n \log n)$  and the number of cuts at most  $2n - 2$  can be obtained by the same argument as in Proof of Theorem 1, we only prove that  $\mathcal{M}_2$  correctly finds an allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$ ,  $\text{len}(A_i) = s_i$  and  $\sum_{i \in N} A_i = C$ .



■ **Figure 4** (a) Example of  $\mathcal{C}_N = (C_i : i \in N)$  and  $(s_i : i \in N)$  with  $\sum_{i \in N} s_i = 1$ . (b) Allocation  $A_N = (A_i : i \in N)$  obtained by  $\mathcal{M}_2$ .

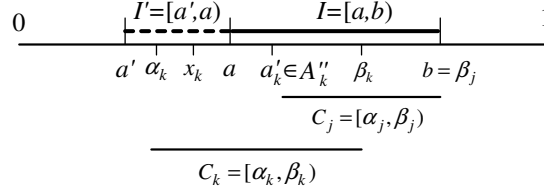
Suppose contrarily that  $\mathcal{M}_2$  could not set  $A_i \subseteq C_i$  with length  $s_i$  for some  $i \in N$ . Let  $j$  be the minimum among such  $i$ s and let  $J = \{1, 2, \dots, j\}$ . Of course,  $j > 1$ , since we assumed that there is an allocation  $A'_N = (A'_i : i \in N)$  to players  $N$  with  $A'_i \subseteq C_i$  and  $\text{len}(A'_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A'_i = C$  (thus  $C_1 = [\alpha_1, \beta_1]$  is of length at least  $s_1$  and  $A_1 = [a_1, b_1] = [\alpha_1, \alpha_1 + s_1] \subseteq C_1$ ). Now we consider valuation intervals  $\mathcal{C}_J = (C_i : i \in J)$ . Note that each  $C_i = [\alpha_i, \beta_i] \in \mathcal{C}_J$  satisfies  $\beta_i \leq \beta_j$ , since  $\mathcal{C}_N = (C_i : i \in N)$  was sorted in the lexicographic order with respect to  $(\beta_i, \alpha_i)$ . Thus,  $\mathcal{M}_2$  could set  $A_i = [a_i, b_i] \setminus \sum_{\ell=1}^{i-1} A_\ell \subseteq C_i = [\alpha_i, \beta_i]$  with length  $s_i$  for each  $i \in J \setminus \{j\}$  but could not set  $A_j = [a_j, b_j] \setminus \sum_{\ell=1}^{j-1} A_\ell \subseteq C_j = [\alpha_j, \beta_j]$  with length  $s_j$ . This implies that  $C_j \setminus \sum_{\ell=1}^{j-1} A_\ell$  is of length  $s'_j < s_j$ . Let

$$A''_i = A_i \quad (i \in J \setminus \{j\}), \quad A''_j = C_j \setminus \sum_{\ell=1}^{j-1} A''_\ell = [a_j, \beta_j] \setminus \sum_{\ell=1}^{j-1} A_\ell.$$

Thus,  $\sum_{i \in J} A''_i$  of allocation  $(A''_i : i \in J)$  consists of several maximal contiguous intervals. Let  $I = [a, b)$  be the rightmost maximal contiguous interval among the maximal contiguous intervals in  $\sum_{i \in J} A''_i$  (Figure 5). Thus,  $b = \beta_j$ . Define  $K \subseteq J$  by

$$K = \{j\} \cup \{i \in J \mid A''_i \cap I \neq \emptyset\}.$$

Now we consider valuation intervals  $\mathcal{C}_K = (C_i : i \in K)$ . Then each  $C_i \in \mathcal{C}_K$  is contained in  $I$ , which can be obtained as follows.



■ **Figure 5** Illustration of  $I = [a, b)$  and  $I' = [a', a)$ .

Of course,  $C_j = [\alpha_j, \beta_j)$  is contained in  $I$ . Actually, since  $C_j \setminus \sum_{\ell=1}^{j-1} A''_\ell = [a_j, \beta_j) \setminus \sum_{\ell=1}^{j-1} A_\ell$  is of length  $s'_j < s_j$  and  $A''_j = C_j \setminus \sum_{\ell=1}^{j-1} A''_\ell = [a_j, \beta_j) \setminus \sum_{\ell=1}^{j-1} A_\ell$ , we have: if  $A''_j = \emptyset$  then  $C_j \subseteq \sum_{\ell=1}^{j-1} A''_\ell$  and a single contiguous interval  $C_j$  is contained in the rightmost maximal contiguous interval  $I$  in  $\sum_{\ell=1}^j A''_\ell = \sum_{\ell=1}^{j-1} A''_\ell$  (i.e.,  $C_j \subseteq I$ ); and otherwise (i.e., if  $A''_j \neq \emptyset$ ),  $C_j \subseteq A''_j \cup \sum_{\ell=1}^{j-1} A''_\ell = \sum_{\ell=1}^j A''_\ell$  and a single contiguous interval  $C_j$  is contained in the rightmost maximal contiguous interval  $I$  in  $\sum_{\ell=1}^j A''_\ell$ .

Now suppose that there were  $i \in K \setminus \{j\}$  such that  $C_i \in \mathcal{C}_K$  is not contained in  $I$ . Thus,  $I = [a, b)$  is a proper subinterval of  $[0, b) = [0, \beta_j)$  (i.e.,  $a > 0$ ) and  $C_i = [\alpha_i, \beta_i) \in \mathcal{C}_K \setminus \{C_j\}$  contains a point  $x$  in  $[0, b) \setminus I = [0, a)$ . Let  $k \in K \setminus \{j\}$  be the minimum among such  $i$ s and let  $x_k$  be a point of  $C_k = [\alpha_k, \beta_k) \in \mathcal{C}_K$  contained in  $[0, a) = [0, b) \setminus I$ . Note that  $C_k \cap I \supseteq A''_k \cap I \neq \emptyset$  since  $k \in K \setminus \{j\} \subseteq J \setminus \{j\}$ . Thus,  $\beta_k \leq \beta_j$  and  $\alpha_k \leq x_k < a \leq a'_k < \beta_k$  for some  $a'_k \in A''_k \cap I \neq \emptyset$ . Furthermore, since we chose  $I = [a, b) \neq [0, b)$  as the rightmost maximal contiguous interval among the maximal contiguous intervals in  $\sum_{i \in J} A''_i$ , we have  $\sum_{i \in J} A''_i \neq [0, b) = [0, \beta_j)$ . Let  $I' = [a', a)$  be the rightmost maximal contiguous interval in  $[0, b) \setminus \sum_{i \in J} A''_i$  (Figure 5). Since  $C_k = [\alpha_k, \beta_k)$  is a contiguous interval and satisfies  $\alpha_k \leq x_k < a \leq a'_k < \beta_k$ , we can choose  $x_k$  with  $x_k \in I' \cap C_k \neq \emptyset$ . Thus,  $x_k \notin A''_k$  by  $I' \cap A''_k \subseteq I' \cap \sum_{i \in J} A''_i = \emptyset$ . Then, however,  $\mathcal{M}_2$  would have tried to include  $x_k$  into  $A''_k$  rather than  $a'_k \in A''_k \cap I \neq \emptyset$ , because  $\mathcal{M}_2$  sets  $A''_k = A_k = [a_k, b_k) \setminus \sum_{\ell=1}^{k-1} A''_\ell \subseteq C_k$  with length  $s_k$  such that  $a_k$  is the leftmost endpoint in  $C_k \setminus \sum_{\ell=1}^{k-1} A''_\ell$ . This is a contradiction.

Thus, each  $C_i \in \mathcal{C}_K$  is contained in  $I$  and  $\bigcup_{i \in K} C_i \subseteq I$ . By the argument above, we have

$$\bigcup_{i \in K} C_i = I = \sum_{i \in K} A''_i,$$

since  $A''_h \cap I = \emptyset$  for  $h \in J \setminus K$  and  $I = \sum_{i \in J} A''_i \cap I = \sum_{i \in K} A''_i \cap I \subseteq \sum_{i \in K} A''_i \subseteq \sum_{i \in K} C_i$  by the definitions of  $I$  and  $K$  and  $A''_i \subseteq C_i$  for each  $i \in K$ . Thus,

$$\sum_{i \in K} \text{len}(A''_i) = s'_j + \sum_{i \in K \setminus \{j\}} s_i = \text{len}(I) = b - a < s_j + \sum_{i \in K \setminus \{j\}} s_i$$

since  $s'_j < s_j$ . However, this is a contradiction, since we assumed that there is an allocation  $A'_N = (A'_i : i \in N)$  to players  $N$  with  $A'_i \subseteq C_i$  and  $\text{len}(A'_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A'_i = C$ , and thus

$$s_j + \sum_{i \in K \setminus \{j\}} s_i = \sum_{i \in K} \text{len}(A'_i) \leq \text{len}\left(\bigcup_{i \in K} C_i\right) = \text{len}(I) = b - a.$$

Thus,  $\mathcal{M}_2$  correctly finds an allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$ ,  $\text{len}(A_i) = s_i$  and  $\sum_{i \in N} A_i = C$ . ◀



## 7 Application to Mechanism of Chen et al. [13]

By Theorem 3, in order to obtain an envy-free and truthful allocation  $A_N = (A_i : i \in N)$  with  $A_i \subseteq C_i$  and  $\text{len}(A_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A_i = C$ , we only need  $(s_i : i \in N)$  such that there is an envy-free and truthful allocation  $A'_N = (A'_i : i \in N)$  to players  $N$  with  $A'_i \subseteq C_i$  and  $\text{len}(A'_i) = s_i$  for each  $i \in N$  and  $\sum_{i \in N} A'_i = C$ . Thus, Theorem 3 can be applied to the mechanism of Chen, et al. [13] where the valuation function  $v_i$  of each player  $i \in N$  is more general and piecewise uniform: Given a cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$  and solid piecewise uniform valuation functions  $(v_i : i \in N)$  such that  $D(v_i) = \{x \in C \mid v_i(x) > 0\}$  of each valuation function  $v_i$  consists of  $m_i \geq 1$  maximal contiguous intervals  $C_{i_1}, \dots, C_{i_{m_i}}$  in  $C$  and  $\bigcup_{i \in N} D(v_i) = C$ .

The mechanism of Chen, et al. [13] finds an envy-free and truthful allocation  $A'_N = (A'_i : i \in N)$  such that  $\sum_{i \in N} A'_i = C$  and  $A'_i = \sum_{j=1}^{m_i} A'_{i_j}$  with  $A'_{i_j} \subseteq C_{i_j}$  for each  $i \in N$  and for each  $j = 1, 2, \dots, m_i$ . Thus, we can set  $s_{i_j} = \text{len}(A'_{i_j})$  and apply Theorem 3 to obtain an envy-free and truthful allocation  $A_N = (A_i : i \in N)$  such that  $A_i = \sum_{j=1}^{m_i} A_{i_j}$  for each  $i \in N$  with  $A_{i_j} \subseteq C_{i_j}$  and  $\text{len}(A_{i_j}) = s_{i_j}$  for each  $j = 1, 2, \dots, m_i$  with at most  $2(\sum_{i \in N} m_i) - 2$  cuts. Note that, we can delete all  $C_{i_j}$  if  $s_{i_j} = \text{len}(A'_{i_j}) = 0$ , and thus, we can assume  $s_{i_j} = \text{len}(A'_{i_j}) > 0$  for each  $i \in N$  and for each  $j = 1, 2, \dots, m_i$ .

In summary, we have the following corollary.

► **Corollary 4.** *Suppose that we are given  $(s_{i_j} : i \in N, j = 1, 2, \dots, m_i)$  such that there is an envy-free and truthful allocation  $A'_N = (A'_i : i \in N)$  to players  $N$  satisfying  $\sum_{i \in N} A'_i = C$  and  $A'_i = \sum_{j=1}^{m_i} A'_{i_j}$  with  $A'_{i_j} \subseteq C_{i_j}$  and  $\text{len}(A'_{i_j}) = s_{i_j} > 0$  for each  $i \in N$  and for each  $j = 1, 2, \dots, m_i$  for the cake-cutting problem with cake  $C = [0, 1)$ ,  $n$  players  $N = \{1, 2, \dots, n\}$  and solid piecewise uniform valuation functions  $(v_i : i \in N)$  such that  $D(v_i) = \{x \in C \mid v_i(x) > 0\}$  of each piecewise uniform valuation function  $v_i$  consists of  $m_i \geq 1$  maximal contiguous intervals  $C_{i_1}, \dots, C_{i_{m_i}}$  in  $C$  and  $\bigcup_{i \in N} D(v_i) = C$  (such  $A'_N = (A'_i : i \in N)$  to players  $N$  can be obtained, for example, by Mechanism of Chen, et al. [13]). Then, Second Mechanism  $\mathcal{M}_2$  (Algorithm 3) correctly finds an envy-free and truthful allocation  $A_N = (A_i : i \in N)$  with  $\sum_{i \in N} A_i = C$  and  $A_i = \sum_{j=1}^{m_i} A_{i_j}$  with  $A_{i_j} \subseteq C_{i_j}$  and  $\text{len}(A_{i_j}) = s_{i_j}$  for each  $i \in N$  and for each  $j = 1, 2, \dots, m_i$  in  $O(\sum_{i \in N} m_i \log \sum_{i \in N} m_i)$  time. Furthermore, the number of cuts made by  $\mathcal{M}_2$  on cake  $C$  is at most  $2(\sum_{i \in N} m_i) - 2$ . Thus, Mechanism of Chen, et al. [13] can be implemented to make at most  $2(\sum_{i \in N} m_i) - 2$  cuts on cake  $C$ .*

## 8 Concluding Remarks

We gave a much simpler envy-free and truthful mechanism with a small number of cuts for the cake-cutting problem posed in [2, 25]. Furthermore, we showed that this approach can be applied to the envy-free and truthful mechanism proposed by Chen, et al. [13] for the more general cake-cutting problem where the valuation function of each player is piecewise uniform. Thus, we can make their envy-free and truthful mechanism use  $2 \sum_{i \in N} m_i - 2$  cuts and settle the problem posed by [2, 25], where  $m_i$  is the number of maximal contiguous intervals in  $D(v_i) = \{x \in C \mid v_i(x) > 0\}$  of each player  $i$ 's piecewise uniform valuation  $v_i$ .

If we require the piecewise uniform valuation  $v_i$  of each player  $i$  to be a single contiguous interval  $C_i$  in cake  $C$ , then Modified Mechanism of Asano and Umeda can be implemented to run in  $O(n^2 \log n)$  time based on parametric flows on the network arising from valuation intervals  $C_i$  [3] (Parametric flows and parametric searching have been studied by many researchers [1, 17, 33]). We expect this would lead to a faster envy-free and truthful mechanism for the general piecewise uniform valuations.



## References

- 1 Hassene Aissi, S. Thomas McCormick, and Maurice Queyranne. Faster algorithms for next breakpoint and max value for parametric global minimum cuts. In *21st International Conference on Integer Programming and Combinatorial Optimization*, pages 27–39, 2020.
- 2 Reza Alijani, Majid Farhadi, Mohammad Ghodsi, Masoud Seddighin, and Ahmad S. Tajik. Envy-free mechanisms with minimum number of cuts. In *31st AAAI Conference on Artificial Intelligence, 2017*, pages 312–318, 2017.
- 3 Takao Asano. Envy-free and truthful cake-cuttings based on parametric flows. In *RIMS Kôkyûroku 2182, Kyoto University, April, 2021*, pages 1–39, 2021.
- 4 Takao Asano and Hiroyuki Umeda. Cake cutting: An envy-free and truthful mechanism with a small number of cuts. In *31st International Symposium on Algorithms and Computation*, pages 15.1–15.16, 2020.
- 5 Takao Asano and Hiroyuki Umeda. An envy-free and truthful mechanism for the cake-cutting problem. In *RIMS Kôkyûroku 2154, Kyoto University, April, 2020*, pages 54–91, 2020.
- 6 Haris Aziz and Simon Mackenzie. A discrete and bounded envy-free cake cutting protocol for any number of agents. In *57th Annual Symposium on Foundations of Computer Science, 2016*, pages 416–427, 2016.
- 7 Haris Aziz and Chun Ye. Cake cutting algorithms for piecewise constant and piecewise uniform valuations. In *10th International Conference on Web and Internet Economics, 2014*, pages 1–14, 2014.
- 8 Xiaohui Bei, Ning Chen, Xia Hua, Biaoshuai Tao, and Endong Yang. Optimal proportional cake cutting with connected pieces. In *26th AAAI Conference on Artificial Intelligence, 2012*, pages 1263–1269, 2012.
- 9 Xiaohui Bei, Ning Chen, Guangda Huzhang, Biaoshuai Tao, and Jiajun Wu. Cake cutting: envy and truth. In *26th International Joint Conference on Artificial Intelligence*, pages 3625–3631, 2017.
- 10 Steven J. Brams, Michal Feldman, John K. Lai, Jamie Morgenstern, and Ariel D. Procaccia. On maxsum fair cake divisions. In *26th AAAI Conference on Artificial Intelligence, 2012*, pages 1285–1291, 2012.
- 11 Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake Cutting to Dispute Resolution*. Cambridge University Press, 1996.
- 12 Ioannis Caragiannis, John K. Lai, and Ariel D. Procaccia. Towards more expressive cake cutting. In *22nd International Joint Conference on Artificial Intelligence, 2011*, pages 127–132, 2011.
- 13 Yiling Chen, John K. Lai, David C. Parkes, and Ariel D. Procaccia. Truth, justice, and cake cutting. *Games and Economic Behavior*, 77:284–297, 2013.
- 14 Xiaotie Deng, Qi Qi, and Amin Saberi. Algorithmic solutions for envy-free cake cutting. *Operations Research*, 60:1461–1476, 2012.
- 15 Jeff Edmonds and Kirk Pruhs. Balanced allocations of cake. In *47th Annual Symposium on Foundations of Computer Science, 2006*, pages 623–634, 2006.
- 16 Jeff Edmonds and Kirk Pruhs. Cake cutting really is not a piece of cake. In *17th ACM-SIAM Symposium on Discrete Algorithms*, pages 271–278, 2006.
- 17 Giorgio Gallo, Michael D. Grigoriadis, and Robert E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18:30–55, 1989.
- 18 Paul W. Goldberg, Alexandros Hollender, and Warut Suksompong. Contiguous cake cutting: hardness results and approximation algorithms. In *34th AAAI Conference on Artificial Intelligence, 2020*, pages 1990–1997, 2020.
- 19 David Kurokawa, John K. Lai, and Ariel D. Procaccia. How to cut a cake before the party ends. In *27th AAAI Conference on Artificial Intelligence, 2013*, pages 555–561, 2013.
- 20 Avishay Maya and Noam Nisan. Incentive compatible two player cake cutting. In *8th International Conference on Web and Internet Economics, 2012*, pages 170–183, 2012.

- 21 Ariel D. Procaccia. Thou shalt covet thy neighbor's cake. In *21st International Joint Conference on Artificial Intelligence*, pages 239–244, 2009.
- 22 Ariel D. Procaccia. Cake cutting: not just child's play. *Commun. ACM*, 56:78–87, 2013.
- 23 Ariel D. Procaccia. Cake cutting algorithms. *Handbook of Computational Social Choice (F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A.D. Procaccia, Eds., Cambridge University Press)*, Chapter 13:313–329, 2016.
- 24 Jack M. Robertson and William A. Webb. *Cake Cutting Algorithms: Be Fair If you Can*. A.K. Peters, 1998.
- 25 Masoud Seddighin, Majid Farhadi, Mohammad Ghodsi, Reza Alijani, and Ahmad S. Tajik. Expand the shares together: envy-free mechanisms with a small number of cuts. *Algorithmica*, 81:1728–1755, 2019.
- 26 Katerina Sherstyuk. How to gerrymander: A formal analysis. *Public Choice*, 95:27–49, 1998.
- 27 Hugo Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.
- 28 Hugo Steinhaus. Sur la division pragmatique. *Econometrica (suppliment)*, 17:315–319, 1949.
- 29 Walter Stromquist. How to cut a cake fairly. *American Mathematical Monthly*, 87:640–644, 1980.
- 30 Walter Stromquist. Envy-free cake divisions cannot be found by finite protocols. *The Electronic Journal of Combinatorics*, 15:#R11 (pp.1–10), 2008.
- 31 Francis E. Su. Rental harmony: Sperner's lemma in fair division. *American Mathematical Monthly*, 106:930–942, 1999.
- 32 William Thomson. Children crying at birthday parties. why? *Economic Theory*, 31:501–521, 2007.
- 33 Takeshi Tokuyama. Minimax parametric optimization problems and multi-dimensional parametric searching. In *33rd ACM Symposium on Theory of Computing*, pages 75–83, 2001.
- 34 David P. Williamson. *Network Flow Algorithms*. Cambridge University Press, 2019.



# Adaptive Regularized Submodular Maximization

Shaojie Tang  

Naveen Jindal School of Management, University of Texas at Dallas, TX, USA

Jing Yuan  

Department of Computer Science, University of Texas at Dallas, TX, USA

---

## Abstract

In this paper, we study the problem of maximizing the difference between an adaptive submodular (revenue) function and a non-negative modular (cost) function. The input of our problem is a set of  $n$  items, where each item has a particular state drawn from some known prior distribution. The revenue function  $g$  is defined over items and states, and the cost function  $c$  is defined over items, i.e., each item has a fixed cost. The state of each item is unknown initially and one must select an item in order to observe its realized state. A policy  $\pi$  specifies which item to pick next based on the observations made so far. Denote by  $g_{avg}(\pi)$  the expected revenue of  $\pi$  and let  $c_{avg}(\pi)$  denote the expected cost of  $\pi$ . Our objective is to identify the best policy  $\pi^o \in \arg \max_{\pi} g_{avg}(\pi) - c_{avg}(\pi)$  under a  $k$ -cardinality constraint. Since our objective function can take on both negative and positive values, the existing results of submodular maximization may not be applicable. To overcome this challenge, we develop a series of effective solutions with performance guarantees. Let  $\pi^o$  denote the optimal policy. For the case when  $g$  is adaptive monotone and adaptive submodular, we develop an effective policy  $\pi^l$  such that  $g_{avg}(\pi^l) - c_{avg}(\pi^l) \geq (1 - \frac{1}{e} - \epsilon)g_{avg}(\pi^o) - c_{avg}(\pi^o)$ , using only  $O(n\epsilon^{-2} \log \epsilon^{-1})$  value oracle queries. For the case when  $g$  is adaptive submodular, we present a randomized policy  $\pi^r$  such that  $g_{avg}(\pi^r) - c_{avg}(\pi^r) \geq \frac{1}{e}g_{avg}(\pi^o) - c_{avg}(\pi^o)$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Approximation algorithms analysis; Mathematics of computing  $\rightarrow$  Approximation algorithms; Mathematics of computing  $\rightarrow$  Submodular optimization and polymatroids

**Keywords and phrases** Adaptive submodularity, approximation algorithms, active learning

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.69

## 1 Introduction

Maximizing a submodular function subject to practical constraints has attracted increased attention recently [3, 16, 17, 7]. Submodularity encodes a natural diminishing returns property, which can be found in a wide variety of machine learning tasks such as active learning [3], virtual marketing [16, 17], sensor placement [7], and data summarization [8]. Under the non-adaptive setting, where one must select a group of items all at once, [11] shows that a classic greedy algorithm achieves  $1 - 1/e$  approximation ratio for the problem of maximizing a monotone and non-negative submodular function subject to a cardinality constraint. For non-monotone and non-negative objectives, [1] obtains an approximation of  $1/e + 0.004$ .

Very recently, [4] studies the problem of maximizing the difference between a monotone non-negative submodular function and a non-negative modular function. Given that the objective function of the above problem may take both positive and negative values, most existing technologies, which require the objective function to take only non-negative values, can not provide nontrivial approximation guarantees. They overcome this challenge by developing a series of effective algorithms. In this paper, we extend their work to the adaptive setting by considering the problem of adaptive regularized submodular maximization, i.e., our goal is to adaptively select a group of items to maximize the difference between an adaptive submodular (revenue) function and a non-negative modular (cost) function. We next provide



© Shaojie Tang and Jing Yuan;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 69; pp. 69:1–69:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

more details about our adaptive setting. Following the framework of adaptive submodular maximization [3], a natural stochastic variant of the classical non-adaptive submodular maximization problem, we assume that each item is in a particular state drawn from a known prior distribution. The state of each item is unknown initially and one must select an item before observing its state. A policy  $\pi$  specifies which item to pick next based on the observations made so far. Note that the decision on selecting an item is irrevocable, that is, we can not discard any item that is previously selected. The revenue function  $g$  is defined over items and states, and the cost function  $c$  is defined over items. Note that there are two sources of randomness that make our problem more complicated than its non-adaptive counterpart. One is the random realization of items' states, and the other one is the random decision that is made by the policy. We use  $g_{avg}(\pi)$  to denote the expected revenue of  $\pi$  and let  $c_{avg}(\pi)$  denote the expected cost of  $\pi$ . Our objective is to identify the best policy:

$$\max_{\pi} g_{avg}(\pi) - c_{avg}(\pi)$$

under a  $k$ -cardinality constraint. The above formulation has its applications in many domains [5, 12]. When  $g_{avg}(\pi)$  represents the revenue of  $\pi$  and  $c_{avg}(\pi)$  encodes the cost of  $\pi$ , the above formulation is to maximize profits. In general, the above formulation may be interpreted as a regularized submodular maximization problem under the adaptive setting. Since our objective function can take on both negative and positive values, the existing results of non-monotone adaptive submodular maximization [14, 15], which require the objective function to take only non-negative values, may not be applicable.

Our contribution is threefold. We first consider the case when the revenue function  $g$  is adaptive monotone and adaptive submodular. Letting  $\pi^o$  denote the optimal policy, we develop an effective policy  $\pi^d$  such that  $g_{avg}(\pi^d) - c_{avg}(\pi^d) \geq (1 - \frac{1}{e})g_{avg}(\pi^o) - c_{avg}(\pi^o)$ , using  $O(kn)$  value oracle queries. Our second result is the development of a faster policy  $\pi^l$  such that  $g_{avg}(\pi^l) - c_{avg}(\pi^l) \geq (1 - \frac{1}{e} - \epsilon)g_{avg}(\pi^o) - c_{avg}(\pi^o)$ , using only  $O(n\epsilon^{-2} \log \epsilon^{-1})$  value oracle queries. For the case when  $g$  is (non-monotone) adaptive submodular, we present a randomized policy  $\pi^r$  such that  $g_{avg}(\pi^r) - c_{avg}(\pi^r) \geq \frac{1}{e}g_{avg}(\pi^o) - c_{avg}(\pi^o)$ .

## 2 Related Work

Submodular maximization is a well-studied topic due to its applications in a wide range of domains including active learning [3], virtual marketing [16, 17], sensor placement [7]. Most of existing studies focus on the non-adaptive setting where one must select a group of items all at once. [11] shows that a classic greedy algorithm, which iteratively selects the item that has the largest marginal revenue on top of the previously selected items, achieves a  $1 - 1/e$  approximation ratio when maximizing a monotone non-negative submodular function subject to a cardinality constraint. The problem of maximizing a sum of a non-negative monotone submodular function and an (arbitrary) modular function is first studied in [13]. Notably, their objective function may take on negative values. [2] develops a faster algorithm using a surrogate objective that varies with time. For the case of a cardinality constraint and a non-negative  $c$ , [4] develops the first practical algorithm. Their results have been enhanced by [5] for the unconstrained case. Recently, [6, 12] extend this study to streaming and distributed settings. Our work is different from theirs in that we focus on the adaptive setting [3, 14, 15]. Moreover, we consider a more general problem of maximizing the difference of a non-negative *non-monotone* adaptive submodular function and a non-negative modular function.

### 3 Preliminaries

In the rest of this paper, we use  $[m]$  to denote the set  $\{0, 1, \dots, m\}$ . We mostly follow [3] and adopt similar notations.

#### 3.1 Items and States

The input of our problem is a set  $E$  of  $n$  items. Each item  $e \in E$  is in a random state  $\Phi(e) \in O$  where  $O$  represents the set of all possible states. We use a function  $\phi : E \rightarrow O$ , called a *realization*, to represent the realized states of all items, i.e.,  $\phi(e)$  represents a realization of  $\Phi(e)$ . There is a known prior probability distribution  $p = \{\Pr[\Phi = \phi] : \phi \in U\}$  over all possible realizations  $U$ . The state  $\Phi(e)$  of each item  $e \in E$  is unknown initially and one must select  $e$  before observing its realized state. If we select multiple items  $S \subseteq E$ , then we are able to observe a *partial realization*  $\psi : S \rightarrow O$  and  $\text{dom}(\psi) = S$  is called the *domain* of  $\psi$ . A partial realization  $\psi$  is said to be consistent with a realization  $\phi$ , denoted  $\phi \sim \psi$ , if they are equal everywhere in  $\text{dom}(\psi)$ . A partial realization  $\psi$  is said to be a *subrealization* of  $\psi'$ , denoted  $\psi \subseteq \psi'$ , if  $\text{dom}(\psi) \subseteq \text{dom}(\psi')$  and they are equal everywhere in the domain  $\text{dom}(\psi)$  of  $\psi$ . Let  $p(\phi | \psi)$  denote the conditional distribution over realizations conditioned on a partial realization  $\psi$ :  $p(\phi | \psi) = \Pr[\Phi = \phi | \Phi \sim \psi]$ . In the rest of this paper, we use uppercase letters to denote random variables, and lowercase letters for realizations. For example,  $\Psi$  refers to a random variable, and  $\psi$  is a realization of  $\Psi$ .

#### 3.2 Revenue and Cost

For a set  $Y \subseteq E$  of items and a realization  $\phi$ , let  $g(Y, \phi)$  represent the revenue of selecting  $Y$  conditioned on  $\phi$ , where  $g$  is called *revenue function*. Moreover, each item  $e \in E$  has a fixed cost  $c_e$ . For any set subset of items  $Y \subseteq E$ , let  $c(Y) = \sum_{e \in Y} c_e$  denote the total cost of  $Y$ , where  $c$  is called *cost function*.

#### 3.3 Problem Formulation

A policy specifies which item to select next based on the partial realization observed so far. Mathematically, we represent a policy using a function  $\pi$  that maps a set of observations to a distribution  $\mathcal{P}(E)$  of  $E$ :  $\pi : 2^E \times O^E \rightarrow \mathcal{P}(E)$ .

► **Definition 1** ([3], Policy Concatenation). *Given two policies  $\pi$  and  $\pi'$ , let  $\pi @ \pi'$  denote a policy that runs  $\pi$  first, and then runs  $\pi'$ , ignoring the observation obtained from running  $\pi$ .*

► **Definition 2** ([3], Level- $t$ -Truncation of a Policy). *Given a policy  $\pi$ , we define its level- $t$ -truncation  $\pi_t$  as a policy that runs  $\pi$  until it selects  $t$  items.*

For each realization  $\phi$ , let  $E(\pi, \phi)$  denote the subset of items selected by  $\pi$  under realization  $\phi$ . Note that  $E(\pi, \phi)$  is a random variable. The expected revenue  $g_{avg}(\pi)$  of a policy  $\pi$  can be written as

$$g_{avg}(\pi) = \mathbb{E}_{\Phi, \Pi}[g(E(\pi, \Phi), \Phi)]$$

where the expectation is taken over possible realizations according to  $p$  and the internal randomness of the policy. Similarly, the expected cost  $c_{avg}(\pi)$  of a policy  $\pi$  can be written as

$$c_{avg}(\pi) = \mathbb{E}_{\Phi, \Pi}[c(E(\pi, \Phi))]$$



## 69:4 Adaptive Regularized Submodular Maximization

We next introduce the conditional expected marginal revenue  $g(e \mid \psi)$  of  $e$  conditioned on a partial realization  $\psi$ :

$$g(e \mid \psi) = \mathbb{E}_{\Phi}[g(\text{dom}(\psi) \cup \{e\}, \Phi) - g(\text{dom}(\psi), \Phi) \mid \Phi \sim \psi]$$

where the expectation is taken over  $\Phi$  with respect to  $p(\phi \mid \psi) = \Pr(\Phi = \phi \mid \Phi \sim \psi)$ .

► **Definition 3** ([3], Adaptive Submodularity). *For any two partial realizations  $\psi$  and  $\psi'$  such that  $\psi \subseteq \psi'$ , we assume that the following holds for each  $e \in E \setminus \text{dom}(\psi')$ :*

$$g(e \mid \psi) \geq g(e \mid \psi') \quad (1)$$

Let  $\Omega = \{\pi \mid \forall \phi \in U^+, |E(\pi, \phi)| \leq k\}$  denote the set of all policies that select at most  $k$  items where  $U^+ = \{\phi \in U \mid p(\phi) > 0\}$ , our objective is listed in below:

$$\max_{\pi \in \Omega} g_{\text{avg}}(\pi) - c_{\text{avg}}(\pi) \quad (2)$$

Before presenting our solutions to the above problem, we introduce some additional notations. By abuse of notation, for any partial realization  $\psi$ , we define  $g(\psi) = \mathbb{E}_{\Phi \sim \psi}[g(\text{dom}(\psi), \Phi)]$ . We next introduce two useful functions:  $G_i$ , the distorted objective function, and  $H_i$ , which is used to analyze the trajectory of  $G_i$ . For any partial realization  $\psi$ , and any iteration  $i \in [k]$  of our algorithms, we define

$$G_i(\psi) = \left(1 - \frac{1}{k}\right)^{k-i} g(\psi) - c(\text{dom}(\psi))$$

For any partial realization  $\psi$ , and any iteration  $i \in [k-1]$  of our algorithms, we define

$$H_i(\psi, e) = \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi) - c_e$$

### 4 Monotone $g$ : Adaptive Distorted Greedy Policy

We start with the case when  $g$  is adaptive submodular and *adaptive monotone* [3], i.e., for any realization  $\psi$ , the following holds for each  $e \in E \setminus \text{dom}(\psi)$ :  $g(e \mid \psi) \geq 0$ . Our approach is a natural extension of the *Distorted-Greedy* algorithm, the first practical non-adaptive algorithm developed in [4]. Note that there are two factors that make our problem more complicated than its non-adaptive counterpart. First, since the objective function is defined over random realization, the key of analysis is to estimate the expected utility under the distribution of realizations  $p$ . Second, the policy itself might produce random outputs even under the same realization, this adds an additional layer of difficulty to the design and analysis of our policy. To address the above complications, we develop an *Adaptive Distorted Greedy Policy*  $\pi^d$  such that  $g_{\text{avg}}(\pi^d) - c_{\text{avg}}(\pi^d) \geq \left(1 - \frac{1}{e}\right) g_{\text{avg}}(\pi^o) - c_{\text{avg}}(\pi^o)$ , where  $\pi^o$  denotes the optimal policy. We next explain the idea of  $\pi^d$  (Algorithm 1), then analyze its performance bound.

#### 4.1 Design of $\pi^d$

We first add a dummy item  $d$  to the ground set, such that,  $c_d = 0$ , and for any partial realization  $\psi$ , we have  $g(d \mid \psi) = 0$ . Let  $E' = E \cup \{d\}$ . We add this to ensure that our policy will not select an item that has a negative profit. Note that  $d$  can be safely removed from the final solution without affecting its performance.  $\pi^d$  performs in  $k$  iterations: It starts with an empty set. In each iteration  $i \in [k-1]$ , let  $\psi_i$  denote the current partial realization,  $\pi^d$  selects an item  $e_i$  that maximizes  $H_i(\psi_i, \cdot)$ :

$$e_i \leftarrow \arg \max_{e \in E'} H_i(\psi_i, e)$$

After observing the state  $\Phi(e_i)$  of  $e_i$ , we update the current partial realization  $\psi_{i+1}$  using  $\psi_i \cup \{(e_i, \Phi(e_i))\}$  and enter the next iteration. This process iterates until all  $k$  items have been selected.

■ **Algorithm 1** Adaptive Distorted Greedy Policy  $\pi^d$ .

- 
- 1:  $S_0 = \emptyset; i = 0; \psi_0 = \emptyset$ .
  - 2: **while**  $i < k$  **do**
  - 3:    $e_i \leftarrow \arg \max_{e \in E'} H_i(\psi_i, e)$ ;
  - 4:    $S_{i+1} \leftarrow S_i \cup \{e_i\}$ ;
  - 5:    $\psi_{i+1} \leftarrow \psi_i \cup \{(e_i, \Phi(e_i))\}; i \leftarrow i + 1$ ;
  - 6: **return**  $S_k$
- 

## 4.2 Performance Analysis

We first present three preparatory lemmas which are used to lower bound the marginal gain in the distorted objective. Recall that  $\Psi_{i+1}$  refers to a random variable, and  $\psi_{i+1}$  is a realization of  $\Psi_{i+1}$ .

► **Lemma 4.** *In each iteration of  $\pi^d$ ,*

$$\mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] = H_i(\psi_i, e_i) + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i)$$

**Proof.** We start with the case when  $e_i \in \text{dom}(\psi_i)$ ,

$$\begin{aligned} & \mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \\ &= \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) - \left(1 - \frac{1}{k}\right)^{k-i} g(\psi_i) \\ &= \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) - \left(1 - \frac{1}{k}\right) \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\ &= \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\ &= H_i(\psi_i, e_i) + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \end{aligned}$$

The last equality is due to  $H_i(\psi_i, e_i) = 0$  when  $e_i \in \text{dom}(\psi_i)$ . We next prove the case when  $e_i \notin \text{dom}(\psi_i)$ ,

$$\begin{aligned} & \mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \\ &= \mathbb{E}_{\Phi \sim \psi_i} \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i \cup \{\Phi(e_i)\}) \right. \\ & \quad \left. - c(\text{dom}(\psi_i) \cup \{e_i\}) - \left( \left(1 - \frac{1}{k}\right)^{k-i} g(\psi_i) - c(\text{dom}(\psi_i)) \right) \right] \\ &= \mathbb{E}_{\Phi \sim \psi_i} \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i \cup \{\Phi(e_i)\}) \right. \\ & \quad \left. - c(\text{dom}(\psi_i) \cup \{e_i\}) - \left( \left(1 - \frac{1}{k}\right)^{k-i} g(\psi_i) - c(\text{dom}(\psi_i)) \right) \right] \\ &= \mathbb{E}_{\Phi \sim \psi_i} \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i \cup \{\Phi(e_i)\}) \right. \\ & \quad \left. - c(\text{dom}(\psi_i) \cup \{e_i\}) - \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} \left(1 - \frac{1}{k}\right) g(\psi_i) - c(\text{dom}(\psi_i)) \right) \right] \\ &= \mathbb{E}_{\Phi \sim \psi_i} \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} (g(\psi_i \cup \{\Phi(e_i)\}) - g(\psi_i)) \right] - c_{e_i} + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \end{aligned}$$

## 69:6 Adaptive Regularized Submodular Maximization

$$\begin{aligned}
&= g(e_i \mid \psi_i) - c_{e_i} + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\
&= H_i(\psi_i, e_i) + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i)
\end{aligned}$$

The fourth equality is due to  $e_i \notin \text{dom}(\psi_i)$ .  $\blacktriangleleft$

► **Lemma 5.** *In each iteration of  $\pi^d$ ,*

$$H_i(\psi_i, e_i) \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{\text{avg}}(\pi^o) - g_{\text{avg}}(\pi_i^d)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{\text{avg}}(\pi^o)]$$

**Proof.** Let  $A_e$  be an indicator that  $e$  is selected by the optimal solution  $\pi^o$  conditioned on a partial realization  $\psi_i$ , then we have

$$\begin{aligned}
H_i(\psi_i, e_i) &= \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e_i \mid \psi_i) - c_{e_i} \\
&= \max_{e \in E'} \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi_i) - c_e \right] \\
&\geq \frac{1}{k} \sum_{e \in E'} \Pr[A_e = 1] \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi_i) - c_e \right] \\
&= \frac{1}{k} \sum_{e \in E'} \Pr[A_e = 1] \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi_i) \right] - \frac{1}{k} \sum_{e \in E'} \Pr[A_e = 1] \times c_e \\
&= \frac{1}{k} \sum_{e \in E'} \Pr[A_e = 1] \left[ \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi_i) \right] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{\text{avg}}(\pi^o)] \\
&\geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{\text{avg}}(\pi^o) - g_{\text{avg}}(\pi_i^d)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{\text{avg}}(\pi^o)]
\end{aligned}$$

The second equality is due to the design of  $\pi^d$ , i.e., it selects an item  $e_i$  that maximizes  $H_i(\psi_i, \cdot)$ . The first inequality is due to  $\sum_{e \in E'} \Pr[A_e = 1] \leq k$  since  $\pi^o$  selects at most  $k$  items. The third equality is due to the assumption that  $\Pr[A_e = 1]$  is the probability that  $e$  is selected by  $\pi^o$  conditioned on  $\psi_i$ . The second inequality is due to  $g$  is adaptive submodular.  $\blacktriangleleft$

► **Lemma 6.** *In each iteration of  $\pi^d$ ,*

$$\mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{\text{avg}}(\pi^o) - \frac{1}{k} c_{\text{avg}}(\pi^o)$$

**Proof.** We first prove that for any fixed partial realization  $\psi_i$ , the following inequality holds:

$$\mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{\text{avg}}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{\text{avg}}(\pi^o)] \quad (3)$$

Due to Lemma 4, we have

$$\begin{aligned}
&\mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \\
&= H_i(\psi_i, e_i) + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\
&\geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{\text{avg}}(\pi^o) - g_{\text{avg}}(\pi_i^d)] \\
&\quad - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{\text{avg}}(\pi^o)] + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\
&= \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{\text{avg}}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{\text{avg}}(\pi^o)]
\end{aligned}$$

The inequality is due to Lemma 5 and the second equality is due to  $\mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi_i^d)] = g(\psi_i)$ . Now we are ready to prove this lemma.

$$\begin{aligned}
& \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\
& \geq \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \Psi_i} [g_{avg}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \Psi_i} [c_{avg}(\pi^o)] \right] \\
& = \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \Psi_i} [g_{avg}(\pi^o)] \right] - \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \mathbb{E}_{\Phi \sim \Psi_i} [c_{avg}(\pi^o)] \right] \\
& = \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{avg}(\pi^o) - \frac{1}{k} c_{avg}(\pi^o)
\end{aligned}$$

The inequality is due to (3). ◀

We next present the first main theorem of this paper.

► **Theorem 7.**  $g_{avg}(\pi^d) - c_{avg}(\pi^d) \geq \left(1 - \frac{1}{e}\right) g_{avg}(\pi^o) - c_{avg}(\pi^o)$ .

**Proof.** According to the definition of  $G_k$ , we have  $\mathbb{E}_{\Psi_k} [G_k(\Psi_k)] = \mathbb{E}_{\Psi_k} \left[ \left(1 - \frac{1}{k}\right)^0 g(\Psi_k) - c(\text{dom}(\Psi_k)) \right] = g_{avg}(\pi^d) - c_{avg}(\pi^d)$  and  $\mathbb{E}_{\Psi_0} [G_0(\Psi_0)] = \mathbb{E}_{\Psi_0} \left[ \left(1 - \frac{1}{k}\right)^k g(\Phi(S_0)) - c(\text{dom}(\Psi_0)) \right] = 0$ . Hence,

$$\begin{aligned}
& g_{avg}(\pi^d) - c_{avg}(\pi^d) = \mathbb{E}_{\Psi_k} [G_k(\Psi_k)] - \mathbb{E}_{\Psi_0} [G_0(\Psi_0)] \\
& = \mathbb{E}_{\Psi_{k-1}} [\mathbb{E}_{\Phi \sim \Psi_{k-1}} [G_k(\Psi_k)]] - \mathbb{E}_{\Psi_0} [\mathbb{E}_{\Phi \sim \Psi_0} [G_0(\Psi_0)]] \\
& = \sum_{i \in [k-1]} (\mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1})]] - \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_i(\Psi_i)]]) \\
& = \sum_{i \in [k-1]} \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\
& \geq \sum_{i \in [k-1]} \left( \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{avg}(\pi^o) - \frac{1}{k} c_{avg}(\pi^o) \right) \\
& = \sum_{i \in [k-1]} \left( \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{avg}(\pi^o) \right) - c_{avg}(\pi^o) \\
& \geq \left(1 - \frac{1}{e}\right) g_{avg}(\pi^o) - c_{avg}(\pi^o)
\end{aligned}$$

The first inequality is due to Lemma 6. ◀

## 5 Monotone $g$ : Linear-time Adaptive Distorted Greedy Policy

We next propose a faster algorithm *Linear-time Adaptive Distorted Greedy Policy*, denoted by  $\pi^l$ , for the case when  $g$  is adaptive monotone. As compared with  $\pi^d$  whose running time is  $O(nk)$ , our new policy  $\pi^l$  achieves nearly the same performance guarantee with  $O(n \log \frac{1}{\epsilon})$  value oracle queries. Our design is inspired by the sampling technique developed in [10] for maximizing a monotone and submodular function. Very recently, [14] extends this approach to the adaptive setting to develop a linear-time adaptive policy for maximizing an adaptive submodular and adaptive monotone function. In this work, we apply this technique to design a linear-time adaptive policy for our adaptive regularized submodular maximization problem. Note that our objectives are not adaptive monotone and they may take negative values.

## 5.1 Design of $\pi^l$

We present the details of our algorithm in Algorithm 2. We first add a set  $D$  of  $k-1$  dummy items to the ground set, such that, each dummy item  $d \in D$  has zero cost, i.e.,  $\forall d \in D, c_d = 0$ , and for any  $d \in D$ , and any partial realization  $\psi$ , we have  $g(d | \psi) = 0$ . Let  $E' = E \cup D$ . We next explain the idea of  $\pi^l$ : It starts with an empty set. In each iteration  $i \in [k-1]$ ,  $\pi^l$  first samples a set  $R_i$  of size  $\frac{n}{k} \log \frac{1}{\epsilon}$  uniformly at random, then adds an item  $e_i$  with the largest  $H_i(\psi_i, \cdot)$  from  $R_i$  to the solution. After observing the state  $\Phi(e_i)$  of  $e_i$ , we update the current partial realization  $\psi_{i+1}$  using  $\psi_i \cup \{(e_i, \Phi(e_i))\}$  and enter the next iteration. This process iterates until all  $k$  items have been selected. It is worth noting that the technique of lazy updates [9] can be used to further accelerate the computation of our algorithms in practice.

■ **Algorithm 2** Linear-time Adaptive Distorted Greedy Policy  $\pi^l$ .

---

```

1:  $S_0 = \emptyset; i = 0; \psi_0 = \emptyset.$ 
2: while  $i < k$  do
3:    $R_i \leftarrow$  a random set sampled uniformly at random from  $E'$ ;
4:    $e_i \leftarrow \arg \max_{e \in R_i} H_i(\psi_i, e);$ 
5:    $S_{i+1} \leftarrow S_i \cup \{e_i\};$ 
6:    $\psi_{i+1} \leftarrow \psi_i \cup \{(e_i, \Phi(e_i))\}; i \leftarrow i + 1;$ 
7: return  $S_k$ 

```

---

## 5.2 Performance Analysis

We first present three preparatory lemmas.

► **Lemma 8.** *In each iteration of  $\pi^l$ ,*

$$\mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] = \mathbb{E}_{e_i} [H_i(\psi_i, e_i)] + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i)$$

The above lemma immediately follows from Lemma 4.

► **Lemma 9.** *In each iteration of  $\pi^l$ ,  $\mathbb{E}_{e_i} [H_i(\psi_i, e_i)] \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o) - g_{avg}(\pi_i^l)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)].$*

**Proof.** Let  $A_e$  be an indicator that  $e$  is selected by the optimal solution  $\pi^o$  conditioned on a partial realization  $\psi_i$ . Let  $B_e$  be an indicator that  $e$  is selected by  $\pi^l$  in iteration  $i$  conditioned on a partial realization  $\psi_i$ . Let  $M(\psi_i)$  denote the top  $k$  items with the largest marginal contribution to  $\psi_i$  in terms of  $H_i(\psi_i, \cdot)$ , i.e.,  $M(\psi_i) \leftarrow \arg \max_{S \subseteq E', |S|=k} \sum_{e \in S} H_i(\psi_i, e)$ . Then we have

$$\begin{aligned} & \mathbb{E}_{e_i} [H_i(\psi_i, e_i)] \\ &= \sum_{e \in E'} \Pr[B_e = 1] \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e | \psi_i) - c_e \right) \\ &\geq \Pr[R_i \cap M(\psi_i) \neq \emptyset] \frac{1}{k} \sum_{e \in M(\psi_i)} \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e | \psi_i) - c_e \right) \\ &\geq (1 - \epsilon) \frac{1}{k} \sum_{e \in M(\psi_i)} \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e | \psi_i) - c_e \right) \\ &\geq (1 - \epsilon) \frac{1}{k} \sum_{e \in E'} \Pr[A_e = 1] \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e | \psi_i) - c_e \right) \\ &\geq (1 - \epsilon) \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o) - g_{avg}(\pi_i^l)] - (1 - \epsilon) \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \end{aligned}$$

The second inequality is due to Lemma 4 in [14], where they show that  $\Pr[R_i \cap M(\psi_i) \neq \emptyset] \geq 1 - \epsilon$  given that  $R_i$  has size of  $\frac{n}{k} \log \frac{1}{\epsilon}$ . The third inequality is due to  $\sum_{e \in E'} \Pr[A_e = 1] \leq k$ . The last inequality is due to  $g$  is adaptive submodular and  $c$  is modular.  $\blacktriangleleft$

► **Lemma 10.** *In each iteration of  $\pi^l$ ,*

$$\begin{aligned} & \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\ & \geq (1 - \epsilon) \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} g_{avg}(\pi^o) - (1 - \epsilon) \frac{1}{k} c_{avg}(\pi^o) \end{aligned}$$

**Proof.** We first show that for any fixed partial realization  $\psi_i$ ,

$$\begin{aligned} & \mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \\ & \geq (1 - \epsilon) \left( \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \right) \end{aligned} \quad (4)$$

Due to Lemma 8, we have

$$\begin{aligned} & \mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \\ & = \mathbb{E}_{e_i} [H_i(\psi_i, e_i)] + \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} g(\psi_i) \\ & \geq (1 - \epsilon) \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o) - g_{avg}(\pi_i^l)] \\ & \quad - (1 - \epsilon) \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] + \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} g(\psi_i) \\ & = (1 - \epsilon) \left( \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \right) \\ & \quad + \epsilon \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} g(\psi_i) \\ & \geq (1 - \epsilon) \left( \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \right) \end{aligned}$$

The first inequality is due to Lemma 9, the second equality is due to  $\mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi_i^l)] = g(\psi_i)$ , and the last inequality is due to  $g$  is non-negative. Now we are ready to prove this lemma.

$$\begin{aligned} & \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\ & \geq (1 - \epsilon) \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} \mathbb{E}_{\Phi \sim \Psi_i} [g_{avg}(\pi^o)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \Psi_i} [c_{avg}(\pi^o)] \right] \\ & = (1 - \epsilon) \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} \mathbb{E}_{\Phi \sim \Psi_i} [g_{avg}(\pi^o)] \right] - \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \mathbb{E}_{\Phi \sim \Psi_i} [c_{avg}(\pi^o)] \right] \\ & = (1 - \epsilon) \frac{1}{k} (1 - \frac{1}{k})^{k-(i+1)} g_{avg}(\pi^o) - (1 - \epsilon) \frac{1}{k} c_{avg}(\pi^o) \end{aligned}$$

The first inequality is due to (4).  $\blacktriangleleft$

We next present the second main theorem of this paper.

► **Theorem 11.**  $g_{avg}(\pi^l) - c_{avg}(\pi^l) \geq (1 - \frac{1}{e} - \epsilon) g_{avg}(\pi^o) - c_{avg}(\pi^o)$ .

**Proof.** According to the definition of  $G_k$ , we have  $\mathbb{E}_{\Psi_k} [G_k(\Psi_k)] = \mathbb{E}_{\Psi_k} [(1 - \frac{1}{k})^0 g(\Psi_k) - c(\text{dom}(\Psi_k))] = g_{avg}(\pi^l) - c_{avg}(\pi^l)$  and  $\mathbb{E}_{\Psi_0} [G_0(\Psi_0)] = \mathbb{E}_{\Psi_0} [(1 - \frac{1}{k})^k g(\Phi(S_0)) - c(\text{dom}(\Psi_0))] = 0$ . Hence,

$$\begin{aligned}
& g_{avg}(\pi^l) - c_{avg}(\pi^l) \\
&= \mathbb{E}_{\Psi_k}[G_k(\Psi_k)] - \mathbb{E}_{\Psi_0}[G_0(\Psi_0)] \\
&= \mathbb{E}_{\Psi_{k-1}}[\mathbb{E}_{\Phi \sim \Psi_{k-1}}[G_k(\Psi_k)]] - \mathbb{E}_{\Psi_0}[\mathbb{E}_{\Phi \sim \Psi_0}[G_0(\Psi_0)]] \\
&= \sum_{i \in [k-1]} (\mathbb{E}_{\Psi_i}[\mathbb{E}_{\Phi \sim \Psi_i}[G_{i+1}(\Psi_{i+1})]] - \mathbb{E}_{\Psi_i}[\mathbb{E}_{\Phi \sim \Psi_i}[G_i(\Psi_i)]]) \\
&= \sum_{i \in [k-1]} \mathbb{E}_{\Psi_i}[\mathbb{E}_{\Phi \sim \Psi_i}[G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\
&\geq \sum_{i \in [k-1]} \left( (1-\epsilon) \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{avg}(\pi^o) - (1-\epsilon) \frac{1}{k} c_{avg}(\pi^o) \right) \\
&= \sum_{i \in [k-1]} \left( (1-\epsilon) \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{avg}(\pi^o) \right) - (1-\epsilon) c_{avg}(\pi^o) \\
&\geq (1-\epsilon) \left(1 - \frac{1}{e}\right) g_{avg}(\pi^o) - (1-\epsilon) c_{avg}(\pi^o) \\
&\geq \left(1 - \frac{1}{e} - \epsilon\right) g_{avg}(\pi^o) - c_{avg}(\pi^o)
\end{aligned}$$

The first inequality is due to Lemma 10. ◀

## 6 Non-monotone $g$ : Adaptive Random Distorted Greedy Policy

We next discuss the case when  $g$  is non-monotone adaptive submodular. We present an *Adaptive Random Distorted Greedy Policy*  $\pi^r$  for this case.

### 6.1 Design of $\pi^r$

The detailed implementation of  $\pi^r$  is listed in Algorithm 3. We first add a set  $D$  of  $k-1$  dummy items to the ground set, such that, for any  $d \in D$ , and any partial realization  $\psi$ , we have  $c_d = 0$  and  $g(d \mid \psi) = 0$ . Let  $E' = E \cup D$ .  $\pi^r$  runs round by round: Starting with an empty set. In each iteration  $i \in [k-1]$ ,  $\pi^r$  randomly selects an item from the set  $M(\psi_i)$ . Recall that  $M(\psi_i)$  is a set of  $k$  items that have the largest  $H_i(\psi_i, \cdot)$ , i.e.,

$$M(\psi_i) \leftarrow \arg \max_{S \subseteq E'; |S|=k} \sum_{e \in S} H_i(\psi_i, e)$$

After observing the state  $\Phi(e_i)$  of  $e_i$ , we update the current partial realization  $\psi_{i+1}$  using  $\psi_i \cup \{(e_i, \Phi(e_i))\}$  and enter the next iteration. This process iterates until all  $k$  items have been selected.

■ **Algorithm 3** Adaptive Random Distorted Greedy Policy  $\pi^r$ .

- 
- 1:  $S_0 = \emptyset; i = 0; \psi_0 = \emptyset$ .
  - 2: **while**  $i < k$  **do**
  - 3:    $M(\psi_i) \leftarrow \arg \max_{S \subseteq E'; |S|=k} \sum_{e \in S} H_i(\psi_i, e)$ ;
  - 4:   sample  $e_i$  uniformly at random from  $M(\psi_i)$ ;
  - 5:    $S_{i+1} \leftarrow S_i \cup \{e_i\}$ ;
  - 6:    $\psi_{i+1} \leftarrow \psi_i \cup \{(e_i, \Phi(e_i))\}$ ;  $i \leftarrow i + 1$ ;
  - 7: **return**  $S_k$
-



## 6.2 Performance Analysis

We first present three preparatory lemmas. The first lemma immediately follows from Lemma 4.

► **Lemma 12.** *In each iteration of  $\pi^r$ ,*

$$\mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] = \mathbb{E}_{e_i} [H_i(\psi_i, e_i)] + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i)$$

► **Lemma 13.** *In each iteration of  $\pi^r$ ,  $\mathbb{E}_{e_i} [H_i(\psi_i, e_i)] \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o @ \pi_i^r)] - g_{avg}(\pi_i^r) - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)]$ .*

**Proof.** Recall that  $A_e$  is an indicator that  $e$  is selected by the optimal solution  $\pi^o$  conditioned on a partial realization  $\psi_i$ ,

$$\begin{aligned} & \mathbb{E}_{e_i} [H_i(\psi_i, e_i)] \\ &= \frac{1}{k} \sum_{e \in M(\psi_i)} \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi_i) - c_e \right) \\ &\geq \frac{1}{k} \sum_{e \in E'} \Pr[A_e = 1] \left( \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(e \mid \psi_i) - c_e \right) \\ &\geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o @ \pi_i^r)] - g_{avg}(\pi_i^r) - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \end{aligned}$$

The equality is due to the design of  $\pi^r$ , i.e., it selects an item  $e_i$  uniformly at random from  $M(\psi_i)$ . The first inequality is due to  $\sum_{e \in E'} \Pr[A_e = 1] \leq k$  since  $\pi^o$  selects at most  $k$  items, and  $M(\psi_i)$  contains a set of  $k$  items that have the largest  $H_i(\psi_i, \cdot)$ . The second inequality is due to  $g$  is adaptive submodular and  $c$  is modular. ◀

► **Lemma 14.** *In each iteration of  $\pi^r$ ,*

$$\mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-1} g_{avg}(\pi^o) - \frac{1}{k} c_{avg}(\pi^o)$$

**Proof.** We first show that for any fixed partial realization  $\psi_i$ ,

$$\mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o @ \pi_i^r)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \quad (5)$$

Due to Lemma 12, we have

$$\begin{aligned} & \mathbb{E}_{\Phi \sim \psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\psi_i)] \\ &= \mathbb{E}_{e_i} [H_i(\psi_i, e_i)] + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\ &\geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o @ \pi_i^r)] - g_{avg}(\pi_i^r) \\ &\quad - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] + \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g(\psi_i) \\ &= \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi^o @ \pi_i^r)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \psi_i} [c_{avg}(\pi^o)] \end{aligned}$$

## 69:12 Adaptive Regularized Submodular Maximization

The first inequality is due to Lemma 13. The second equality is due to  $\mathbb{E}_{\Phi \sim \psi_i} [g_{avg}(\pi_i^r)] = g(\psi_i)$ . The last inequality is due to  $g$  is non-negative. Now we are ready to prove this lemma.

$$\begin{aligned}
& \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\
& \geq \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \Psi_i} [g_{avg}(\pi^o @ \pi_i^r)] - \frac{1}{k} \mathbb{E}_{\Phi \sim \Psi_i} [c_{avg}(\pi^o)] \right] \\
& = \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \mathbb{E}_{\Phi \sim \Psi_i} [g_{avg}(\pi^o @ \pi_i^r)] \right] - \mathbb{E}_{\Psi_i} \left[ \frac{1}{k} \mathbb{E}_{\Phi \sim \Psi_i} [c_{avg}(\pi^o)] \right] \\
& = \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} g_{avg}(\pi^o @ \pi_i^r) - \frac{1}{k} c_{avg}(\pi^o) \\
& \geq \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-(i+1)} \left(1 - \frac{1}{k}\right)^i g_{avg}(\pi^o) - \frac{1}{k} c_{avg}(\pi^o) \\
& = \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-1} g_{avg}(\pi^o) - \frac{1}{k} c_{avg}(\pi^o)
\end{aligned}$$

The first inequality is due to (5), and the second inequality is due to Lemma 1 in [14], where they show that  $g_{avg}(\pi^o @ \pi_i^r) \geq \left(1 - \frac{1}{k}\right)^i g_{avg}(\pi^o)$ . ◀

We next present the third main theorem of this paper.

► **Theorem 15.**  $g_{avg}(\pi^r) - c_{avg}(\pi^r) \geq \frac{1}{e} g_{avg}(\pi^o) - c_{avg}(\pi^o)$ .

**Proof.** According to the definition of  $G_k$ , we have  $\mathbb{E}_{\Psi_k} [G_k(\Psi_k)] = \mathbb{E}_{\Psi_k} [(1 - \frac{1}{k})^0 g(\Psi_k) - c(\text{dom}(\Psi_k))] = g_{avg}(\pi^r) - c_{avg}(\pi^r)$  and  $\mathbb{E}_{\Psi_0} [G_0(\Psi_0)] = \mathbb{E}_{\Psi_0} [(1 - \frac{1}{k})^k g(\Phi(S_0)) - c(\text{dom}(\Psi_0))] = 0$ . Hence,

$$\begin{aligned}
g_{avg}(\pi^r) - c_{avg}(\pi^r) &= \mathbb{E}_{\Psi_k} [G_k(\Psi_k)] - \mathbb{E}_{\Psi_0} [G_0(\Psi_0)] \\
&= \mathbb{E}_{\Psi_{k-1}} [\mathbb{E}_{\Phi \sim \Psi_{k-1}} [G_k(\Psi_k)]] - \mathbb{E}_{\Psi_0} [\mathbb{E}_{\Phi \sim \Psi_0} [G_0(\Psi_0)]] \\
&= \sum_{i \in [k-1]} (\mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1})]] - \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_i(\Psi_i)]]) \\
&= \sum_{i \in [k-1]} \mathbb{E}_{\Psi_i} [\mathbb{E}_{\Phi \sim \Psi_i} [G_{i+1}(\Psi_{i+1}) - G_i(\Psi_i)]] \\
&\geq \sum_{i \in [k-1]} \left( \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-1} g_{avg}(\pi^o) - \frac{1}{k} c_{avg}(\pi^o) \right) \\
&= \sum_{i \in [k-1]} \frac{1}{k} \left(1 - \frac{1}{k}\right)^{k-1} g_{avg}(\pi^o) - c_{avg}(\pi^o) \\
&\geq \frac{1}{e} g_{avg}(\pi^o) - c_{avg}(\pi^o)
\end{aligned}$$

The first inequality is due to Lemma 14. ◀

## 7 Conclusion

In this paper, we study the adaptive regularized submodular maximization problem. Because our objective function may take both negative and positive values, most existing technologies of submodular maximization do not apply to our setting. We develop a series of effective policies for this problem.

---

**References**

---


- 1 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1433–1452. SIAM, 2014.
- 2 Moran Feldman. Guess free maximization of submodular and linear sums. *Algorithmica*, pages 1–26, 2020.
- 3 Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.
- 4 Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In *International Conference on Machine Learning*, pages 2634–2643. PMLR, 2019.
- 5 Tianyuan Jin, Yu Yang, Renchi Yang, Jieming Shi, Keke Huang, and Xiaokui Xiao. Unconstrained submodular maximization with modular costs: Tight approximation and application to profit maximization. *Proceedings of the VLDB Endowment*, 14(10), 2021.
- 6 Ehsan Kazemi, Shervin Minaee, Moran Feldman, and Amin Karbasi. Regularized submodular maximization at scale. In *International Conference on Machine Learning*, pages 5356–5366. PMLR, 2021.
- 7 Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, volume 7, pages 1650–1654, 2007.
- 8 Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520, 2011.
- 9 Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*, pages 234–243. Springer, 1978.
- 10 Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- 11 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical programming*, 14(1):265–294, 1978.
- 12 Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. An efficient framework for balancing submodularity and cost. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1256–1266, 2021.
- 13 Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. *Mathematics of Operations Research*, 42(4):1197–1218, 2017.
- 14 Shaojie Tang. Beyond pointwise submodularity: Non-monotone adaptive submodular maximization in linear time. *Theoretical Computer Science*, 850:249–261, 2021.
- 15 Shaojie Tang. Beyond pointwise submodularity: Non-monotone adaptive submodular maximization subject to knapsack and  $k$ -system constraints. In *International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences*. Springer, 2021.
- 16 Shaojie Tang and Jing Yuan. Influence maximization with partial feedback. *Operations Research Letters*, 48(1):24–28, 2020.
- 17 Jing Yuan and Shaojie Tang. No time to observe: adaptive influence maximization with partial feedback. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3908–3914, 2017.



# $\Gamma$ -Graphic Delta-Matroids and Their Applications

Donggyu Kim ✉

Department of Mathematical Sciences, KAIST, Daejeon, South Korea  
Discrete Mathematics Group, Institute for Basic Science, Daejeon, South Korea

Duksang Lee ✉ 

Department of Mathematical Sciences, KAIST, Daejeon, South Korea  
Discrete Mathematics Group, Institute for Basic Science, Daejeon, South Korea

Sang-il Oum ✉ 

Discrete Mathematics Group, Institute for Basic Science, Daejeon, South Korea  
Department of Mathematical Sciences, KAIST, Daejeon, South Korea

---

## Abstract

For an abelian group  $\Gamma$ , a  $\Gamma$ -labelled graph is a graph whose vertices are labelled by elements of  $\Gamma$ . We prove that a certain collection of edge sets of a  $\Gamma$ -labelled graph forms a delta-matroid, which we call a  $\Gamma$ -graphic delta-matroid, and provide a polynomial-time algorithm to solve the separation problem, which allows us to apply the symmetric greedy algorithm of Bouchet to find a maximum weight feasible set in such a delta-matroid. We present two algorithmic applications on graphs; MAXIMUM WEIGHT PACKING OF TREES OF ORDER NOT DIVISIBLE BY  $k$  and MAXIMUM WEIGHT  $S$ -TREE PACKING. We also discuss various properties of  $\Gamma$ -graphic delta-matroids.

**2012 ACM Subject Classification** Mathematics of computing  $\rightarrow$  Matroids and greedoids

**Keywords and phrases** delta-matroid, group-labelled graph, greedy algorithm, tree packing

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.70

**Related Version** *Full Version:* <https://arxiv.org/abs/2104.11383>

**Funding** Supported by the Institute for Basic Science (IBS-R029-C1).

## 1 Introduction

We introduce the class of  $\Gamma$ -graphic delta-matroids arising from graphs whose vertices are labelled by elements of an abelian group  $\Gamma$ . This allows us to show that the following problems are solvable in polynomial time by using the symmetric greedy algorithm [1].

MAXIMUM WEIGHT PACKING OF TREES OF ORDER NOT DIVISIBLE BY  $k$

**Input:** An integer  $k \geq 2$ , a graph  $G$ , and a weight  $w : E(G) \rightarrow \mathbb{Q}$ .

**Problem:** Find vertex-disjoint trees  $T_1, T_2, \dots, T_m$  for some  $m$  such that  $|V(T_i)| \not\equiv 0 \pmod{k}$  for each  $i \in \{1, \dots, m\}$  and  $\sum_{i=1}^m \sum_{e \in E(T_i)} w(e)$  is maximized.

For a vertex set  $S$  of a graph  $G$ , a subgraph of  $G$  is an  $S$ -tree if it is a tree intersecting  $S$ .

MAXIMUM WEIGHT  $S$ -TREE PACKING

**Input:** A graph  $G$ , a nonempty subset  $S$  of  $V(G)$ , and a weight  $w : E(G) \rightarrow \mathbb{Q}$ .

**Problem:** Find vertex-disjoint  $S$ -trees  $T_1, T_2, \dots, T_m$  for some  $m$  such that  $\bigcup_{i=1}^m V(T_i) = V(G)$  and  $\sum_{i=1}^m \sum_{e \in E(T_i)} w(e)$  is maximized.

Let  $\Gamma$  be an abelian group. We assume that  $\Gamma$  is an additive group. A  $\Gamma$ -labelled graph is a pair  $(G, \gamma)$  of a graph  $G$  and a map  $\gamma : V(G) \rightarrow \Gamma$ . A subgraph  $H$  of  $G$  is  $\gamma$ -nonzero if, for each component  $C$  of  $H$ ,



© Donggyu Kim, Duksang Lee, and Sang-il Oum;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 70; pp. 70:1–70:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- (G1)  $\sum_{v \in V(C)} \gamma(v) \neq 0$  or  $\gamma|_{V(C)} \equiv 0$ , and
- (G2) if  $\gamma|_{V(C)} \equiv 0$ , then  $G[V(C)]$  is a component of  $G$ .

A subset  $F$  of  $E(G)$  is  $\gamma$ -nonzero in  $G$  if a subgraph  $(V(G), F)$  is  $\gamma$ -nonzero. A subset  $F$  of  $E(G)$  is *acyclic* in  $G$  if a subgraph  $(V(G), F)$  has no cycle.

Bouchet [1] introduced delta-matroids which are set systems  $(E, \mathcal{F})$  satisfying certain axioms. Our first theorem proves that the set of acyclic  $\gamma$ -nonzero sets in a  $\Gamma$ -labelled graph  $(G, \gamma)$  forms a delta-matroid, which we call a  $\Gamma$ -graphic delta-matroid. For sets  $X$  and  $Y$ , let  $X \Delta Y = (X - Y) \cup (Y - X)$ .

► **Theorem 1.** *Let  $\Gamma$  be an abelian group and  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. If  $\mathcal{F}$  is the set of acyclic  $\gamma$ -nonzero sets in  $G$ , then the following hold.*

- (1)  $\mathcal{F} \neq \emptyset$ .
- (2) For  $X, Y \in \mathcal{F}$  and  $e \in X \Delta Y$ , there exists  $f \in X \Delta Y$  such that  $X \Delta \{e, f\} \in \mathcal{F}$ .

Bouchet [1] proved that the symmetric greedy algorithm finds a maximum weight set in  $\mathcal{F}$  for a delta-matroid  $(E, \mathcal{F})$ . But it requires the *separation oracle*, which determines, for two disjoint subsets  $X$  and  $Y$  of  $E$ , whether there exists a set  $F \in \mathcal{F}$  such that  $X \subseteq F$  and  $F \cap Y = \emptyset$ . We provide the separation oracle that runs in polynomial time for  $\Gamma$ -graphic delta-matroids given by  $\Gamma$ -labelled graphs. As a consequence, we prove the following theorem.

**MAXIMUM WEIGHT ACYCLIC  $\gamma$ -NONZERO SET**

**Input:** A  $\Gamma$ -labelled graph  $(G, \gamma)$  and a weight  $w : E(G) \rightarrow \mathbb{Q}$ .

**Problem:** Find an acyclic  $\gamma$ -nonzero set  $F$  in  $G$  maximizing  $\sum_{e \in F} w(e)$ .

► **Theorem 2.** *MAXIMUM WEIGHT ACYCLIC  $\gamma$ -NONZERO SET is solvable in polynomial time.*

From Theorem 2, we can easily deduce that both MAXIMUM WEIGHT PACKING OF TREES OF ORDER NOT DIVISIBLE BY  $k$  and MAXIMUM WEIGHT  $S$ -TREE PACKING are solvable in polynomial time.

► **Corollary 3.** *MAXIMUM WEIGHT PACKING OF TREES OF ORDER NOT DIVISIBLE BY  $k$  is solvable in polynomial time.*

**Proof.** Let  $\Gamma = \mathbb{Z}_k$  and  $\gamma : V(G) \rightarrow \mathbb{Z}_k$  be a map such that  $\gamma(v) = 1$  for each  $v \in V(G)$ . Then, an edge set  $F$  is an acyclic  $\gamma$ -nonzero set in  $(G, \gamma)$  if and only if there exist vertex-disjoint trees  $T_1, \dots, T_m$  for some  $m$  such that  $\bigcup_{i=1}^m E(T_i) = F$  and  $|V(T_i)| \not\equiv 0 \pmod{k}$  for each  $i \in \{1, \dots, m\}$ . ◀

► **Corollary 4.** *MAXIMUM WEIGHT  $S$ -TREE PACKING is solvable in polynomial time.*

**Proof.** We may assume that every component of  $G$  has a vertex in  $S$ . Let  $\Gamma = \mathbb{Z}$  and  $\gamma : V(G) \rightarrow \mathbb{Z}$  be a map such that

$$\gamma(v) = \begin{cases} 1 & \text{if } v \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Then, an edge set  $F$  is an acyclic  $\gamma$ -nonzero set in  $(G, \gamma)$  if and only if there exist vertex-disjoint  $S$ -trees  $T_1, \dots, T_m$  for some  $m$  such that  $\bigcup_{i=1}^m V(T_i) = V(G)$  and  $\bigcup_{i=1}^m E(T_i) = F$ . ◀

One of the major motivations to introduce  $\Gamma$ -graphic delta-matroids is to generalize the concept of graphic delta-matroids introduced by Oum [8], which are precisely  $\mathbb{Z}_2$ -graphic delta-matroids. Oum [8] proved that every minor of graphic delta-matroids is graphic. We will prove that every minor of a  $\Gamma$ -graphic delta-matroid is  $\Gamma$ -graphic.

A delta-matroid  $(E, \mathcal{F})$  is *even* if  $|X \Delta Y|$  is even for all  $X, Y \in \mathcal{F}$ . Oum [8] proved that every graphic delta-matroid is even. We characterize even  $\Gamma$ -graphic delta-matroids as follows.

► **Theorem 5.** *Let  $\Gamma$  be an abelian group. Then a  $\Gamma$ -graphic delta-matroid is even if and only if it is graphic.*

Bouchet [2] proved that for a symmetric or skew-symmetric matrix  $A$  over a field  $\mathbb{F}$ , the set of index sets of nonsingular principal submatrices of  $A$  forms a delta-matroid, which we call a delta-matroid *representable over  $\mathbb{F}$* . Oum [8] proved that every graphic delta-matroid is representable over  $\text{GF}(2)$ . Our next theorem partially characterizes a pair of an abelian group  $\Gamma$  and a field  $\mathbb{F}$  such that every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ .

If  $\mathbb{F}_1$  is a subfield of a field  $\mathbb{F}_2$ , then  $\mathbb{F}_2$  is an *extension field* of  $\mathbb{F}_1$ , denoted by  $\mathbb{F}_2/\mathbb{F}_1$ . The *degree* of a field extension  $\mathbb{F}_2/\mathbb{F}_1$ , denoted by  $[\mathbb{F}_2 : \mathbb{F}_1]$ , is the dimension of  $\mathbb{F}_2$  as a vector space over  $\mathbb{F}_1$ .

► **Theorem 6.** *Let  $p$  be a prime,  $k$  be a positive integer, and  $\mathbb{F}$  be a field of characteristic  $p$ . If  $[\mathbb{F} : \text{GF}(p)] \geq k$ , then every  $\mathbb{Z}_p^k$ -graphic delta-matroid is representable over  $\mathbb{F}$ .*

For a prime  $p$ , an abelian group is an *elementary abelian  $p$ -group* if every nonzero element has order  $p$ .

► **Theorem 7.** *Let  $\mathbb{F}$  be a finite field of characteristic  $p$  and  $\Gamma$  be an abelian group. If every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ , then  $\Gamma$  is an elementary abelian  $p$ -group.*

Theorems 6 and 7 allow us to partially characterize pairs of a finite field  $\mathbb{F}$  and an abelian group  $\Gamma$  for which every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$  as follows. We omit its easy proof.

► **Corollary 8.** *Let  $\Gamma$  be a finite abelian group of order at least 2 and  $\mathbb{F}$  be a finite field.*

- (i) *For every prime  $p$  and integers  $1 \leq k \leq \ell$ , every  $\mathbb{Z}_p^k$ -graphic delta-matroid is representable over  $\text{GF}(p^\ell)$ .*
- (ii) *If every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ , then  $\Gamma$  is isomorphic to  $\mathbb{Z}_p^k$  and  $\mathbb{F}$  is isomorphic to  $\text{GF}(p^\ell)$  for a prime  $p$  and positive integers  $k$  and  $\ell$ .*

We suspect that the following could be the complete characterization.

► **Conjecture 9.** *Let  $\Gamma$  be a finite abelian group of order at least 2 and  $\mathbb{F}$  be a finite field. Then every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$  if and only if  $(\Gamma, \mathbb{F}) = (\mathbb{Z}_p^k, \text{GF}(p^\ell))$  for some prime  $p$  and positive integers  $k \leq \ell$ .*

This paper is organized as follows. In Section 2, we review some terminologies and results on delta-matroids and graphic delta-matroids. In Section 3, we introduce  $\Gamma$ -graphic delta-matroids. We show that the class of  $\Gamma$ -graphic delta-matroids is closed under taking minors in Section 4. In Section 5, we present a polynomial-time algorithm to solve MAXIMUM WEIGHT ACYCLIC  $\gamma$ -NONZERO SET, proving Theorem 2. We characterize even  $\Gamma$ -graphic delta-matroids in Section 6. In Section 7, we prove Theorems 6 and 7. We provide some proofs in the full version when lemmas and theorems are marked by \*.

## 2 Preliminaries

In this paper, all graphs are finite and may have parallel edges and loops. A graph is *simple* if it has neither loops nor parallel edges. For a graph  $G$ , *contracting* an edge  $e$  is an operation to obtain a new graph  $G/e$  from  $G$  by deleting  $e$  and identifying ends of  $e$ . For a set  $X$  and



a positive integer  $s$ , let  $\binom{X}{s}$  be the set of  $s$ -element subsets of  $X$ . For two sets  $A$  and  $B$ , let  $A\Delta B = (A - B) \cup (B - A)$ . For a function  $f : X \rightarrow Y$  and a subset  $A \subseteq X$ , we write  $f|_A$  to denote the restriction of  $f$  on  $A$ .

**Delta-matroids.** Bouchet [1] introduced delta-matroids. A *delta-matroid* is a pair  $M = (E, \mathcal{F})$  of a finite set  $E$  and a nonempty set  $\mathcal{F}$  of subsets of  $E$  such that if  $X, Y \in \mathcal{F}$  and  $x \in X\Delta Y$ , then there is  $y \in Y\Delta X$  such that  $X\Delta\{x, y\} \in \mathcal{F}$ . We write  $E(M) = E$  to denote the *ground set* of  $M$ . An element of  $\mathcal{F}$  is called a *feasible set*. An element of  $E$  is a *loop* of  $M$  if it is not contained in any feasible set of  $M$ . An element of  $E$  is a *coloop* of  $M$  if it is contained in every feasible set of  $M$ .

**Minors.** For a delta-matroid  $M = (E, \mathcal{F})$  and a subset  $X$  of  $E$ , we can obtain a new delta-matroid  $M\Delta X = (E, \mathcal{F}\Delta X)$  from  $M$  where  $\mathcal{F}\Delta X = \{F\Delta X : F \in \mathcal{F}\}$ . This operation is called *twisting* a set  $X$  in  $M$ . A delta-matroid  $N$  is *equivalent* to  $M$  if  $N = M\Delta X$  for some set  $X$ .

If there is a feasible subset of  $E - X$ , then  $M \setminus X = (E - X, \mathcal{F} \setminus X)$  is a delta-matroid where  $\mathcal{F} \setminus X = \{F \in \mathcal{F} : F \cap X = \emptyset\}$ . This operation of obtaining  $M \setminus X$  is called the *deletion* of  $X$  in  $M$ . A delta-matroid  $N$  is a *minor* of a delta-matroid  $M$  if  $N = M\Delta X \setminus Y$  for some subsets  $X, Y$  of  $E$ .

A delta-matroid is *normal* if  $\emptyset$  is feasible. A delta-matroid is *even* if  $|X\Delta Y|$  is even for all feasible sets  $X$  and  $Y$ . It is easy to see that all minors of even delta-matroids are even.

The following theorem gives the minimal obstruction for even delta-matroids, which is implied by Bouchet [3, Lemma 5.4].

► **Theorem 10** (Bouchet [3]). *A delta-matroid is even if and only if it does not have a minor isomorphic to  $(\{e\}, \{\emptyset, \{e\}\})$ .*

It is easy to observe the following.

► **Lemma 11.** *Let  $N$  be a minor of a delta-matroid  $M$  such that  $|E(M)| > |E(N)|$ . Then there exists an element  $e \in E(M) - E(N)$  such that  $N$  is a minor of  $M \setminus e$  or a minor of  $M\Delta\{e\} \setminus e$ .*

**Representable delta-matroids.** For an  $R \times C$  matrix  $A$  and subsets  $X$  of  $R$  and  $Y$  of  $C$ , we write  $A[X, Y]$  to denote the  $X \times Y$  submatrix of  $A$ . For an  $E \times E$  square matrix  $A$  and a subset  $X$  of  $E$ , we write  $A[X]$  to denote  $A[X, X]$ , which is called an  $X \times X$  *principal* submatrix of  $A$ .

For an  $E \times E$  square matrix  $A$ , let  $\mathcal{F}(A) = \{X \subseteq E : A[X] \text{ is nonsingular}\}$ . We assume that  $A[\emptyset]$  is nonsingular and so  $\emptyset \in \mathcal{F}(A)$ . Bouchet [2] proved that,  $(E, \mathcal{F}(A))$  is a delta-matroid if  $A$  is an  $E \times E$  symmetric or skew-symmetric matrix. A delta-matroid  $M = (E, \mathcal{F})$  is *representable over* a field  $\mathbb{F}$  if  $\mathcal{F} = \mathcal{F}(A)\Delta X$  for a symmetric or skew-symmetric matrix  $A$  over  $\mathbb{F}$  and a subset  $X$  of  $E$ . Since  $\emptyset \in \mathcal{F}(A)$ , it is natural to define representable delta-matroids with twisting so that the empty set is not necessarily feasible in representable delta-matroids.

A delta-matroid is *binary* if it is representable over  $\text{GF}(2)$ . Note that all diagonal entries of a skew-symmetric matrix are zero, even if the characteristic of a field is 2.

► **Proposition 12** (Bouchet [2]). *Let  $M = (E, \mathcal{F})$  be a delta-matroid. Then  $M$  is normal and representable over a field  $\mathbb{F}$  if and only if there is an  $E \times E$  symmetric or skew-symmetric matrix  $A$  over  $\mathbb{F}$  such that  $\mathcal{F} = \mathcal{F}(A)$ .*

► **Lemma 13** (Geelen [5, page 27]). *Let  $M$  be a delta-matroid representable over a field  $\mathbb{F}$ . Then  $M$  is even if and only if  $M$  is representable by a skew-symmetric matrix over  $\mathbb{F}$ .*

**Pivoting.** For a finite set  $E$  and a symmetric or skew-symmetric  $E \times E$  matrix  $A$ , if  $A$  is represented by

$$A = \begin{matrix} & X & Y \\ X & \alpha & \beta \\ Y & \gamma & \delta \end{matrix}$$

after selecting a linear ordering of  $E$  and  $A[X] = \alpha$  is nonsingular, then let

$$A * X = \begin{matrix} & X & Y \\ X & \alpha^{-1} & \alpha^{-1}\beta \\ Y & -\gamma\alpha^{-1} & \delta - \gamma\alpha^{-1}\beta \end{matrix}$$

This operation is called *pivoting*. Tucker [11] proved that when  $A[X]$  is nonsingular,  $A * X[Y]$  is nonsingular if and only if  $A[X\Delta Y]$  is nonsingular for each subset  $Y$  of  $E$ . Hence, if  $X$  is a feasible set of a delta-matroid  $M = (E, \mathcal{F}(A))$ , then  $M\Delta X = (E, \mathcal{F}(A * X))$ . It implies that all minors of delta-matroids representable over a field  $\mathbb{F}$  are representable over  $\mathbb{F}$  [4].

**Greedy algorithm.** Let  $M = (E, \mathcal{F})$  be a set system such that  $E$  is finite and  $\mathcal{F} \neq \emptyset$ . A pair  $(X, Y)$  of disjoint subsets  $X$  and  $Y$  of  $E$  is *separable* in  $M$  if there exists a set  $F \in \mathcal{F}$  such that  $X \subseteq F$  and  $Y \cap F = \emptyset$ . The following theorem characterizes delta-matroids in terms of a greedy algorithm. Note that this greedy algorithm requires an oracle which answers whether a pair  $(X, Y)$  of disjoint subsets  $X$  and  $Y$  of  $E$  is separable in  $M$ .

► **Theorem 14** (Bouchet [1]; see Moffatt [7]). *Let  $M = (E, \mathcal{F})$  be a set system such that  $E$  is finite and  $\mathcal{F} \neq \emptyset$ . Then  $M$  is a delta-matroid if and only if the symmetric greedy algorithm in Algorithm 1 gives a set  $F \in \mathcal{F}$  maximizing  $\sum_{e \in F} w(e)$  for each  $w : E \rightarrow \mathbb{R}$ .*

**Graphic delta-matroids.** Oum [8] introduced graphic delta-matroid. A *graft* is a pair  $(G, T)$  of a graph  $G$  and a subset  $T$  of  $V(G)$ . A subgraph  $H$  of  $G$  is  *$T$ -spanning* in  $G$  if  $V(H) = V(G)$ , for each component  $C$  of  $H$ , either

- (i)  $|V(C) \cap T|$  is odd, or
- (ii)  $V(C) \cap T = \emptyset$  and  $G[V(C)]$  is a component of  $G$ .

An edge set  $F$  of  $G$  is  *$T$ -spanning* in  $G$  if a subgraph  $(V(G), F)$  is  $T$ -spanning in  $G$ . For a graft  $(G, T)$ , let  $\mathcal{G}(G, T) = (E(G), \mathcal{F})$  where  $\mathcal{F}$  is the set of acyclic  $T$ -spanning sets in  $G$ . Oum [8] proved that  $\mathcal{G}(G, T)$  is an even binary delta-matroid. A delta-matroid is *graphic* if it is equivalent to  $\mathcal{G}(G, T)$  for a graft  $(G, T)$ .

### 3 Delta-matroids from group-labelled graphs

Let  $\Gamma$  be an abelian group. A  $\Gamma$ -labelled graph  $(G, \gamma)$  is a pair of a graph  $G$  and a map  $\gamma : V(G) \rightarrow \Gamma$ . We say  $\gamma \equiv 0$  if  $\gamma(v) = 0$  for all  $v \in V(G)$ . A  $\Gamma$ -labelled graph  $(G, \gamma)$  and a  $\Gamma'$ -labelled graph  $(G', \gamma')$  are *isomorphic* if there are a graph isomorphism  $f$  from  $G$  to  $G'$  and a group isomorphism  $\phi : \Gamma \rightarrow \Gamma'$  such that  $\phi(\gamma(v)) = \gamma'(f(v))$  for each  $v \in V(G)$ .

■ **Algorithm 1** Symmetric greedy algorithm.

---

```

1: function SYMMETRIC GREEDY ALGORITHM( $M, w$ )      ▷  $M = (E, \mathcal{F})$  and  $w : E \rightarrow \mathbb{R}$ 
2:   Enumerate  $E = \{e_1, e_2, \dots, e_n\}$  such that  $|w(e_1)| \geq |w(e_2)| \geq \dots \geq |w(e_n)|$ 
3:    $X \leftarrow \emptyset$  and  $Y \leftarrow \emptyset$ 
4:   for  $i \leftarrow 1$  to  $n$  do
5:     if  $w(e_i) \geq 0$  then
6:       if  $(X \cup \{e_i\}, Y)$  is separable then
7:          $X \leftarrow X \cup \{e_i\}$ 
8:       else
9:          $Y \leftarrow Y \cup \{e_i\}$ 
10:      end if
11:     else
12:       if  $(X, Y \cup \{e_i\})$  is separable then
13:          $Y \leftarrow Y \cup \{e_i\}$ 
14:       else
15:          $X \leftarrow X \cup \{e_i\}$ 
16:       end if
17:     end if
18:   end for
19: end function
20: return  $X$                                      ▷  $X \in \mathcal{F}$ 

```

---

A subgraph  $H$  of  $G$  is  $\gamma$ -nonzero if, for each component  $C$  of  $H$ ,

(G1)  $\sum_{v \in V(C)} \gamma(v) \neq 0$  or  $\gamma|_{V(C)} \equiv 0$ , and

(G2) if  $\gamma|_{V(C)} \equiv 0$ , then  $G[V(C)]$  is a component of  $G$ .

An edge set  $F$  of  $E(G)$  is  $\gamma$ -nonzero in  $G$  if a subgraph  $(V(G), F)$  is  $\gamma$ -nonzero. An edge set  $F$  of  $E(G)$  is *acyclic* in  $G$  if a subgraph  $(V(G), F)$  has no cycle.

For an abelian group  $\Gamma$  and a  $\Gamma$ -labelled graph  $(G, \gamma)$ , let  $\mathcal{F}$  be the set of acyclic  $\gamma$ -nonzero sets in  $G$ . Now we are ready to show Theorem 1, which proves that  $(E(G), \mathcal{F})$  is a delta-matroid. We denote  $(E(G), \mathcal{F})$  by  $\mathcal{G}(G, \gamma)$ . A delta-matroid  $M$  is  $\Gamma$ -graphic if there exist a  $\Gamma$ -labelled graph  $(G, \gamma)$  and  $X \subseteq E(G)$  such that  $M = \mathcal{G}(G, \gamma) \Delta X$ .

► **Theorem 1.** *Let  $\Gamma$  be an abelian group and  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. If  $\mathcal{F}$  is the set of acyclic  $\gamma$ -nonzero sets in  $G$ , then the following hold.*

(1)  $\mathcal{F} \neq \emptyset$ .

(2) For  $X, Y \in \mathcal{F}$  and  $e \in X \Delta Y$ , there exists  $f \in X \Delta Y$  such that  $X \Delta \{e, f\} \in \mathcal{F}$ .

**Proof.** By considering each component, we may assume that  $G$  is connected. If  $\gamma \equiv 0$ , then we choose a vertex  $v$  of  $G$  and a map  $\gamma' : V(G) \rightarrow \Gamma$  such that  $\gamma'(u) \neq 0$  if and only if  $u = v$ . Then the set of acyclic  $\gamma$ -nonzero sets in  $G$  is equal to the set of acyclic  $\gamma'$ -nonzero sets in  $G$ . Hence, we can assume that  $\gamma$  is not identically zero. Therefore, a subgraph  $H$  of  $G$  is  $\gamma$ -nonzero if and only if  $\sum_{u \in V(C)} \gamma(u) \neq 0$  for each component  $C$  of  $H$ .

Let us first prove (1), stating that  $\mathcal{F} \neq \emptyset$ . Let  $S = \{v \in V(G) : \gamma(v) \neq 0\}$  and  $T$  be a spanning tree of  $G$ . Then by the assumption, we have  $S \neq \emptyset$ . We may assume that  $\sum_{u \in V(G)} \gamma(u) = 0$  because otherwise  $E(T)$  is acyclic  $\gamma$ -nonzero in  $G$ . Let  $e$  be an edge of  $T$  such that one of two components  $C_1$  and  $C_2$  of  $T \setminus e$  has exactly one vertex in  $S$ . Then  $\sum_{u \in V(C_1)} \gamma(u) = -\sum_{u \in V(C_2)} \gamma(u) \neq 0$ . So  $E(T) - \{e\}$  is acyclic  $\gamma$ -nonzero in  $G$ , and (1) holds.

Now let us prove (2). We proceed by induction on  $|E(G)|$ . It is obvious if  $|E(G)| = 0$ . If there is an edge  $g = vw$  in  $X \cap Y$ , then let  $\gamma' : V(G/g) \rightarrow \Gamma$  such that, for each vertex  $x$  of  $G/g$ ,

$$\gamma'(x) = \begin{cases} \gamma(v) + \gamma(w) & \text{if } x \text{ is the vertex of } G/g \text{ corresponding to } g, \\ \gamma(x) & \text{otherwise.} \end{cases}$$

Then both  $X - \{g\}$  and  $Y - \{g\}$  are acyclic  $\gamma'$ -nonzero sets in  $G/g$ . Let  $e \in (X - \{g\}) \Delta (Y - \{g\}) = X \Delta Y$ . By the induction hypothesis, there exists  $f \in X \Delta Y$  such that  $(X - \{g\}) \Delta \{e, f\}$  is an acyclic  $\gamma'$ -nonzero set in  $G/g$ .

We now claim that  $X \Delta \{e, f\}$  is an acyclic  $\gamma$ -nonzero set in  $G$ . It is obvious that  $X \Delta \{e, f\}$  is acyclic in  $G$ . If  $\gamma' \equiv 0$ , then  $\gamma(v) = -\gamma(w) \neq 0$  and  $\gamma(u) = 0$  for every  $u$  in  $V(G) - \{v, w\}$ . Then  $X$  is not  $\gamma$ -nonzero, contradicting our assumption. Hence,  $\gamma' \neq 0$  and let  $C$  be a component of  $(V(G), X \Delta \{e, f\})$ . If  $C$  contains  $g$ , then  $\sum_{u \in V(C)} \gamma(u) = \sum_{u \in V(C/g)} \gamma'(u) \neq 0$ . If  $C$  does not contain  $g$ , then  $\sum_{u \in V(C)} \gamma(u) = \sum_{u \in V(C)} \gamma'(u) \neq 0$ . It implies that  $X \Delta \{e, f\}$  is  $\gamma$ -nonzero in  $G$ , so the claim is verified.

Therefore we may assume that  $X \cap Y = \emptyset$ . Let  $H_1 = (V(G), X)$  and  $H_2 = (V(G), Y)$ .

► **Case 1.**  $e \in X$ .

Let  $C$  be the component of  $H_1$  containing  $e$  and  $C_1, C_2$  be two components of  $C \setminus e$ . If both  $\sum_{u \in V(C_1)} \gamma(u)$  and  $\sum_{u \in V(C_2)} \gamma(u)$  are nonzero, then  $X \Delta \{e\}$  is acyclic  $\gamma$ -nonzero and so we can choose  $f = e$ . So we may assume that  $\sum_{u \in V(C_1)} \gamma(u) = 0$  and therefore

$$\sum_{u \in V(C_2)} \gamma(u) = \sum_{u \in V(C)} \gamma(u) - \sum_{u \in V(C_1)} \gamma(u) \neq 0.$$

If there exists  $f \in Y$  joining a vertex in  $V(C_1)$  to a vertex in  $V(G) - V(C_1)$ , then  $X \Delta \{e, f\}$  is acyclic  $\gamma$ -nonzero. Therefore, we may assume that there is a component  $D_1$  of  $H_2$  such that  $V(D_1) \subseteq V(C_1)$ . Since  $\sum_{u \in V(D_1)} \gamma(u) \neq 0$ , there is a vertex  $x$  of  $D_1$  such that  $\gamma(x) \neq 0$ . So  $\gamma|_{V(C_1)} \neq 0$  and there is an edge  $f$  of  $C_1$  such that one of the components of  $C_1 \setminus f$ , say  $U$ , has exactly one vertex  $v$  with  $\gamma(v) \neq 0$ . If  $U'$  is the component of  $C_1 \setminus f$  other than  $U$ , then  $\sum_{u \in V(U')} \gamma(u) = -\sum_{u \in V(U)} \gamma(u) \neq 0$ . So  $X \Delta \{e, f\}$  is acyclic  $\gamma$ -nonzero.

► **Case 2.**  $e \in Y$ .

Let  $\tilde{H} = (V(G), X \cup \{e\})$ . If  $\tilde{H}$  contains a cycle  $D$ , then, since  $X$  and  $Y$  are acyclic,  $D$  is a unique cycle of  $\tilde{H}$  and there is an edge  $f \in E(D) - Y$ . Then  $X \Delta \{e, f\}$  is acyclic  $\gamma$ -nonzero. Therefore, we can assume that  $e$  joins two distinct components  $C', C''$  of  $H_1$ .

Since  $\sum_{u \in V(C')} \gamma(u) \neq 0$ , there is an edge  $f$  of  $C'$  such that one of the components of  $C' \setminus f$ , say  $U$ , has exactly one vertex  $v$  with  $\gamma(v) \neq 0$ . If  $U'$  is the component of  $C' \setminus f$  other than  $U$ , then  $\sum_{u \in V(U')} \gamma(u) = -\sum_{u \in V(U)} \gamma(u) \neq 0$ . So  $X \Delta \{e, f\}$  is acyclic  $\gamma$ -nonzero. ◀

## 4 Minors of group-labelled graphs

Let  $\Gamma$  be an abelian group. Now we define minors of  $\Gamma$ -labelled graphs as follows. Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph and  $e = uv$  be an edge of  $G$ . Then  $(G, \gamma) \setminus e = (G \setminus e, \gamma)$  is the  $\Gamma$ -labelled graph obtained by *deleting* the edge  $e$  from  $(G, \gamma)$ . For an isolated vertex  $v$  of  $G$ ,  $(G, \gamma) \setminus v = (G \setminus v, \gamma|_{V(G) - \{v\}})$  is the  $\Gamma$ -labelled graph obtained by *deleting* the vertex  $v$  from  $(G, \gamma)$ . If  $e$  is not a loop, then let  $(G, \gamma)/e = (G/e, \gamma')$  such that, for each  $x \in V(G/e)$ ,

$$\gamma'(x) = \begin{cases} \gamma(u) + \gamma(v) & \text{if } x \text{ is the vertex of } G/e \text{ corresponding to } e, \\ \gamma(x) & \text{otherwise.} \end{cases}$$

## 70:8 $\Gamma$ -Graphic Delta-Matroids and Their Applications

If  $e$  is a loop, then let  $(G, \gamma)/e = (G, \gamma) \setminus e$ . Contracting the edge  $e$  is an operation obtaining  $(G, \gamma)/e$  from  $(G, \gamma)$ . For an edge set  $X = \{e_1, \dots, e_t\}$ , let  $(G, \gamma)/X = (G, \gamma)/e_1/\dots/e_t$  and  $(G, \gamma) \setminus X = (G \setminus X, \gamma)$ . A  $\Gamma$ -labelled graph  $(G', \gamma')$  is a *minor* of  $(G, \gamma)$  if  $(G', \gamma')$  is obtained from  $(G, \gamma)$  by deleting some edges, contracting some edges, and deleting some isolated vertices. Let  $\kappa(G, \gamma)$  be the number of components  $C$  of  $G$  such that  $\gamma(x) = 0$  for all  $x \in V(C)$ . An edge  $e$  of  $G$  is a  $\gamma$ -bridge if  $\kappa((G, \gamma) \setminus e) > \kappa(G, \gamma)$ . A non-loop edge  $e = uv$  of  $G$  is a  $\gamma$ -tunnel if, for the component  $C$  of  $G$  containing  $e$ , the following hold:

- (i) For each  $x \in V(C)$ ,  $\gamma(x) \neq 0$  if and only if  $x \in \{u, v\}$ .
- (ii)  $\gamma(u) + \gamma(v) = 0$ .

From the definition of a  $\gamma$ -tunnel, it is easy to see that an edge  $e$  is a  $\gamma$ -tunnel in  $G$  if and only if  $\kappa((G, \gamma)/e) > \kappa(G, \gamma)$ .

The following lemmas are analogous to properties of graphic delta-matroids in Oum [8, Propositions 8, 9, 10, and 11].

► **Lemma 15 (\*)**. *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph and  $e$  be an edge of  $G$ . The following are equivalent.*

- (i) *Every acyclic  $\gamma$ -nonzero set in  $G$  contains  $e$ .*
- (ii) *The edge  $e$  is a  $\gamma$ -bridge in  $G$ .*
- (iii) *Every  $\gamma$ -nonzero set in  $G$  contains  $e$ .*

► **Lemma 16 (\*)**. *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. Then, for an edge  $e$  of  $G$ ,*

$$\mathcal{G}((G, \gamma) \setminus e) = \begin{cases} \mathcal{G}(G, \gamma) \setminus e & \text{if } e \text{ is not a } \gamma\text{-bridge,} \\ \mathcal{G}(G, \gamma) \Delta \{e\} \setminus e & \text{otherwise.} \end{cases}$$

► **Lemma 17 (\*)**. *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph and  $e$  be a non-loop edge of  $G$ . Then the following are equivalent.*

- (i) *No acyclic  $\gamma$ -nonzero set in  $G$  contains  $e$ .*
- (ii) *The edge  $e$  is a  $\gamma$ -tunnel in  $G$ .*
- (iii) *No  $\gamma$ -nonzero set in  $G$  contains  $e$ .*

► **Lemma 18 (\*)**. *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. Then, for an edge  $e$  of  $G$ ,*

$$\mathcal{G}((G, \gamma)/e) = \begin{cases} \mathcal{G}(G, \gamma) \Delta \{e\} \setminus e & \text{if } e \text{ is neither a loop nor a } \gamma\text{-tunnel,} \\ \mathcal{G}(G, \gamma) \setminus e & \text{otherwise.} \end{cases}$$

We omit the proof of the following lemma.

► **Lemma 19**. *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph and  $v$  be an isolated vertex of  $G$ . Then  $\mathcal{G}((G, \gamma) \setminus v) = \mathcal{G}(G \setminus v, \gamma|_{V(G) - \{v\}})$ .*

► **Proposition 20**. *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph and  $M = \mathcal{G}(G, \gamma) \Delta X$  for some  $X \subseteq E(G)$ .*

- (i) *If  $(G', \gamma')$  is a minor of  $(G, \gamma)$ , then  $\mathcal{G}(G', \gamma')$  is a minor of  $M$ .*
- (ii) *If  $M'$  is a minor of  $M$ , then there exists a minor  $(G', \gamma')$  of  $(G, \gamma)$  such that  $M' = \mathcal{G}(G', \gamma') \Delta X'$  for some  $X' \subseteq E(G')$ .*

**Proof.** We may assume that  $X = \emptyset$ . Lemmas 16, 18, and 19 imply (i) and Lemmas 11, 16, 18, and 19 imply (ii). ◀

## 5 Maximum weight acyclic $\gamma$ -nonzero set

In this section, we prove that one can find a maximum weight acyclic  $\gamma$ -nonzero set in a  $\Gamma$ -labelled graph  $(G, \gamma)$  in polynomial time by applying the symmetric greedy algorithm on  $\Gamma$ -graphic delta-matroids. Let us first state the problem.

MAXIMUM WEIGHT ACYCLIC  $\gamma$ -NONZERO SET

**Input:** A  $\Gamma$ -labelled graph  $(G, \gamma)$  and a weight  $w : E(G) \rightarrow \mathbb{Q}$ .

**Problem:** Find an acyclic  $\gamma$ -nonzero set  $F$  in  $G$  maximizing  $\sum_{e \in F} w(e)$ .

Recall that Theorem 14 allows us to find a maximum weight feasible set in a delta-matroid by using the symmetric greedy algorithm in Algorithm 1. As we proved that the set of acyclic  $\gamma$ -nonzero sets in a  $\Gamma$ -labelled graph  $(G, \gamma)$  forms a  $\Gamma$ -graphic delta-matroid in Section 3, we can apply Theorem 14 to solve MAXIMUM WEIGHT ACYCLIC  $\gamma$ -NONZERO SET, but it requires a subroutine that decides in polynomial time whether a pair of two disjoint sets  $X$  and  $Y$  of  $E(G)$  is separable in  $\mathcal{G}(G, \gamma)$ . In the remainder of this section, we focus on developing this subroutine.

We assume that the addition of two elements of  $\Gamma$  and testing whether an element of  $\Gamma$  is zero can be done in time polynomial in the length of the input.

► **Theorem 21.** *Given a  $\Gamma$ -labelled graph  $(G, \gamma)$  and disjoint subsets  $X, Y$  of  $E(G)$ , one can decide in polynomial time whether  $G$  has an acyclic  $\gamma$ -nonzero set  $F$  such that  $X \subseteq F$  and  $Y \cap F = \emptyset$ .*

To prove Theorem 21, we will characterize separable pairs  $(X, Y)$  in  $\mathcal{G}(G, \gamma)$ . Recall that, for a  $\Gamma$ -labelled graph  $(G, \gamma)$ ,  $\kappa(G, \gamma)$  is the number of components  $C$  of  $G$  such that  $\gamma|_{V(C)} \equiv 0$ .

► **Lemma 22.** *Let  $\Gamma$  be an abelian group and  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. Then  $\kappa((G, \gamma) \setminus e) \geq \kappa(G, \gamma)$  and  $\kappa((G, \gamma)/e) \geq \kappa(G, \gamma)$  for every edge  $e$  of  $G$ .*

**Proof.** We may assume that  $G$  is connected and  $\kappa(G, \gamma) = 1$ . Then  $\gamma \equiv 0$  and therefore  $\kappa((G, \gamma) \setminus e) \geq 1$  and  $\kappa((G, \gamma)/e) = 1$ . ◀

► **Lemma 23.** *Let  $\Gamma$  be an abelian group,  $(G, \gamma)$  be a  $\Gamma$ -labelled graph, and  $X$  be an acyclic set of edges of  $G$ . Let  $\gamma' : V(G/X) \rightarrow \Gamma$  be a map such that  $(G/X, \gamma') = (G, \gamma)/X$ . Then the following hold.*

- (1) *If  $\kappa((G, \gamma)/X) = \kappa(G, \gamma)$  and  $F$  is an acyclic  $\gamma'$ -nonzero set in  $G/X$ , then  $F \cup X$  is an acyclic  $\gamma$ -nonzero set in  $G$ .*
- (2) *If  $\kappa((G, \gamma)/X) > \kappa(G, \gamma)$ , then  $G$  has no acyclic  $\gamma$ -nonzero set containing  $X$ .*

**Proof.** Let us first prove (1). By considering each component, we may assume that  $G$  is connected. Since  $X$  is acyclic,  $F \cup X$  is acyclic in  $G$ .

If  $\kappa((G, \gamma)/X) = \kappa(G, \gamma) = 1$ , then  $\gamma \equiv 0$  and  $F$  is the edge set of a spanning tree of  $G/X$  by (G2). Hence  $F \cup X$  is the edge set of a spanning tree of  $G$ , which implies that  $F \cup X$  is acyclic  $\gamma$ -nonzero in  $G$ .

If  $\kappa((G, \gamma)/X) = \kappa(G, \gamma) = 0$ , then let  $H' = (V(G/X), F)$  be a subgraph of  $G/X$  and  $H = (V(G), F \cup X)$  be a subgraph of  $G$ . Then, for each component  $C$  of  $H$ , there exists a component  $C'$  of  $H'$  such that  $C' = C/(E(C) \cap X)$ . Then  $\sum_{u \in V(C)} \gamma(u) = \sum_{u \in V(C')} \gamma'(u) \neq 0$  by (G1). Hence  $F \cup X$  is an acyclic  $\gamma$ -nonzero set in  $G$  and (1) holds.

Now let us prove (2). We proceed by induction on  $|X|$ .

## 70:10 $\Gamma$ -Graphic Delta-Matroids and Their Applications

If  $|X| = 1$ , then  $e \in X$  is a  $\gamma$ -tunnel and by Lemma 17, there is no acyclic  $\gamma$ -nonzero set containing  $X$ . So we may assume that  $|X| > 1$ . Let  $e \in X$  and  $X' = X - \{e\}$ .

By the induction hypothesis, we may assume that  $\kappa((G, \gamma)/X') = \kappa(G, \gamma)$ . Let  $\gamma'' : V(G/X') \rightarrow \Gamma$  be a map such that  $(G/X', \gamma'') = (G, \gamma)/X'$ . Since  $\kappa((G, \gamma)/X) = \kappa((G, \gamma)/X'/e) > \kappa((G, \gamma)/X')$ , by the induction hypothesis,  $G/X'$  has no acyclic  $\gamma''$ -nonzero set containing  $e$ . Therefore,  $G$  has no acyclic  $\gamma$ -nonzero set containing  $X$ . ◀

► **Lemma 24.** *Let  $\Gamma$  be an abelian group,  $(G, \gamma)$  be a  $\Gamma$ -labelled graph, and  $Y$  be a set of edges of  $G$ . Then the following hold.*

(1) *If  $\kappa((G, \gamma) \setminus Y) = \kappa(G, \gamma)$  and  $F$  is an acyclic  $\gamma$ -nonzero set in  $G \setminus Y$ , then  $F$  is an acyclic  $\gamma$ -nonzero set in  $G$ .*

(2) *If  $\kappa((G, \gamma) \setminus Y) > \kappa(G, \gamma)$ , then  $G$  has no acyclic  $\gamma$ -nonzero set  $F$  such that  $Y \cap F = \emptyset$ .*

**Proof.** Let us first prove (1). By considering each component, we may assume that  $G$  is connected.

If  $\kappa((G, \gamma) \setminus Y) = \kappa(G, \gamma) = 1$ , then  $\gamma \equiv 0$  and the set  $F$  is the edge set of a spanning tree of  $G \setminus Y$  by (G2). Then  $F$  is an acyclic  $\gamma$ -nonzero set in  $G$ .

If  $\kappa((G, \gamma) \setminus Y) = \kappa(G, \gamma) = 0$ , then for each component  $C$  of  $G \setminus Y$ , we have  $\gamma|_{V(C)} \not\equiv 0$ . Then,  $\sum_{v \in V(C)} \gamma(v) \neq 0$  for each component  $C$  of  $(V(G), F)$ . So  $F$  is an acyclic  $\gamma$ -nonzero set in  $G$ .

Let us show (2). We proceed by induction on  $|Y|$ . If  $|Y| = 1$ , then  $e \in Y$  is a  $\gamma$ -bridge so it is done by Lemma 15. Now we assume  $|Y| \geq 2$ . Let  $e \in Y$  and  $Y' = Y - \{e\}$ . By the induction hypothesis, we may assume that  $\kappa(G \setminus Y', \gamma) = \kappa(G, \gamma)$ . Since  $\kappa(G \setminus Y, \gamma) = \kappa(G \setminus Y' \setminus e, \gamma) > \kappa(G \setminus Y', \gamma)$ , by the induction hypothesis, every acyclic  $\gamma$ -nonzero set in  $G \setminus Y'$  contains  $e$ . Since every acyclic  $\gamma$ -nonzero set  $F$  in  $G$  not intersecting  $Y'$  is an acyclic  $\gamma$ -nonzero set in  $G \setminus Y'$ , every acyclic  $\gamma$ -nonzero set in  $G$  intersects  $Y$ . ◀

► **Proposition 25.** *Let  $\Gamma$  be an abelian group and  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. Let  $X$  and  $Y$  be disjoint subsets of  $E(G)$  such that  $X$  is acyclic in  $G$ . Then  $\kappa((G, \gamma)/X \setminus Y) = \kappa(G, \gamma)$  if and only if  $G$  has an acyclic  $\gamma$ -nonzero set  $F$  such that  $X \subseteq F$  and  $Y \cap F = \emptyset$ .*

**Proof.** Let us prove the forward direction. By Lemma 22,  $\kappa((G, \gamma)/X \setminus Y) = \kappa((G, \gamma)/X) = \kappa(G, \gamma)$ . Let  $\gamma' : V(G/X \setminus Y) \rightarrow \Gamma$  be a map such that  $(G/X \setminus Y, \gamma') = (G, \gamma)/X \setminus Y$ . By (1) of Theorem 1, there exists an acyclic  $\gamma'$ -nonzero set  $F'$  in  $G/X \setminus Y$ . Since  $\kappa((G, \gamma)/X \setminus Y) = \kappa((G, \gamma)/X)$ ,  $F'$  is acyclic  $\gamma'$ -nonzero in  $G/X$  by (1) of Lemma 24. Since  $\kappa((G, \gamma)/X) = \kappa(G, \gamma)$ ,  $F := F' \cup X$  is acyclic  $\gamma$ -nonzero in  $G$  by (1) of Lemma 23. Therefore,  $F$  is an acyclic  $\gamma$ -nonzero set in  $G$  such that  $X \subseteq F$  and  $Y \cap F = \emptyset$ .

Now let us prove the backward direction. Let  $F$  be an acyclic  $\gamma$ -nonzero set in  $G$  such that  $X \subseteq F$  and  $Y \cap F = \emptyset$ . Let  $\gamma' : V(G/X) \rightarrow \Gamma$  be a map such that  $(G/X, \gamma') = (G, \gamma)/X$ . Then  $F - X$  is an acyclic  $\gamma'$ -nonzero set in  $G/X$  not intersecting  $Y$ , so we have  $\kappa((G, \gamma)/X \setminus Y) = \kappa((G, \gamma)/X)$  by (2) of Lemma 24. Since  $F$  is an acyclic  $\gamma$ -nonzero set containing  $X$  in  $G$ , we have  $\kappa((G, \gamma)/X) = \kappa(G, \gamma)$  by (2) of Lemma 23. ◀

**Proof of Theorem 21.** Given a  $\Gamma$ -labelled graph  $(G, \gamma)$  and disjoint subsets  $X, Y$  of  $E(G)$ , we can compute  $\kappa((G, \gamma)/X \setminus Y)$  in polynomial time and therefore, by Proposition 25, we can decide whether there exists an acyclic  $\gamma$ -nonzero set  $F$  in  $G$  such that  $X \subseteq F$  and  $Y \cap F = \emptyset$ . ◀

Now we are ready to show Theorem 2

► **Theorem 2.** *MAXIMUM WEIGHT ACYCLIC  $\gamma$ -NONZERO SET is solvable in polynomial time.*



**Proof.** Let  $M = \mathcal{G}(G, \gamma)$  be a  $\Gamma$ -graphic delta-matroid. The set of acyclic  $\gamma$ -nonzero sets in  $G$  is equal to the set of feasible sets of  $M$ . By Theorem 21, we can decide in polynomial time whether a pair  $(X, Y)$  of disjoint subsets  $X$  and  $Y$  of  $E(G)$  is separable in  $M$ . It implies that the symmetric greedy algorithm in Algorithm 1 for  $M$  and  $w$  runs in polynomial time. By Theorem 14, we can obtain an acyclic  $\gamma$ -nonzero set  $F$  in  $G$  maximizing  $\sum_{e \in F} w(e)$ . ◀

## 6 Even $\Gamma$ -graphic delta-matroids

In this section, we show that every even  $\Gamma$ -graphic delta-matroid is graphic.

► **Lemma 26** (\*). *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph, and  $\eta : V(G) \rightarrow \mathbb{Z}_2$  such that  $\eta(v) = 0$  if and only if  $\gamma(v) = 0$  for each  $v \in V(G)$ . If  $\mathcal{G}(G, \gamma)$  is even, then, for each connected subgraph  $H$  of  $G$ ,  $\sum_{u \in V(H)} \eta(u) = 0$  if and only if  $\sum_{u \in V(H)} \gamma(u) = 0$ .*

► **Proposition 27.** *Let  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. If  $\mathcal{G}(G, \gamma)$  is even, then there is a map  $\eta : V(G) \rightarrow \mathbb{Z}_2$  such that  $\mathcal{G}(G, \gamma) = \mathcal{G}(G, \eta)$ .*

**Proof.** Let  $\eta : V(G) \rightarrow \mathbb{Z}_2$  is a map such that, for every  $u \in V(G)$ ,  $\eta(u) = 0$  if and only if  $\gamma(u) = 0$ . Let  $F$  be a set of edges of  $G$ . Then, for each component  $C$  of  $(V(G), F)$ ,  $\gamma|_{V(C)} \equiv 0$  if and only if  $\eta|_{V(C)} \equiv 0$  and, by Lemma 26,  $\sum_{u \in V(C)} \gamma(u) \neq 0$  if and only if  $\sum_{u \in V(C)} \eta(u) \neq 0$ . Therefore,  $F$  is acyclic  $\gamma$ -nonzero in  $G$  if and only if it is acyclic  $\eta$ -nonzero in  $G$ . ◀

We are ready to prove Theorem 5.

► **Theorem 5.** *Let  $\Gamma$  be an abelian group. Then a  $\Gamma$ -graphic delta-matroid is even if and only if it is graphic.*

**Proof of Theorem 5.** Let  $M$  be an even  $\Gamma$ -graphic delta-matroid. By twisting, we may assume that  $M = \mathcal{G}(G, \gamma)$  for a  $\Gamma$ -labelled graph  $(G, \gamma)$ . By Proposition 27,  $M$  is  $\mathbb{Z}_2$ -graphic. Conversely, Oum [8, Theorem 5] proved that every graphic delta-matroid is even. ◀

## 7 Representations of $\Gamma$ -graphic delta-matroids

We aim to study the condition on an abelian group  $\Gamma$  and a field  $\mathbb{F}$  such that every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ . Recall that a delta-matroid  $M = (E, \mathcal{F})$  is representable over  $\mathbb{F}$  if there is an  $E \times E$  symmetric or skew-symmetric  $A$  over  $\mathbb{F}$  such that  $\mathcal{F} = \{F \subseteq E : A[X]$  is nonsingular $\} \Delta X$  for some  $X \subseteq E$ . If every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ , then to prove this, we will construct symmetric matrices over  $\mathbb{F}$  representing  $\Gamma$ -graphic delta-matroids.

For a graph  $G = (V, E)$ , let  $\vec{G}$  be an orientation obtained from  $G$  by arbitrarily assigning a direction to each edge. Let  $I_{\vec{G}} = (a_{ve})_{v \in V, e \in E}$  be a  $V \times E$  matrix over  $\mathbb{F}$  such that, for a vertex  $v \in V$  and an edge  $e \in E$ ,

$$a_{ve} = \begin{cases} 1 & \text{if } v \text{ is the head of a non-loop edge } e \text{ in } \vec{G}, \\ -1 & \text{if } v \text{ is the tail of a non-loop edge } e \text{ in } \vec{G}, \\ 0 & \text{otherwise.} \end{cases}$$

► **Lemma 28.** *Let  $G = (V, E)$  be a graph and  $\vec{G}_1, \vec{G}_2$  be orientations of  $G$ . If  $W \subseteq V$ ,  $F \subseteq E$ , and  $|W| = |F|$ , then  $\det(I_{\vec{G}_1}[W, F]) = \pm \det(I_{\vec{G}_2}[W, F])$ .*

## 70:12 $\Gamma$ -Graphic Delta-Matroids and Their Applications

**Proof.** The matrix  $I_{\vec{G}_1}$  can be obtained from  $I_{\vec{G}_2}$  by multiplying  $-1$  to some columns. ◀

By slightly abusing the notation, we simply write  $I_G$  to denote  $I_{\vec{G}}$  for some orientation  $\vec{G}$  of  $G$ . The following two lemmas are easy exercises.

► **Lemma 29** (see Oxley [9, Lemma 5.1.3]). *Let  $G$  be a graph and  $F$  be an edge set of  $G$ . Then  $F$  is acyclic if and only if column vectors of  $I_G$  indexed by the elements of  $F$  are linearly independent.*

► **Lemma 30** (see Matoušek and Nešetřil [6, Lemma 8.5.3]). *Let  $G = (V, E)$  be a tree. Then  $\det(I_G[V - \{v\}, E]) = \pm 1$  for any vertex  $v \in V$ .*

► **Lemma 31** (\*). *Let  $\Gamma$  be an abelian group with at least one nonzero element, and  $(G, \gamma)$  be a  $\Gamma$ -labelled graph. Then there is a  $\Gamma$ -labelled graph  $(H, \eta)$  such that*

- (i)  $\eta(v) \neq 0$  for each vertex  $v \in V(H)$  and
- (ii)  $(G, \gamma)$  is a minor of  $(H, \eta)$ .

► **Theorem 32** (Binet-Cauchy theorem). *Let  $X$  and  $Y$  be finite sets. Let  $M$  be an  $X \times Y$  matrix and  $N$  be a  $Y \times X$  matrix with  $|Y| \geq |X| = s$ . Then*

$$\det(MN) = \sum_{S \in \binom{Y}{s}} \det(M[X, S]) \cdot \det(N[S, X]).$$

It is straightforward to prove the following lemma from the Binet-Cauchy theorem.

► **Corollary 33.** *Let  $X, Y, Z$  be finite sets. Let  $L, M, N$  be  $X \times Y, Y \times Z, Z \times X$  matrices, respectively, with  $|Y|, |Z| \geq |X| = s$ . Then*

$$\det(LMN) = \sum_{S \in \binom{Y}{s}, T \in \binom{Z}{s}} \det(L[X, S]) \cdot \det(M[S, T]) \cdot \det(N[T, X]).$$

► **Theorem 6** (\*). *Let  $p$  be a prime,  $k$  be a positive integer, and  $\mathbb{F}$  be a field of characteristic  $p$ . If  $[\mathbb{F} : \text{GF}(p)] \geq k$ , then every  $\mathbb{Z}_p^k$ -graphic delta-matroid is representable over  $\mathbb{F}$ .*

Now we show that for some pairs of an abelian group  $\Gamma$  and a finite field  $\mathbb{F}$ , not every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ . Let  $R(n; m)$  be the Ramsey number that is the minimum integer  $t$  such that any coloring of edges of  $K_t$  into  $m$  colors induces a monochromatic copy of  $K_n$ .

► **Theorem 34** (Ramsey [10]). *For positive integers  $m$  and  $n$ ,  $R(n; m)$  is finite.*

► **Corollary 35.** *Let  $k$  be a positive integer and  $\mathbb{F}$  be a finite field of order  $m$ . If  $N \geq R(k; m)$ , then each  $N \times N$  symmetric matrix  $A$  over  $\mathbb{F}$  has a  $k \times k$  principal submatrix  $A'$  such that all non-diagonal entries are equal.*

► **Lemma 36** (\*). *Let  $\mathbb{F}$  be a field. If every  $\mathbb{Z}_2$ -graphic delta-matroid is representable over  $\mathbb{F}$ , then the characteristic of  $\mathbb{F}$  is 2.*

► **Theorem 7** (\*). *Let  $\mathbb{F}$  be a finite field of characteristic  $p$ , and  $\Gamma$  be an abelian group. If every  $\Gamma$ -graphic delta-matroid is representable over  $\mathbb{F}$ , then  $\Gamma$  is an elementary abelian  $p$ -group.*

---

**References**

---

- 1 André Bouchet. Greedy algorithm and symmetric matroids. *Mathematical Programming*, 38(2):147–159, 1987. doi:10.1007/BF02604639.
- 2 André Bouchet. Representability of  $\Delta$ -matroids. In *Combinatorics (Eger, 1987)*, volume 52 of *Colloq. Math. Soc. János Bolyai*, pages 167–182. North-Holland, Amsterdam, 1988.
- 3 André Bouchet. Maps and  $\Delta$ -matroids. *Discrete Mathematics*, 78(1-2):59–71, 1989. doi:10.1016/0012-365X(89)90161-1.
- 4 André Bouchet and Alain Duchamp. Representability of  $\Delta$ -matroids over  $\text{GF}(2)$ . *Linear Algebra Appl.*, 146:67–78, 1991. doi:10.1016/0024-3795(91)90020-W.
- 5 James Ferdinand Geelen. *Matchings, matroids and unimodular matrices*. ProQuest LLC, Ann Arbor, MI, 1996. Thesis (Ph.D.)—University of Waterloo (Canada).
- 6 Jiří Matoušek and Jaroslav Nešetřil. *Invitation to discrete mathematics*. Oxford University Press, Oxford, second edition, 2009.
- 7 Iain Moffatt. Delta-matroids for graph theorists. In *Surveys in combinatorics 2019*, volume 456 of *London Math. Soc. Lecture Note Ser.*, pages 167–220. Cambridge Univ. Press, Cambridge, 2019.
- 8 Sang-il Oum. Excluding a bipartite circle graph from line graphs. *Journal of Graph Theory*, 60(3):183–203, 2009. doi:10.1002/jgt.20353.
- 9 James Oxley. *Matroid theory*, volume 21 of *Oxford Graduate Texts in Mathematics*. Oxford University Press, Oxford, second edition, 2011. doi:10.1093/acprof:oso/9780198566946.001.0001.
- 10 Frank P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30(s2):264–286, 1930. doi:10.1112/plms/s2-30.1.264.
- 11 Alan W. Tucker. A combinatorial equivalence of matrices. In *Proc. Sympos. Appl. Math., Vol. 10*, pages 129–140. American Mathematical Society, Providence, R.I., 1960.



# An Approximation Algorithm for Maximum Stable Matching with Ties and Constraints

Yu Yokoi  

National Institute of Informatics, Hitotsubashi, Chiyoda-ku, Tokyo, Japan

---

## Abstract

We present a polynomial-time  $\frac{3}{2}$ -approximation algorithm for the problem of finding a maximum-cardinality stable matching in a many-to-many matching model with ties and laminar constraints on both sides. We formulate our problem using a bipartite multigraph whose vertices are called workers and firms, and edges are called contracts. Our algorithm is described as the computation of a stable matching in an auxiliary instance, in which each contract is replaced with three of its copies and all agents have strict preferences on the copied contracts. The construction of this auxiliary instance is symmetric for the two sides, which facilitates a simple symmetric analysis. We use the notion of matroid-kernel for computation in the auxiliary instance and exploit the base-orderability of laminar matroids to show the approximation ratio.

In a special case in which each worker is assigned at most one contract and each firm has a strict preference, our algorithm defines a  $\frac{3}{2}$ -approximation mechanism that is strategy-proof for workers.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Theory of computation → Algorithmic game theory

**Keywords and phrases** Stable matching, Approximation algorithm, Matroid, Strategy-proofness

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.71

**Related Version** *Full Version*: <https://arxiv.org/abs/2107.03076>

**Funding** This work was supported by JSPS KAKENHI Grant Number JP18K18004, JST PRESTO Grant Number JPMJPR212B, and the joint project of Kyoto University and Toyota Motor Corporation, titled “Advanced Mathematical Science for Mobility Society”.

**Acknowledgements** The author thanks the anonymous reviewers for their helpful comments.

## 1 Introduction

The *college admission problem* (CA) is a many-to-one generalization of the well-known *stable marriage problem* [18,32,34], introduced by Gale and Shapley [16]. An instance of CA involves two disjoint agent sets called students and colleges. Each agent has a strict linear order of preference over agents on the opposite side, and each college has an upper quota for the number of assigned students. It is known that any instance of CA has a stable matching, we can find it efficiently, and all stable matchings have the same cardinality.

Recently, matching problems with constraints have been studied extensively [6,9,15,27,28]. Motivated by the matching scheme used in the higher education sector in Hungary, Biró et al. [4] studied CA *with common quotas*. In this problem, in addition to individual colleges, certain subsets of colleges, called *bounded sets*, have upper quotas. Such constraints are also called *regional caps* or *distributional constraints*, and they have been studied in [17,29]. Meanwhile, motivated by academic hiring, Huang [21] introduced the *classified stable matching problem*. This is an extension of CA in which each individual college has quotas for subsets of students, called *classes*. Its many-to-many generalizations have been studied in [14,44].<sup>1</sup>

---

<sup>1</sup> In [14,17,21,44], not only upper quotas but also lower quotas are considered. With lower quotas, the existence of stable matching is not guaranteed. In this paper, we consider only upper quotas.



© Yu Yokoi;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 71; pp. 71:1–71:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For these models, the laminar structure of constraints is commonly found to be the key to the existence of a stable matching. A family  $\mathcal{L}$  of sets is called *laminar* if any  $L, L' \in \mathcal{L}$  satisfy  $L \subseteq L'$  or  $L \supseteq L'$  or  $L \cap L' = \emptyset$  (also called *nested* or *hierarchical*). In [4, 21], the authors showed that a stable matching exists in their models if regions or classes form laminar families, whereas the existence is not guaranteed in the general case. Furthermore, in the laminar case, a stable matching can be found efficiently, and all stable matchings have the same cardinality. Applications with laminar constraints have been discussed in [29].

The purpose of this paper is to introduce ties to a matching model with laminar constraints. In the previous studies described above, the preferences of agents were assumed to be strictly ordered. However, ties naturally arise in real problems. Matching models with ties have been studied widely in the literature [18, 23, 34], where the preference of an agent is said to contain a *tie* if she is indifferent between two or more agents on the opposite side. When ties are allowed, the existence of a stable matching is maintained; however, stable matchings vary in cardinalities. As it is desirable to produce a large matching in practical applications, we consider the problem of finding a maximum-cardinality stable matching.

Such a problem is known to be difficult even in the simple matching model without constraints. The problem of finding a maximum stable matching in the setting of *stable marriage with ties and incomplete lists*, called MAX-SMTI, is NP-hard [24, 35], as is obtaining an approximation ratio within  $\frac{33}{29}$  [43]. For its approximability, several algorithms with improved approximation ratios have been proposed [25, 26, 30, 31, 36, 38]. The current best ratio is  $\frac{3}{2}$  by a polynomial-time algorithm proposed by McDermid [36] as well as linear-time algorithms proposed by Paluch [38] and Király [31]. The  $\frac{3}{2}$ -approximability extends to the settings of CA with ties [31] and the student-project allocation problem with ties [8].

**Our Contribution.** We present a polynomial-time  $\frac{3}{2}$ -approximation algorithm for the problem of finding a maximum-cardinality stable matching in a many-to-many matching model with ties and laminar constraints on both sides. We call this problem MAX-SMTI-LC and formulate it using a bipartite multigraph, where we call the two vertex sets *workers* and *firms*, respectively, and each edge a *contract*. Each agent has upper quotas on a laminar family defined on incident contracts. Our formulation can deal with each agent's constraints, such as *classified stable matching*. Furthermore, distributional constraints such as CA *with common quotas* can be handled by considering a dummy agent that represents a consortium of the agents on one side (see the remark at the end of Section 2). Our algorithm runs in  $O(k \cdot |E|^2)$  time, where  $E$  is the set of contracts and  $k$  is the maximum level of nesting of laminar constraints. The *level of nesting* of a laminar family  $\mathcal{L}$  is the maximum length of a chain  $L_1 \subsetneq L_2 \subsetneq \dots \subsetneq L_k$  of members of  $\mathcal{L}$ ; hence,  $k \leq |E|$ .

Our algorithm is described as the computation of a stable matching in an auxiliary instance. Here, we explain the ideas underlying the construction of the auxiliary instance, which is inspired by the algorithms of Király [31] and Hamada, Miyazaki, and Yanagisawa [19].

First, we briefly explain Király's  $\frac{3}{2}$ -approximation algorithm for MAX-SMTI [31]. In this algorithm, each worker makes proposals from top to bottom in her list sequentially, as with the worker-oriented Gale–Shapley algorithm. A worker rejected by all firms is given a second chance for proposals. Each firm prioritizes a worker in the second cycle over a worker in the first cycle if they are tied in its preference list. This idea of *promotion* is used to handle ties in firms' preference lists. To handle ties in workers' lists, Király's algorithm lets each worker prioritize a currently unmatched firm over a currently matched firm if they are tied in her preference list. This priority rule depends on the states of firms at each moment, which makes the algorithm complicated when we introduce constraints on both sides.

Then, we introduce the idea of the algorithm of Hamada et al. [19], who proposed a worker-strategy-proof algorithm for MAX-SMTI that attains the  $\frac{3}{2}$ -approximation ratio when ties appear only in workers' lists. They modified Király's algorithm such that each worker's proposal order is predetermined and is not affected by the history of the algorithm. Their algorithm can be seen as a Gale–Shapley-type algorithm in which each worker makes proposals twice to each firm in a tie before proceeding to the next tie, and each firm prioritizes second proposals over first proposals regardless of its preference. By combining their algorithm with the promotion operation of Király's algorithm, we obtain a Gale–Shapley-type algorithm in which each worker makes at most three proposals to each firm.

Based on these observations, we propose a method for transforming a MAX-SMTI-LC instance  $I$  into an auxiliary instance  $I^*$ , which is also a MAX-SMTI-LC instance. Each contract  $e_i$  in  $I$  is replaced with three copies  $x_i, y_i, z_i$  in  $I^*$ . Each agent has a strict preference on the copied contracts, which reflects the priority rules in the algorithms of Király and Hamada et al. The instance  $I^*$  has an upper bound 1 for each triple  $\{x_i, y_i, z_i\}$  and also has constraints corresponding to those in  $I$ . The construction of  $I^*$  is completely symmetric for workers and firms. We show that, for any stable matching  $M^*$  of  $I^*$ , its projection  $M := \{e_i \mid \{x_i, y_i, z_i\} \cap M^* \neq \emptyset\}$  is a  $\frac{3}{2}$ -approximate solution for  $I$ . Both the stability and the approximation ratio of  $M$  are implied by the stability of  $M^*$  in  $I^*$ , and the process of computing  $M^*$  is irrelevant. Thus, our method enables us to conduct a symmetric and static analysis even with constraints.

Because the auxiliary instance  $I^*$  has no ties, we can find a stable matching of  $I^*$  efficiently by using the matroid framework of Fleiner [12, 13]. In the analysis of the approximation ratio, we exploit the fact that the family of feasible sets defined by laminar constraints forms a matroid with a property called *base-orderability*.

In the last section, we show that the result of Hamada et al. [19] mentioned above is generalized to a many-to-one matching setting with laminar constraints on the firm side. In other words, if we restrict MAX-SMTI-LC such that each worker is assigned at most one contract and each firm has a strict preference, then we can provide a worker-strategy-proof mechanism that returns a  $\frac{3}{2}$ -approximate solution. We obtain this conclusion using the strategy-proofness result of Hatfield and Milgrom [20].

**Paper Organization.** The remainder of this paper is organized as follows. Section 2 formulates our matching model, while Section 3 describes our algorithm. Section 4 presents a lemma on base-orderable matroids that is the key to our proof of the approximation ratio. Sections 5 and 6 are devoted to the proofs of correctness and time complexity, respectively. Section 7 investigates strategy-proof approximation mechanisms for our model.

Throughout the paper, we denote the set of non-negative integers by  $\mathbf{Z}_+$ . For a subset  $S \subseteq E$  and an element  $e \in E$ , we denote  $S + e := S \cup \{e\}$  and  $S - e := S \setminus \{e\}$ .

## 2 Problem Formulation

An instance of the *stable matching with ties and laminar constraints*, which we call SMTI-LC, is a tuple  $I = (W, F, E, \{\mathcal{L}_a, q_a, P_a\}_{a \in W \cup F})$  defined as follows. Let  $W$  and  $F$  be disjoint finite sets called *workers* and *firms*, respectively. We call  $a \in W \cup F$  an *agent* when we do not distinguish between workers and firms. We are provided a set  $E$  of *contracts*. Each contract  $e \in E$  is associated with one worker and one firm, denoted by  $\partial_W(e)$  and  $\partial_F(e)$ , respectively. Multiple contracts are allowed to exist between a worker–firm pair. Then,  $(W, F; E)$  is represented as a bipartite multigraph in which  $W$  and  $F$  are vertex sets, and each  $e \in E$  is an edge connecting  $\partial_W(e)$  and  $\partial_F(e)$ . For each  $a \in W \cup F$ , we denote the set of associated contracts by  $E_a$ , i.e.,



## 71:4 Maximum Stable Matching with Ties and Constraints

$$E_w := \{e \in E \mid \partial_W(e) = w\} \quad (w \in W), \quad E_f := \{e \in E \mid \partial_F(e) = f\} \quad (f \in F).$$

Then, the family  $\{E_w \mid w \in W\}$  forms a partition of  $E$ , as does  $\{E_f \mid f \in F\}$ .

Each agent  $a \in W \cup F$  has a laminar family  $\mathcal{L}_a$  of subsets of  $E_a$  and a quota function  $q_a: \mathcal{L}_a \rightarrow \mathbf{Z}_+$ . For any subset  $M \subseteq E$  of contracts and an agent  $a \in W \cup F$ , we denote by  $M_a := M \cap E_a$  the set of contracts assigned to  $a$ . We say that  $M$  is *feasible* for  $a \in W \cup F$  if

$$\forall L \in \mathcal{L}_a: |M_a \cap L| \leq q_a(L).$$

A set  $M \subseteq E$  is called a *matching* if it is feasible for all agents in  $W \cup F$ .

Each agent  $a \in W \cup F$  has a preference list  $P_a$  that consists of all elements in  $E_a$  and may contain ties. In this paper, a preference list is written in one row, from left to right according to preference, where two or more contracts with equal preference are included in the same parentheses. For example, if the preference list  $P_a$  of an agent  $a \in W \cup F$  is represented as

$$P_a: e_2 \ (e_1 \ e_4) \ e_3,$$

then  $e_2$  is  $a$ 's top choice,  $e_1$  and  $e_4$  are the second choices with equal preference, and  $e_3$  is the last choice. For contracts  $e, e' \in E_a$ , we write  $e \succ_a e'$  if  $a$  prefers  $e$  to  $e'$ . Furthermore, we write  $e \succeq_a e'$  if  $e \succ_a e'$  or  $a$  is indifferent between  $e$  and  $e'$  (including the case  $e = e'$ ).

For a matching  $M \subseteq E$ , a contract  $e \in E \setminus M$ , and an associated agent  $a \in \{\partial_W(e), \partial_F(e)\}$ , we say that  $e$  is *free* for  $a$  in  $M$  if

- $M_a + e$  is feasible for  $a$ , or
- there is  $e' \in M_a$  such that  $e \succ_a e'$  and  $M_a + e - e'$  is feasible for  $a$ .

In other words, a contract  $e$  is free for an agent  $a$  if  $a$  has an incentive to add  $e$  to the current assignment possibly at the expense of some less preferred contract  $e'$ . A contract  $e \in E \setminus M$  *blocks*  $M$  if  $e$  is free for both  $\partial_W(e)$  and  $\partial_F(e)$ . A matching  $M$  is *stable* if there is no contract in  $E \setminus M$  that blocks  $M$ .

The goal of our problem MAX-SMTI-LC is to find a maximum-cardinality stable matching for a given SMTI-LC instance. Because MAX-SMTI-LC is a generalization of the NP-hard problem MAX-SMTI, we consider the approximability. Similarly to the case of MAX-SMTI, for the problem MAX-SMTI-LC, a 2-approximate solution can be easily obtained using an arbitrary tie-breaking method (see the full version [45] for the proof). In the next section, we present a  $\frac{3}{2}$ -approximation algorithm.

► **Remark.** We demonstrate that SMTI-LC includes several models investigated in previous works, which implies that our algorithm finds  $\frac{3}{2}$ -approximate solutions for the problems of finding maximum-cardinality stable matchings in those models with ties.

First, SMTI and the stable  $b$ -matching problem are special cases such that  $E \subseteq W \times F$  and  $\mathcal{L}_a = \{E_a\}$  for every  $a \in W \cup F$ . Furthermore, the two-sided laminar classified stable matching problem [14, 21], if lower quotas are absent, is a special case with  $E \subseteq W \times F$ .

To represent CA with laminar common quotas [4], let  $W$  be the set of students and let  $F := \{f\}$ , where  $f$  is regarded as a consortium of all colleges in  $C$ . The set of contracts is defined by  $E := \{(w, f, c) \mid \text{a college } c \in C \text{ is acceptable for a student } w \in W\}$ , where  $\partial_W(e) = w$ ,  $\partial_F(e) = f$  for any  $e = (w, f, c)$ . Note that  $E = E_f$ . A quota for a region  $C' \subseteq C$  is then represented as a quota for the set  $\{(w, f, c) \in E \mid c \in C'\} \subseteq E_f$ . Thus, laminar common quotas can be represented as constraints on a laminar family on  $E_f$ .

For the student-project allocation problem [8], let  $W$  and  $F$  be the sets of students and lecturers, respectively, and  $E := \{(w, f, p) \mid \text{a project } p \text{ acceptable for } w \in W \text{ is offered by } f \in F\}$ . Let  $E_{f,p} \subseteq E_f$  be the set of contracts associated with a project  $p$  offered by a lecturer  $f$ . Then, the lecturer's upper quota and projects' upper quotas define two-level laminar constraints on the family  $\mathcal{L}_f = \{E_f\} \cup \{E_{f,p} \mid p \text{ is offered by } f\}$ .

For the above-mentioned settings, we can appropriately set the preferences of agents such that the stability in the previous works coincides with the stability in SMTI-LC.

### 3 Algorithm

Our approximation algorithm for MAX-SMTI-LC consists of three steps: (i) construction of an auxiliary instance, (ii) computation of any stable matching of this auxiliary instance, and (iii) mapping the obtained matching to a matching of the original instance. In what follows, we describe how to construct an auxiliary instance  $I^*$  from a given instance  $I$  and how to map a matching of  $I^*$  to that of  $I$ .

Let  $I = (W, F, E, \{\mathcal{L}_a, q_a, P_a\}_{a \in W \cup F})$  be an instance of MAX-SMTI-LC, where the set  $E$  of contracts is represented as  $E = \{e_i \mid i = 1, 2, \dots, n\}$ . We construct an auxiliary instance  $I^* = (W, F, E^*, \{\mathcal{L}_a^*, q_a^*, P_a^*\}_{a \in W \cup F})$ , which is also an SMTI-LC instance; however, each preference list  $P_a^*$  does not contain ties.

The sets of workers and firms in  $I^*$  are the same as those in  $I$ . The set  $E^*$  of contracts in  $I^*$  is given as  $E^* = \{x_i, y_i, z_i \mid i = 1, 2, \dots, n\}$ , where  $x_i, y_i$ , and  $z_i$  are copies of  $e_i$ ; hence,  $\partial_W(x_i) = \partial_W(y_i) = \partial_W(z_i) = \partial_W(e_i)$  and  $\partial_F(x_i) = \partial_F(y_i) = \partial_F(z_i) = \partial_F(e_i)$ . We define a mapping  $\pi : 2^{E^*} \rightarrow 2^E$  by  $\pi(S^*) = \{e_i \mid \{x_i, y_i, z_i\} \cap S^* \neq \emptyset\}$  for any  $S^* \subseteq E^*$ .

For any agent  $a \in W \cup F$ , the laminar family  $\mathcal{L}_a^*$  and the quota function  $q_a^* : \mathcal{L}_a^* \rightarrow \mathbf{Z}_+$  are defined as follows. For each  $e_i \in E_a$ , we have  $\{x_i, y_i, z_i\} \in \mathcal{L}_a^*$  and  $q_a^*(\{x_i, y_i, z_i\}) = 1$ . For each  $L \in \mathcal{L}_a$ , we have  $L^* := \{x_i, y_i, z_i \mid e_i \in L\} \in \mathcal{L}_a^*$  and  $q_a^*(L^*) = q_a(L)$ . These are all that  $\mathcal{L}_a^*$  contains. Then, for any set  $M^* \subseteq E^*$  of contracts, we see that  $M^*$  is feasible for  $a$  in  $I^*$  if and only if  $M^*$  contains at most one copy of each  $e_i \in E_a$  and the set  $\pi(M^*)$  is feasible for  $a$  in  $I$ .

The preference list  $P_w^*$  of each worker  $w \in W$  is defined as follows. Take a tie  $(e_{i_1} e_{i_2} \cdots e_{i_\ell})$  in  $P_w$ . We replace it with a strict linear order of  $2\ell$  contracts  $x_{i_1} x_{i_2} \cdots x_{i_\ell} y_{i_1} y_{i_2} \cdots y_{i_\ell}$ . Apply this operation to all the ties in  $P_w$ , where we regard a contract not included in any tie as a tie of length one. Next, at the end of the resultant list, append the original list  $P_w$  with each  $e_i$  replaced with  $z_i$  and all the parentheses omitted. Here is a demonstration. If the preference list of a worker  $w$  is

$$P_w : ( e_2 \ e_6 ) \ e_1 \ ( e_3 \ e_4 ),$$

then her list in  $I^*$  is

$$P_w^* : x_2 \ x_6 \ y_2 \ y_6 \ x_1 \ y_1 \ x_3 \ x_4 \ y_3 \ y_4 \ z_2 \ z_6 \ z_1 \ z_3 \ z_4.$$

The preference list  $P_f^*$  of each firm  $f \in F$  is defined in the same manner, where the roles of  $x_i$  and  $z_i$  are interchanged. For example, if the preference list of a firm  $f$  is

$$P_f : e_3 \ ( e_2 \ e_4 \ e_7 ) \ e_5,$$

then its list in  $I^*$  is

$$P_f^* : z_3 \ y_3 \ z_2 \ z_4 \ z_7 \ y_2 \ y_4 \ y_7 \ z_5 \ y_5 \ x_3 \ x_2 \ x_4 \ x_7 \ x_5.$$

Thus, we have defined the auxiliary instance  $I^*$ . As this is again an SMTI-LC instance, a stable matching of  $I^*$  is defined as before. The existence of a stable matching of  $I^*$  is guaranteed by the existing framework of Fleiner [12,13], as will be explained in Section 6. Here is the main theorem of this paper, which states that any stable matching of  $I^*$  defines a  $\frac{3}{2}$ -approximate solution for  $I$ .

► **Theorem 1.** *For a stable matching  $M^*$  of  $I^*$ , let  $M := \pi(M^*)$ . Then,  $M$  is a stable matching of  $I$  with  $|M| \geq \frac{2}{3}|M_{\text{OPT}}|$ , where  $M_{\text{OPT}}$  is a maximum-cardinality stable matching of  $I$ .*

We prove Theorem 1 in Section 5. This theorem guarantees the correctness of Algorithm 1.

■ **Algorithm 1**  $\frac{3}{2}$ -approximation algorithm for MAX-SMTI-LC.

---

**Input:** An instance  $I = (W, F, E, \{\mathcal{L}_a, q_a, P_a\}_{a \in W \cup F})$ .

**Output:** A stable matching  $M$  with  $|M| \geq \frac{2}{3}|M_{\text{OPT}}|$ , where  $M_{\text{OPT}}$  is an optimal solution.

- 1: Construct an auxiliary instance  $I^*$ .
  - 2: Find any stable matching  $M^*$  of  $I^*$ .
  - 3: Let  $M = \pi(M^*)$  and return  $M$ .
- 

Clearly, the first and third steps of Algorithm 1 can be performed efficiently. Furthermore, the second step can be executed in polynomial time by applying the generalized Gale–Shapley algorithm of Fleiner [12,13]. In Section 6, we will explain this more precisely and present the time complexity represented in the following theorem.

► **Theorem 2.** *One can find a stable matching  $M$  of  $I$  with  $|M| \geq \frac{2}{3}|M_{\text{OPT}}|$  in  $O(k \cdot |E|^2)$  time, where  $M_{\text{OPT}}$  is a maximum-cardinality stable matching and  $k$  is the maximum level of nesting of laminar families  $\mathcal{L}_a$  ( $a \in W \cup F$ ).*

## 4 Base-orderable Matroids

For the proofs of Theorems 1 and 2, we introduce some concepts related to matroids (see, e.g., Oxley [37] for more information on matroids).

For a finite set  $E$  and a family  $\mathcal{I} \subseteq 2^E$ , a pair  $(E, \mathcal{I})$  is called a *matroid* if the following three conditions hold: (I1)  $\emptyset \in \mathcal{I}$ , (I2)  $S \subseteq T \in \mathcal{I}$  implies  $S \in \mathcal{I}$ , and (I3) for any  $S, T \in \mathcal{I}$  with  $|S| < |T|$ , there exists  $e \in T \setminus S$  such that  $S + e \in \mathcal{I}$ .

For a matroid  $(E, \mathcal{I})$ , each member of  $\mathcal{I}$  is called an *independent set*. An independent set is called a *base* if it is inclusion-wise maximal in  $\mathcal{I}$ . We denote the family of all bases by  $\mathcal{B}$ . By the matroid axiom (I3), it follows that  $|B_1| = |B_2|$  holds for any bases  $B_1, B_2 \in \mathcal{B}$ .

► **Definition 3** (Base-orderable Matroid). *A matroid  $(E, \mathcal{I})$  is called base-orderable if for any two bases  $B_1, B_2 \in \mathcal{B}$ , there exists a bijection  $\varphi: B_1 \rightarrow B_2$  with the property that, for every  $e \in B_1$ , both  $B_1 - e + \varphi(e)$  and  $B_2 + e - \varphi(e)$  are bases.*

A class of base-orderable matroids includes *gammoids* (see [7] and [42, Theorem 42.12]), and gammoids include laminar matroids described below (see [10] and [11, Section 2.3.1]).

► **Example 4** (Laminar Matroid). For a laminar family  $\mathcal{L}$  on  $E$  and a function  $q: \mathcal{L} \rightarrow \mathbf{Z}_+$ , define  $\mathcal{I} = \{S \subseteq E \mid \forall L \in \mathcal{L}: |S \cap L| \leq q(L)\}$ . Then,  $(E, \mathcal{I})$  is a base-orderable matroid.

A matroid is *laminar* if it can be defined in the above-mentioned manner for some  $\mathcal{L}$  and  $q$ .

Base-orderability is known to be closed under the following operations (see, e.g., [5, 22]).

**Contraction.**<sup>2</sup> For a matroid  $(E, \mathcal{I})$  and any  $S \in \mathcal{I}$ , define  $\mathcal{I}_S := \{T \subseteq E \setminus S \mid S \cup T \in \mathcal{I}\}$ . Then,  $(E \setminus S, \mathcal{I}_S)$  is a matroid. If  $(E, \mathcal{I})$  is base-orderable, then so is  $(E \setminus S, \mathcal{I}_S)$ .

**Truncation.** For a matroid  $(E, \mathcal{I})$  and any integer  $p \in \mathbf{Z}_+$ , define  $\mathcal{I}_p := \{S \in \mathcal{I} \mid |S| \leq p\}$ . Then,  $(E, \mathcal{I}_p)$  is a matroid. If  $(E, \mathcal{I})$  is base-orderable, then so is  $(E, \mathcal{I}_p)$ .

**Direct Sum.** For matroids  $(E_j, \mathcal{I}_j)$  ( $j = 1, 2, \dots, \ell$ ) such that  $E_j$  are all pairwise disjoint, let  $E := E_1 \cup E_2 \cup \dots \cup E_\ell$  and  $\mathcal{I} := \{S_1 \cup S_2 \cup \dots \cup S_\ell \mid S_j \in \mathcal{I}_j \text{ (} j = 1, 2, \dots, \ell)\}$ . Then,  $(E, \mathcal{I})$  is a matroid. If all  $(E_j, \mathcal{I}_j)$  are base-orderable, then so is  $(E, \mathcal{I})$ .

On the intersection of two base-orderable matroids, we show the following property, which plays a key role in proving the  $\frac{3}{2}$ -approximation ratio of our algorithm. This generalizes the fact that, if (one-to-one) bipartite matchings  $M$  and  $N$  satisfy  $|M| < \frac{2}{3}|N|$ , then  $M \Delta N$  contains a connected component that forms an alternating path of length at most three.

► **Lemma 5.** *For base-orderable matroids  $(E, \mathcal{I}_1)$  and  $(E, \mathcal{I}_2)$ , suppose that  $S, T \in \mathcal{I}_1 \cap \mathcal{I}_2$  and  $|S| < \frac{2}{3}|T|$ . If  $S + e \notin \mathcal{I}_1 \cap \mathcal{I}_2$  for every  $e \in T \setminus S$ , then there exist distinct elements  $e_i, e_j, e_k$  such that  $e_i, e_k \in T \setminus S$ ,  $e_j \in S \setminus T$ , and the following conditions hold:*

- $S + e_i \in \mathcal{I}_1$ ,
- both  $S + e_i - e_j$  and  $T - e_i + e_j$  belong to  $\mathcal{I}_2$ ,
- both  $S - e_j + e_k$  and  $T + e_j - e_k$  belong to  $\mathcal{I}_1$ ,
- $S + e_k \in \mathcal{I}_2$ .

**Proof.** By the matroid axiom (I3), there is a subset  $A_1 \subseteq T \setminus S$  such that  $|A_1| = |T| - |S|$  and  $S_1 := S \cup A_1 \in \mathcal{I}_1$ . Then,  $|S_1| = |T|$ ; hence,  $|S_1 \setminus T| = |T \setminus S_1|$ . Let  $(E', \mathcal{I}'_1)$  be a matroid obtained from  $(E, \mathcal{I}_1)$  by contracting  $S_1 \cap T$  and truncating with size  $|S_1 \setminus T|$ , i.e.,  $E' = E \setminus (S_1 \cap T)$  and  $\mathcal{I}'_1 := \{R \subseteq E' \mid R \cup (S_1 \cap T) \in \mathcal{I}_1, |R| \leq |S_1 \setminus T|\}$ . Then,  $S_1 \setminus T$  and  $T \setminus S_1$  are bases of  $(E', \mathcal{I}'_1)$ . As  $(E', \mathcal{I}'_1)$  is base-orderable, there is a bijection  $\varphi_1: S_1 \setminus T \rightarrow T \setminus S_1$  such that both  $(S_1 \setminus T) - e + \varphi_1(e)$  and  $(T \setminus S_1) + e - \varphi_1(e)$  are bases of  $(E', \mathcal{I}'_1)$  for every  $e \in S_1 \setminus T$ . By the definition of  $\mathcal{I}'_1$ , this implies that both  $S - e + \varphi_1(e)$  and  $T + e - \varphi_1(e)$  belong to  $\mathcal{I}_1$  for every  $e \in S_1 \setminus T$ . By the same argument, there exists  $A_2 \subseteq T \setminus S$  such that  $|A_2| = |T| - |S|$  and  $S_2 := S \cup A_2 \in \mathcal{I}_2$ , and there exists a bijection  $\varphi_2: S_2 \setminus T \rightarrow T \setminus S_2$  such that both  $S - e + \varphi_2(e)$  and  $T + e - \varphi_2(e)$  belong to  $\mathcal{I}_2$  for every  $e \in S_2 \setminus T$ .

We represent  $\varphi_1$  and  $\varphi_2$  using a bipartite graph as follows. Note that, for each  $\ell \in \{1, 2\}$ , we have  $S_\ell \setminus T = S \setminus T$  and  $T \setminus S_\ell = T \setminus (S \cup A_\ell) \subseteq T \setminus S$ . Let  $S \setminus T$  and  $T \setminus S$  be two vertex sets and let  $M_\ell := \{(e, \varphi_\ell(e)) \mid e \in S \setminus T\}$  for  $\ell = 1, 2$ . Then, each  $M_\ell$  is a one-to-one matching that covers  $S \setminus T$  and  $T \setminus (S \cup A_\ell)$ . Note that the sets  $A_1, A_2 \subseteq S \setminus T$  are mutually disjoint since, otherwise, some  $e \in A_1 \cap A_2$  satisfies  $S + e \in \mathcal{I}_1 \cap \mathcal{I}_2$ , which contradicts the assumption. Then,  $|T \setminus (S \cup A_1 \cup A_2)| = |T \setminus S| - |A_1| - |A_2| = |T \setminus S| - 2|T| + 2|S|$ . Therefore, at most  $2(|T \setminus S| - 2|T| + 2|S|)$  vertices in  $S \setminus T$  are adjacent to  $T \setminus (S \cup A_1 \cup A_2)$  via the edges in  $M_1 \cup M_2$ . Because  $|S \setminus T| - 2(|T \setminus S| - 2|T| + 2|S|) = -3|S| + 2|T| + |S \cap T| > -3 \cdot \frac{2}{3}|T| + 2|T| + |S \cap T| \geq 0$ , there exists  $\tilde{e} \in S \setminus T$  that is not adjacent to  $T \setminus (S \cup A_1 \cup A_2)$  via  $M_1 \cup M_2$ . This implies that  $\varphi_2(\tilde{e}) \in A_1$  and  $\varphi_1(\tilde{e}) \in A_2$ ; hence,  $S + \varphi_2(\tilde{e}) \in \mathcal{I}_1$  and  $S + \varphi_1(\tilde{e}) \in \mathcal{I}_2$ . Let  $e_i := \varphi_2(\tilde{e})$ ,  $e_j := \tilde{e}$ , and  $e_k := \varphi_1(\tilde{e})$ . Then, these three elements satisfy all the required conditions. ◀

<sup>2</sup> Contraction is defined for any subset of  $E$  [37]; however this paper uses only contraction by independent sets.

## 5 Correctness

This section is devoted to showing Theorem 1, which establishes the correctness of Algorithm 1.

As in Section 3, let  $I$  be an SMTI-LC instance with  $E = \{e_i \mid i = 1, 2, \dots, n\}$  and let  $I^*$  be the auxiliary instance  $I^*$ , whose contract set is  $E^* = \{x_i, y_i, z_i \mid i = 1, 2, \dots, n\}$ .

For any agent  $a \in W \cup F$ , let  $E_a^* = \{x_i, y_i, z_i \mid e_i \in E_a\}$  and define families  $\mathcal{I}_a$  and  $\mathcal{I}_a^*$  by

$$\begin{aligned}\mathcal{I}_a &= \{S \subseteq E_a \mid \forall L \in \mathcal{L}_a : |S \cap L| \leq q_a(L)\}, \\ \mathcal{I}_a^* &= \{S^* \subseteq E_a^* \mid \forall L^* \in \mathcal{L}_a^* : |S^* \cap L^*| \leq q_a^*(L^*)\},\end{aligned}$$

i.e.,  $\mathcal{I}_a$  and  $\mathcal{I}_a^*$  are the families of feasible sets in  $I$  and  $I^*$ , respectively. Then,  $(E_a, \mathcal{I}_a)$  and  $(E_a^*, \mathcal{I}_a^*)$  are laminar matroids and base-orderable. The definitions of  $\mathcal{L}_a^*$  and  $q_a^*$  imply the following fact. Recall that  $\pi : 2^{E^*} \rightarrow 2^E$  is defined by  $\pi(S^*) = \{e_i \mid \{x_i, y_i, z_i\} \cap S^* \neq \emptyset\}$ .

► **Observation 6.** *For a set  $S^* \subseteq E_a^*$ , we have  $S^* \in \mathcal{I}_a^*$  if and only if  $|\{x_i, y_i, z_i\} \cap S^*| \leq 1$  for every  $e_i \in E_a$  and  $\pi(S^*) \in \mathcal{I}_a$ .*

Take any stable matching  $M^*$  of  $I^*$  and let  $M := \pi(M^*)$ . As  $M^*$  is feasible in  $I^*$ , it contains at most one copy of each contract  $e_i$ . For any  $e_i \in M$ , we denote by  $\pi^{-1}(e_i)$  the unique element in  $\{x_i, y_i, z_i\} \cap M^*$ .

By the definitions of the preference lists  $\{P_a^*\}_{a \in W \cup F}$  in  $I^*$ , we can observe the following properties. For any agent  $a \in W \cup F$  and contracts  $e, e' \in E_a^*$ , we write  $e \succ_a^* e'$  if  $a$  prefers  $e$  to  $e'$  with respect to  $P_a^*$ . Recall that  $P_a^*$  does not contain ties, while  $P_a$  may contain.

► **Observation 7.** *For any  $e_i \in E \setminus M$  and  $e_j \in M$ , the following conditions hold.*

- *For any agent  $a \in W \cup F$ , if  $e_i, e_j \in E_a$  and  $e_i \succ_a e_j$ , then  $y_i \succ_a^* \pi^{-1}(e_j)$  holds regardless of which of  $\{x_i, y_i, z_i\}$  is  $\pi^{-1}(e_i)$ .*
- *For any worker  $w \in W$ , if  $e_i, e_j \in E_w$  and  $\pi^{-1}(e_j) \succ_w^* x_i$ , then we have either  $[\pi^{-1}(e_j) = x_j \text{ and } e_j \succeq_w e_i]$  or  $[\pi^{-1}(e_j) = y_j \text{ and } e_j \succ_w e_i]$ .*
- *For any firm  $f \in F$ , if  $e_i, e_j \in E_f$  and  $\pi^{-1}(e_j) \succ_f^* z_i$ , then we have either  $[\pi^{-1}(e_j) = z_j \text{ and } e_j \succeq_f e_i]$  or  $[\pi^{-1}(e_j) = y_j \text{ and } e_j \succ_f e_i]$ .*

First, we show the stability of  $M$  in  $I$ . For each agent  $a \in W \cup F$ , we write  $M_a^* = M^* \cap E_a^*$ , which implies that  $\pi(M_a^*) = M_a$ .

► **Lemma 8.** *The set  $M$  is a stable matching of  $I$ .*

**Proof.** Since  $M^*$  is feasible for all agents in  $I^*$ , Observation 6 implies that  $M = \pi(M^*)$  is feasible for all agents in  $I$ , i.e.,  $M$  is a matching in  $I$ .

Suppose, to the contrary, that  $M$  is not stable. Then, some contract  $e_i \in E \setminus M$  blocks  $M$ . Let  $w = \partial_W(e_i)$  and  $f = \partial_F(e_i)$ . Then,  $e_i$  is free for both  $w$  and  $f$  in  $M$ . We now show that  $y_i$  is free for both  $w$  and  $f$  in  $M^*$ , which contradicts the stability of  $M^*$ .

As  $e_i$  is free for  $w$  in  $I$ , we have (i)  $M_w + e_i \in \mathcal{I}_w$  or (ii) there exists  $e_j \in M_a$  such that  $e_i \succ_w e_j$  and  $M_a + e_i - e_j \in \mathcal{I}_w$ . Note that  $e_i \in E \setminus M$  implies  $\{x_i, y_i, z_i\} \cap M^* = \emptyset$ . In case (i), we have  $\pi(M_w^* + y_i) = M_w + e_i \in \mathcal{I}_w$ , which implies  $M_w^* + y_i \in \mathcal{I}_w^*$ ; hence,  $y_i$  is free for  $w$  in  $M^*$ . In case (ii), we have  $\pi(M_w^* + y_i - \pi^{-1}(e_j)) = M_w + e_i - e_j \in \mathcal{I}_w$ , which implies  $M_w^* + y_i - \pi^{-1}(e_j) \in \mathcal{I}_w^*$ . Furthermore, as  $e_i \succ_w e_j$ , the first statement of Observation 7 implies  $y_i \succ_w^* \pi^{-1}(e_j)$ . Thus, in each case,  $y_i$  is free for  $w$  in  $M^*$ .

Similarly, we can show that  $y_i$  is free for  $f$  in  $M^*$ . Thus,  $y_i$  blocks  $M^*$ , a contradiction. ◀

Next, we show the approximation ratio using Lemma 5. Note that  $\{E_w \mid w \in W\}$  is a partition of  $E$ , as is  $\{E_f \mid f \in F\}$ . Let  $(E, \mathcal{I}_W)$  be the direct sum of base-orderable matroids  $\{(E_w, \mathcal{I}_w) \mid w \in W\}$  and  $(E, \mathcal{I}_F)$  be the direct sum of  $\{(E_f, \mathcal{I}_f) \mid f \in F\}$ . Then, they are both base-orderable matroids on  $E$ .

By the definitions of  $\mathcal{I}_W$  and  $\mathcal{I}_F$ , for any subset  $N \subseteq E$ , we have  $N \in \mathcal{I}_W \cap \mathcal{I}_F$  if and only if  $N_a := N \cap E_a$  is feasible for each  $a \in W \cup F$ , i.e.,  $N$  is a matching. Furthermore, for any matching  $N \in \mathcal{I}_W \cup \mathcal{I}_F$  and contract  $e_i \in E \setminus N$ , which is associated with a worker  $w = \partial_W(e_i)$  (and a firm  $f = \partial_F(e_i)$ ), the condition  $N + e_i \in \mathcal{I}_W$  is equivalent to  $N_w + e_i \in \mathcal{I}_w$ . In addition, if  $N + e_i \notin \mathcal{I}_W$ , we have  $N + e_i - e_j \in \mathcal{I}_W$  if and only if  $e_j \in N_w$  and  $N_w + e_i - e_j \in \mathcal{I}_w$ . The same statements hold when  $w$  and  $W$  are replaced with  $f$  and  $F$ , respectively.

► **Lemma 9.** *The set  $M$  satisfies  $|M| \geq \frac{2}{3}|M_{\text{OPT}}|$ , where  $M_{\text{OPT}}$  is a maximum-cardinality stable matching of  $I$ .*

**Proof.** Set  $N := M_{\text{OPT}}$  for notational simplicity. Since  $M$  and  $N$  are stable matchings,  $M, N \in \mathcal{I}_W \cap \mathcal{I}_F$ . In addition,  $M + e_i \notin \mathcal{I}_W \cap \mathcal{I}_F$  for any  $e_i \in N \setminus M$  since, otherwise,  $e_i$  blocks  $M$ . Suppose, to the contrary, that  $|M| < \frac{2}{3}|N|$ . Then, by Lemma 5 and the definitions of  $\mathcal{I}_W$  and  $\mathcal{I}_F$ , there exist three contracts  $e_i, e_j, e_k$  such that  $e_i, e_k \in N \setminus M$ ,  $e_j \in M \setminus N$ , and the following conditions hold:

- $M_w + e_i \in \mathcal{I}_w$ ,
- both  $M_f + e_i - e_j$  and  $N_f - e_i + e_j$  belong to  $\mathcal{I}_f$ ,
- both  $M_{w'} - e_j + e_k$  and  $N_{w'} + e_j - e_k$  belong to  $\mathcal{I}_{w'}$ ,
- $M_{f'} + e_k \in \mathcal{I}_{f'}$ ,

where  $w = \partial_W(e_i)$ ,  $f = \partial_F(e_i) = \partial_F(e_j)$ ,  $w' = \partial_W(e_j) = \partial_W(e_k)$ ,  $f' = \partial_F(e_k)$ .

Since  $e_i \notin M$  and  $M_w + e_i \in \mathcal{I}_w$ , we have  $M_w^* + z_i \in \mathcal{I}_w^*$ ; hence,  $z_i$  is free for the worker  $w = \partial_W(z_i)$  in  $M^*$ . Then, the stability of  $M^*$  implies that  $z_i$  is not free for the firm  $f = \partial_F(z_i)$ . Since  $\pi(M_f^* + z_i - \pi^{-1}(e_j)) = M_f + e_i - e_j \in \mathcal{I}_f$  implies  $M_f^* + z_i - \pi^{-1}(e_j) \in \mathcal{I}_f^*$ , we should have  $\pi^{-1}(e_j) \succ_f^* z_i$ . Then, the third statement of Observation 7 implies that we have either  $[\pi^{-1}(e_j) = z_j$  and  $e_j \succeq_f e_i]$  or  $[\pi^{-1}(e_j) = y_j$  and  $e_j \succ_f e_i]$ .

Meanwhile, since  $e_k \notin M$  and  $M_{f'} + e_k \in \mathcal{I}_{f'}$ , we have  $M_{f'}^* + x_k \in \mathcal{I}_{f'}^*$ ; hence,  $x_k$  is free for the firm  $f' = \partial_W(x_k)$  in  $M^*$ . As  $M^*$  is stable, then  $x_k$  is not free for the worker  $w' = \partial_W(x_k)$ . Since  $\pi(M_{w'}^* + x_k - \pi^{-1}(e_j)) = M_{w'} + e_k - e_j \in \mathcal{I}_{w'}$  implies  $M_{w'}^* + x_k - \pi^{-1}(e_j) \in \mathcal{I}_{w'}^*$ , we should have  $\pi^{-1}(e_j) \succ_{w'}^* x_k$ . Then, the second statement of Observation 7 implies that we have either  $[\pi^{-1}(e_j) = x_j$  and  $e_j \succeq_{w'} e_k]$  or  $[\pi^{-1}(e_j) = y_j$  and  $e_j \succ_{w'} e_k]$ .

Because we cannot have  $\pi^{-1}(e_j) = z_j$  and  $\pi^{-1}(e_j) = x_j$  simultaneously, we must have  $\pi^{-1}(e_j) = y_j$ ,  $e_j \succ_f e_i$ , and  $e_j \succ_{w'} e_k$ . As we have  $N_f - e_i + e_j \in \mathcal{I}_f$  and  $N_{w'} + e_j - e_k \in \mathcal{I}_{w'}$ , these preference relations imply that  $e_j$  blocks  $N$ , which contradicts the stability of  $N$ . ◀

**Proof of Theorem 1.** Combining Lemmas 8 and 9, we obtain Theorem 1. ◀

## 6 Time Complexity

We explain how to implement the second step of Algorithm 1 and estimate its time complexity, which establishes Theorem 2. For this purpose, we introduce the notion of a matroid-kernel, which is a matroid generalization of a stable matching proposed by Fleiner [12, 13]. Note that it is defined not only for base-orderable matroids but for general matroids.

## 6.1 Matroid-kernels

A triple  $\mathcal{M} = (E, \mathcal{I}, \succ)$  is called an *ordered matroid* if  $(E, \mathcal{I})$  is a matroid and  $\succ$  is a strict linear order on  $E$ . For an ordered matroid  $\mathcal{M} = (E, \mathcal{I}, \succ)$  and an independent set  $S \in \mathcal{I}$ , an element  $e \in E \setminus S$  is said to be *dominated* by  $S$  in  $\mathcal{M}$  if  $S + e \notin \mathcal{I}$  and there is no element  $e' \in S$  such that  $e \succ e'$  and  $S + e - e' \in \mathcal{I}$ .

Let  $\mathcal{M}_1 = (E, \mathcal{I}_1, \succ_1)$  and  $\mathcal{M}_2 = (E, \mathcal{I}_2, \succ_2)$  be two ordered matroids on the same ground set  $E$ . Then, a set  $S \subseteq E$  is called an  $\mathcal{M}_1\mathcal{M}_2$ -kernel if  $S \in \mathcal{I}_1 \cap \mathcal{I}_2$  and any element  $e \in E \setminus S$  is dominated by  $S$  in  $\mathcal{M}_1$  or  $\mathcal{M}_2$ .

In [12], an algorithm for finding a matroid-kernel has been described using choice functions defined as follows. For an ordered matroid  $\mathcal{M} = (E, \mathcal{I}, \succ)$ , give indices of elements in  $E$  such that  $E = \{e^1, e^2, \dots, e^n\}$  and  $e^1 \succ e^2 \succ \dots \succ e^n$ . Define a function  $C_{\mathcal{M}} : 2^E \rightarrow 2^E$  by letting  $C_{\mathcal{M}}$  be the output of the following greedy algorithm for every  $S \subseteq E$ . Let  $T^0 := \emptyset$  and define  $T^\ell$  for  $\ell = 1, 2, \dots, n$  by

$$T^\ell := \begin{cases} T^{\ell-1} + e^\ell & \text{if } e^\ell \in S \text{ and } T^{\ell-1} + e^\ell \in \mathcal{I}, \\ T^{\ell-1} & \text{otherwise;} \end{cases}$$

then, let  $\mathcal{M}(S) := T^n$ .

Let  $C_{\mathcal{M}_1}, C_{\mathcal{M}_2}$  be the choice functions defined from  $\mathcal{M}_1 = (E, \mathcal{I}_1, \succ_1)$ ,  $\mathcal{M}_2 = (E, \mathcal{I}_2, \succ_2)$ , respectively. In [12, Theorem 2], Fleiner showed that an  $\mathcal{M}_1\mathcal{M}_2$ -kernel can be found using the following algorithm, which can be regarded as a generalization of the Gale–Shapley algorithm. First, set  $R \leftarrow \emptyset$ . Then, repeat the following three steps: (1)  $S \leftarrow C_{\mathcal{M}_1}(E \setminus R)$ , (2)  $T \leftarrow C_{\mathcal{M}_2}(S \cup R)$ , and (3)  $R \leftarrow (S \cup R) \setminus T$ . Stop the repetition if  $R$  is not changed at (3) and return  $T$  at that moment. In terms of the ordinary Gale–Shapley algorithm,  $R$ ,  $S$ , and  $T$  correspond to the sets of contracts that are rejected by firms thus far, proposed by workers, and accepted by firms, respectively.

► **Theorem 10** (Fleiner [12, 13]). *For any pair of ordered matroids  $\mathcal{M}_1$  and  $\mathcal{M}_2$  on the same ground set  $E$ , there exists an  $\mathcal{M}_1\mathcal{M}_2$ -kernel. One can find an  $\mathcal{M}_1\mathcal{M}_2$ -kernel in  $O(|E| \cdot \text{EO})$  time, where EO is the time required to compute  $C_{\mathcal{M}_1}(S)$  and  $C_{\mathcal{M}_2}(S)$  for any  $S \subseteq E$ .*

## 6.2 Implementation of Our Algorithm

We show that the second step of Algorithm 1 is reduced to a computation of a matroid-kernel.

For an auxiliary instance  $I^*$  defined in Section 2, note that  $\{E_w^* \mid w \in W\}$  is a partition of  $E^*$  and let  $(E^*, \mathcal{I}_W^*)$  be the direct sum of  $\{(E_w^*, \mathcal{I}_w^*)\}_{w \in W}$ . Furthermore, let  $\succ_W$  be a strict linear order on  $E^*$  that is consistent with the workers' preferences  $\{P_w^*\}_{w \in W}$  in  $I^*$ . For example, obtain  $\succ_W$  by concatenating the lists  $P_w^*$  of all workers in an arbitrary order. Then,  $\mathcal{M}_W = (E^*, \mathcal{I}_W^*, \succ_W)$  is an ordered matroid on the contract set  $E^*$ . As  $\{E_f^* \mid f \in F\}$  is also a partition of  $E^*$ , we can define an ordered matroid  $\mathcal{M}_F = (E^*, \mathcal{I}_F^*, \succ_F)$  in the same manner from  $\{(E_f^*, \mathcal{I}_f^*)\}_{f \in F}$  and  $\{P_f^*\}_{f \in F}$ .

We show that  $\mathcal{M}_W\mathcal{M}_F$ -kernels are equivalent to stable matchings of  $I$ . This has already been shown in several previous works [14, 44]. We present a proof for the completeness.

► **Lemma 11.**  *$M^* \subseteq E^*$  is a stable matching of  $I^*$  if and only if  $M^*$  is an  $\mathcal{M}_W\mathcal{M}_F$ -kernel.*

**Proof.** By the definitions of  $(E^*, \mathcal{I}_W^*)$  and  $(E^*, \mathcal{I}_F^*)$ , a set  $M^* \subseteq E^*$  is feasible for all agents in  $I^*$  if and only if  $M^* \in \mathcal{I}_W^* \cap \mathcal{I}_F^*$ . Recall that a contract  $e \in E^* \setminus M^*$  is free for the associated worker  $w := \partial_W(e)$  if  $M_w^* + e \in \mathcal{I}_w^*$  or there exists  $e' \in M_w^*$  such that  $e \succ_w^* e'$  and  $M_w^* + e - e' \in \mathcal{I}_w^*$ . By the definition of  $\mathcal{I}_W^*$ , we have  $M_w^* + e \in \mathcal{I}_w^*$  if and only if



$M^* + e \in \mathcal{I}_W^*$ . In addition, if  $M_w^* + e \notin \mathcal{I}_w^*$ , then  $M_w^* + e - e' \in \mathcal{I}_w^*$  holds for  $e' \in M_w$  if and only if  $M^* + e - e' \in \mathcal{I}_W^*$ . Because  $\succ_W$  is consistent with  $\succ_w^*$ , these imply that  $e$  is free for  $w = \partial_W(e)$  in  $M^*$  if and only if  $e$  is not dominated by  $M^*$  in  $\mathcal{M}_W$ . Similarly, we can show that  $e$  is free for the associated firm  $f := \partial_F(e)$  in  $M^*$  if and only if  $e$  is not dominated by  $M^*$  in  $\mathcal{M}_F$ . Thus, the equivalence holds.  $\blacktriangleleft$

► **Lemma 12.** *For any subset  $S^* \subseteq E^*$ , we can compute  $C_{\mathcal{M}_W}(S^*)$  and  $C_{\mathcal{M}_F}(S^*)$  in  $O(k^* \cdot |E^*|)$  time, where  $k^*$  is the maximum level of nesting of laminar families  $\mathcal{L}_a^*$  ( $a \in W \cup F$ ).*

**Proof.** We only explain the computation of  $C_{\mathcal{M}_W}(S^*)$  because that of  $C_{\mathcal{M}_F}(S^*)$  is similar.

Let  $\mathcal{L}$  be the union of  $\{\mathcal{L}_w^*\}_{w \in W}$  and define  $q : \mathcal{L} \rightarrow \mathbf{Z}_+$  by setting  $q(L) = q_w^*(L)$  for each  $w \in W$  and  $L \in \mathcal{L}_w^*$ . Then,  $\mathcal{L}$  is a laminar family on  $E^*$  and the matroid  $(E^*, \mathcal{I}_W^*)$  is defined by  $\mathcal{L}$  and  $q$ . The maximum level of nesting of  $\mathcal{L}$  is again  $k^*$ .

Referring to [4], we represent  $\mathcal{L}$  by a forest  $G$  whose node set is  $\{v_L \mid L \in \mathcal{L}\}$ . Node  $v_L$  is the parent of  $v_{L'}$  in  $G$  if  $L \subseteq L'$  and there is no  $L'' \in \mathcal{L}$  such that  $L \subsetneq L'' \subsetneq L'$ . Note that  $\mathcal{L}$  contains the set  $\{x_i, y_i, z_i\}$  for every  $e_i \in E$ , which is inclusion-wise minimal in  $\mathcal{L}$ . Therefore, the node  $v_i := v_{\{x_i, y_i, z_i\}}$  is a leaf for any  $e_i \in E$ , and any leaf has this form.

We compute the sequence  $T^0, T^1, \dots, T^{|E^*|}$  of sets in the definition of  $C_{\mathcal{M}_W}(S^*)$  as follows. For each  $v_L$ , we store a pointer to its parent, the value of  $q(L)$ , and the value of  $|T^{\ell-1} \cap L|$ . For each  $e^\ell \in E^*$ , we have  $T^{\ell-1} + e^\ell \in \mathcal{I}_W^*$  if and only if there is no ancestor node  $v_L$  of  $v_i$  with  $q(L) = |T^{\ell-1} \cap L|$ , where  $v_i$  is the leaf with  $e^\ell \in \{x_i, y_i, z_i\}$ . Then, we can check whether  $T^{\ell-1} + e^\ell \in \mathcal{I}_W^*$  in  $O(k^*)$  time by following the path of the parent pointers from  $v_i$ . When  $T^\ell = T^{\ell-1} + e^\ell$ , we update the stored values  $|T^{\ell-1} \cap L|$  to  $|T^\ell \cap L|$  for each  $L \in \mathcal{L}$  with  $e^\ell \in L$ . This is also performed in  $O(k^*)$  time by following the path of the parent pointers.  $\blacktriangleleft$

**Proof of Theorem 2.** As we have Theorem 1, what is left is to show the time complexity. The set  $E^*$  of contracts in  $I^*$  satisfies  $|E^*| = 3|E|$ . The maximum level of nesting of laminar families  $\mathcal{L}_a^*$  in  $I^*$  is  $k + 1$ . By Theorem 10 and Lemmas 11 and 12, then the second step of Algorithm 1 is computed in  $O((k + 1) \cdot |E^*|^2) = O(k \cdot |E|^2)$  time. Since the first and third steps can be performed in  $O(k \cdot |E|^2)$  time, Algorithm 1 runs in  $O(k \cdot |E|^2)$  time.  $\blacktriangleleft$

► **Remark.** Our analysis depends on the fact that the feasible set family defined by laminar constraints forms the independent set family of a base-orderable matroid. Actually, we can extend Theorem 1 to a setting where the family of feasible sets of each agent  $a \in W \cup F$  is represented by the independent set family  $\mathcal{I}_a$  of an arbitrary base-orderable matroid. To construct  $I^*$  in this case, we define  $E^*$  and  $\{P_a^*\}_{a \in W \cup F}$  as in Section 3 and define the feasible set family  $\mathcal{I}_a^*$  by  $\mathcal{I}_a^* = \{S^* \subseteq E_a^* \mid |\{x_i, y_i, z_i\} \cap S^*| \leq 1 \text{ for any } e_i \in E_a \text{ and } \pi(S^*) \in \mathcal{I}_a\}$ . We can easily show that  $(E_a^*, \mathcal{I}_a^*)$  is also a base-orderable matroid and apply the arguments in Sections 5 and 6, except Lemma 12. Given a membership oracle for each  $\mathcal{I}_a$  available, Algorithm 1 runs in  $O(\tau \cdot |E|^2)$  time in this case, where  $\tau$  is the time for an oracle call.

## 7 Strategy-Proof Approximation Mechanisms

In this section, we investigate approximation ratios for MAX-SMTI-LC attained by strategy-proof mechanisms. First, note that our setting SMTI-LC is a generalization of the stable marriage model of Gale and Shapley [16]; hence, Roth's impossibility theorem [40] implies that there is no mechanism that returns a stable matching and is strategy-proof for agents on both sides. As with many existing works on strategy-proofness in two-sided matching models, we consider one-sided strategy-proofness in the setting of many-to-one matching. Many-to-one matching models have various applications such as assignment of residents to hospitals [39, 41] and students to high schools [1–3]. In such applications, strategy-proofness for residents or students is a desirable property preventing their strategic behavior.

## 7.1 Model and Definitions

We define a setting of SMTI-OLC, which is a many-to-one variant of SMTI-LC. (Here, OLC stands for “one-sided laminar constraints”). In SMTI-OLC, each worker is assigned at most one contract and hence has no laminar constraints. An instance of SMTI-OLC is described as  $I = (W, F, E, \{P_w\}_{w \in W}, \{\mathcal{L}_f, q_f, P_f\}_{f \in F})$ . To consider strategies of workers, we slightly change the assumption on each  $P_w$ . In Section 2, it is assumed that  $P_w$  contains all contracts in  $E_w$ . Here, we allow each worker to submit a preference list  $P_w$  that is defined on any subset of  $E_w$  and regard contracts not appearing in  $P_w$  as unacceptable for  $w$ . Let  $E^\circ$  be the set of acceptable contracts, that is,  $E^\circ = \{e \in E \mid e \text{ appears in } P_w, \text{ where } w = \partial_W(e)\}$ .

A set  $M \subseteq E$  is called a *matching* if  $M \subseteq E^\circ$ ,  $|M_w| \leq 1$  for every worker  $w \in W$ , and  $M$  is feasible for every firm  $f \in F$ . For a matching  $M$ , a contract  $e \in E \setminus M$  *blocks*  $M$  if it is free for both  $\partial_W(e)$  and  $\partial_F(e)$ , where we say that  $e$  is *free* for the associated worker  $w := \partial_W(e)$  if  $e \in E^\circ$  and either  $w$  is assigned no contract in  $M$  or prefers  $e$  to the contract assigned in  $M$ . A matching  $M$  is *stable* if there is no contract that blocks  $M$ . The auxiliary instance  $I^* = (W, F, E^*, \{P_w^*\}_{w \in W}, \{\mathcal{L}_f^*, q_f^*, P_f^*\}_{f \in F})$  of  $I$  is defined similarly as in Section 3.

We remark that SMTI-OLC can be seen as a special case of SMTI-LC, although the assumption on workers’ preference lists is slightly different from that of SMTI-LC. From an SMTI-OLC instance  $I$ , define  $I^\circ = (W, F, E^\circ, \{\mathcal{L}_a^\circ, q_a^\circ, P_a^\circ\}_{a \in W \cup F})$  as follows. For each worker  $w \in W$ , set  $\mathcal{L}_w^\circ = \{E_w^\circ\}$ ,  $q_w^\circ(E_w^\circ) = 1$ , and  $P_w^\circ = P_w$ . For each firm  $f \in F$ , set  $\mathcal{L}_f^\circ = \{L \cap E^\circ \mid L \in \mathcal{L}_f\}$ ,  $q_f^\circ(L \cap E^\circ) = q_f(L)$  for each  $L \in \mathcal{L}_f$ , and let  $P_f^\circ$  be the restriction of  $P_f$  on  $E_f^\circ$  (i.e., delete the elements in  $E_f \setminus E_f^\circ$  from  $P_f$ ). Then,  $I^\circ$  is an instance of SMTI-LC in Section 2. By definition, we can observe that a subset  $M \subseteq E$  is a stable matching of  $I$  if and only if it is a stable matching of  $I^\circ$ . Therefore, we can apply Algorithm 1 to SMTI-OLC instances.

For subsets  $M, N \subseteq E$ , a worker  $w \in W$ , and a preference list  $P_w$ , we say that  $w$  *weakly prefers*  $M$  to  $N$  with respect to  $P_w$  if either (i)  $w$  is assigned a contract appearing in  $P_w$  only in  $M$  or (ii)  $w$  is assigned a contract appearing in  $P_w$  in both  $M$  and  $N$  and does not strictly prefer the one assigned in  $N$  with respect to  $P_w$ . A stable matching  $M$  of an SMTI-OLC instance  $I$  is *worker-optimal* if, for any other stable matching  $N$  of  $I$ , every worker  $w$  weakly prefers  $M$  to  $N$ .

A *mechanism* is a mapping from SMTI-OLC instances to matchings. Here, we define the *worker-strategy-proofness* of a mechanism. Let  $A$  be a mechanism. For any instance  $I$  and any worker  $w$ , let  $I'$  be an instance obtained from  $I$  by replacing  $w$ ’s list  $P_w$  with some other list  $P'_w$ . Let  $M$  and  $M'$  be the outputs of  $A$  for instances  $I$  and  $I'$ , respectively. We say that  $A$  is *worker-strategy-proof* if  $w$  weakly prefers  $M$  to  $M'$  with respect to the original list  $P_w$  regardless of the choices of  $I$ ,  $w$ , and  $P'_w$ .

## 7.2 Approximation Mechanisms

Before providing our results on SMTI-OLC, we introduce some existing results on special cases of SMTI-OLC. We first present a result on the setting without ties.

► **Lemma 13.** *In a restriction of SMTI-OLC in which all agents have strict preferences, a mechanism that returns the worker-optimal stable matching is worker-strategy-proof.*

Lemma 13 is a natural consequence of the results shown in previous works [17, 33]. For the completeness, the full version [45] provides the proof, which uses the fact that SMTI-OLC can be reduced to the model of Hatfield and Milgrom [20] if there are no ties.

Next, we introduce the results of Hamada et al. [19] on MAX-SMTI, which is a special case of MAX-SMTI-OLC in which every agent is assigned at most one contract.

► **Theorem 14** (Hamada et al. [19, Theorem 2]). *For MAX-SMTI, there is a worker-strategy-proof mechanism that returns a 2-approximate solution. On the other hand, for any  $\epsilon > 0$ , there is no worker-strategy-proof mechanism that returns a  $(2 - \epsilon)$ -approximate solution.*

► **Theorem 15** (Hamada et al. [19, Theorem 4]). *For a restriction of MAX-SMTI in which ties appear in only workers' preference lists, there is a worker-strategy-proof mechanism that returns a  $\frac{3}{2}$ -approximate solution. On the other hand, for any  $\epsilon > 0$ , there is no worker-strategy-proof mechanism that returns a  $(\frac{3}{2} - \epsilon)$ -approximate solution.*

The first statement of Theorem 14 is attained by a naive mechanism that first breaks ties in an increasing order of the indices and then finds the worker-optimal stable matching of the resultant instance. This method naturally extends to the setting of SMTI-OLC and yields the following theorem. See the full version [45] for the proof.

► **Theorem 16.** *For SMTI-OLC, there is a worker-strategy-proof mechanism that returns a stable matching  $M$  with  $|M| \geq \frac{1}{2}|M_{\text{OPT}}|$  in  $O(k \cdot |E|^2)$  time, where  $M_{\text{OPT}}$  is a maximum-cardinality stable matching and  $k$  is the maximum level of nesting of  $\mathcal{L}_f$  ( $f \in F$ ).*

Since SMTI-OLC is a generalization of SMTI, the second statement (i.e., the hardness part) of Theorem 14 immediately extends to MAX-SMTI-OLC. Therefore, for the general SMTI-OLC, there is no worker-strategy-proof mechanism with an approximation ratio better than 2.

However, in a special case in which firms' lists contain no ties, Algorithm 1 in Section 3 defines a worker-strategy-proof mechanism whose approximation ratio is  $\frac{3}{2}$ . That is, we can extend the first statement of Theorem 15 to the setting of SMTI-OLC. According to the second statement of Theorem 15, this is the best approximation ratio attained by a worker-strategy-proof mechanism.

► **Theorem 17.** *For a restriction of SMTI-OLC in which ties appear in only workers' lists, there is a worker-strategy-proof mechanism that returns a stable matching  $M$  with  $|M| \geq \frac{2}{3}|M_{\text{OPT}}|$  in  $O(k \cdot |E|^2)$  time, where  $M_{\text{OPT}}$  is a maximum-cardinality stable matching and  $k$  is the maximum level of nesting of laminar families  $\mathcal{L}_f$  ( $f \in F$ ).*

We provide a mechanism that meets the requirements in Theorem 17. Our mechanism is regarded as a possible realization of Algorithm 1. In the second step of Algorithm 1, we should choose the worker-optimal stable matching of the auxiliary instance  $I^*$ . Our mechanism is described as follows.

1. Given an instance  $I$  (in which ties appear in only workers' lists), construct  $I^*$ .
2. Find the worker-optimal stable matching  $M^*$  of  $I^*$ .
3. Let  $M = \pi(M^*)$  and return  $M$ .

In the proof of Theorem 10 (Fleiner [12, p.113]), it is shown that one can find the  $\mathcal{M}_1$ -optimal  $\mathcal{M}_1\mathcal{M}_2$ -kernel in  $O(|E| \cdot \text{EO})$  time. The arguments in Section 6 then imply that one can find the worker-optimal stable matching of  $I^*$  in  $O(k \cdot |E|^2)$  time. As we have Theorem 2, showing the strategy-proofness of the above-mentioned mechanism completes the proof of Theorem 17. To this end, we show the following lemma.

► **Lemma 18.** *Let  $I$  be an SMTI-OLC instance with  $E = \{e_i \mid i = 1, 2, \dots, n\}$  and let  $I^*$  be the auxiliary instance. If ties appear in only workers' lists in  $I$ , then the worker-optimal stable matching  $M^*$  of  $I^*$  satisfies  $M^* \cap \{z_i \mid i = 1, 2, \dots, n\} = \emptyset$ .*

**Proof.** Suppose, to the contrary, that  $z_i \in M^*$  for some index  $i$ . Then  $N := M^* - z_i + y_i$  is a matching of  $I^*$  and  $w := \partial_W(z_i) = \partial_W(y_i)$  prefers  $N$  to  $M^*$ . We intend to show that  $N$  is stable in  $I^*$ . Take any  $e \in E^* \setminus N = (E^* \setminus M^*) + z_i - y_i$ . If  $e = z_i$ , then it does not block  $N$  because  $y_i \succ_w^* z_i$ . If  $e \neq z_i$ , then the assignment of  $\partial_W(e)$  does not change in  $M^*$  and  $N$ , and hence  $e$  can block  $N$  only if  $f := \partial_F(e) = \partial_F(z_i)$  and  $z_i \succ_f^* e \succ_f^* y_i$ . This is impossible because no contract lies between  $z_i$  and  $y_i$  in  $P_f^*$  as the list  $P_f$  of the firm  $f$  is strict. Thus,  $N$  is a stable matching of  $I^*$ , which contradicts the worker-optimality of  $M^*$ . ◀

**Proof of Theorem 17.** As we have Theorem 2, what is left is to show that our mechanism is worker-strategy-proof. Let  $I = (W, F, E, \{P_w\}_{w \in W}, \{\mathcal{L}_f, q_f, P_f\}_{f \in F})$  be an instance of the setting in the statement and let  $E = \{e_i \mid i = 1, 2, \dots, n\}$ . Furthermore, let  $I'$  be obtained from  $I$  by replacing  $P_w$  with some other list  $P'_w$ . Let  $M^*$  and  $N^*$  be the worker-optimal stable matchings of the auxiliary instances defined from  $I$  and  $I'$ , respectively. Note that the two auxiliary instances have no ties and they differ only in the preference list of  $w$ . Then, Lemma 13 implies that  $w$  weakly prefers  $M^*$  to  $N^*$  with respect to  $P_w^*$ . In other words, either (i)  $w$  is assigned a contract on  $P_w^*$  only in  $M^*$ , or (ii)  $w$  is assigned a contract on  $P_w^*$  in both  $M^*$  and  $N^*$  and does not strictly prefer the one assigned in  $N^*$  w.r.t.  $P_w^*$ . By Lemma 18,  $w$  is not assigned a contract of type  $z_i$  in  $M^*$  or  $N^*$ . Then, the definition of  $P_w^*$  implies that  $w$  weakly prefers  $\pi(M^*)$  to  $\pi(N^*)$  w.r.t.  $P_w$ . Thus the mechanism is worker-strategy-proof. ◀

---

## References

- 1 A. Abdulkadiroğlu, P. A. Pathak, and A. E. Roth. The New York city high school match. *American Economic Review*, 95:364–367, 2005.
- 2 A. Abdulkadiroğlu, P. A. Pathak, and A. E. Roth. Strategy-proofness versus efficiency in matching with indifferences: Redesigning the NYC high school match. *American Economic Review*, 99(5):1954–1978, 2009.
- 3 A. Abdulkadiroğlu, P. A. Pathak, A. E. Roth, and T. Sönmez. The Boston public school match. *American Economic Review*, 95:368–371, 2005.
- 4 P. Biró, T. Fleiner, R. W. Irving, and D. F. Manlove. The college admissions problem with lower and common quotas. *Theoretical Computer Science*, 411(34):3136–3153, 2010.
- 5 J. E. Bonin and T. J. Savitsky. An infinite family of excluded minors for strong base-orderability. *Linear Algebra and its Applications*, 488:396–429, 2016.
- 6 S. Braun, N. Dwenger, D. Kübler, and A. Westkamp. Implementing quotas in university admissions: An experimental analysis. *Games and Economic Behavior*, 85:232–251, 2014.
- 7 R. A. Brualdi. Induced matroids. *Proceedings of the American Mathematical Society*, 29(2):213–221, 1971.
- 8 F. Cooper and D. Manlove. A  $3/2$ -approximation algorithm for the student-project allocation problem. In *Proc. 17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103, pages 8:1–8:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 9 L. Ehlers, I. E. Hafalir, M. B. Yenmez, and M. A. Yildirim. School choice with controlled choice constraints: Hard bounds versus soft bounds. *Journal of Economic Theory*, 153:648–683, 2014.
- 10 T. Fife and J. Oxley. Laminar matroids. *European Journal of Combinatorics*, 62:206–216, 2017.
- 11 L. Finkelstein. Two algorithms for the matroid secretary problem. Master’s thesis, Technion-Israel Institute of Technology, Faculty of Industrial and Management Engineering, 2011.
- 12 T. Fleiner. A matroid generalization of the stable matching polytope. In *Proc. Eighth International Conference on Integer Programming and Combinatorial Optimization (IPCO 2001)*, volume 2081 of *LNCS*, pages 105–114. Springer-Verlag, Berlin & Heidelberg, 2001.
- 13 T. Fleiner. A fixed-point approach to stable matchings and some applications. *Mathematics of Operations Research*, 28(1):103–126, 2003.

- 14 T. Fleiner and N. Kamiyama. A matroid approach to stable matchings with lower quotas. *Mathematics of Operations Research*, 41(2):734–744, 2016.
- 15 D. Fragiadakis and P. Troyan. Improving matching under hard distributional constraints. *Theoretical Economics*, 12(2):863–908, 2017.
- 16 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.
- 17 M. Goto, A. Iwasaki, Y. Kawasaki, R. Kurata, Y. Yasuda, and M. Yokoo. Strategyproof matching with regional minimum and maximum quotas. *Artificial Intelligence*, 235:40–57, 2016.
- 18 D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, 1989.
- 19 K. Hamada, S. Miyazaki, and H. Yanagisawa. Strategy-proof approximation algorithms for the stable marriage problem with ties and incomplete lists. In *Proc. 30th International Symposium on Algorithms and Computation (ISAAC 2019)*, volume 149 of *LIPICs*, pages 9:1–9:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 20 J. W. Hatfield and P. R. Milgrom. Matching with contracts. *American Economic Review*, 95(4):913–935, 2005.
- 21 C. C. Huang. Classified stable matching. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA2010)*, pages 1235–1253. SIAM, Philadelphia, 2010.
- 22 A. W. Ingleton. Transversal matroids and related structures. In *Higher Combinatorics* (M. Aigner eds.), pages 117–131. Reidel, Dordrecht, 1977.
- 23 R. W. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48(3):261–272, 1994.
- 24 K. Iwama, D. F. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. In *Proc. 26th International Colloquium on Automata, Languages, and Programming (ICALP1999)*, pages 443–452. Springer, 1999.
- 25 K. Iwama, S. Miyazaki, and N. Yamauchi. A 1.875-approximation algorithm for the stable marriage problem. In *Proc. 18th annual ACM-SIAM symposium on Discrete algorithms (SODA2007)*, pages 288–297. SIAM, Philadelphia, 2007.
- 26 K. Iwama, S. Miyazaki, and N. Yamauchi. A  $(2 - c\frac{1}{\sqrt{n}})$ -approximation algorithm for the stable marriage problem. *Algorithmica*, 51(3):342–356, 2008.
- 27 Y. Kamada and F. Kojima. Efficient matching under distributional constraints: Theory and applications. *American Economic Review*, 105(1):67–99, 2015.
- 28 Y. Kamada and F. Kojima. Stability concepts in matching under distributional constraints. *Journal of Economic Theory*, 168:107–142, 2017.
- 29 Y. Kamada and F. Kojima. Stability and strategy-proofness for matching with constraints: A necessary and sufficient condition. *Theoretical Economics*, 13(2):761–793, 2018.
- 30 Z. Király. Better and simpler approximation algorithms for the stable marriage problem. *Algorithmica*, 60(1):3–20, 2011.
- 31 Z. Király. Linear time local approximation algorithm for maximum stable marriage. *Algorithms*, 6(3):471–484, 2013.
- 32 D. E. Knuth. *Stable Marriage and Its Relation to Other Combinatorial Problems*. American Mathematical Society, Providence, 1996.
- 33 F. Kojima, A. Tamura, and M. Yokoo. Designing matching mechanisms under constraints: An approach from discrete convex analysis. *Journal of Economic Theory*, 176:803–833, 2018.
- 34 D. F. Manlove. *Algorithmics of Matching under Preferences*. World Scientific Publishing, Singapore, 2013.
- 35 D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.
- 36 E. Mcdermid. A  $3/2$ -approximation algorithm for general stable marriage. In *Proc. 36th International Colloquium on Automata, Languages, and Programming (ICALP2009)*, pages 689–700. Springer, 2009.

## 71:16 Maximum Stable Matching with Ties and Constraints

- 37 J. G. Oxley. *Matroid Theory* (2nd ed.). Oxford University Press, Oxford, 2011.
- 38 K. Paluch. Faster and simpler approximation of stable matchings. *Algorithms*, 7(2):189–202, 2014.
- 39 A. E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *The Journal of Political Economy*, 92(6):991–1016, 1984.
- 40 A. E. Roth. On the allocation of residents to rural hospitals: A general property of two-sided matching markets. *Econometrica*, 54(2):425–427, 1986.
- 41 A. E. Roth and E. Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *American economic review*, 89(4):748–780, 1999.
- 42 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer-Verlag, Heidelberg, 2003.
- 43 H. Yanagisawa. *Approximation algorithms for stable marriage problems*. Ph.D. Thesis, Kyoto University, 2007.
- 44 Y. Yokoi. A generalized polymatroid approach to stable matchings with lower quotas. *Mathematics of Operations Research*, 42(1):238–255, 2017.
- 45 Y. Yokoi. An approximation algorithm for maximum stable matching with ties and constraints. *arXiv preprint*, 2021. [arXiv:2107.03076](https://arxiv.org/abs/2107.03076).



# A Faster Algorithm for Maximum Flow in Directed Planar Graphs with Vertex Capacities

Julian Enoch ✉

Department of Computer Science, University of Texas at Dallas, TX, USA

Kyle Fox ✉

Department of Computer Science, University of Texas at Dallas, TX, USA

Dor Mesica ✉

Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Israel

Shay Mozes ✉ 

Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Israel

---

## Abstract

We give an  $O(k^3 \Delta n \log n \min(k, \log^2 n) \log^2(nC))$ -time algorithm for computing maximum integer flows in planar graphs with integer arc *and vertex* capacities bounded by  $C$ , and  $k$  sources and sinks. This improves by a factor of  $\max(k^2, k \log^2 n)$  over the fastest algorithm previously known for this problem [Wang, SODA 2019].

The speedup is obtained by two independent ideas. First we replace an iterative procedure of Wang that uses  $O(k)$  invocations of an  $O(k^3 n \log^3 n)$ -time algorithm for maximum flow algorithm in a planar graph with  $k$  apices [Borradaile et al., FOCS 2012, SICOMP 2017], by an alternative procedure that only makes one invocation of the algorithm of Borradaile et al. Second, we show two alternatives for computing flows in the  $k$ -apex graphs that arise in our modification of Wang's procedure faster than the algorithm of Borradaile et al. In doing so, we introduce and analyze a sequential implementation of the parallel highest-distance push-relabel algorithm of Goldberg and Tarjan [JACM 1988].

**2012 ACM Subject Classification** Theory of computation → Network flows

**Keywords and phrases** flow, planar graphs, vertex capacities

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.72

**Related Version** *Preprint*: <https://arxiv.org/abs/2101.11300>

**Funding** Enoch and Fox were partially supported by NSF grant CCF-1942597. Mesica and Mozes were partially supported by Israel Science Foundation grant No. 592/17.

## 1 Introduction

The maximum flow problem has been extensively studied in many different settings and variations. This work concerns two related variants of the maximum flow problem in planar graphs. The first variant is the problem of computing a maximum flow in a directed planar network with integer arc *and vertex* capacities, and  $k$  sources and sinks. The second variant, which is used in algorithms for the first variant, is the problem of computing a maximum flow in a directed network that is nearly planar; there is a set of  $k$  vertices, called apices, whose removal turns the graph planar.

The problem of maximum flow in a planar graph with vertex capacities has been studied in several works since the 1990s [9, 14, 7, 2, 13]. For a more detailed survey of the history of this problem and other relevant results see [13] and references therein. Vertex capacities pose a challenge in planar graphs because the standard reduction from a flow network with vertex capacities to a flow network with only arc capacities does not preserve planarity. The problem can be solved by algorithms for maximum flow in sparse graphs (i.e., graphs with  $n$  vertices



© Julian Enoch, Kyle Fox, Dor Mesica, and Shay Mozes;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 72; pp. 72:1–72:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



and  $O(n)$  edges that are not necessarily planar). The fastest such algorithms currently known are an  $O(n^2/\log n)$ -time algorithm [11] for sparse graphs, and an  $O(n^{4/3+o(1)}C^{1/3})$ -time algorithm for sparse graphs with integer capacities bounded by  $C$  [8]. Until recently, there was no planarity exploiting algorithm for the case of more than a single source and a single sink. Significant progress on this problem was recently made by Wang [13]. Wang developed an  $O(k^5\Delta n \log^3 n \log^2(nC))$ -time algorithm, where  $k$  is the number of sources and sinks,  $\Delta$  is the maximum vertex degree, and  $C$  is the largest capacity of a single vertex.<sup>1</sup> This is faster than using the two algorithms for general sparse graphs mentioned above when  $k = \tilde{O}(n^{1/5}/(\Delta \log^2 C) + (nC)^{1/15}/\Delta)$ .

Wang’s algorithm uses multiple calls to an algorithm of Borradaile et al. [2] for computing a maximum flow in a  $k$ -apex graph with only arc capacities. The algorithm of Borradaile et al. [2] is based on an approach originally suggested by Hochstein and Weihe [6] for a slightly more restricted problem. In Borradaile et al.’s approach, a maximum flow in a  $k$ -apex graph with  $n$  vertices is computed by simulating the Push-Relabel algorithm of Goldberg and Tarjan [4] on a complete graph with  $k$  vertices, corresponding to the  $k$  apices of the input graph. Whenever the Push-Relabel algorithm pushes flow on an arc of the complete graph, the push operation is simulated by sending flow between the two corresponding apices in the input  $k$ -apex graph. This can be done efficiently using an  $O(n \log^3 n)$  time multiple-source multiple-sink (MSMS) maximum flow algorithm in planar graphs, which is the main result of the paper of Borradaile et al. [2]. Overall, their algorithm for maximum flow in  $k$ -apex graphs takes  $O(k^3 n \log^3 n)$  time. Flow in  $k$ -apex graphs can also be computed using the algorithms for sparse graphs mentioned above. The  $O(k^3 n \log^3 n)$ -time algorithm of Borradaile et al. is faster than these algorithms when  $k = \tilde{O}(n^{1/3}/\log^2 C + (nC)^{1/9})$ .

## 1.1 Our results and techniques

We improve the running time of Wang’s algorithm to  $O(k^3\Delta n \log n \min(k, \log^2 n) \log^2(nC))$ . This is faster than Wang’s result by a factor of  $\max(k^2, k \log^2 n)$ , extending the range of values of  $k$  for which the planarity exploiting algorithm is the fastest known algorithm for the problem to  $k = \tilde{O}(n^{1/3}/(\Delta \log^2 C) + (nC)^{1/9}/\Delta)$ . The improvement is achieved by two main ideas. At the heart of Wang’s algorithm is an iterative procedure for eliminating excess flow from vertices violating the capacity constraints. Each iteration consists of computing a circulation with some desired properties. Wang computes this circulation using  $O(k)$  calls to the algorithm of Borradaile et al. for maximum flow in  $k$ -apex graphs. We show how to compute this circulation using a constant number of invocations of the algorithm for  $k$ -apex graphs. This idea alone improves on Wang’s algorithm by a factor of  $k$ .

To further improve the running time, we modify the algorithm of Borradaile et al. for maximum flow in  $k$ -apex graphs [2]. The algorithm of Borradaile et al. uses the Push-Relabel algorithm of Goldberg and Tarjan [4]. We introduce a sequential implementation of the parallel highest-distance Push-Relabel algorithm. In this algorithm, which we call batch-highest-distance, a single operation, Bulk-Push, pushes flow on multiple arcs simultaneously, instead of just on a single arc as in Goldberg and Tarjan’s Push operation. More specifically, we simultaneously push flow on all admissible arcs whose tails have maximum height (see Section 3). This is reminiscent of parallel and distributed Push-Relabel algorithms [4, 3],

---

<sup>1</sup> In a very recent personal communication, Wang informed the authors of a subtle bug in the published version [13] of his result that led to the  $\Delta$  factor not appearing in the claimed running time. The full version of Wang’s paper (which has not been published yet) fixes the bug, and we refer to his corrected claims throughout the current work.

but our algorithm is sequential, not parallel. We prove that the total number of Bulk-Push operations performed by the batch-highest-distance algorithm is  $O(k^2)$  (this should be compared to  $O(k^3)$  Push operations for the FIFO or highest-distance Push-Relabel algorithms). We then show that, in the case of the  $k$ -apex graphs that show up in Wang's algorithm, we can implement each Bulk-Push operation using a constant number of invocations of the  $O(n \log^3 n)$ -time MSMS maximum flow algorithm for planar graphs [2]. Hence, we can find a maximum flow in such  $k$ -apex graphs in  $O(k^2 n \log^3 n)$  time, which is faster by a factor of  $k$  than the time required by the algorithm of Borradaile et al.

We also give another way to modify the algorithm of Borradaile et al. for maximum flow in  $k$ -apex graphs; the second way is better when  $k = o(\log^2 n)$ . We observe that the structure of the  $k$ -apex graphs that arise in Wang's algorithm allows us to implement each of the  $O(k^3)$  Push operations of the FIFO Push-Relabel algorithm used by Borradaile et al. in just  $O(n \log n)$  time. This is done using a procedure due to Miller and Naor [10] for the case when all sources and sinks lie on a single face.

**Roadmap.** In Section 2 we provide preliminary background and notations. Section 3 describes the sequential implementation of the parallel highest-distance Push-Relabel algorithm, and its use in an algorithm for finding maximum flow in  $k$ -apex graphs. In Section 4 we describe how to use this Push-Relabel variant to obtain an improved algorithm for computing maximum flow in planar graphs with vertex capacities.

## 2 Preliminaries

All the graphs we consider in this paper are directed. For a graph  $G$  we use  $V(G)$  and  $E(G)$  to denote the vertex set and arc set of  $G$ , respectively. For any vertex  $v \in V(G)$ , let  $\deg(v)$  denote the degree of  $v$  in  $G$ .

For a path  $P$  we denote by  $P[u, v]$  the subpath of  $P$  that starts at  $u$  and ends at  $v$ . We denote by  $P \circ Q$  the concatenation of two paths  $P, Q$  such that the first vertex of  $Q$  is the last vertex of  $P$ .

A *flow network* is a directed graph  $G$  with a capacity function  $c : V(G) \cup E(G) \rightarrow [0, \infty)$  on the vertices and arcs of  $G$ , along with two disjoint sets  $S, T \subset V(G)$  called *sources* and *sinks*, respectively. We assume without loss of generality that sources and sinks have infinite capacities, and that, for any arc  $e = (u, v) \in E(G)$ , the *reverse* arc  $(v, u)$ , denoted  $\text{rev}(e)$  is also in  $E(G)$ , and has capacity  $c(\text{rev}(e)) = 0$ .

Let  $\rho : E(G) \rightarrow [0, \infty)$ . To avoid clutter we write  $\rho(u, v)$  instead of  $\rho((u, v))$ . For each vertex  $v$  let  $\rho^{\text{in}}(v) = \sum_{(u, v) \in E(G)} \rho(u, v)$ , and  $\rho^{\text{out}}(v) = \sum_{(v, u) \in E(G)} \rho(v, u)$ . The function  $\rho$  is called a *preflow* if it satisfies the following conservation constraint: for all  $v \in V(G) \setminus (S)$ ,  $\rho^{\text{in}}(v) \geq \rho^{\text{out}}(v)$ . The *excess of a vertex  $v$  with respect to a preflow  $\rho$*  is defined by  $\text{ex}(\rho, v) = \rho^{\text{in}}(v) - \rho^{\text{out}}(v)$ . A preflow is *feasible on an arc  $e \in E(G)$*  if  $\rho(e) \leq c(e)$ . It is *feasible on a vertex  $v \in V(G)$*  if  $\rho^{\text{in}}(v) \leq c(v)$ . A preflow is said to be *feasible* if, in addition to the conservation constraint, it is feasible on all arcs and vertices. The value of a preflow  $\rho$  is defined as  $|\rho| = \sum_{s \in S} \rho^{\text{out}}(s) - \rho^{\text{in}}(s)$ . A preflow  $f$  satisfying  $\text{ex}(f, v) = 0$  for all  $v \in V(G) \setminus (S \cup T)$ , is called a *flow*. A flow whose value is 0 is called a *circulation*. A *maximum flow* is a feasible flow whose value is maximum.

► **Remark 1.** The problem of finding a maximum flow in a flow network with multiple sources and sinks can be reduced to the single-source, single-sink case by adding a super source  $s$  and super sink  $t$ , and infinite-capacity arcs  $(s, s_i)$  and  $(t_i, t)$  for every  $s_i \in S$  and  $t_i \in T$ . If the

original network is planar then this transformation adds two apices to the graph. Throughout the paper, whenever we refer to the graph  $G$ , we mean the graph  $G$  after this transformation, i.e., with a single source, the apex  $s$ , and a single sink, the apex  $t$ .

The *violation of a flow  $f$  at a vertex  $v$*  is defined by  $\text{vio}(f, v) = \max\{0, f^{\text{in}}(v) - c(v)\}$ . Thus, if  $f$  is a feasible flow then  $\text{vio}(f, v) = 0$  for all vertices  $v$ . The violation of the flow  $f$  is defined to be  $\text{vio}(f) = \max_{v \in V(G)} \text{vio}(f, v)$ .<sup>2</sup>

A preflow  $\rho$  is *acyclic* if there is no cycle  $C$  such that  $\rho(e) > 0$  for every arc  $e \in C$ . A preflow *saturates* an arc  $e$  if  $\rho(e) = c(e)$ .

The sum of two preflows  $\rho$  and  $\eta$  is defined as follows. For every arc  $e \in E(G)$ ,  $(\rho + \eta)(e) = \max\{0, \rho(e) + \eta(e) - \rho(\text{rev}(e)) - \eta(\text{rev}(e))\}$ . Multiplying the preflow  $\rho$  by some constant  $c$  to get the flow  $c\rho$  is defined as  $(c\rho)(e) = c \cdot \rho(e)$  for all  $e \in E(G)$ .

The *residual capacity*  $c_\rho(e)$  of an arc  $e$  with respect to a preflow  $\rho$  is  $c(e) - \rho(e) + \rho(\text{rev}(e))$ . The *residual graph* of a flow network  $G$  with respect to a preflow  $\rho$  is the graph  $G$  where the capacity of every arc  $e \in E(G)$  is set to  $c_\rho(e)$ . It is denoted by  $G_\rho$ . A path of  $G$  is called *augmenting* or *residual* (with respect to a preflow  $\rho$ ) if it is also a path of  $G_\rho$ .

Suppose  $G$  and  $H$  are flow networks such that every arc in  $G$  is also an arc in  $H$ . If  $f'$  is a (pre)flow in  $H$  then the *restriction* of  $f'$  to  $G$  is the (pre)flow  $f$  in  $G$  defined by  $f(e) = f'(e)$  for all  $e \in E(G)$ .

### 3 An algorithm for maximum flow in $k$ -apex graphs

In this section we introduce a sequential implementation of the parallel highest-distance Push-Relabel algorithm of Goldberg and Tarjan [4], and use it in the algorithm of Borradaile et al. [2] for maximum flow in  $k$ -apex graphs. We first give a high-level description of the Push-Relabel algorithm.

#### 3.1 The Push-Relabel algorithm [4]

Let  $H$  be a flow network (not necessarily planar) with source  $s$  and sink  $t$ , arc capacities  $c : E(H) \rightarrow \mathbb{R}$ , and no finite vertex capacities. The Push-Relabel algorithm maintains a feasible *preflow* function,  $\rho$ , on the arcs of  $H$ . A vertex  $u$  is called *active* if  $\text{ex}(\rho, u) > 0$ . The algorithm starts with a preflow that is zero on all arcs, except for the arcs leaving the source  $s$ , which are saturated. Thus, all the neighbors of  $s$  are initially active. When the algorithm terminates, no vertex is active and the preflow function is guaranteed to be a maximum flow. The algorithm also maintains a label function  $h$  (also known as distance or height function) over the vertices of  $H$ . The label function  $h : V(H) \rightarrow \mathbb{N}$  is *valid* if  $h(s) = |V(H)|$ ,  $h(t) = 0$  and  $h(u) \leq h(v) + 1$  for every residual arc  $(u, v) \in E(H_\rho)$ .

The algorithm progresses by performing two basic operations, **Push** and **Relabel**. A **Push** operation applies to an arc  $(u, v)$  if  $(u, v)$  is residual,  $\text{ex}(\rho, u) > 0$ , and  $h(u) = h(v) + 1$ . The operation moves excess flow from  $u$  to  $v$  by increasing the flow on  $e$  by  $\min\{\text{ex}(\rho, u), c(e) - \rho(e)\}$ .

The other basic operation, **Relabel**( $u$ ), assigns  $u$  the label  $h(u) = \min\{h(v) : (u, v) \in E(H_\rho)\} + 1$  and applies to  $u$  only if  $u$  is active and  $h(u)$  is not greater than the label of any neighbor of  $u$  in  $H_\rho$ . In other words, **Relabel** applies to an active vertex  $u$  only if the excess flow in  $u$  cannot be pushed out of  $u$  (because  $h(u)$  is not high enough). The algorithm performs applicable **Push** and **Relabel** operations until no vertex is active.

<sup>2</sup> We define violations only with respect to flows (rather than preflows) because we will only discuss preflows in the context of flow networks without finite vertex capacities.

To fit our purposes, we think of the algorithm as one that only maintains explicitly the excess  $\text{ex}(\rho, v)$  and residual capacity  $c_\rho(e)$  of each vertex  $v$  and arc  $e$  of  $H$ . The preflow  $\rho$  is implicit. In this view, a  $\text{Push}(u, v)$  operation decreases  $\text{ex}(\rho, u)$  and  $c_\rho(u, v)$  by  $\min\{\text{ex}(\rho, u), c_\rho(u, v)\}$  and increases  $\text{ex}(\rho, v)$  and  $c_\rho(v, u)$  by the same amount.

We reformulate Goldberg and Tarjan's correctness proof of the generic Push-Relabel algorithm to fit this view.

► **Lemma 2** ([4]). *Any algorithm that performs applicable Push and Relabel operations in any order satisfies the following properties and invariants:*

- (1)  $\text{ex}(\rho, \cdot)$  and  $c_\rho(\cdot)$  are non-negative.<sup>3</sup>
- (2) The function  $h$  is a valid labeling function.
- (3) For all  $v \in V$ , the value of  $h(v)$  never decreases, and strictly increases when  $\text{Relabel}(v)$  is called.
- (4)  $h(v) \leq 2|V(H)| - 1$  for all  $v \in V(H)$ .
- (5) Immediately after  $\text{Push}(u, v)$  is performed, either  $(u, v)$  is saturated or  $u$  is inactive.

**Proof.** Properties (1) and (5) are immediate from the definition of Push and the fact that excess and residual capacities only change during Push operations. Property (2) corresponds to Lemma 3.1 in [4], Property (3) is proved in Lemma 3.6 in [4], and Property (4) in Lemma 3.7 in [4]. ◀

► **Lemma 3** ([4, Lemma 3.3]). *Properties (1), (2) imply that there is no augmenting path from  $s$  to  $t$  at any point of the algorithm.*

► **Lemma 4** ([4, Lemma 3.8]). *Properties (3), (4) imply that the number of Relabel operations is at most  $2|V(H)| - 1$  per vertex and at most  $2|V(H)|^2$  overall.*

► **Lemma 5** ([4, Lemmas 3.9, 3.10]). *Properties (1)-(5) imply that the number of Push operations is  $O(|V(H)|^2|E(H)|)$ .*

By Lemmas 4 and 5, the algorithm terminates. Upon termination no vertex is active, so the implicit preflow  $\rho$  is in fact a feasible flow. By Lemma 3  $\rho$  is a maximum flow from  $s$  to  $t$ .

Variants of the Push-Relabel algorithm differ in the order in which applicable Push and Relabel operations are applied. Some variants, such as FIFO, highest-distance, maximal-excess, etc., guarantee faster termination than the  $O(|V(H)|^2|E(H)|)$  guarantee given above.

## 3.2 A sequential implementation of the parallel highest-distance Push-Relabel algorithm

We present a sequential implementation of the parallel highest-distance Push-Relabel algorithm, which we call Batch-Highest-Distance. This algorithm attempts to push flow on multiple edges simultaneously in an operation called Bulk-Push. In that sense, it resembles the parallel version of the highest-distance Push-Relabel algorithm. It is important to note, however, that Bulk-Push is a sequential operation and not a parallel/distributed one.

We define Bulk-Push, a batched version of the Push operation.  $\text{Bulk-Push}(U, W)$  operates on two sets of vertices,  $U$  and  $W$ . It is applicable under the following requirements:

- (i)  $\text{ex}(u) > 0$  for all  $u \in U$ .
- (ii) There exists an integer  $h$  such that  $h(u) = h$  and  $h(w) = h - 1$  for all  $u \in U$  and  $w \in W$ .
- (iii) There is a residual arc  $(u, w)$  for some  $u \in U$  and  $w \in W$ .

<sup>3</sup> This corresponds to the function  $\rho$  being a feasible preflow.

Note that in a regular Push-Relabel algorithm, conditions (i) and (ii) imply that  $\text{Push}(u, w)$  is applicable to any residual arc  $(u, w)$  with  $u \in U$  and  $w \in W$ . Condition (iii) guarantees there is at least one such arc. **Bulk-Push** pushes as much excess flow as possible from vertices in  $U$  to vertices in  $W$  so that after **Bulk-Push** the following property holds:

**(5\*)** Immediately after  $\text{Bulk-Push}(U, W)$  is called, for all  $u \in U$  and  $w \in W$ , either  $(u, w)$  is saturated or  $u$  is inactive.

We replace property (5) with the more general property (5\*) in Lemma 2 and Lemma 5. With this modification, Lemmas 2, 3, 4 and 5 apply to our sequential implementation. The proofs from [4] need no change except replacing **Push** with **Bulk-Push**. Hence, our variant terminates correctly with a maximum flow from  $s$  to  $t$ .

► **Remark.** One may think of  $\text{Bulk-Push}(U, W)$  as performing in parallel all **Push** operations on arcs whose tail is in  $U$  and whose head is in  $W$ . However, not every maximum flow with sources  $U$  and sinks  $W$  can be achieved as the sum of flows pushed by multiple **Push** operations. For example, consider the case where  $U$  consists of a single vertex  $u$ , with  $\text{ex}(u) = 2$ ,  $W = \{w_1, w_2\}$ , and the residual capacities of  $(u, w_1)$  and  $(u, w_2)$  are both 2.  $\text{Bulk-Push}(U, W)$  may push one unit of excess flow from  $u$  on each of  $(u, w_1)$  and  $(u, w_2)$ , but  $\text{Push}(u, w_i)$  would push 2 units of flow on  $(u, w_i)$ , and no flow on the other arc. Therefore, the correctness of this variant cannot be argued just by simulating **Bulk-Push** by multiple **Push** operations. Instead we chose to argue correctness by stating the generalized property (5\*).

We now discuss a concrete policy for choosing which **Bulk-Push** and **Relabel** operations to perform in the above algorithm. This policy is similar, but not identical, to the highest-distance Push-Relabel algorithm [4, 3]. As long as there is an active vertex, the algorithm repeatedly executes the following two steps, which together are called a *pulse*. Let  $h_{max}$  be the maximum label of an active vertex. That is,  $h_{max} = \max\{h(v) : \text{ex}(\rho, v) > 0\}$ . Let  $H_{max}$  be the set of all the active vertices whose height is  $h_{max}$ . In the first step of the pulse, the algorithm invokes  $\text{Bulk-Push}(H_{max}, W)$  where  $W$  is the set of all vertices  $w \in V$  such that  $h(w) = h_{max} - 1$ .<sup>4</sup> In the second step of the pulse, the algorithm applies the **Relabel** operation to all remaining active vertices in  $H_{max}$  in arbitrary order.

■ **Algorithm 1** Batch-Highest-Distance( $G, c$ ).

---

```

1: Initialize  $h(\cdot)$ ,  $c_\rho(\cdot)$  and  $\text{ex}(\cdot)$ 
2: while there exists an active vertex do
3:    $h_{max} \leftarrow \max\{h(v) : \text{ex}(\rho, v) > 0\}$ 
4:    $H_{max} \leftarrow \{v \in V(H) : \text{ex}(\rho, v) > 0, h(v) = h_{max}\}$ 
5:    $W \leftarrow \{w \in V(H) : h(w) = h_{max} - 1\}$ 
6:    $\text{Bulk-Push}(H_{max}, W)$ 
7:   Relabel all active vertices in  $H_{max}$  in arbitrary order
8: end while

```

---

► **Remark.** The crucial difference between this policy and the highest-distance Push-Relabel algorithm [4, 3] is that in the highest-distance algorithm a vertex  $u$  with height  $h_{max}$  is relabeled as soon as no more **Push** operations can be applied to  $u$ . In contrast, our variant first pushes flow from all vertices with height  $h_{max}$  and only then relabels all of them.

<sup>4</sup> Formally it may be that  $\text{Bulk-Push}(H_{max}, W)$  is not applicable because condition (iii) is not satisfied, e.g., when  $W = \emptyset$ . In such cases **Bulk-Push** does not push any flow. Condition (iii) is essential for the termination of the generic generalized algorithm, which may repeat such empty calls to **Bulk-Push** indefinitely. However, we prove in Lemma 6 that in our specific policy there are  $O(|V(H)|^2)$  pulses, regardless of the flow pushed (or not pushed) by **Bulk-Push** in each pulse.

We partition the pulses into two types according to whether any vertices are relabeled in the relabel step of the pulse. A pulse in which at least one vertex is relabeled is called *saturating*. All other pulses are called *non-saturating*.<sup>5</sup>

By Lemma 4, the total number of Relabel operations executed by the batch-highest-distance algorithm is  $O(|V(H)|^2)$ . We now prove the same bound for Bulk-Push operations.

► **Lemma 6.** *The number of pulses (and hence also the number of calls to Bulk-Push) executed by the batch-highest-distance algorithm is  $O(|V(H)|^2)$ .*

**Proof.** Note that the Relabel step of a saturating pulse consists of at least one call to Relabel which strictly increases the height of an active vertex  $v$  whose height (before the increase) was  $h_{max}$ . Hence, a saturating pulse strictly increases the value of  $h_{max}$ . The fact that the height of each vertex never decreases and is bound by  $2|V(H)|$  implies that (i) there are  $O(|V(H)|^2)$  saturating pulses, and (ii) the total increase in  $h_{max}$  over all saturating Bulk-Push operations is  $O(|V(H)|^2)$ .

As for non-saturating pulses, note that since excess flow is always pushed to a vertex with lower height, the push step of a pulse does not create excess in any vertex with height greater than or equal to  $h_{max}$ , so all vertices with height greater than  $h_{max}$  remain inactive during the pulse. By property (5\*), for every  $u \in H_{max}$  and  $w \in W$ , either  $(u, w)$  is saturated, or  $u$  is inactive. Since the pulse is non-saturating, it follows that all the vertices in  $H_{max}$  become inactive during the pulse. Hence, the value of  $h_{max}$  strictly decreases during a non-saturating pulse. Since  $h_{max} \geq 0$ , the total decrease in  $h_{max}$  is also  $O(|V(H)|^2)$ , so there are  $O(|V(H)|^2)$  non-saturating pulses. ◀

Note that we do not claim that implementing the Bulk-Push operation by applying applicable  $\text{Push}(u, w)$  operations for  $u \in U$ ,  $w \in W$  until no more such operations can be applied would result in fewer Push operations than the  $O(|V(H)|^2|E(H)|)$  bound of Lemma 5 for the generic Push-Relabel algorithm. However, in Section 4 we will show a situation where each call to Bulk-Push can be efficiently implemented using a single invocation of a multiple-source multiple-sink algorithm in a planar graph.

### 3.3 The algorithm of Borradaile et al. for $k$ -apex graphs [2]

The algorithm of Borradaile et al. [2, Section 5] uses the framework of Hochstein and Weihe [6]. Let  $H$  be a graph with a set  $V^\times$  of  $k$  apices. Denote  $V_0 = V(H) \setminus V^\times$ . The goal is to compute a maximum flow in  $H$  from a source  $s \in V(H)$  to a sink  $t \in V(H)$ . We assume that  $s$  and  $t$  are apices. This is without loss of generality since treating  $s$  and  $t$  as apices leaves the number of apices in  $O(k)$ . Let  $K^\times$  be a complete graph over  $V^\times$ . The algorithm computes a maximum flow  $\rho$  from  $s$  to  $t$  in  $H$  by simulating a maximum flow computation from  $s$  to  $t$  in  $K^\times$  using the Push-Relabel algorithm. Whenever a Push operation is performed on an arc  $(u, v)$  of  $K^\times$  it is implemented by pushing flow from  $u$  to  $v$  in the graph  $H_{uv}$ , induced by  $V_0 \cup \{u, v\}$  on the residual graph of  $H$  with respect to the flow computed so far. Note that, because no vertex of  $V_0$  is an apex of  $H$ ,  $H_{uv}$  is a 2-apex graph with apices  $u, v$ . Borradaile et al. use this fact to compute a maximum flow from  $u$  to  $v$  in  $H_{uv}$  as follows. They split  $u$  into multiple copies, each incident to a different vertex  $w$  for which  $(u, w)$  is an arc of  $H_{uv}$ . A similar process is then applied to  $v$ . Note that the resulting graph is planar. A maximum flow from  $u$  to  $v$  in  $H_{uv}$  is equivalent to a maximum flow with sources

<sup>5</sup> This is a generalization of the notions of saturating and non-saturating Push operations in [4].



the copies of  $u$  and sinks the copies of  $v$  in the resulting graph. This flow can be computed by the multiple-source multiple-sink maximum flow algorithm (the main result in [2]) in  $O(|V(H)| \log^3 |V(H)|)$  time.

The correctness of implementing the Push-Relabel algorithm on  $K^\times$  in this way was proved by Hochstein and Weihe [6] by proving essentially that the algorithm satisfies the properties in Lemma 2. Borradaile et al. used the FIFO policy of Push-Relabel, which guarantees that the number of Push operations is  $O(k^3)$ , so the overall running time of their algorithm is  $O(k^3 |V(H)| \log^3 |V(H)|)$ .

### 3.4 An algorithm for maximum flow in $k$ -apex graphs

We use the algorithm of Borradaile et al. for maximum flow in  $k$ -apex graphs from the previous section, but use our new batch-highest-distance Push-Relabel algorithm instead of the FIFO Push-Relabel algorithm to compute the maximum flow in  $K^\times$ . In order to implement the batch-highest-distance algorithm on  $K^\times$  we only need to maintain the excess  $\text{ex}(\rho, v)$  and labels  $h(v)$  of each vertex  $v \in K^\times$ , and to be able to implement Bulk-Push so that after the execution, property (5\*) is fulfilled. We do not define a flow function in  $K^\times$  nor do we explicitly maintain residual capacities of arcs of  $K^\times$ . Instead, we maintain a preflow  $\rho$  in  $H$ , and say an arc  $(u, v)$  of  $K^\times$  is residual if and only if there exists a residual path from  $u$  to  $v$  in  $H_\rho$  that is internally disjoint from the vertices of  $V^\times$ . Under this definition, there is no path of residual arcs in  $K^\times$  starting at  $s$  and ending at  $t$  if and only if there is no such path in  $H$ . Since  $K^\times$  has  $O(k)$  vertices, by Lemma 6, the algorithm performs  $O(k^2)$  pulses.

We next describe how a Bulk-Push( $U, W$ ) operation in  $K^\times$  is implemented. Let  $A = U \cup W$ . Let  $H_A$  be the graph obtained from  $H_\rho$  by deleting the vertices  $V^\times \setminus A$ . Bulk-Push( $U, W$ ) in  $K^\times$  is implemented by pushing a maximum flow in  $H_A$  with sources the vertices  $U$  and sinks the vertices  $W$ , with the additional restriction that the amount of flow leaving each vertex  $u \in U$  is at most the excess of  $u$ . The efficiency of the procedure depends on how fast we can compute the maximum flow in  $H_A$ . We denote the time to execute a single Bulk-Push operation in the graph  $H_A$  by  $T_{BP}$ . Note that  $T_{BP} = \Omega(k)$ , as it takes  $\Omega(k)$  time to construct  $H_A$  from  $H_\rho$ .

The proof of correctness is an easy adaptation of the proof of Hochstein and Weihe [6]. We cannot use their proof without change because Hochstein and Weihe considered only Push operations along a single arc of  $K^\times$  rather than the Bulk-Push operations which involves more than a single pair of vertices of  $K^\times$ .

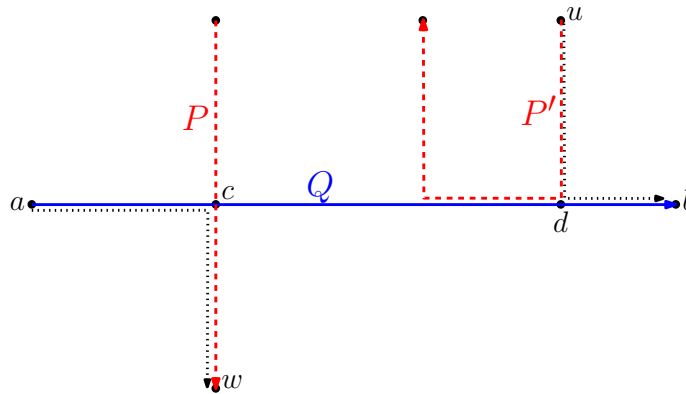
► **Lemma 7.** *Maximum flow in  $k$ -apex graphs can be computed in  $O(k^2 \cdot T_{BP})$  time.*

**Proof.** We first show that the properties (1)-(4) in the statement of Lemma 2, and the generalized property (5\*) from Section 3.2 hold. Property (1) holds since Bulk-Push( $U, W$ ) limits the amount of flow pushed from each vertex  $u \in U$  by the excess of  $u$ . Properties (3) and (4) hold without change since Relabel is not changed.

To show property (5\*) holds, recall that an arc  $(u, w)$  of  $K^\times$  is residual if there exists, in the residual graph  $H_\rho$  with respect to the current preflow  $\rho$ , a residual path from  $u$  to  $w$  that is internally disjoint from any vertex of  $V^\times$ . With this definition it is immediate that property (5\*) holds, since our implementation of Bulk-Push( $U, W$ ) pushes a maximum flow in  $H_A$  from  $U$  that is limited by the excess flow in each vertex of  $U$ . Hence, after Bulk-Push( $U, W$ ) is executed, for every  $u \in U$  and  $w \in W$ , either there is no residual path from  $u$  to  $w$  in  $H_\rho$  that is internally disjoint from  $V^\times$ , or  $u$  is inactive.

As for property (2), since we did not change Relabel,  $h$  remains valid after calls to Relabel. It remains to show that  $h$  remains a valid labeling after Bulk-Push( $U, W$ ). Consider two vertices  $a, b \in V^\times$ . We will show that after Bulk-Push( $U, W$ ), either the arc  $(a, b)$  of  $K^\times$  is





■ **Figure 1** Illustration of property (2) in the proof of Lemma 7. A  $\text{Bulk-Push}(U, W)$  operation pushes flow along paths  $P$  and  $P'$  (dashed red paths). If  $Q$  (blue solid path) is residual after the Bulk-Push operation then the dotted black paths were residual before.

saturated (i.e., is no residual path from  $a$  to  $b$  in  $H_\rho$ ), or  $h(a) \leq h(b) + 1$ . The flow pushed (in  $H_A$ ) by the call  $\text{Bulk-Push}(U, W)$  can be decomposed into a set  $\mathcal{P}$  of flow paths, each of which starts at a vertex of  $U$  and ends at a vertex of  $W$ .

Assume that after performing  $\text{Bulk-Push}(U, W)$  there is an augmenting  $a$ -to- $b$  path,  $Q$  in  $H_\rho$ . If  $Q$  does not intersect any path  $P \in \mathcal{P}$  then  $Q$  was residual before  $\text{Bulk-Push}(U, W)$  was called, so  $h(a) \leq h(b) + 1$  because  $h$  was a valid labeling before the call. Otherwise,  $Q$  intersects some path in  $\mathcal{P}$ . Let  $c, d$  be the first and last vertices of  $Q$  that also belong to paths in  $\mathcal{P}$ . Let  $P, P' \in \mathcal{P}$  be paths such that  $c \in P$  and  $d \in P'$ . Let  $w \in W$  be the last vertex of  $P$  and let  $u \in U$  be the first vertex of  $P'$ . See Figure 1. Then, before  $\text{Bulk-Push}(U, W)$  was called,  $Q[a, c] \circ P[c, w]$  was a residual path from  $a$  to  $w$ , and  $P'[u, d] \circ Q[d, b]$  was a residual path from  $u$  to  $b$ . Since  $h$  was a valid labeling before the call, we have

$$h(u) \leq h(b) + 1 \quad \text{and} \quad h(a) \leq h(w) + 1.$$

Since  $h(u) = h(w) + 1$  it follows that

$$h(a) \leq h(w) + 1 = h(u) \leq h(b) + 1,$$

showing property (2).

We have shown that properties (1)-(4) and (5\*) hold. Hence, by Lemmas 6 and 4, the algorithm terminates after performing  $O(|V^\times|^2) = O(k^2)$  Bulk-Push and Relabel operations. Since each Relabel takes  $O(k)$  time, and each Bulk-Push takes  $\Omega(k)$  time, the total running time of the algorithm is  $O(k^2 \cdot T_{BP})$ . By Lemma 3, when the algorithm terminates there is no residual path from  $s$  to  $t$  in  $K^\times$ . By our definition of residual arcs of  $K^\times$  this implies that there is no residual path from  $s$  to  $t$  in  $H_\rho$ , so  $\rho$  is a maximum flow from  $s$  to  $t$  in  $H$ . ◀

#### 4 A faster algorithm for maximum flow with vertex capacities

In this section, we give a faster algorithm for computing a maximum flow in a directed planar graph with integer arc and vertex capacities bounded by  $C$ , parameterized by the number  $k$  of terminal vertices (sources and sinks). The fastest algorithm currently known for this problem is by Wang [13] and runs in  $O(k^5 \Delta n \text{polylog}(nC))$  time. We first sketch Wang’s algorithm, only detailing the parts that will be modified in our algorithm in Section 4.2.

## 4.1 Wang's algorithm

Wang's algorithm uses some auxiliary graphs, in which only the arcs are capacitated.

► **Definition 8** (The graph  $G^\circ$ ). *For a flow network  $G$ , the network  $G^\circ$  is obtained by the following procedure. For each vertex  $v \in V(G)$ , replace  $v$  with an undirected cycle  $C_v$  with  $d = \deg(v)$  vertices  $v_1, \dots, v_d$ .<sup>6</sup> Each arc in  $C_v$  has capacity  $c(v)/2$ . Connect each arc incident to  $v$  with a different vertex  $v_i$ , preserving the clockwise order of the arcs so that the graph remains planar.*

Recall that if  $H$  and  $G$  are two graphs such that every arc of  $G$  is also an arc of  $H$ , then the restriction of a flow  $f'$  in  $H$  to  $G$  is a flow  $f$  in  $G$  such that  $f(e) = f'(e)$  for all  $e \in E(G)$ . Thus we can speak of the restriction of a flow  $f^\circ$  in  $G^\circ$ , to a flow  $f$  in  $G$ .

Kaplan and Nussbaum [7] used the reduction from  $G$  to  $G^\circ$  to compute a maximum flow in directed planar graphs with vertex capacities and a single source and a single sink. Their algorithm computes a maximum flow  $f^\circ$  in the graph  $G^\circ$ . Let  $f$  be the restriction of  $f^\circ$  to  $G$ . Kaplan and Nussbaum then proved that, provided there is only a single source and a single sink, the values of the maximum flow in  $G^\circ$  and in  $G$  are the same and that if  $f$  is acyclic then it is a feasible maximum flow in  $G$ .

However, when dealing with multiple source and sinks this approach fails because the resulting flow  $f$  may violate vertex capacities in  $G$ . In fact, the value of the maximum flow in  $G^\circ$  may be greater than the value of the maximum flow in  $G$ . Wang proves that the set  $X$  of vertices whose capacities are violated by  $f$  (after canceling flow cycles) has size at most  $k - 2$ , and that the sum of the violations of the vertices in  $X$  is at most  $(k - 2)C$ . In order to overcome this obstacle, Wang's algorithm uses binary search to find the value  $\lambda^*$  of the maximum flow in  $G$ .

Let  $\lambda$  be the current candidate value for  $\lambda^*$ . The algorithm computes a flow  $f^\circ$  with value  $\lambda$  in the graph  $G^\circ$  (recall,  $G^\circ$  has a super source  $s$  connected to all sources, and a super sink  $t$ , connected to all sinks. Thus  $G^\circ$  has an apex set of size 2). Let  $f$  be the restriction of  $f^\circ$  to  $G$ . As long as  $\text{vio}(f) > 2k\Delta$ , the algorithm improves  $f$ . This improvement phase, which will be described shortly, is the crux of the algorithm. If  $\text{vio}(f) \leq 2k\Delta$ , then  $O(k^2\Delta)$  iterations of the classical Ford-Fulkerson algorithm suffice to get rid of all the remaining violations.

The improvement phase of the algorithm is based on finding a circulation  $g$  that cancels the violations on the infeasible vertices and does not create too much violations on other vertices. It can then be shown that adding  $1/(k\Delta) \cdot g$  to the flow  $f$  decreases  $\text{vio}(f)$  by a multiplicative factor of roughly  $1 - 1/(k\Delta)$ . After  $O(k\Delta \log(kC))$  iterations of the improvement step,  $\text{vio}(f)$  is at most  $2k\Delta$ .

In order to find the circulation  $g$  Wang defines the following auxiliary graph.

► **Definition 9** (The graph  $G^\times$ ). *Let  $f$  be a flow in  $G$ . Let  $X$  be the set of infeasible vertices. I.e., vertices  $x \in V(G)$  s.t.  $f^{\text{in}}(x) > c(x)$ . The graph  $G^\times$  is defined as follows. Starting with  $G^\circ$ , for each vertex  $x \in X$ , replace the cycle representing  $x$  with two vertices  $x^{\text{in}}$ ,  $x^{\text{out}}$  and an arc  $(x^{\text{in}}, x^{\text{out}})$  of capacity  $c(x)$ .*

*Every arc of capacity  $c$  going from a vertex  $u \notin C_x$  to a vertex in  $C_x$  becomes an arc  $(u, x^{\text{in}})$  of capacity  $c$ . Similarly, every arc of capacity  $c$  going from a vertex of  $C_x$  to a vertex  $u \notin C_x$  becomes an arc  $(x^{\text{out}}, u)$  with capacity  $c$ .*

<sup>6</sup> By undirected cycle we mean that there are directed arcs in both directions between every pair of consecutive vertices of the cycle  $C_v$ .

Since arcs of  $G^\circ$  can be identified with arcs of  $G^\times$ , the flow  $f^\circ$  can be viewed as a flow  $f^\times$  in  $G^\times$ , setting  $f^\times(x^{in}, x^{out}) = \sum_{(u, x_i) \in E(G^\circ)} f^\circ(u, x_i)$ . The flow  $f^\times$  can then be restricted to a flow in  $G$ .

Wang proves that in order to find the circulation  $g$  described above, it suffices to compute a circulation  $g^\times$  in  $G^\times$  that satisfies the following properties:

1.  $f^\times + g^\times$  is feasible in  $G^\times$ .
2. The restriction of  $f^\times + g^\times$  to  $G$  has no violations on vertices of  $X$ .
3. The restriction of  $f^\times + g^\times$  to  $G$  has at most  $(k - 2)\Delta \cdot \text{vio}(f)$  violation on any vertex in  $V(G) \setminus X$ .

The desired circulation  $g$  is the restriction of  $g^\times$  to  $G$ . If no such  $g^\times$  exists then  $g$  does not exist, which implies that  $\lambda > \lambda^*$ . Otherwise,  $\lambda \leq \lambda^*$ . The binary search for  $\lambda^*$  then continues with a different value of  $\lambda$ .

Wang essentially shows that any algorithm for finding  $g^\times$  in  $O(T)$  time, where  $T = \Omega(n)$ , yields an algorithm for maximum flow with vertex capacities in  $O(k\Delta T \log(kC) \log(nC))$  time. The additional terms stem from the  $O(k\Delta \log(kC))$  iterations of the improvement step, and the  $\log(nC)$  steps of the binary search. Wang shows how to compute  $g^\times$  in  $T = O(k^4 n \log^3 n)$  time, by eliminating the violation at each vertex of  $X$  one after the other in some auxiliary graph obtained from  $G^\times$ . Thus, the overall running time of his algorithm is  $O(k^5 \Delta n \log^3 n \log(kC) \log(nC))$ .

## 4.2 A faster algorithm for computing $g^\times$

We propose a faster way of computing the circulation  $g^\times$  by eliminating the violations in all the vertices of  $X$  in a single shot. Doing so correctly requires some care in defining the appropriate capacities in the auxiliary graph, since we only know that for each  $x \in X$ ,  $g^\times$  should eliminate at least  $\text{vio}(f, x)$  units of flow from  $x$ , but the actual amount of flow eliminated from  $x$  may have to be larger. This issue does not come up when resolving the violations one vertex at a time as was done by Wang.

Define an auxiliary graph  $H$  as follows. Starting with  $G_{f^\times}^\times$ , the residual graph of  $G^\times$  with respect to  $f^\times$ ,

- For each  $x \in X$ , set the capacity of the arc  $(x^{in}, x^{out})$  to be 0 and the capacity of  $(x^{out}, x^{in})$  to be  $c(x)$ .
- Add a super source  $s'$  and arcs  $(s', x^{in})$  with capacity  $\text{vio}(f, x)$  for every  $x \in X$ .
- Add a super sink  $t'$  and arcs  $(x^{out}, t')$  with capacity  $\text{vio}(f, x)$  for every  $x \in X$ .

Note that  $\{s, t\} \cup \bigcup_{x \in X} \{x^{in}, x^{out}\} \cup \{s', t'\}$  is an apex set of size  $O(k)$  in  $H$  (recall from Remark 1 that  $s$  and  $t$  are the super source and super sink of the original graph  $G$ ).

The circulation  $g^\times$  can be found using the following algorithm. Find a maximum flow  $h'$  from  $s'$  to  $t'$  in  $H$  using Lemma 7. Convert  $h'$  to an acyclic flow  $h$  of the same value using the algorithm of Sleator and Tarjan [12] (cf. [13, Lemma 2.5]). If  $h$  does not saturate every arc incident to  $s'$  and  $t'$ , return that the desired circulation  $g$  does not exist. Otherwise,  $h$  can be extended to the desired circulation  $g^\times$  by setting  $g^\times(x^{out}, x^{in}) = h(x^{out}, x^{in}) + \text{vio}(f, x)$  for every  $x \in X$  and  $g^\times(e) = h(e)$  for all other arcs.

The following lemma shows that any single Bulk-Push operation in the algorithm of Lemma 7 on  $H$  can be implemented by a constant number of calls to the  $O(n \log^3 n)$ -time multiple-source multiple-sink maximum flow algorithm in planar graphs of Borradaile et al. [2]. There are two challenges that need to be overcome. First, the graph  $H$  is  $O(k)$ -apex graph rather than planar. Second, the algorithm of Borradaile et al. computes a maximum flow from multiple sources to multiple sinks, not a maximum flow under the restriction that each source sends at most some given limit. This is not a problem in the case of a single

source, or a limit on just the total value of the flow, since then some of the flow pushed can be “undone”. When each of the multiple sources has a different limit, undoing the flow from one source can create residual paths from another source that did not yet reach its limit.

► **Lemma 10.** *Any single Bulk-Push operation in the execution of the algorithm of Lemma 7 on the graph  $H$  defined above can be implemented in  $O(n \log^3 n)$  time.*

**Proof.** Let  $V^\times = \{s, t\} \cup \bigcup_{x \in X} \{x^{in}, x^{out}\} \cup \{s', t'\}$  be the set of apices of  $H$ . Recall that the algorithm of Lemma 7 invokes the batch-highest-distance Push-Relabel algorithm on a complete graph  $K^\times$  over  $V^\times$ , and maintains a corresponding preflow in  $H$ . Consider a single Bulk-Push( $U, W$ ) operation from a set of apices  $U$  to a set of apices  $W$ . Let  $\rho$  denote the preflow pushed in  $H$  up to this Bulk-Push operation. Let  $A = U \cup W$ . To correctly implement Bulk-Push( $U, W$ ), we find a flow  $\rho'$  with sources  $U$  and sinks  $W$  in the graph  $H$ , which satisfies the following properties:

- (i) For every  $u \in U$ ,  $\text{ex}(\rho + \rho', u) \geq 0$ , and
- (ii) For every  $u \in U$  and  $w \in W$ , either  $\text{ex}(\rho + \rho', u) = 0$  or there is no residual path in  $H_{\rho+\rho'}$  from  $u$  to  $w$  that is internally disjoint from  $V^\times$ .

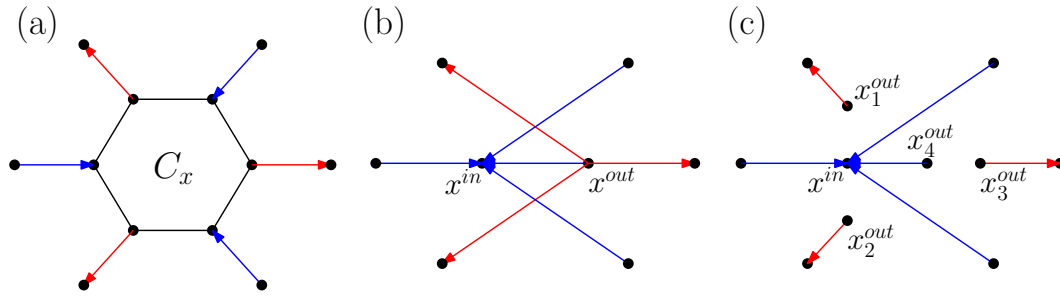
Condition (i) guarantees that  $\rho'$  does not push more flow from a vertex  $u \in U$  than the current excess of  $u$ . Condition (ii) is condition (5\*) from Section 3.2.

Let  $H''$  be the graph obtained from  $H_\rho$  by deleting the vertices  $V^\times \setminus A$ . Note that the absence of residual paths that are internally disjoint from  $V^\times$  in  $H''$  is equivalent to the absence of such paths in  $H$ . We will compute  $\rho'$  using a constant number of invocations of the  $O(n \log^3 n)$ -time multiple-source multiple-sink maximum flow algorithm in planar graphs of Borradaile et al. [2]. Instead of invoking this algorithm on  $H''$ , which is not planar, we shall invoke it on modified versions of  $H''$  which are planar.

Starting with  $H''$ , we split each vertex  $w \in W$  into  $\deg(w)$  copies. Each arc  $e$  that was incident to  $w$  before the split is now incident to a distinct copy of  $w$ , and is embedded so that it does not cross any other arc in the graph. Let  $H'$  denote the resulting graph, and let  $W'$  denote the set of vertices created as a result of splitting all the vertices of  $W$ .

The set  $W'$  replaces  $W$  as the set of sinks of the flow  $\rho'$  we need to compute. Note that  $U$  is an apex set in  $H'$ . We then build the flow  $\rho'$  gradually, by computing the following steps, each using a single invocation of the multiple-source multiple-sink maximum flow algorithm of Borradaile et al. in  $O(n \log^3 n)$  time. In what follows, when we say that the flow  $\rho'$  satisfies condition (ii) for a subset  $U'$  of  $U$  we mean that for every  $u \in U'$  and  $w \in W$ , either  $\text{ex}(\rho + \rho', u) = 0$  or there is no residual path in  $H_{\rho+\rho'}$  from  $u$  to  $w$  that is internally disjoint from  $V^\times$ .

- (1) If  $s \in U$ , starting with  $H'$ , we split  $s$  into  $\deg(s)$  copies. Each arc  $e$  that was incident to  $s$  before the split is now incident to a distinct copy of  $s$ , and is embedded so that it does not cross any other arc in the graph. We also delete all other vertices of  $U$ . We compute in the resulting graph, which is planar, a maximum flow with sources the copies of  $s$  and the sinks  $W'$ . Let  $\rho'_s$  be the flow computed. If  $|\rho'_s| > \text{ex}(\rho, s)$ , we decrease  $|\rho'_s|$  by pushing  $|\rho'_s| - \text{ex}(\rho, s)$  units of flow back from  $W'$  to the copies of  $s$ . This can be done in  $O(n)$  time in reverse topological order w.r.t.  $\rho'_s$  (cf. [2, Section 1.4]). We view  $\rho'_s$  as a flow in  $H$ , and set  $\rho' = \rho'_s$ . By construction  $\rho'$  satisfies condition (i), and satisfies condition (ii) for the subset  $\{s\}$ .
- (2) If  $t \in U$ , starting with  $H'_{\rho'}$ , we repeat step (1) with  $t$  taking the role of  $s$  to compute a flow  $\rho'_t$ . Set  $\rho' \leftarrow \rho' + \rho'_t$ . By construction of  $\rho'_t$ ,  $\rho'$  now satisfies condition (i), and satisfies condition (ii) for the subset  $U \cap \{s, t\}$ .
- (3) Let  $U^{in}$  be the set  $U \cap \{x^{in} : x \in X\}$ . If  $U^{in} \neq \emptyset$ , starting with  $H'_{\rho'}$ , we delete all the vertices of  $U$  that are not in  $U^{in}$ . Note that, since the resulting graph does not contain  $s, t, s', t'$ , nor any  $x^{out}$  for any  $x \in X$ , and since arcs incident to  $x^{in}$  only cross those



■ **Figure 2** Illustration of the auxiliary graphs used in the algorithm of Lemma 10. Only a portion of the graphs around some vertex  $x \in X$  is shown. (a) the graph  $G^\circ$ . (b) the graph  $H$ . Note that the only crossings are between arcs incident to  $x^{in}$  and arcs incident to  $x^{out}$ . (c) the graph  $H'$  in the case that  $x^{out}$  belongs to  $W$ .  $x^{out}$  is split into multiple copies, eliminating all arc crossings.

incident to  $x^{out}$ , the resulting graph is planar. For every  $x^{in} \in U^{in}$  we add a vertex  $x'$  and an arc  $(x', x^{in})$  with capacity  $\text{ex}(\rho, x^{in})$ . The resulting graph is still planar. We compute a maximum flow  $\rho'_{in}$  with sources  $\{x' : x^{in} \in U^{in}\}$  and sinks  $W'$ . We view  $\rho'_{in}$  as a flow in  $H$ , and set  $\rho' \leftarrow \rho + \rho'_{in}$ . By construction of  $\rho'_{in}$ ,  $\rho'$  now satisfies condition (i), and satisfies condition (ii) for the subset  $U \cap (\{s, t\} \cup U^{in})$ .

- (4) We repeat step (3) with  $out$  taking the role of  $in$  to compute a flow  $\rho'_{out}$ . By construction of  $\rho'_{in}$ ,  $\rho'$  now satisfies condition (i), and satisfies condition (ii) for  $U \cap (\{s, t\} \cup U^{in} \cup U^{out})$ . Since  $s'$  and  $t'$  are the source and sink of the flow computed by the Push-Relabel algorithm, they are never active vertices, so they never belong to  $U$ . Hence  $\{s, t\} \cup U^{in} \cup U^{out} \supseteq U$ , and conditions (i) and (ii) are fully satisfied by  $\rho'$ . ◀

Using Lemma 10 and Lemma 7, we get the following lemma.

▶ **Lemma 11.** *The algorithm described above finds a circulation  $g^\times$  such that*

1.  $f^\times + g^\times$  is feasible in  $G^\times$ .
  2. The restriction of  $f^\times + g^\times$  to  $G$  has no violations at vertices of  $X$ .
  3. The restriction of  $f^\times + g^\times$  to  $G$  has violation at most  $(k - 2)\Delta \cdot \text{vio}(f)$  at any vertex in  $V(G) \setminus X$ .
- in  $O(k^2 n \log^3 n)$  time if such a circulation exists.

**Proof.** We first analyze the running time. Computing the graph  $H$  can be done in  $O(n)$  time. Computing the flow  $h'$  in  $H$  using the algorithm of Lemma 7 takes  $O(k^2 \cdot T_{BP})$  time. By Lemma 10,  $T_{BP} = O(n \log^3 n)$  for the graph  $H$ . Transforming  $h'$  to an acyclic flow  $h$  using the algorithm of Sleator and Tarjan [12] takes  $O(n \log n)$  time. Finally, computing  $g^\times$  from  $h$  takes  $O(n)$  time. Hence, the total time to compute  $g^\times$  is  $O(k^2 n \log^3 n)$ .

In order to prove the correctness of the algorithm, we will first prove that there exists a feasible flow  $h$  in  $H$  that saturates every arc incident to  $s'$  and  $t'$  if and only if there exists a circulation  $g^\times$  in  $G^\times$  that satisfies conditions (1) and (2) in the statement of the lemma.

( $\Leftarrow$ ) Assume the circulation  $g^\times$  exists in  $G^\times$ . Define a flow  $h$  in  $H$  as follows. For every arc  $e \in E(H)$  not of the form  $(x^{out}, x^{in})$  set  $h(e) = g^\times(e)$ . For every  $x \in X$ , set  $h(x^{out}, x^{in}) = g^\times(x^{out}, x^{in}) - \text{vio}(f, x)$ ,  $h(s, x^{in}) = \text{vio}(f, x)$  and  $h(x^{out}, t) = \text{vio}(f, x)$ . Since the restriction of  $f^\times + g^\times$  to  $G$  has no violations on the vertices of  $x$ ,  $g^\times(x^{out}, x^{in}) \geq \text{vio}(f, x)$ , so  $h(x^{out}, x^{in}) \geq 0$  and  $h$  is a well defined flow. By definition, the flow  $h$  saturates every arc incident to  $s'$  and  $t'$ .

To show that  $h$  is feasible in  $H$  it is enough to show that  $h(x^{out}, x^{in}) \leq c(x)$  for every  $x \in X$  (on all other arcs  $h$  is feasible because  $g^\times$  is feasible in  $G_{f^\times}^\times$ ). Let  $x \in X$ . Since  $f^\times + g^\times$  is feasible in  $G^\times$ ,  $g^\times(x^{out}, x^{in}) \leq f^\times(x^{in}, x^{out})$ . Since  $h(x^{out}, x^{in}) = g^\times(x^{out}, x^{in}) - \text{vio}(f, x)$ ,  $h(x^{out}, x^{in}) \leq f^\times(x^{in}, x^{out}) - \text{vio}(f, x) = c(x)$ .

( $\Rightarrow$ ) Assume there exist a feasible flow  $h$  in  $H$  that saturates every arc incident to  $s'$  and  $t'$ , and let  $g^\times$  be the circulation obtained from  $h$  as described above. We show that  $f^\times + g^\times$  is feasible in  $G^\times$ . On all arcs  $e$  not of the form  $(x^{out}, x^{in})$ ,  $g^\times(e) = h(e)$  and the capacity of  $e$  in  $G_{f^\times}^\times$  equals the capacity of  $e$  in  $H$ . Therefore, since  $h$  is a feasible flow in  $H$ ,  $g^\times$  is feasible on  $e$  in  $G_{f^\times}^\times$ , so  $f^\times + g^\times$  is feasible on  $e$  in  $G^\times$ . We now focus on the arcs  $(x^{out}, x^{in})$  for each  $x \in X$ . Let  $e = (x^{out}, x^{in})$ . Observe that  $0 \leq h(e) \leq c(x)$ . Since  $g^\times(e) = h(e) + \text{vio}(f, x)$  we have that  $\text{vio}(f, x) \leq g^\times(e) \leq c(x) + \text{vio}(f, x) = f^\times(e)$ . Since  $(f^\times + g^\times)(x^{in}, x^{out}) = f^\times(x^{in}, x^{out}) - g^\times(x^{out}, x^{in})$ , we have  $0 \leq (f^\times + g^\times)(x^{in}, x^{out}) \leq c(x)$ , so  $f^\times + g^\times$  is feasible in  $G^\times$ .

To finish proving the ( $\Rightarrow$ ) direction, we show that the restriction of  $f^\times + g^\times$  to  $G$  has no violations on the vertices of  $X$ . By definition of  $G^\times$  and of residual graph, the only arcs in  $G_{f^\times}^\times$  that can carry flow out of  $x^{in}$  are the reverses of the arcs that carry flow into  $x$  in  $f$ , and the only arcs that can carry flow into  $x^{out}$  are the reverses of the arcs that carry flow out of  $x$  in  $f$ . We will show that  $(f + g)^{in}(x) \leq c(x)$  by considering separately the contribution of the flow on arcs of  $G$  that in  $G^\times$  are incident to  $x^{out}$ , and arcs of  $G$  that in  $G^\times$  are incident to  $x^{in}$ .

The only arc of  $f^\times$  that carries flow into  $x^{out}$  is  $(x^{in}, x^{out})$ . Thus, there is no arc  $e$  of  $G$  such that  $f^\times(e)$  carries flow into  $x^{out}$ . Since  $g^\times$  only carries flow into  $x^{out}$  along the reverses of arcs that carry flow out of  $x^{out}$  in  $f^\times$  and since for every such arc  $e'$ ,  $g^\times(e') \leq f^\times(\text{rev}(e'))$ , there is also no arc  $e$  of  $G$  such that  $(f^\times + g^\times)(e)$  carries flow into  $x^{out}$ .

The total flow that  $f^\times$  carries into  $x^{in}$  is  $c(x) + \text{vio}(f, x)$ . Let  $z$  denote the total amount of flow that  $g^\times$  carries into  $x^{in}$ . Since the only arc incident to  $x^{in}$  that carries flow in  $g^\times$  and does not belong to  $G$  is  $(x^{out}, x^{in})$ , the total amount of flow that  $g^\times$  carries into  $x^{in}$  on arcs that belong to  $G$  is  $z - g^\times(x^{out}, x^{in})$ . On the other hand,  $g^\times$  carries  $z$  units of flow out of  $x^{in}$ , and all of this flow is pushed along the reverses of arcs that carry flow into  $x^{in}$  in  $f^\times$  (and also belong to  $G$ ). Hence, the total amount of flow that  $f^\times + g^\times$  carries into  $x^{in}$  on arcs that belong to  $G$  is  $c(x) + \text{vio}(f, x) + (z - g^\times(x^{out}, x^{in})) - z$ . Therefore,

$$\begin{aligned} (f + g)^{in}(x) &= c(x) + \text{vio}(f, x) - g^\times(x^{out}, x^{in}) \\ &= c(x) + \text{vio}(f, x) - (h(x^{out}, x^{in}) + \text{vio}(f, x)) \\ &= c(x) - h(x^{out}, x^{in}) \\ &\leq c(x). \end{aligned}$$

We have thus shown that the algorithm computes a flow  $g^\times$  satisfying conditions (1) and (2) in the statement of the lemma. To see that condition (3) is also satisfied, note that the value of the flow  $h$  is  $\sum_{x \in X} \text{vio}(f, x) \leq (k-2) \cdot \text{vio}(f)$ . Since  $h$  is acyclic,  $h^{in}(v) \leq (k-2) \cdot \text{vio}(f)$  for all  $v \in H$ . Since for all  $v \in V(G) \setminus X$ ,  $f^{in}(v) \leq c(v)$ , and  $h^{in}(v) \leq \Delta(g^\times)^{in}(v)$ , it follows that the violation of  $f^\times + g^\times$  at  $v$  is at most  $(k-2)\Delta \cdot \text{vio}(f)$ .  $\blacktriangleleft$

Using the  $O(k^2 n \log^3 n)$ -time algorithm of Lemma 11 in the improvement phase of Wang's algorithm instead of using Wang's  $O(k^4 n \log^3 n)$ -time procedure for this phase results in a running time of  $O(k^3 \Delta n \text{polylog}(nC))$  for finding a maximum flow in  $G$ .



### 4.3 The case $k = o(\log^2 n)$

We now provide an algorithm that is faster for small  $k = o(\log^2 n)$ . Specifically, we describe an algorithm for computing the circulation  $g^\times$  that runs in  $O(k^3 n \log n)$  time instead of the  $O(k^2 n \log^3 n)$  time required by the algorithm of Lemma 11.

We use the same auxiliary graph  $H$  as defined above and again compute a maximum flow  $h'$  from  $s'$  to  $t'$  in  $H$ . Let  $V^\times = \{s, t\} \cup \bigcup_{x \in X} \{x^{in}, x^{out}\} \cup \{s', t'\} \cup S \cup T$  be the set of apices of  $H$  along with the original sources  $S$  and sinks  $T$  of  $G$ , and let  $K^\times$  be the complete graph on  $V^\times$ . Instead of using the batch-highest-distance Push-Relabel algorithm as in Lemma 7, we more directly follow the strategy of Borradaile et al. [2] by simulating a maximum flow computation from  $s$  to  $t$  in  $K^\times$  using the FIFO Push-Relabel algorithm. We do not wish to directly use the multiple-source multiple-sink flow algorithm of Borradaile et al. [2], because then each of the  $O(k^3)$  Push operations would take  $O(n \log^3 n)$  time.

► **Lemma 12.** *Any single (individual arc) Push operation in the graph  $H$  defined above can be implemented in  $O(n \log n)$  time.*

**Proof.** Consider a single Push( $u, v$ ) operation where  $u, v \in V^\times$ . Let  $\rho$  denote the preflow pushed in  $H$  by the FIFO Push-Relabel algorithm up to this Push operation. We find a flow  $\rho'$  with source  $u$  and sink  $v$  in the graph  $H$  such that either  $\text{ex}(\rho + \rho', u) = 0$  or  $\text{ex}(\rho + \rho', u) > 0$  and there is no residual path in  $H_{\rho+\rho'}$  from  $u$  to  $v$  that is internally disjoint from  $V^\times$ .

Let  $H'$  be the graph obtained from  $H_\rho$  by deleting vertices  $V^\times \setminus \{u, v\}$ . Instead of invoking the  $O(n \log^3 n)$ -time multiple-source multiple-sink maximum flow algorithm of Borradaile et al. [2], we will compute  $\rho'$  as follows. As before, we must consider a few different cases.

- If  $u = s$  or  $v = s$ , then  $v \in S$  or  $u \in S$ , respectively. We push up to  $\text{ex}(\rho, u)$  units of flow directly along the arc  $(u, v)$  in constant time, either saturating the arc or reducing the excess flow in  $u$  to 0. We may similarly push directly along the arc  $(u, v)$  in constant time if one of  $u$  or  $v$  is one of  $t, s',$  or  $t'$  instead.
- If  $u \in \{x_1^{in}, x_1^{out}\}$  and  $v \in \{x_2^{in}, x_2^{out}\}$  for two distinct vertices  $x_1, x_2 \in X$ , then the graph  $H'$  is planar. We add a vertex  $u'$  and an arc  $(u', u)$  with capacity  $\text{ex}(\rho, u)$  and compute the maximum flow  $\rho'$  with source  $u'$  and sink  $v$  using the single-source single-sink maximum flow algorithm in planar graphs of Borradaile and Klein [1].
- If neither of the above cases apply, then  $u, v \in \{x^{in}, x^{out}\}$  for some  $x \in X$ . If arc  $(u, v)$  has positive residual capacity, we push up to  $\text{ex}(\rho, u)$  units of flow directly along it in constant time. Similar to Step 1 in the proof of Lemma 10, starting with  $H'$ , we split  $u$  into  $\text{deg}(u)$  copies so that each arc that was incident to  $u$  is now incident to a distinct copy of  $u$ . Similarly, we split  $v$  into  $\text{deg}(v)$  copies so each arc that was incident to  $v$  is now incident to a distinct copy of  $v$ . The resulting graph is planar, and all copies of  $u$  and  $v$  lie on a common face. As mentioned by Borradaile et al. [2, p. 1280], we can then plug the linear time shortest paths in planar graphs algorithm of Henzinger et al. [5] into a divide-and-conquer procedure of Miller and Naor [10] to compute a maximum flow  $\rho'_u$  with sources the copies of  $u$  and sinks the copies of  $v$  in  $O(n \log n)$  time. Again, if the value of this flow is greater than the excess of  $u$ , we push the appropriate amount of flow back to the copies of  $u$  in  $O(n)$  time. Finally, we view  $\rho'_u$  as a flow in  $H$  to set  $\rho' = \rho'_u$ . ◀

We immediately get a variation of Lemma 11 with a running time of  $O(k^3 n \log n)$ . By using the better of the two procedures for computing  $g^\times$ , we get our main theorem.

► **Theorem 13.** *A maximum flow in an  $n$ -vertex planar flow network  $G$  with integer arc and vertex capacities bounded by  $C$  can be computed in  $O(k^3 \Delta n \log n \min(k, \log^2 n) \log(kC) \log(nC))$  time.*



## References

- 1 Glencora Borradaile and Philip N. Klein. An  $O(n \log n)$  algorithm for maximum  $st$ -flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.
- 2 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.
- 3 Joseph Cheriyan and S. N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. *SIAM J. Comput.*, 18(6):1057–1086, 1989. doi:10.1137/0218072.
- 4 Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.
- 5 Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997. doi:10.1006/jcss.1997.1493.
- 6 Jan M. Hochstein and Karsten Weihe. Maximum  $s$ - $t$ -flow with  $k$  crossings in  $O(k^3 n \log n)$  time. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 843–847. SIAM, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283473>.
- 7 Haim Kaplan and Yahav Nussbaum. Maximum flow in directed planar graphs with vertex capacities. *Algorithmica*, 61(1):174–189, 2011. doi:10.1007/s00453-010-9436-7.
- 8 Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost  $O(m^{4/3})$  time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 119–130. IEEE, 2020. doi:10.1109/FOCS46700.2020.00020.
- 9 Samir Khuller and Joseph Naor. Flow in planar graphs with vertex capacities. *Algorithmica*, 11(3):200–225, 1994. doi:10.1007/BF01240733.
- 10 Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995. doi:10.1137/S0097539789162997.
- 11 James B. Orlin. Max flows in  $O(nm)$  time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.
- 12 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.
- 13 Yipu Wang. Maximum integer flows in directed planar graphs with vertex capacities and multiple sources and sinks. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 554–568. SIAM, 2019. doi:10.1137/1.9781611975482.35.
- 14 Xianchao Zhang, Weifa Liang, and Guoliang Chen. Computing maximum flows in undirected planar networks with both edge and vertex capacities. In Xiaodong Hu and Jie Wang, editors, *Computing and Combinatorics, 14th Annual International Conference, COCOON 2008, Dalian, China, June 27-29, 2008, Proceedings*, volume 5092 of *Lecture Notes in Computer Science*, pages 577–586. Springer, 2008. doi:10.1007/978-3-540-69733-6\_57.

# Maximum-Weight Matching in Sliding Windows and Beyond

Leyla Biabani ✉

Department of Computer Science, TU Eindhoven, The Netherlands

Mark de Berg ✉

Department of Computer Science, TU Eindhoven, The Netherlands

Morteza Monemizadeh ✉

Department of Computer Science, TU Eindhoven, The Netherlands

---

## Abstract

We study the maximum-weight matching problem in the sliding-window model. In this model, we are given an adversarially ordered stream of edges of an underlying edge-weighted graph  $G(V, E)$ , and a parameter  $L$  specifying the *window size*, and we want to maintain an approximation of the maximum-weight matching of the current graph  $G(t)$ ; here  $G(t)$  is defined as the subgraph of  $G$  consisting of the edges that arrived during the time interval  $[\max(t - L, 1), t]$ , where  $t$  is the current time. The goal is to do this with  $\tilde{O}(n)$  space, where  $n$  is the number of vertices of  $G$ . We present a deterministic  $(3.5 + \varepsilon)$ -approximation algorithm for this problem, thus significantly improving the  $(6 + \varepsilon)$ -approximation algorithm due to Crouch and Stubbs [5].

We also present a generic machinery for approximating subadditive functions in the sliding-window model. A function  $f$  is called *subadditive* if for every disjoint substreams  $A, B$  of a stream  $S$  it holds that  $f(AB) \leq f(A) + f(B)$ , where  $AB$  denotes the concatenation of  $A$  and  $B$ . We show that given an  $\alpha$ -approximation algorithm for a subadditive function  $f$  in the insertion-only model we can maintain a  $(2\alpha + \varepsilon)$ -approximation of  $f$  in the sliding-window model. This improves upon recent result Krauthgamer and Reitblat [14], who obtained a  $(2\alpha^2 + \varepsilon)$ -approximation.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** maximum-weight matching, sliding-window model, approximation algorithm, and subadditive functions

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.73

**Funding** The work in this paper is supported by the Dutch Research Council (NWO) through Gravitation-grant NETWORKS-024.002.003.

## 1 Introduction

A *matching* in a graph  $G(V, E)$  is a subset  $M \subseteq E$  of pairwise non-adjacent edges. Matchings play an important role in applications and they form a central concept in combinatorial and algorithmic graph theory. They have been studied extensively; there is even a book, by Lovasz and Plummer, dedicated completely to matching theory [16]. One of the most studied algorithmic problems concerning matchings is the *maximum-matching problem*. In the unweighted version, the goal is to compute a maximum-cardinality matching and in the weighted version – here every edge in the input graph has a non-negative weight – the goal is to compute a maximum-weight matching (MWM). In the classical offline setting we are given the graph  $G$  completely in advance and we have enough space to store all vertices and edges. In this setting, the fastest algorithm to compute a maximum matching is still the 30-years-old algorithm due to Micali and Vazirani [17] with running time  $O(m\sqrt{n})$ , where  $n := |V|$  and  $m := |E|$ .



© Leyla Biabani, Mark de Berg, and Morteza Monemizadeh;  
licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 73; pp. 73:1–73:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study the maximum-weight matching problem in a streaming setting. Here we are given a stream  $S$  of edges of an underlying graph  $G(V, E)$  and we only have  $\tilde{O}(n) := O(n \cdot \text{polylog}(n))$  storage available. This means that we cannot store all edges from  $G$ , and it becomes impossible to compute a solution that is guaranteed to be optimal. The goal thus becomes to maintain a matching on the current graph whose total weight is as close to optimal as possible. In the *insertion-only model*, the current graph  $G(t)$  is the graph whose edge set consists of all edges that have arrived up to the current time  $t$ . We will be working in the *sliding-window model* [6], where we are given a window length  $L$  and the current graph  $G(t)$  consists of the last  $L$  edges from the stream. Thus, at any time  $t$  we are interested in approximating the maximum-weight matching of the edges that are within the window  $[\max(t - L, 1), t]$ .

**Our results.** We present a  $(3.5 + \varepsilon)$ -approximation for the maximum-weight matching problem in the sliding-window model. The following theorem states our main result more formally.

► **Theorem 1.** *Let  $G(V, E)$  be a graph with  $n = |V|$  vertices and let  $w : E \rightarrow \mathbb{R}^+$  be a function that assigns a weight  $w(e)$  to each edge  $e \in E$  such that the ratio of the maximum edge weight to the minimum edge weight is at most  $W$  for some  $W = n^{O(1)}$ . Then, for any given  $\varepsilon > 0$ , there exists a deterministic algorithm that maintains a  $(3.5 + \varepsilon)$ -approximate maximum-weight matching in the sliding-window model using  $\tilde{O}(n)$  space.*

A comparison of our result with the existing streaming algorithms for maximum-weight matchings is given in Table 1.

The algorithm in Theorem 1 takes the algorithm by Paz and Schwartzman [18] as a starting point. Our  $(3.5 + \varepsilon)$ -approximation is then based on two contributions: a mechanism to convert an algorithm in the insertion-only model into an algorithm in the sliding-window model, and an intricate analysis of the resulting algorithm in the case of maximum-weight matchings. The mechanism to obtain a sliding-window algorithm from an insertion-only algorithm actually works in a much more general setting, namely when we want to compute subadditive functions, defined as follows. Let  $f$  be a function that assigns a value to a data stream. Then  $f$  is called *subadditive* if for every disjoint substreams  $A, B$  of a stream  $S$  it holds that  $f(AB) \leq f(A) + f(B)$ , where  $AB$  denotes the concatenation of  $A$  and  $B$ . We say that  $f$  is *monotone* if for all  $A \subseteq B$  we have  $f(A) \leq f(B)$ . Observe that (the cost of a) maximum weight matching is a subadditive and monotone function. We prove the following result in Section 4.

► **Theorem 2.** *Let  $0 < \varepsilon \leq 1/2$  and  $\alpha \geq 1$  be two parameters. Let  $f$  be a function defined on streams that is subadditive, non-negative, and monotone. Let  $\sigma := f_{\max}/f_{\min+}$ , where  $f_{\min+} := \min\{f(X) : X \text{ is a substream of the input and } f(X) > 0\}$  and  $f_{\max} := \max\{f(X) : X \text{ is a substream of the input}\}$ . Suppose there is an algorithm in the insertion-only streaming model that  $\alpha$ -approximates  $f$  using space  $s$ . Then, there is an algorithm in the sliding-window model that maintains a  $(2\alpha + \varepsilon)$ -approximation of  $f$  using  $O(\varepsilon^{-1}s \cdot \log \sigma)$  space.*

**Previous Work.** Given the prominence of the matching problem, it is not surprising that it has already received considerable attention in the streaming setting.

In the insertion-only model, a maximal matching – a matching  $M$  such that adding any edge from  $E$  to  $M$  would violate the condition that the edges are pairwise non-adjacent – can be computed greedily using  $O(n)$  space. This immediately gives a 2-approximation for maximum-cardinality matching (MCM) [9]. If the stream is in adversarial order, obtaining an

approximation ratio better than 2 for the MCM problem is one of the most important open problems in streaming algorithms. Interestingly, Kapralov [12] showed that any streaming algorithm that achieves an approximation ratio better than  $e/(1-e) \approx 1.58$  must use  $n^{1+\Omega(\frac{1}{\log \log n})}$  storage. In the *random-order model*, where the edges from  $E$  arrive in random order, better approximation ratios than 2 have been obtained [13, 10, 7, 8, 1]. In particular, very recently, Bernstein [1] showed that we can compute a  $(\frac{3}{2} + \varepsilon)$ -approximate matching in random order streams using  $\tilde{O}(n)$  space.

Maximum-weighted matchings (MWM) have also been studied in the insertion-only model. Crouch and Stubbs [5] presented a technique that makes it possible to turn a  $c$ -approximation algorithm for the MCM problem into a  $2(1+\varepsilon)c$ -approximation algorithm for the MWM problem. A combination of this reduction and the greedy matching achieves a  $(4+\varepsilon)$ -approximation algorithm for the maximum-weight matching problem in the insertion-only model. In a breakthrough result, Paz and Schwartzman [18] developed a  $(2+\varepsilon)$ -approximation algorithm for the maximum-weight matching problem in the insertion-only model, using  $O(n \log^2 n)$  bits of space. Later, Ghaffari and Wajc [11] improved the space of Paz and Schwartzman's algorithm to  $O(n \log n)$  bits of space.

We now turn our attention to the sliding-window model. Braverman and Ostrovsky [2] introduced *smooth histograms*, a powerful framework to maintain a class of functions, called smooth function in the sliding-window model. Crouch, McGregor, and Stubbs [4] showed that the smooth-histograms technique can be applied to maintain an approximately maximal matching that achieves a  $(3+\varepsilon)$ -approximation of the maximum-cardinality matching. They also presented a 9.027-approximation algorithm for the maximum-weight matching in the sliding-window model. This can be improved using the already mentioned (more recent) technique by Crouch and Stubbs [5] to turn a  $c$ -approximation algorithm for the MCM problem into a  $2(1+\varepsilon)c$ -approximation algorithm for the MWM problem. Using this reduction and the algorithm of [4] we can achieve  $(6+\varepsilon)$ -approximation weighted matching in the sliding-window model. This reduction, which yields the best previously known result for the MWM problem in the sliding-window model, partitions the edges into weight classes and maintains a matching on the edges in each weight class. At the end of the stream, it greedily merges the matchings from largest weight class to smallest weight class. However, our algorithm is based on the algorithm by Paz and Schwartzman [18] that is explained in Section 2.

We next explain the previous work on subadditive functions in data streams. Let  $f$  be a function defined on streams, which is subadditive, non-negative, and monotone. Recently, Krauthgamer and Reitblat [14] showed that given a streaming algorithm  $\mathcal{ALG}$  that  $\alpha$ -approximates  $f$  using space  $s$ , we can develop a sliding-window algorithm that  $(2\alpha^2 + \varepsilon)$ -approximates  $f$  using space  $O(\varepsilon^{-1}s \cdot \log \sigma)$ , where  $\sigma$  is ratio of the maximum value of  $f$  to the minimum value of  $f$ . They also showed that if  $\mathcal{ALG}$  is monotone and subadditive, we can reduce the approximation factor of the sliding-window algorithm down to  $(2\alpha + \varepsilon)$ -factor. Unfortunately, it is not always easy to show a streaming algorithm is monotone and/or subadditive. In some cases, this is not in fact, the case. On the other hand, Theorem 2 shows that we can achieve  $(2\alpha + \varepsilon)$ -approximation factor independent of the monotonicity or subadditivity of the streaming algorithm  $\mathcal{ALG}$ . We should mention that there is a similar result for constrained submodular maximization in the sliding-window model due to Chen, Nguyen, and Zhang [3].

**Notation and terminology.** In this paper, we assume we are given a weighted graph  $G(V, E)$  with  $n = |V|$  vertices and a weight function  $w : E \rightarrow \mathbb{R}^+$  that assigns a non-negative weight  $w(e)$  to each edge  $e = (u, v) \in E$ , where the ratio of the maximum edge weight to the

■ **Table 1** Overview of results on the maximum-weight matching problem.

Model	Problem	Approximation	Adversarial Order	Reference
insertion-only	MCM	2	✓	[9]
		$1.5 + \varepsilon$	Random Order	[1]
	MWM	$4 + \varepsilon$	✓	[5]
		$2 + \varepsilon$	✓	[18]
sliding-window	MCM	$3 + \varepsilon$	✓	[4]
	MWM	9.027	✓	[4]
		$6 + \varepsilon$	✓	[5]
		$3.5 + \varepsilon$	✓	[this paper]

minimum edge weight is upper-bounded by  $W = n^{O(1)}$ . We assume that the graph is simple, that is, it does not have parallel edges. We denote by  $N_G(v) = \{u \in V : \exists(u, v) \in E\}$  the set of neighbors of a vertex  $v$  in  $G$ . We drop the subscript  $G$  and write  $N(v)$  when  $G$  is clear from the context. Similarly, we denote by  $N_G(e)$  the set of edges that are neighbors of an edge  $e = (u, v)$ , that is, those edges such that one of their endpoints is in the  $\{u, v\}$ . A *matching*  $M$  of  $G$  is a set of pairwise non-adjacent edges, that is, a set where no two edges share a common vertex. The weight of the matching  $M$  is defined as  $w(M) = \sum_{e \in M} w(e)$ . A *maximum matching* of graph  $G(V, E)$  is a matching of maximum weight. Throughout the paper, when we fix a maximum-weight matching of a graph  $G$ , we denote it by  $M_{\text{opt}}(G)$ , or simply  $M_{\text{opt}}$  when  $G$  is clear from the context. We say a matching  $M$  is an  $\alpha$ -*approximate weighted matching*, for some  $\alpha > 1$ , when  $w(M) \geq (1/\alpha) \cdot w(M_{\text{opt}})$ . Finally, with a slight abuse of notation, we will often not distinguish between sets of edges and streams of edges. As an example, given a stream  $S$  of edges of an underlying graph  $G(V, E)$ , we do not distinguish between  $G(V, E)$  and  $G(V, S)$ .

## 2 Maximum-weight matching in the insertion-only model

As mentioned in the introduction, in a breakthrough paper [18], Paz and Schwartzman showed that there exists an algorithm in the insertion-only model that computes a  $(2 + \varepsilon)$ -approximate weighted matching of a graph  $G(V, E)$ . Their algorithm forms the basis of our  $(3.5 + \varepsilon)$ -approximation algorithm in the sliding-window model. In this section, we explain Paz and Schwartzman’s algorithm and we give various properties – some already proved by Ghaffari and Wajc [11], some new – that we later use.

**Overview of algorithm.** Before the stream starts, we initialize an empty stack. The streaming algorithm then processes the edges in the stream one by one. While doing so, it maintains a *potential*  $\phi(v)$  for every vertex  $v \in V$  (which is initialized to zero before the stream starts). For each arriving edge  $e = (u, v)$ , it is decided whether or not to push  $e$  on the stack based on its weight  $w(e)$  and on the potential of its endpoints. In particular, if  $w(e) \geq (1 + \varepsilon) \cdot (\phi(u) + \phi(v))$ , then  $e$  is added to the stack. When  $e$  is added to the stack, we assign a *reduced weight*  $w'(e)$  to it, defined as  $w'(e) := w(e) - (\phi(u) + \phi(v))$ ; from now on we refer to  $w(e)$  as the *original weight* of  $e$ . If  $e$  is added to the stack, we add  $w'(e)$  to the potential of its endpoints. It will be convenient to set  $w'(e) := 0$  when  $e$  is not added to the stack.

After all edges have been handled in this manner, the matching is computed greedily: we pop edges from the stack and add them to the matching  $M_{\text{alg}}(G)$  if they have no neighboring edge in  $M_{\text{alg}}(G)$ .

---

**Algorithm 1** MWM-STREAMING [18, 11].
 

---

**Streaming:**

```

1: while a new edge  $e = (u, v)$  of the stream  $S$  is revealed do
2:   if  $w(e) < (1 + \varepsilon) \cdot (\phi(u) + \phi(v))$  then  $w'(e) \leftarrow 0$ 
3:   else
4:      $w'(e) \leftarrow w(e) - (\phi(u) + \phi(v))$   $\triangleright w'(e)$  is the reduced weight of  $e$ 
5:      $\phi(u) \leftarrow \phi(u) + w'(e)$ ;  $\phi(v) \leftarrow \phi(v) + w'(e)$   $\triangleright$  update potentials
6:      $Stack.push(e)$ 

```

---

**Postprocessing:**

```

1: Let  $M_{alg}(G) \leftarrow \emptyset$  be an empty matching set.
2: while  $Stack \neq \emptyset$  do
3:    $e \leftarrow Stack.pop()$ 
4:   if  $M_{alg}(G) \cap N(e) = \emptyset$  then  $M_{alg}(G) \leftarrow M_{alg}(G) \cup \{e\}$ .
5: return  $M_{alg}(G)$ .

```

---

**Properties of the algorithm.** Let  $t_e$  be the arrival time of the edge  $e$ . Let  $P(e) := \{e' \in E : e' \in N(e) \cup \{e\} \text{ and } t_{e'} \leq t_e\}$  be the set of neighbor edges of  $e$  that arrive in the stream before  $e$  plus the edge  $e$  itself. Observe that if  $e$  is not added to the stack, then  $w(e) < (1 + \varepsilon) \cdot (\phi(u) + \phi(v))$ ; and if this happens, the reduced weight must be  $w'(e) = 0$ .

The first lemma shows that the original weight of every edge in the graph is upper-bounded by  $(1 + \varepsilon)$ -factor of the sum of the potentials that are assigned to its end-points regardless of the fact that the edge is pushed onto the stack or not. The following lemma was proved by Ghaffari and Wajc [11, Observation 3.2].

► **Lemma 3** ([11]). *After the algorithm has finished we have  $w(e) \leq (1 + \varepsilon) \cdot (\phi(v) + \phi(u))$  for each  $e = (u, v)$  in  $E$ .*

The next lemma gives a relation between the original weight  $w(e)$  of an edge  $e$  and the reduced weights  $w'(e')$  of the edges  $e' \in P(e)$ . It is a special case of Lemma 3.4 of [11], where it was stated that  $w(e) \geq \sum_{e' \in P(e)} w'(e')$ . As follows from their proof, we can only have  $w(e) > \sum_{e' \in P(e)} w'(e')$  when there are parallel edges. As we assume in this paper that  $G$  is a simple graph (i.e., does not contain any parallel edge) we state the result as follows.

► **Lemma 4** ([11]). *Suppose that the graph  $G(V, E)$  is a simple graph. Let  $e \in E$  be an edge that is added to the stack. Then  $w(e) = \sum_{e' \in P(e)} w'(e')$ .*

Lemma 4 is about edges that are added to the stack. The next lemma states a similar result for any edge  $e$ , regardless of whether it is added to the stack.

► **Lemma 5.** *For each  $e \in E$  we have  $w(e) \leq (1 + \varepsilon) \sum_{e' \in P(e)} w'(e')$ .*

**Proof.** If  $e = (u, v)$  is added to the stack, the inequality holds according to Lemma 4. Suppose that  $e$  is not added to the stack. We use  $\phi_e^-(u)$  and  $\phi_e^-(v)$  for the potential values just before the arrival of  $e$ . As  $e$  is not added to the stack, we have  $w(e) < (1 + \varepsilon) \cdot (\phi_e^-(u) + \phi_e^-(v))$ . Let  $e'$  be an edge in  $P(e) \setminus \{e\}$ . As  $G$  does not contain parallel edges,  $e'$  increases exactly one of  $\phi(u)$  or  $\phi(v)$  by  $w'(e')$ . (This also holds when  $w'(e') = 0$ .) Therefore,  $\phi_e^-(u) + \phi_e^-(v) = \sum_{e' \in P(e) \setminus \{e\}} w'(e')$ . Thus,

$$w(e) < (1 + \varepsilon) \cdot (\phi_e^-(v) + \phi_e^-(u)) = (1 + \varepsilon) \cdot \sum_{e' \in P(e) \setminus \{e\}} w'(e') \leq (1 + \varepsilon) \cdot \sum_{e' \in P(e)} w'(e') . \quad \blacktriangleleft$$



We also need Lemma 3.1 of [11], which states that the weight of the reported matching  $M_{\text{alg}}(G)$  is lower-bounded by the sum of reduced weights of all edges of the graph  $G$ , which is in fact, one half of the sum of potentials that we place on vertices. The latter is a  $(1 + \varepsilon)$ -approximation of the optimal weight  $w(M_{\text{opt}}(G))$ .

► **Lemma 6** ([11]). *Let  $M_{\text{alg}}(G)$  be the matching reported by Algorithm MWM-STREAMING. Then,*

$$w(M_{\text{alg}}(G)) \geq \sum_{e \in E} w'(e) = \frac{1}{2} \sum_{v \in V} \phi(v) \geq \frac{1}{2(1 + \varepsilon)} \cdot w(M_{\text{opt}}(G)) .$$

Suppose we have two substreams  $B$  and  $BC$ , i.e.,  $B$  is a prefix of  $BC$ . The next lemma shows that the reduced weight of an edge  $e \in B$  is the same when the algorithm is run on  $B$  as when it is run on  $BC$ . It immediately follows from the fact that  $w'(e)$  is set when  $e$  is processed, and it will not be changed afterwards.

► **Lemma 7.** *Let  $S$  be a stream of edges of an underlying simple weighted graph  $G(V, E)$ . Let  $B, C$  be two disjoint segments of the stream  $S$ . Let  $e \in B$  be an arbitrary edge in  $B$ . Let  $w'_B(e)$  and  $w'_{BC}(e)$  be the reduced weights of  $e$  when we run Algorithm MWM-STREAMING on the streams  $B$  and  $BC$ , respectively. Then,  $w'_B(e) = w'_{BC}(e)$ .*

**Making MWM-Streaming monotone.** Later, when we develop our sliding-window algorithm, we need Algorithm MWM-STREAMING to be monotone, that is, we need that the weight of the reported matching for a stream  $BC$  is at least the weight of the reported matching for  $B$ . Unfortunately, this need not be the case, because in the reporting phase the edges are popped from the stack in reverse order of their arrival and added to the matching greedily.

However, we can simply make Algorithm MWM-STREAMING monotone by maintaining the best solution over all prefixes encountered so far. More precisely, while we run the algorithm on the input stream  $S$  as usual, we maintain  $M_{\text{mon}}(S) := \arg \max\{w(M_{\text{alg}}(T)) : T \text{ is a prefix of } S\}$ . To this end, we just run the Postprocessing subroutine after each arrival on a copy of the current stack, and update  $M_{\text{mon}}(S)$  if the computed solution is better. We denote by MWM-MONOTONE this monotone version of Algorithm MWM-STREAMING.

Next, we state two useful properties of the monotone matching  $M_{\text{mon}}(S)$ .

► **Lemma 8.** *Let  $S$  be a stream of edges of an underlying simple weighted graph  $G(V, E)$ . Suppose that we have two disjoint substreams  $B$  and  $C$  of the stream  $S$ . Then,*

- **Monotonicity Property 1:**  $w(M_{\text{alg}}(B)) \leq w(M_{\text{mon}}(B)) \leq w(M_{\text{opt}}(B))$ .
- **Monotonicity Property 2:**  $w(M_{\text{mon}}(B)) \leq w(M_{\text{mon}}(BC))$ .

### 3 Maximum-weight matching in the sliding-window model

In this section, we develop our 3.5-approximation algorithm for the maximum-weight matching problem in the sliding-window model. As the first step, we next provide a generic framework for computing monotone bounded functions in the sliding-window model. Our framework is based on the *smooth-histogram techniques* developed by Braverman and Ostrovsky [2] for the sliding-window model. The smooth histogram technique was later used by Crouch, McGregor, and Stubbs [4] to develop algorithms for graph problems in the sliding-window model.

The idea behind this technique is that at any time  $t$ , we have a logarithmic number of insertion-only algorithms running in parallel, that each started at different times. As our answer at time  $t$  we then report, roughly speaking, the answer given by the “oldest”



still running algorithm; the technique ensures that this algorithm started shortly after time  $t - L$ , so that the answer is a good approximation of the actual answer. The next definition formalizes the properties we need to apply the technique. Suppose we want to approximate a function  $f$  in a sliding-window setting, and we have an algorithm  $\mathcal{ALG}$  for the insertion-only setting. The idea is that if  $A, B$  are two substreams such that  $f(ABC) \leq \alpha \cdot \mathcal{ALG}(BC)$  for any substream  $C$ , then we can  $\alpha$ -approximate  $f(ABC)$  by  $\mathcal{ALG}(BC)$ .

► **Definition 9** ( $(f, \alpha, \beta)$ -lookahead algorithm). *Let  $\beta \in (0, 1)$  and  $\alpha \in \mathbb{R}^+$  be two parameters. Let  $f$  be a real-valued monotone function defined on subsets of a ground set  $\mathcal{X}$ . Let  $S$  be a stream of items of the set  $\mathcal{X}$ . Let  $\mathcal{ALG}$  be a streaming algorithm. We say the streaming algorithm  $\mathcal{ALG}$  is an  $(f, \alpha, \beta)$ -lookahead algorithm if for any partitioning of  $S$  into three disjoint sub-streams  $A, B$ , and  $C$  with  $\mathcal{ALG}(B) \geq (1 - \beta) \cdot \mathcal{ALG}(AB)$ , we have  $f(ABC) \leq \alpha \cdot \mathcal{ALG}(BC)$ .*

The following theorem shows that if we have such a lookahead algorithm, we can develop a sliding-window algorithm for the function  $f$ . The proof of this theorem is based on the techniques developed in [4] and we give the full proof for the sake of completeness.

► **Theorem 10.** *Let  $0 < \beta < 1$  and  $\alpha \geq 1$  be two parameters. Let  $S$  be a stream of items from an underlying ground set  $\mathcal{X}$ . Let  $f$  be a monotone function defined on subsets of a ground set  $\mathcal{X}$ . Let  $\sigma := f_{\max}/f_{\min^+}$ , where  $f_{\min^+} := \min\{f(X) : X \text{ is a substream of } S \text{ and } f(X) > 0\}$  and  $f_{\max} := \max\{f(X) : X \text{ is a substream of the input}\}$ . Suppose there exists an  $(f, \alpha, \beta)$ -lookahead algorithm that uses space  $s$ . Then there exists a sliding-window algorithm that maintains an  $\alpha$ -approximation of  $f$  using  $O(\beta^{-1} \cdot s \log \sigma)$  space.*

**Proof.** We first present our algorithm SLIDINGLOOKAHEAD, which is based on the smooth histograms developed by Braverman and Ostrovsky [2] for the sliding-window model.

■ **Algorithm 2** SLIDINGLOOKAHEAD (Based on Smooth Histogram [2, 14]).

---

**Initialization:**

- 1: Let  $k \leftarrow 0$  be the number of buckets.
- 

**Streaming:**

- 1: **while** a new item  $e$  of the stream  $S$  is revealed at time  $t$  **do**
  - 2:     Create an empty bucket  $B_{k+1}$ .
  - 3:     Let  $\mathcal{ALG}_{k+1}$  be an instance of  $\mathcal{ALG}$  for the bucket  $B_{k+1}$ .
  - 4:     **for**  $i = 1, \dots, k + 1$  **do**
  - 5:         Feed  $e$  to the bucket  $B_i$ , and update its associated instance  $\mathcal{ALG}_i$ .
  - 6:     **for**  $i = 1, \dots, k - 1$  **do**
  - 7:         Let  $j > i$  be the largest index for which  $\mathcal{ALG}(B_j) \geq (1 - \beta) \cdot \mathcal{ALG}(B_i)$ .
  - 8:         Delete buckets  $B_r$  for  $i < r < j$  and their associated instances  $\mathcal{ALG}_r$ .
  - 9:     **if**  $W \subseteq B_2$  **then** Delete bucket  $B_1$  and its associated instance  $\mathcal{ALG}_1$ .
  - 10:     Let  $k$  be number of remaining buckets.
  - 11:     Renumber buckets and their associated instances.
- 

**Output:**

- 1: **if**  $W = B_1$  **then return** the solution of the  $\mathcal{ALG}_1$ , which equals  $\mathcal{ALG}(B_1)$ .
  - 2: **else return** the solution of the  $\mathcal{ALG}_2$ , which equals  $\mathcal{ALG}(B_2)$ .
-

Let  $W = [\max(t - L, 1), t]$  be the active window. With a slight abuse of notation, we will use  $W$  both for the time interval of the active window as well as for the set of items that arrive during this time interval. We consider  $k$  buckets  $B_1, \dots, B_k$  such that  $B_1 \supseteq W \supseteq B_2 \supseteq \dots \supseteq B_k$ . We later show that  $k = O(\varepsilon^{-1} \log \sigma)$ . For each bucket, we instantiate an instance of  $(f, \alpha, \beta)$ -lookahead algorithm  $\mathcal{ALG}$ . Upon arrival of a new item  $e$  at a time  $t$ , we create a new bucket  $B_{k+1}$  and add  $e$  to all buckets  $B_1, \dots, B_{k+1}$ . Next, if there exists a sequence of buckets whose values of  $\mathcal{ALG}$  are too close, we keep the bucket in this sequence that has the lowest index and delete the rest of the buckets in this sequence.

Observe that the active window  $W$  is always sandwiched between the buckets  $B_1$  and  $B_2$ . If at any time  $t$ , the active window  $W$  is covered by the bucket  $B_2$  (i.e.,  $W \subseteq B_2$ ), we then delete  $B_1$  and renumber the buckets accordingly. The output of smooth histogram is reported by the instance  $\mathcal{ALG}$  of the bucket  $B_2$ .

Next, we prove the theorem which is based on the combination of the proofs of Lemma 3 and Theorem 4 in [4]. (We should mention that the proof in [4] is for  $\alpha = 3 + \varepsilon$ , but we prove it for any  $\alpha$ .)

Let  $\mathcal{ALG}$  be a  $(f, \alpha, \beta)$ -lookahead algorithm using space  $s$ . We show that SLIDINGLOOKAHEAD maintains an  $\alpha$ -approximation of  $f$ , using  $O(\beta^{-1} \cdot s \log \sigma)$  space. First, we prove the approximation ratio. Later, we show that  $k = O(\beta^{-1} \log \sigma)$  at any time  $t$ .

Let  $W = [\max(t - L, 1), t]$  be the active window and let  $0 < \beta < 1$ . Recall that the buckets  $B_1, \dots, B_k$  satisfy  $B_1 \supseteq W \supseteq B_2 \supseteq \dots \supseteq B_k$ . If  $W = B_1$ , we return  $\mathcal{ALG}(B_1) = \mathcal{ALG}(W)$ , which satisfies  $f(W) \leq \alpha \cdot \mathcal{ALG}(B_1)$ . Now, suppose that  $W \neq B_1$ . Since we report  $\mathcal{ALG}(B_2)$ , we must show that  $f(W) \leq \alpha \cdot \mathcal{ALG}(B_2)$ . To this end, let  $t^*$  be the first time that buckets  $B_1$  and  $B_2$  became adjacent; it is the time when buckets in between  $B_1$  and  $B_2$  were removed in step 8 of the streaming phase. We denote these buckets at time  $t^*$  by  $B_1^*$  and  $B_2^*$ . So,  $\mathcal{ALG}(B_2^*) > (1 - \beta)\mathcal{ALG}(B_1^*)$ . Moreover,  $B_1 = B_1^*C$ ,  $B_2 = B_2^*C$  for the stream  $C$  consisting of the items that arrived since time  $t^*$ . Now, according to the  $(f, \alpha, \beta)$ -lookahead property of  $\mathcal{ALG}$ , we can conclude  $f(B_1) \leq \alpha \cdot \mathcal{ALG}(B_2)$ . As  $f$  is monotone, we have:

$$f(W) \leq f(B_1) \leq \alpha \cdot \mathcal{ALG}(B_2) .$$

Recall that  $k$  is the number of buckets. Next, we prove that at any time  $t$ , we have  $k = O(\beta^{-1} \log \sigma)$ . Recall that for each index  $i \in [1..k - 2]$ , we have

$$\mathcal{ALG}(B_{i+2}) < (1 - \beta) \cdot \mathcal{ALG}(B_i).$$

Since  $1/(1 - \beta) > 1 + \beta$ , we have

$$\mathcal{ALG}(B_i) > \frac{1}{1 - \beta} \cdot \mathcal{ALG}(B_{i+2}) > (1 + \beta) \cdot \mathcal{ALG}(B_{i+2}).$$

Thus,

$$\mathcal{ALG}(B_1) \geq (1 + \beta)^{\lfloor k/2 \rfloor} \cdot \mathcal{ALG}(B_k) .$$

Recall that  $\frac{f(B_1)}{f(B_k)} \leq \sigma$ . This essentially means that  $\mathcal{ALG}(B_1) \leq \sigma \alpha \cdot \mathcal{ALG}(B_k)$ . Therefore,  $k \leq O(\log_{1+\beta}(\sigma \alpha)) = O(\beta^{-1} \cdot \log \sigma)$ .  $\blacktriangleleft$

Our main result in this paper is that there exists a lookahead algorithm for the maximum weight matching problem. We state this result formally next.

► **Lemma 11.** *Let  $G(V, E)$  be a graph with  $n = |V|$  vertices and let  $w : E \rightarrow \mathbb{R}^+$  be a function that assigns a non-negative weight  $w(e)$  to each edge  $e \in E$ . Let  $0 < \varepsilon \leq \frac{1}{10}$  and  $0 < \beta \leq \frac{\varepsilon}{9}$  be two parameters. Let  $f$  be defined as the weight of a maximum weight matching of  $G$ . Then, Algorithm MWM-MONOTONE is a  $(f, (3.5 + \varepsilon), \beta)$ -lookahead algorithm. That is, for any partitioning of a stream  $S$  of edges of  $G$  into three disjoint sub-streams  $A$ ,  $B$ , and  $C$  with  $w(M_{\text{mon}}(B)) \geq (1 - \beta) \cdot w(M_{\text{mon}}(AB))$ , we have  $w(M_{\text{opt}}(ABC)) \leq (3.5 + \varepsilon) \cdot w(M_{\text{mon}}(BC))$ .*

Thus, we can use the machinery developed in Theorem 10 to obtain a  $(3.5 + \varepsilon)$ -approximate sliding-window algorithm for the maximum-weight matching.

We first define some notation and prove auxiliary tools to later prove Lemma 11.

**Notation.** Suppose we have a simple weighted graph  $G(V, E)$ , whose edges are revealed in a streaming fashion and let that stream be  $S$ . Suppose we partition the stream  $S$  into three disjoint consecutive substreams  $A$ ,  $B$ , and  $C$ . Let  $M_{\text{opt}}(ABC)$  be a fixed maximum-weight matching of  $G$ . Let  $X \in \{A, B, C, AB, BC\}$ .

- We denote by  $M_{\text{opt}}(ABC) \cap X$  those edges of the maximum-weight matching  $M_{\text{opt}}(ABC)$  that belong to the substream  $X$ .
- We denote by  $M_{\text{alg}}(X)$  the reported matching of Algorithm MWM-STREAMING if we invoke it on the input substream  $X$ .
- We let  $\phi_X(v)$  be the potential that is assigned to a vertex  $v \in V$  if we execute Algorithm MWM-STREAMING on the input substream  $X$ .
- We let  $w'_X(e)$  be the reduced weight that we assign to an edge  $e \in E$  if we execute Algorithm MWM-STREAMING on the input substream  $X$ .

A key concept in our analysis is the so-called *critical subgraph* of a graph  $G$ . We take advantage of this concept to show upper bounds for  $w(M_{\text{opt}}(ABC) \cap AB)$  and  $w(M_{\text{opt}}(ABC) \cap C)$  in Lemmas 13–16.

► **Definition 12 (Critical Subgraph).** *Consider a graph  $G$  specified by a stream  $S$  of edges. Let  $A, B, C$  be disjoint substreams of  $S$  such that  $S = ABC$ . Then, the critical subgraph of  $G$  with respect to the matching  $M_{\text{opt}}(ABC)$  and the substreams  $A, B, C$  is the subgraph  $H = (V_H, E_H)$  such that*

- $E_H := \{e \in B \mid e \text{ has two neighbors in } M_{\text{opt}}(ABC) \cap C\}$ .
- $V_H := \{v \in V \mid \exists u \in V : (u, v) \in E_H\}$ , i.e.,  $V_H$  is the set of endpoints of the edges in  $E_H$ .

**Outline of the proof of Lemma 11.** We show an upper-bound for  $w(M_{\text{opt}}(ABC))$  based on  $w(M_{\text{alg}}(AB))$  and  $w(M_{\text{alg}}(BC))$ . In particular, we show that

$$w(M_{\text{opt}}(ABC)) \leq 2(1 + \varepsilon)w(M_{\text{alg}}(AB)) + 2(1 + \varepsilon)w(M_{\text{alg}}(BC)) - \mathcal{Z} . \quad (1)$$

The slack term  $\mathcal{Z}$  shows that there might be a double counting in the above inequality. Intuitively, this makes sense as the substream  $B$  is repeated in the first and the second terms.

However, for the moment, suppose this is not the case and  $\mathcal{Z} = 0$ . Assume that  $w(M_{\text{alg}}(B)) \geq (1 - \beta) \cdot w(M_{\text{alg}}(AB))$ , that is, the sets  $A$  and  $B$  (and our algorithm) are such that  $M_{\text{alg}}(B)$  gives a good approximation of  $M_{\text{alg}}(AB)$ . Then, we have

$$\begin{aligned} w(M_{\text{opt}}(ABC)) &\leq 2 \cdot \frac{(1 + \varepsilon)}{(1 - \beta)} \cdot w(M_{\text{alg}}(B)) + 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(BC)) \\ &\leq 2 \cdot \frac{(1 + \varepsilon)}{(1 - \beta)} \cdot w(M_{\text{mon}}(B)) + 2(1 + \varepsilon) \cdot w(M_{\text{mon}}(BC)) \\ &\leq 4 \cdot (1 + O(\varepsilon)) \cdot w(M_{\text{mon}}(BC)) , \end{aligned}$$

## 73:10 Maximum-Weight Matching in Sliding Windows and Beyond

where for the second inequality we switch from Algorithm MWM-STREAMING to its monotone version which is Algorithm MWM-STREAMING-MONOTONE and the third inequality uses Monotonicity Property 2 of Lemma 8.

Observe that this result already improves upon the  $(6 + \varepsilon)$ -approximation algorithm in the sliding-window model due to Crouch and Stubbs [5]. To obtain an even better bound, we show that  $\mathcal{Z}$  is lower-bounded by a constant factor of the optimal matching of the substream  $B$ . In particular, we show that Inequality (1) holds for  $\mathcal{Z} = \frac{1}{2(1+\varepsilon)} \cdot w(M_{\text{opt}}(B))$ . Since  $w(M_{\text{opt}}(B)) \geq w(M_{\text{mon}}(B))$ , we then have

$$\begin{aligned} w(M_{\text{opt}}(ABC)) &\leq 2 \frac{(1+\varepsilon)}{(1-\beta)} w(M_{\text{mon}}(B)) + 2(1+\varepsilon)w(M_{\text{mon}}(BC)) - \frac{1}{2(1+\varepsilon)} w(M_{\text{mon}}(B)) \\ &\leq (3.5 + O(\varepsilon))w(M_{\text{mon}}(BC)) . \end{aligned}$$

**The complete proof of Lemma 11.** We find the bound on  $w(M_{\text{opt}}(ABC))$  that we claimed in Equation 1 in three steps. First in Lemma 13, we find an upper-bound on the contribution of the substream  $AB$  towards  $M_{\text{opt}}(ABC)$ . Next, we upper-bound the weight of edges of the optimal matching  $M_{\text{opt}}(ABC)$  that are in the substream  $C$ . This is done in Lemma 14. Finally, in Lemma 15 we obtain a lower-bound for the slack term  $\mathcal{Z}$  of Equation 1.

► **Lemma 13.**  $w(M_{\text{opt}}(ABC) \cap AB) \leq 2(1+\varepsilon)w(M_{\text{alg}}(AB)) - (1+\varepsilon) \cdot \sum_{v \in V_H} \phi_{AB}(v)$  .

**Proof.** By definition,  $w(M_{\text{opt}}(ABC) \cap AB) = \sum_{e^* \in M_{\text{opt}}(ABC) \cap AB} w(e^*)$ . Let us consider an arbitrary edge  $e^* = (u, v) \in M_{\text{opt}}(ABC) \cap AB$ . Using Lemma 3, we have  $w(e^*) \leq (1+\varepsilon) \cdot (\phi_{AB}(u) + \phi_{AB}(v))$ . Observe that the vertices  $u$  and  $v$  cannot be in  $V_H$ . Thus, we obtain

$$w(M_{\text{opt}}(ABC) \cap AB) \leq (1+\varepsilon) \cdot \sum_{v \in V \setminus V_H} \phi_{AB}(v) = (1+\varepsilon) \cdot \sum_{v \in V} \phi_{AB}(v) - (1+\varepsilon) \cdot \sum_{v \in V_H} \phi_{AB}(v) .$$

Now we use Lemma 6 that shows  $\sum_{v \in V} \phi_{AB}(v) \leq 2w(M_{\text{alg}}(AB))$ . Hence,

$$w(M_{\text{opt}}(ABC) \cap AB) \leq 2(1+\varepsilon) \cdot w(M_{\text{alg}}(AB)) - (1+\varepsilon) \cdot \sum_{v \in V_H} \phi_{AB}(v) . \quad \blacktriangleleft$$

The next lemma shows that the contribution of optimal edges that are in the substream  $C$  is upper-bounded by twice the weight of the reported matching of the substream  $BC$  minus the reduced weight of edges of  $B$  that are not in the critical subgraph  $H$ .

► **Lemma 14.**  $w(M_{\text{opt}}(ABC) \cap C) \leq 2(1+\varepsilon) \cdot w(M_{\text{alg}}(BC)) - (1+\varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_B(e)$  .

**Proof.** Recall that for every edge  $e \in E$ ,  $w'_{BC}(e)$  is the reduced weight of  $e$  if we run Algorithm MWM-STREAMING on the input stream  $BC$ , where we define  $w'_{BC}(e) = 0$  if  $e \notin BC$ . To prove the lemma, we will show that

$$w(M_{\text{opt}}(ABC) \cap C) \leq 2(1+\varepsilon) \cdot \sum_{e \in BC} w'_{BC}(e) - (1+\varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_{BC}(e) . \quad (2)$$

Suppose for now that Inequality (2) is correct. According to Lemma 7, for every  $e \in B$ , we have  $w'_{BC}(e) = w'_B(e)$ . Thus,  $\sum_{e \in B \setminus E_H} w'_{BC}(e) = \sum_{e \in B \setminus E_H} w'_B(e)$ .

Next, we use Lemma 6, where we replace the graph  $G$  in that lemma with the subgraph  $G'(V, BC)$  to show that  $\sum_{e \in BC} w'_{BC}(e) \leq w(M_{\text{alg}}(BC))$ .

Putting everything together, we prove the statement of this lemma as follows:

$$\begin{aligned}
w(M_{\text{opt}}(ABC) \cap C) &\leq 2(1 + \varepsilon) \cdot \sum_{e \in BC} w'_{BC}(e) - (1 + \varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_{BC}(e) \\
&\leq 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(BC)) - (1 + \varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_{BC}(e) \\
&= 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(BC)) - (1 + \varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_B(e) .
\end{aligned}$$

It remains to prove Inequality (2). Let  $e^* \in M_{\text{opt}}(ABC) \cap C$ . We use Lemma 5, where we replace the graph  $G$  in that lemma with the subgraph  $G'(V, BC)$ . Then, Lemma 5 shows that  $w(e^*) \leq (1 + \varepsilon) \cdot \sum_{e' \in P_{BC}(e^*)} w'_{BC}(e')$ , where  $P_{BC}(e) := \{e' \in BC : e' \in N_{G'}(e) \cup \{e\} \text{ and } t_{e'} \leq t_e\}$ . (Here  $t_{e'}$  and  $t_e$  are the arrival time of edges  $e$  and  $e'$  in the substream  $BC$ .)

Hence,

$$w(M_{\text{opt}}(ABC) \cap C) = \sum_{e^* \in M_{\text{opt}}(ABC) \cap C} w(e^*) \leq (1 + \varepsilon) \sum_{e^* \in M_{\text{opt}}(ABC) \cap C} \sum_{e' \in P_{BC}(e^*)} w'_{BC}(e')$$

Consider the double summation  $\sum_{e^* \in M_{\text{opt}}(ABC) \cap C} \sum_{e' \in P_{BC}(e^*)} w'_{BC}(e')$ . Every edge  $e' \in BC$  appears in  $P_{BC}(e^*)$  for at most two edges  $e^*$ , because each endpoint of  $e'$  is incident to at most one edge from  $M_{\text{opt}}(ABC)$ . Hence, the double summation can be bounded by  $2 \cdot \sum_{e \in BC} w'_{BC}(e)$ . If, however,  $e'$  appears twice then  $e' \in E_H$ , that is,  $e'$  is part of the critical subgraph  $H$ . This means that the edges in  $B \setminus E_H$  appear at most once in the sum. Therefore,

$$w(M_{\text{opt}}(ABC) \cap C) \leq 2(1 + \varepsilon) \cdot \sum_{e \in BC} w'_{BC}(e) - (1 + \varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_{BC}(e),$$

which proves Inequality (2) and finishes the proof of the lemma.  $\blacktriangleleft$

**► Lemma 15.** *Suppose we execute Algorithm MWM-STREAMING on the substream  $AB$ . Let  $\phi_{AB}(v)$  be the potential assigned to a vertex  $v \in V_H$  at the end of this algorithm. Then,  $(1 + \varepsilon) \cdot \sum_{v \in V_H} \phi_{AB}(v) \geq \sum_{e \in E_H} w'_B(e)$ .*

**Proof.** Observe that  $E_H \subseteq B \subseteq AB$ . Let us consider an arbitrary edge  $e = (u, v)$  in  $E_H$ . By applying Lemma 3 to the graph  $G'(V, AB)$  we obtain  $w(e) \leq (1 + \varepsilon)(\phi_{AB}(u) + \phi_{AB}(v))$ . Fix a maximum-weight matching  $M_{\text{opt}}(H)$  of the critical subgraph  $H(V_H, E_H)$ . Then,

$$\begin{aligned}
w(M_{\text{opt}}(H)) &= \sum_{e=(u,v) \in M_{\text{opt}}(H)} w(e) \leq \sum_{(u,v) \in M_{\text{opt}}(H)} (1 + \varepsilon)(\phi_{AB}(u) + \phi_{AB}(v)) \\
&\leq (1 + \varepsilon) \sum_{u \in V_H} \phi_{AB}(u) ,
\end{aligned}$$

where the last inequality is due to the fact that the degree of any vertex  $u \in V_H$  in the matching  $M_{\text{opt}}(H)$  is at most one. We next find a lower bound for  $w(M_{\text{opt}}(H))$ , which in turns yields a lower bound for  $(1 + \varepsilon) \sum_{u \in V_H} \phi_{AB}(v)$ .

Now, suppose we execute Algorithm MWM-STREAMING on the stream  $B$ . However, during the postprocessing phase, once we pop an edge  $e$  from the stack, we add it to the matching  $M_{\text{alg}}$  if both end-points of  $e$  are free and  $e \in E_H$ . Thus, the reported matching  $M_{\text{alg}}$  of the substream  $B$  is influenced by the edge set  $E_H$ . We denote by  $M_{\text{alg}}(B|E_H)$  the

## 73:12 Maximum-Weight Matching in Sliding Windows and Beyond

reported matching of  $B$  that is influenced by  $E_H$ . Observe that  $M_{\text{alg}}(B|E_H)$  is a matching of the critical subgraph  $H(V_H, E_H)$ . Therefore,  $w(M_{\text{alg}}(B|E_H)) \leq w(M_{\text{opt}}(H))$ . We next find a lower bound for  $w(M_{\text{alg}}(B|E_H))$ .

Consider the graph  $G'(V, B)$ . Let us fix an arbitrary edge  $e \in B$ . Recall that  $P_B(e) = \{e' \in B : e' \in N_{G'}(e) \cup \{e\} \text{ and } t_{e'} \leq t_e\}$ . (Here  $t_{e'}$  and  $t_e$  are the arrival time of edges  $e$  and  $e'$  in the substream  $B$ .) Observe that if  $e \in M_{\text{alg}}(B|E_H)$ , then  $e$  was added to the stack at some point. Using Lemma 4, we then have  $w(e) = \sum_{e' \in P_B(e)} w'_B(e')$ . Therefore,

$$\sum_{e \in M_{\text{alg}}(B|E_H)} w(e) = \sum_{e \in M_{\text{alg}}(B|E_H)} \sum_{e' \in P_B(e)} w'_B(e').$$

Now, let us consider an arbitrary edge  $e' \in E_H$ . For  $e'$  we have three cases. The first case is that  $e'$  is not in the stack, which implies  $w'_B(e') = 0$ . The second case is that  $e' \in M_{\text{alg}}(B|E_H)$ , so  $e'$  contributes to the sum  $\sum_{e \in M_{\text{alg}}(B|E_H)} w(e)$ . The third case is that  $e'$  has a neighbor  $e \in M_{\text{alg}}(B|E_H)$ , so that  $e' \in P_B(e)$ . Considering all the three cases we conclude

$$\sum_{e \in M_{\text{alg}}(B|E_H)} w(e) = \sum_{e \in M_{\text{alg}}(B|E_H)} \sum_{e' \in P_B(e)} w'_B(e') \geq \sum_{e \in E_H} w'_B(e).$$

Putting everything together, we obtain the statement of this lemma as follows:

$$(1 + \varepsilon) \cdot \sum_{v \in V_H} \phi_{AB}(v) \geq w(M_{\text{opt}}(H)) \geq w(M_{\text{alg}}(B|E_H)) \geq \sum_{e \in E_H} w'_B(e) . \quad \blacktriangleleft$$

We are now finally ready to prove Inequality (1) on page 9, with  $\mathcal{Z} = \frac{1}{2(1+\varepsilon)}w(M_{\text{opt}}(B))$ .

► **Lemma 16.** *Let  $S = ABC$  be a stream of edges of an underlying weighted graph  $G(V, E)$ . Then, there exists the following upper bound for the weight of  $M_{\text{opt}}(ABC)$ :*

$$w(M_{\text{opt}}(ABC)) \leq 2(1 + \varepsilon)w(M_{\text{alg}}(AB)) + 2(1 + \varepsilon)w(M_{\text{alg}}(BC)) - \frac{1}{2(1 + \varepsilon)}w(M_{\text{opt}}(B)) .$$

**Proof.** First of all, observe that we can decompose the edges of the optimal matching  $M_{\text{opt}}(ABC)$  into the subset of edges that are in the substreams  $AB$  and  $C$ . Thus, we have

$$w(M_{\text{opt}}(ABC)) = w(M_{\text{opt}}(ABC) \cap AB) + w(M_{\text{opt}}(ABC) \cap C) .$$

Using Lemma 14 and Lemma 13, we then obtain

$$\begin{aligned} w(M_{\text{opt}}(ABC)) &\leq 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(AB)) - (1 + \varepsilon) \cdot \sum_{v \in V_H} \phi_{AB}(v) \\ &\quad + 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(BC)) - (1 + \varepsilon) \cdot \sum_{e \in B \setminus E_H} w'_B(e) . \end{aligned}$$

Next, we replace the negative term on the sum of the potentials for vertices in  $V_H$  with its lower-bound as in Lemma 15 .

$$\begin{aligned} &w(M_{\text{opt}}(ABC)) \\ &\leq 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(AB)) - \sum_{e \in E_H} w'_B(e) + 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(BC)) - (1 + \varepsilon) \sum_{e \in B \setminus E_H} w'_B(e) \\ &= 2(1 + \varepsilon) \cdot w(M_{\text{alg}}(AB)) + 2(1 + \varepsilon)w(M_{\text{alg}}(BC)) - \sum_{e \in B} w'_B(e) . \end{aligned}$$

Applying Lemma 6 to the subgraph  $G'(V, B)$  gives  $\sum_{e \in B} w'_B(e) \geq \frac{1}{2(1+\varepsilon)} \cdot w(M_{\text{opt}}(G'(V, B)))$ . Hence,

$$w(M_{\text{opt}}(ABC)) \leq 2(1+\varepsilon) \cdot w(M_{\text{alg}}(AB)) + 2(1+\varepsilon) \cdot w(M_{\text{alg}}(BC)) - \frac{1}{2(1+\varepsilon)} \cdot w(M_{\text{opt}}(B)) \quad \blacktriangleleft$$

With the help of Lemma 16 we can now prove Lemma 11.

**Proof of Lemma 11.** First of all using Lemma 16, we have the following upper-bound for the weight of  $M_{\text{opt}}(ABC)$ :

$$w(M_{\text{opt}}(ABC)) \leq 2(1+\varepsilon)w(M_{\text{alg}}(AB)) + 2(1+\varepsilon)w(M_{\text{alg}}(BC)) - \frac{1}{2(1+\varepsilon)}w(M_{\text{opt}}(B)) \quad .$$

Now, we switch from Algorithm MWM-STREAMING to its monotone version, which is Algorithm MWM-MONOTONE. Monotonicity Property 1 of Lemma 8 states that for the substream  $B$  we have  $w(M_{\text{alg}}(B)) \leq w(M_{\text{mon}}(B)) \leq w(M_{\text{opt}}(B))$ . The same bounds also hold for the substream  $AB$  and  $BC$ . Therefore,

$$\begin{aligned} w(M_{\text{opt}}(ABC)) &\leq 2(1+\varepsilon)w(M_{\text{mon}}(AB)) + 2(1+\varepsilon)w(M_{\text{mon}}(BC)) - \frac{1}{2(1+\varepsilon)}w(M_{\text{mon}}(B)) \\ &\leq \frac{2(1+\varepsilon)}{1-\beta} \cdot w(M_{\text{mon}}(B)) + 2(1+\varepsilon)w(M_{\text{mon}}(BC)) - \frac{1}{2(1+\varepsilon)}w(M_{\text{mon}}(B)) \\ &\leq 1.5(1+3\varepsilon) \cdot w(M_{\text{mon}}(B)) + 2(1+\varepsilon)w(M_{\text{mon}}(BC)) \quad , \end{aligned}$$

where for the second inequality we use the assumption of this lemma, which is

$$w(M_{\text{mon}}(B)) \geq (1-\beta) \cdot w(M_{\text{mon}}(AB)) \quad .$$

From Monotonicity Property 2 of Lemma 8, which is  $w(M_{\text{mon}}(B)) \leq w(M_{\text{mon}}(BC))$ , we now for  $\varepsilon \leq \frac{1}{10}$  conclude

$$\begin{aligned} w(M_{\text{opt}}(ABC)) &\leq 1.5(1+3\varepsilon)w(M_{\text{mon}}(B)) + 2(1+\varepsilon)w(M_{\text{mon}}(BC)) \\ &\leq (3.5+\varepsilon)w(M_{\text{mon}}(BC)) \quad . \end{aligned} \quad \blacktriangleleft$$

#### 4 Subadditive functions in the sliding-window model

In this section, we prove Theorem 2 and then we show that using this theorem, we can improve the approximation factor of quite a few submodular matching problems in the sliding-window model. We first explain the notations that we use in this section.

Let  $f$  be a function defined on streams, which is subadditive, non-negative, and monotone. Let  $\sigma := f_{\text{max}}/f_{\text{min}+}$ , where

$$f_{\text{min}+} := \min\{f(X) : X \text{ is a substream of the input and } f(X) > 0\} \quad ,$$

and

$$f_{\text{max}} := \max\{f(X) : X \text{ is a substream of the input}\}$$

Suppose we are given a streaming algorithm  $\mathcal{ALG}$  that  $\alpha$ -approximates  $f$  using space  $s$ . Very recently Krauthgamer and Reitblat [14] showed that we can use the streaming algorithm  $\mathcal{ALG}$  to develop a sliding-window algorithm that  $(2\alpha^2 + \varepsilon)$ -approximates  $f$  using space  $O(\varepsilon^{-1}s \cdot \log \sigma)$ . They also showed that if the streaming algorithm  $\mathcal{ALG}$  is monotone and subadditive, we can reduce the approximation factor of the sliding-window algorithm down to  $(2\alpha + \varepsilon)$ -factor. Unfortunately, in some cases, although  $f$  is subadditive,  $\mathcal{ALG}$  is not



subadditive, or it is not easy to show that  $\mathcal{ALG}$  is also subadditive. So, the  $(2\alpha + \varepsilon)$ -factor cannot always be a result of [14]. Nevertheless, Theorem 2 shows that we can obtain a  $(2\alpha + \varepsilon)$ -approximation algorithm in the sliding-window model, independent of the monotonicity or subadditivity of the streaming algorithm  $\mathcal{ALG}$ . Our main result in this paper is that there exists a lookahead algorithm for the maximum weight matching problem. We state this result formally next.

Next, we prove Theorem 2. To this end, in Lemma 17, we show how to transform a non-monotone algorithm  $\mathcal{ALG}$  into a monotone algorithm  $\mathcal{ALG}_{\text{mon}}$ . Later, in Lemma 18 we show that a monotone algorithm  $\mathcal{ALG}_{\text{mon}}$  that  $\alpha$ -approximates a subadditive monotone function  $f$  is in fact, a  $(f, 2\alpha + \varepsilon(\beta), \beta)$ -lookahead algorithm. Thus, we can use the machinery of Theorem 10 to develop  $(2\alpha + \varepsilon)$ -approximation sliding-window algorithms for  $f$  using the streaming algorithm  $\mathcal{ALG}_{\text{mon}}$ .

► **Lemma 17.** *Let  $f$  be a non-negative and monotone function defined on streams. Let  $\mathcal{ALG}$  be a streaming algorithm that  $\alpha$ -approximates  $f$  using space  $s$ , where  $\alpha \geq 1$ . Then, there is a monotone algorithm  $\mathcal{ALG}_{\text{mon}}$  that  $\alpha$ -approximates  $f$  using  $O(s)$  space.*

**Proof.** Let  $S$  be a stream of items of the domain  $\mathcal{X}$  of the subadditive  $f$ . In order to make the algorithm  $\mathcal{ALG}$  monotone, we store the maximum  $f$  of all prefixes of the stream  $S$ . Formally, we define  $\mathcal{ALG}_{\text{mon}}$  as follow:  $\mathcal{ALG}_{\text{mon}}(S) := \max\{\mathcal{ALG}(T) : T \text{ is a prefix of } S\}$ .

To prove the approximation factor, we show that  $\frac{1}{\alpha} \cdot f(S) \leq \mathcal{ALG}_{\text{mon}}(S) \leq f(S)$ . Observe that  $\mathcal{ALG}$  returns a  $\alpha$ -approximation of  $f(S)$ . Thus,  $\frac{1}{\alpha} \cdot f(S) \leq \mathcal{ALG}(S) \leq \mathcal{ALG}_{\text{mon}}(S)$ .

Now, it is enough to show that  $\mathcal{ALG}_{\text{mon}}(S) \leq f(S)$ . Let  $T^*$  be the prefix of  $S$  for which  $\mathcal{ALG}$  returns the maximum value among all prefixes of  $S$ . That is,  $\mathcal{ALG}_{\text{mon}}(S) = \mathcal{ALG}(T^*)$ . Since  $f$  is monotone  $f(T^*) \leq f(S)$ , we then have

$$\mathcal{ALG}_{\text{mon}}(S) = \mathcal{ALG}(T^*) \leq f(T^*) \leq f(S) . \quad \blacktriangleleft$$

► **Lemma 18.** *Let  $0 < \varepsilon \leq 1/2$ ,  $\alpha \geq 1$  and  $0 < \beta \leq \varepsilon/2\alpha$  be three parameters. Let  $f$  be a subadditive, non-negative, and monotone function defined on streams. Suppose we have a monotone streaming algorithm  $\mathcal{ALG}_{\text{mon}}$  that  $\alpha$ -approximates  $f$ . Then, Algorithm  $\mathcal{ALG}_{\text{mon}}$  is a  $(f, (2\alpha + \varepsilon), \beta)$ -lookahead algorithm. That is, for any partitioning of a stream  $S$  of items of the domain  $\mathcal{X}$  of  $f$  into three disjoint sub-streams  $A$ ,  $B$ , and  $C$  with  $\mathcal{ALG}_{\text{mon}}(B) \geq (1 - \beta) \cdot \mathcal{ALG}_{\text{mon}}(AB)$ , we have*

$$f(ABC) \leq (2\alpha + \varepsilon) \cdot \mathcal{ALG}_{\text{mon}}(BC) .$$

**Proof.** First of all since  $f$  is subadditive, we have  $f(ABC) \leq f(A) + f(BC)$ . Also, the function  $f$  is monotone, which means that  $f(A) \leq f(AB)$ . Therefore,  $f(ABC) \leq f(AB) + f(BC)$ .

Since  $\mathcal{ALG}_{\text{mon}}$  is an  $\alpha$ -approximate algorithm for  $f$ , we then have

$$f(ABC) \leq \alpha \cdot \mathcal{ALG}_{\text{mon}}(AB) + \alpha \cdot \mathcal{ALG}_{\text{mon}}(BC) .$$

We assume that  $\mathcal{ALG}_{\text{mon}}(B) \geq (1 - \beta) \cdot \mathcal{ALG}_{\text{mon}}(AB)$ . Using which, we have

$$f(ABC) \leq \frac{\alpha}{(1 - \beta)} \cdot \mathcal{ALG}_{\text{mon}}(B) + \alpha \cdot \mathcal{ALG}_{\text{mon}}(BC) .$$

Since  $\mathcal{ALG}_{\text{mon}}$  is a monotone algorithm, we have  $\mathcal{ALG}_{\text{mon}}(B) \leq \mathcal{ALG}_{\text{mon}}(BC)$ . Thus,

$$\begin{aligned} f(ABC) &\leq \frac{\alpha}{(1 - \beta)} \cdot \mathcal{ALG}_{\text{mon}}(BC) + \alpha \cdot \mathcal{ALG}_{\text{mon}}(BC) \\ &\leq \alpha(1 + 2\beta) \cdot \mathcal{ALG}_{\text{mon}}(BC) + \alpha \cdot \mathcal{ALG}_{\text{mon}}(BC) \\ &\leq (2\alpha + \varepsilon) \cdot \mathcal{ALG}_{\text{mon}}(BC) , \end{aligned}$$

by setting  $0 < \beta \leq \frac{\varepsilon}{2\alpha}$  and using the Maclaurin series  $\frac{1}{1 - \varepsilon} = 1 + \varepsilon + \varepsilon^2 + \varepsilon^3 + \dots \leq 1 + 2\varepsilon$  for  $\varepsilon \leq 1/2$ . ◀

Having Theorem 2 in hand, we can improve quite a few sliding-window algorithms that have  $(2\alpha^2 + \varepsilon)$ -approximation factor. As an example, if we have  $\alpha$ -approximation streaming algorithms for the maximum submodular matching, maximum submodular  $b$ -matching, and in general, maximum submodular rank  $p$  hypergraph  $b$ -matching, we can then use Theorem 2 to develop sliding-window algorithms for all these problems with the approximation factor of  $(2\alpha + \varepsilon)$  which significantly improves upon the best known approximation factor of  $(2\alpha^2 + \varepsilon)$  of these problems due to Krauthgamer and Reitblat [14].

As an example, we consider next the maximum submodular matching, which is defined as follows. We are given a simple graph  $G(V, E)$ . A non-negative submodular function  $w : 2^E \rightarrow \mathbb{R}^{\geq 0}$  is defined on subsets of edges. The goal is to find a matching  $M_{\text{opt}}$ , whose submodular function  $w(M_{\text{opt}})$  is maximum. We next show that the maximum submodular matching is indeed a subadditive function.

► **Lemma 19.** *Let  $w$  be a non-negative submodular function defined on subsets of edges of a simple graph  $G(V, E)$ . Then, the maximum submodular matching of  $G(V, E)$  is subadditive.*

**Proof.** Let us consider an arbitrary stream  $S$  of edges of the underlying graph  $G(V, E)$ . Suppose we partition  $S$  into two disjoint consecutive substreams  $A$  and  $B$ . Since  $w$  is non-negative, we then have  $w(M_{\text{opt}}(AB)) \leq w(M_{\text{opt}}(AB)) + w(\emptyset)$ .

Next, we use the property of submodular functions that for every two sets  $X, Y$  we have  $w(X \cup Y) + w(X \cap Y) \leq w(X) + w(Y)$ . Let  $X = M_{\text{opt}}(AB) \cap A$  and  $Y = M_{\text{opt}}(AB) \cap B$ . Since  $A$  and  $B$  are disjoint substreams, we then have  $X \cup Y = M_{\text{opt}}(AB)$  and  $w(X \cap Y) = \emptyset$ . Therefore, we have

$$w(M_{\text{opt}}(AB)) \leq w(M_{\text{opt}}(AB) \cap A) + w(M_{\text{opt}}(AB) \cap B) .$$

Finally, since  $M_{\text{opt}}(AB) \cap A$  and  $M_{\text{opt}}(AB) \cap B$  are valid matchings for  $A$  and  $B$ , respectively, then  $w(M_{\text{opt}}(AB) \cap A) \leq w(M_{\text{opt}}(A))$  and  $w(M_{\text{opt}}(AB) \cap B) \leq w(M_{\text{opt}}(B))$ . Thus, we have

$$w(M_{\text{opt}}(AB)) \leq w(M_{\text{opt}}(A)) + w(M_{\text{opt}}(B)) ,$$

as we need. ◀

For the maximum monotone submodular matching problem, Levin and Wajc [15] develop a 5.828-approximation streaming algorithm. If we plug this streaming algorithm into the generic machinery of Theorem 2, we obtain 11.656-approximation sliding-window algorithm for this problem. This significantly improves upon the 67.93-approximation factor that the result of [14] achieves.

On the other hand, for the maximum non-monotone submodular matching problem, there exists a 7.464-approximation streaming algorithm due to Levin and Wajc [15]. We use this streaming algorithm for the the generic machinery of Theorem 2 and obtain 14.9-approximation sliding-window algorithm for this problem improving the 111.4-approximation factor that the result of [14] achieves.

---

## References

- 1 Aaron Bernstein. Improved bounds for matching in random-order streams. In *Proceedings 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPIcs*, pages 12:1–12:13, 2020. doi:10.4230/LIPIcs.ICALP.2020.12.
- 2 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *Proceedings 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007)*, pages 283–293, 2007. doi:10.1109/FOCS.2007.55.

- 3 Jiecao Chen, Huy L. Nguyen, and Qin Zhang. Submodular maximization over sliding windows. *CoRR*, abs/1611.00129, 2016. [arXiv:1611.00129](#).
- 4 Michael S. Crouch, Andrew McGregor, and Daniel M. Stubbs. Dynamic graphs in the sliding-window model. In *Proceedings 21st Annual European Symposium on Algorithms (ESA 2013)*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013. [doi:10.1007/978-3-642-40450-4\\_29](#).
- 5 Michael S. Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014*, volume 28 of *LIPICs*, pages 96–104, 2014. [doi:10.4230/LIPICs.APPROX-RANDOM.2014.96](#).
- 6 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. [doi:10.1137/S0097539701398363](#).
- 7 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016*, pages 608–614, 2016. [doi:10.1109/ICDMW.2016.0092](#).
- 8 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings 31st ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1773–1785, 2020. [doi:10.1137/1.9781611975994.108](#).
- 9 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. [doi:10.1016/j.tcs.2005.09.013](#).
- 10 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019*, pages 491–500. ACM, 2019. [doi:10.1145/3293611.3331603](#).
- 11 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming  $(2+\epsilon)$ -approximate matching. In *Proceedings 2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69 of *OASICS*, pages 13:1–13:8, 2019. [doi:10.4230/OASICS.SOSA.2019.13](#).
- 12 Michael Kapralov. Better bounds for matchings in the streaming model. In Sanjeev Khanna, editor, *Proceedings 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1679–1697, 2013. [doi:10.1137/1.9781611973105.121](#).
- 13 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242, 2012. [doi:10.1007/978-3-642-32512-0\\_20](#).
- 14 Robert Krauthgamer and David Reitblat. Almost-smooth histograms and sliding-window graph algorithms. *CoRR*, abs/1904.07957, 2019. [arXiv:1904.07957](#).
- 15 Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. *CoRR*, abs/2008.10062, 2020. [arXiv:2008.10062](#).
- 16 L. Lovasz and M.D. Plummer. *Matching Theory*. North-Holland, 1986.
- 17 Silvio Micali and Vijay V. Vazirani. An  $O(\sqrt{|V||E|})$  algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science (FOCS 1980)*, pages 17–27, 1980. [doi:10.1109/SFCS.1980.12](#).
- 18 Ami Paz and Gregory Schwartzman. A  $(2 + \epsilon)$ -approximation for maximum weight matching in the semi-streaming model. In Philip N. Klein, editor, *Proceedings 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 2153–2161, 2017. [doi:10.1137/1.9781611974782.140](#).

# Quantum Advantage with Shallow Circuits Under Arbitrary Corruption

Atsuya Hasegawa ✉

The University of Tokyo, Japan

François Le Gall ✉

Nagoya University, Japan

---

## Abstract

Recent works by Bravyi, Gosset and König (Science 2018), Bene Watts et al. (STOC 2019), Coudron, Stark and Vidick (QIP 2019) and Le Gall (CCC 2019) have shown unconditional separations between the computational powers of shallow (i.e., small-depth) quantum and classical circuits: quantum circuits can solve in constant depth computational problems that require logarithmic depth to solve with classical circuits. Using quantum error correction, Bravyi, Gosset, König and Tomamichel (Nature Physics 2020) further proved that a similar separation still persists even if quantum circuits are subject to local stochastic noise.

In this paper, we consider the case where *any* constant fraction of the qubits (for instance, huge blocks of qubits) may be arbitrarily corrupted at the end of the computation. We make a first step forward towards establishing a quantum advantage even in this extremely challenging setting: we show that there exists a computational problem that can be solved in constant depth by a quantum circuit but such that even solving *any* large subproblem of this problem requires logarithmic depth with bounded fan-in classical circuits. This gives another compelling evidence of the computational power of quantum shallow circuits.

In order to show our result, we consider the Graph State Sampling problem (which was also used in prior works) on expander graphs. We exploit the “robustness” of expander graphs against vertex corruption to show that a subproblem hard for small-depth classical circuits can still be extracted from the output of the corrupted quantum circuit.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** Quantum computing, circuit complexity, constant-depth circuits

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2021.74

**Funding** JSPS KAKENHI grants Nos. JP16H01705, JP19H04066, JP20H00579, JP20H04139 and by the MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) grant No. JPMXS0120319794.

**Acknowledgements** AH is grateful to Hidefumi Hiraishi and Hiroshi Imai for helpful discussions and continuous supports. FLG would also like to thank Robert König for useful discussions.

## 1 Introduction

**Background.** Quantum computing was introduced in the early 1980s as a quantum mechanical model of the Turing machine that has a potential to simulate things that a classical computer could not [11, 25]. In 1994, Peter Shor developed a polynomial-time quantum algorithm for factoring integers [43], which gave an exponential speedup over the most efficient known classical algorithms for this task.

While initially realizing a physical quantum computer was thought to be extremely challenging, nowadays various types of high-fidelity processors capable of quantum algorithms have been developed [10, 24, 36, 38, 45]. These devices with noise and relatively small scale are called NISQ (Noisy Intermediate-Scale Quantum) devices [41]. For such devices, “quantum supremacy” [40] has been recently reported [6, 46].



© Atsuya Hasegawa and François Le Gall;

licensed under Creative Commons License CC-BY 4.0

32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 74; pp. 74:1–74:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While Shor’s algorithm and such quantum supremacy results strongly suggest that quantum computation is more powerful than classical computation, they are not mathematical proofs. While the superiority of quantum computation has been formally shown in constrained models such as query complexity [4] and communication complexity [22, 34] and considering complexity classes relative to an oracle [12, 42], almost no definite answer is known in standard computational models such as Turing machines or general circuits. Since the complexity class BQP (the class of the problems that can be solved efficiently by a quantum computer) satisfies the inclusions  $P \subseteq BPP \subseteq BQP \subseteq PSPACE$ , an unconditional separation between BPP and BQP would imply a separation between P and PSPACE, which would be a significant breakthrough. Therefore, unconditional separations between the computational powers of quantum computers and classical computers in a general setting are expected to be very hard to obtain.

However, with several assumptions from computational complexity such as non-collapse of the polynomial hierarchy or some conjectures on the hardness of the permanent, the superiority of quantum computation has been shown in the circuit model: even approximate or noisy probabilistic distributions of small depth quantum circuits are hard to simulate for classical computers [1, 2, 3, 13, 16, 17, 18, 44]. A recent breakthrough by Bravyi, Gosset and König [14] showed an *unconditional* separation between the computational powers of small-depth quantum and classical circuits: they constructed a computational problem that can be solved by quantum circuits of constant depth composed of one- and two-qubit gates acting locally on a grid and showed that any probabilistic classical circuit with bounded fan-in gates solving this problem on all inputs must have depth  $\Omega(\log n)$ , where  $n$  denotes the input size. The computational problem they use is a relation problem (i.e., for any input there are several possible outputs). Besides its theoretical importance, this separation is also important since shallow quantum circuits are likely to be easy to implement on physical devices experimentally due to their robustness to noise and decoherence.

There are several results related to this separation. Coudron, Stark and Vidick [21] and Le Gall [26] showed a similar separation in the average case setting, instead of the worst case setting considered in the original version of [14]: there exists a relation problem such that constant-depth quantum circuits can solve the relation on all inputs, but any  $O(\log n)$  depth randomized bounded fan-in classical circuits cannot solve it on *most* inputs with high probability. Bene Watts et al. [9] showed that a similar separation holds against classical circuits using unbounded fan-in gates and, considering interactive tasks, Grier and Schaeffer [28] showed even stronger classical lower bounds.

Bravyi et al. [15] additionally proved, using quantum error-correction, that a similar separation holds even if quantum circuits are corrupted by local stochastic noise (see Definition 1 below). The computational problem used in [15] is a generalized version, defined on a 3D grid, of the magic square game [35, 39], which is a nonlocal game with two cooperating players Alice and Bob who cannot communicate. The noise model is as follows.

► **Definition 1** (Definition 9 in [15]). *Consider a random  $n$ -qubit Pauli error  $E \in \{I, X, Y, Z\}^{\otimes n}$  and let  $Supp(E) \subseteq [n]$  denote its support, i.e., the subset of qubits acted on by either  $X, Y$ , or  $Z$ . For any constant  $p \in [0, 1]$ ,  $E$  is called  $p$ -local stochastic noise if*

$$\Pr[F \subseteq Supp(E)] \leq p^{|F|} \quad \text{for all } F \subseteq [n].$$

In this model, an error is given as applying a gate which is  $X, Y$  or  $Z$ . This definition assumes that when picking an arbitrary subset of qubits, the probability of all qubits are corrupted is an exponentially small function of the size of the subset. This property, which implies that a subset of size  $\Omega(\log n)$  contains with probability  $1 - 1/\text{poly}(n)$  at least one qubit that

is not corrupted, is crucial in [15] to use quantum error correction. Note that, considering interactive tasks, Grier, Ju and Schaeffer [27] showed even stronger classical lower bounds in the same noise setting.

**The Graph State Sampling problem.** Before presenting our results, let us describe in more details the computational problem introduced in [14], which is called the 2D Hidden Linear Function problem and corresponds to an extension of the Bernstein-Vazirani problem [12].

We actually describe a slightly more general computational problem that we name the “Graph State Sampling problem” and denote  $\rho(G)$ . Here  $G = (V, E)$  is a graph specifying the problem. For any graph  $G$ , the problem  $\rho(G)$  is a relation  $\rho(G) \subseteq \{0, 1\}^{|V|+|E|} \times \{0, 1\}^{|V|}$ . Given an input  $x \in \{0, 1\}^{|V|+|E|}$  for this relation problem, which we interpret as a pair  $x = (y, H)$  with  $y \in \{0, 1\}^{|V|}$  and  $H$  being a subgraph of  $G$ , we ask to output any bit string  $z \in \{0, 1\}^{|V|}$  that may appear with nonzero probability when measuring the graph state corresponding to the subgraph  $H$  in a basis determined by the bit string  $y$ . We refer to Section 5.1 for details of the definition of the problem.

The 2D Hidden Linear Function problem considered in [14] is essentially the problem  $\rho(G)$  where  $G$  is the family of 2D grid graphs. Since the graph states of subgraphs of a 2D grid can be constructed by constant-depth quantum circuits whose gates act locally on the grid graphs, the problem  $\rho(G)$  can be solved by a constant-depth quantum circuit. At the same time, Bravyi et al. [14] prove that no small-depth classical circuits can solve this problem using an argument based on the existence of quantum nonlocality in a triangle (first shown by Barrett et al. [7]). The main result in [14] can essentially be restated as follows.<sup>1</sup>

► **Theorem 2** ([14]). *There exist a constant  $\alpha > 0$  such that the following holds for all sufficiently large 2D grid graphs  $G$ :*

- (i)  $\rho(G)$  can be solved on all inputs with certainty by a constant-depth quantum circuit on  $\Theta(|G|)$  qubits composed of one- and two-qubit gates;
- (ii) no bounded-fanin classical probabilistic circuit whose depth is less than  $\alpha \log(|G|)$  can solve with high probability  $\rho(G)$  on all inputs.

**Description of our results.** In this paper we show the following result.

► **Theorem 3.** *There exist constants  $\alpha > 0$  and  $\epsilon > 0$ , and a family of graphs  $(G_i)_{i \in \mathbb{N}}$  with  $\lim_{i \rightarrow \infty} |G_i| = \infty$  such that the following holds for all sufficiently large  $i$ :*

- (i)  $\rho(G_i)$  can be solved on all inputs with certainty by a constant-depth quantum circuit on  $\Theta(|G_i|)$  qubits composed of one- and two-qubit gates;
- (ii) for any induced subgraph  $S_i$  of  $G_i$  such that  $|S_i| \geq (1 - \epsilon)|G_i|$ , no bounded-fanin classical probabilistic circuit whose depth is less than  $\alpha \log(|S_i|)$  can solve with high probability  $\rho(S_i)$  on all inputs.

Item (ii) of Theorem 3, which is proved by considering the Graph State Sampling problem over expander graphs, gives a significantly stronger hardness guarantee than in Theorem 2 and thus provides us further compelling evidence of the computational power of quantum shallow circuits.

We stress that Theorem 3 does not claim a quantum advantage for noisy shallow quantum circuits: Theorem 3 simply shows that there exists a problem that can be computed by shallow quantum circuit but such that a shallow classical circuit cannot solve *any* (large)

<sup>1</sup> In this paper, for any graph  $G$  we use the notation  $|G|$  to denote the size of the vertex set of  $G$ .



subproblem of it. We can nevertheless interpret Item (ii) as follows. A quantum circuit  $\mathcal{C}$  solving the relation  $\rho(G_i)$  has  $|G_i|$  output qubits, which are measured at the end of the computation to give the output string  $z \in \{0, 1\}^{|G_i|}$  that is a solution for the relation. Assume that an adversary chooses up to  $\epsilon|G_i|$  qubits among these  $|G_i|$  qubits and corrupts them in an arbitrary way (or, essentially equivalently, corrupts the bits of the measurement outcomes corresponding to these positions). Let  $S_i$  denote the set of qubits that are not corrupted by the adversary. Since  $\rho(S_i)$  corresponds to a subproblem<sup>2</sup> of  $\rho(G_i)$ , and since  $\mathcal{C}$  before the corruption solved the problem  $\rho(G_i)$  on all inputs, even after the corruption the part of the output of  $\mathcal{C}$  corresponding to the qubits in  $S_i$  gives a correct solution to the problem  $\rho(S_i)$ . On the other hand, Item (ii) of Theorem 3 shows that no small-depth classical circuit can solve  $\rho(S_i)$ . (Note that in Item (ii) we even allow the classical circuit to depend on  $S_i$ .)

Let us compare this model of corruption of qubits with the model of noise considered in [15]. As already mentioned, in the error model of [15] (Definition 1 above), the probability that all qubits in a given set of size  $\Theta(\log n)$ , where  $n$  denotes the total number of qubits, are corrupted by the noise is polynomially small and thus can be neglected. In comparison, Theorem 3 deals with the case where any subset of qubits of size as large as  $\Theta(n)$  can be corrupted, and shows that the quantum advantage is still preserved in this case. In this sense, our result gives a further compelling evidence of the computational power of quantum shallow circuits.

**Brief overview of our techniques and organization of the paper.** The family of graph  $(G_i)_{i \in \mathbb{N}}$  used to prove Theorem 3 is a class of expander graphs of constant degree.<sup>3</sup> Item (i) of Theorem 3 essentially follows from the fact the graph state of a bounded-degree graph can be created by a constant-depth quantum circuit composed of one- and two-qubit gates. The proof of Item (ii) of Theorem 3 exploits the “robustness” of expander graphs against corruption of vertices. More precisely, we show that even after corrupting a constant fraction of vertices, an expander graph still has a large grid minor (see Lemma 8 in Section 3). Finally, exploiting the existence of a large grid minor, we can use arguments based on quantum nonlocality (very similarly to the arguments used in prior works [14, 26]) to conclude that any classical circuit solving the Graph State Sampling problem on the expander graph requires logarithmic depth.

After giving preliminaries in Section 2, in Section 3 we present graph-theoretical results about expander graphs. In particular, we prove Lemma 8 about the existence of large grid minors in corrupted expander graphs. In Section 4, we review the result about the nonlocality of a triangle quantum graph state used in prior works. In Section 5, we define our computational problem and prove Theorem 3.

## 2 Preliminaries

**Graph theory.** All graphs considered in this paper are undirected. We write a graph as  $G = (V, E)$  where  $V$  is the vertex set and  $E$  is the edge set.  $|G|$  means the number of vertex of graph  $G$ , i.e.,  $|V|$ . The degree of a vertex is the number of edges that are incident to the vertex. We denote  $\deg(v)$  the degree of a vertex  $v$ . Given a graph  $G = (V, E)$  and any vertex set  $U \subseteq V$ , we denote  $N_G(U) = \{v \in V \setminus U : v \text{ has a neighbor in } U\}$  the external neighborhood of  $U$  in  $G$ .

<sup>2</sup> This property can immediately be derived from the formal definition of the computational problem given in Section 5.1

<sup>3</sup> For technical reasons, we actually consider a family of graphs of the form  $G \times K_2$ , where  $G$  is an expander graph of constant degree and  $K_2$  is the graph consisting of a single edge.



Let us describe the definition of graph minors. Graph  $\Gamma$  is a graph minor of graph  $G$  if it is isomorphic to a graph obtained from  $G$  by deleting edges and vertices and by contracting edges. Note that if a graph  $\Gamma$  is a minor of a subgraph  $S$  of a graph  $G$ ,  $\Gamma$  is also a minor of  $G$ . The definition is equivalent to the following definition, that if graph  $\Gamma$  is a minor of graph  $G$ , we can decompose  $G$  to connected subgraphs which connect to each other like  $\Gamma$ . We will use this definition later for explicit explanations.

► **Definition 4** (Minor, Definition 1 in [33]). *A graph  $\Gamma$  is a minor of a graph  $G$  if for every vertex  $u \in \Gamma$  there is a connected subgraph  $G_u$  of  $G$  such that all subgraphs  $G_u$  are vertex disjoint, and  $G$  contains an edge between  $G_u$  and  $G_{u'}$  whenever  $\{u, u'\}$  is an edge of  $\Gamma$ .*

Next, we will define the product of graphs  $G \times H$  of graphs  $G = (V_G, E_G)$  and  $H = (V_H, E_H)$ . The vertex set of  $G \times H$  is the Cartesian product  $V_G \times V_H$  and an edge is spanned between  $(u_G, u_H)$  and  $(v_G, v_H)$  if and only if  $u_G = v_G$  and  $\{u_H, v_H\} \in E_H$ , or  $u_H = v_H$  and  $\{u_G, v_G\} \in E_G$ . There are several ways to define graph products but we will use the definition above. In this paper, we particularly use  $G \times K_2$ , which  $K_2$  is the complete graph of two vertices. Given a graph  $G = (V, E)$ , the graph product  $G \times K_2$  is with  $2|V|$  vertices and  $2|E| + |V|$  edges.

Lastly, we refer to the Vizing's theorem, which is about edge coloring. Edge coloring is to assign colors to edges so that the edges of the same color are not incident.

► **Lemma 5** (Vizing's theorem [23]). *Every simple undirected graph can be edge colored using a number of colors that is the maximum degree or the maximum degree+1.*

**Quantum circuits.** The textbook [37] is a good reference about notations of quantum computation in our paper. We will use the Pauli  $X$ ,  $Y$  and  $Z$  gates, the Hadamard gate and the  $S$  and  $T$  gates as single qubit gates:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, S = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}, T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix},$$

where  $i$  denotes the imaginary unit of complex numbers. (Note the Phase gate  $S$  differs from the standard one in [37].) We also use the controlled Pauli  $Z$  gate (or  $CZ = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z$ ) as two-qubit gate.

Let us explain about quantum circuits. An  $n$ -qubit quantum circuit is initialized to  $|0^n\rangle$  and then arbitrary gates are applied to the state. We can apply gates at one time if each gate is applied to disjoint sets of qubits. The depth of a quantum circuit is  $d$  if the whole operation of circuits can be decomposed to  $U_d \dots U_2 U_1$  where each  $U_j$  is a tensor product of one- and two-qubit gates which act on disjoint sets of qubits.

Next, we describe measurements of quantum states. Mathematically, (projective) measurements are projections to some orthogonal bases. In this paper, We will use two kinds of measurements with the  $X$  basis and the  $Y$  basis. The orthogonal two states of  $X$  basis are  $\{|+\rangle, |-\rangle\}$  and the ones of  $Y$  basis are  $\left\{\frac{|0\rangle+i|1\rangle}{\sqrt{2}}, \frac{|0\rangle-i|1\rangle}{\sqrt{2}}\right\}$ . Note that the measurements of  $X$  and  $Y$  basis are equivalent to the measurements of the computational basis if we apply  $H$  and  $HS$  gates before the measurements respectively.

**Quantum graph states.** Quantum graph states are a certain type of entangled states corresponding to graphs first introduced by [30]. Let  $G = (V, E)$  be a finite simple graph. Define an associated  $|V|$  qubit graph state  $|\Phi_G\rangle$  by

$$|\Phi_G\rangle = \left( \prod_{e \in E} CZ_e \right) H^{\otimes |V|} |0^{|V|}\rangle. \quad (1)$$

The graph state  $|\Phi_G\rangle$  is a stabilizer state with stabilizer group generated by the operators  $g_v = X_v \left( \prod_{w: \{w,v\} \in E} Z_w \right)$  for all  $v \in V$ .

**Classical circuits.** A classical circuit is specified by a directed acyclic graphs. Vertices with no incoming and outgoing edges are inputs and outputs respectively and all other vertices are called gates. We must specify a function of each gate  $\{0, 1\}^k \rightarrow \{0, 1\}$  where  $k$  is the fan-in. We assume a classical probabilistic circuit receives an arbitrary binary  $x$  as an input and outputs a binary  $z$  and it also could input a random string  $r$  drawn from some arbitrary distribution. We say an input bit and an output bit are correlated iff the value of the output bit depends on the value of the input bit. For each input bit  $x_i$ , we denote the lightcone  $L_C(x_i)$  the set of output bits correlated with  $x_i$  through a classical circuit  $C$ . Likewise, the lightcone  $L_C(z_i)$  is the set of input bits correlated with an output bit  $z_i$ . In this paper, we are interested in small-depth classical circuits with bounded fan-in. We also say that a classical probabilistic circuit solves the relation  $R$  on all inputs if and only if the circuit takes any  $x \in \{0, 1\}^n$  and a random string  $r$  as input and outputs  $z \in \{0, 1\}^m$  such that  $xRz$  with high probability.

### 3 Expander graphs and their properties

Expander graphs are highly sparse but well connected graphs. Notable applications of the graphs have been found in mathematics and computer science [8, 19, 29]. Before giving the definition, we define the expansion ratio  $h(G)$  of graph  $G$ . There are several ways to define the expansion ratio, for example edge expansion, vertex expansion and spectral expansion, but these are related to each other. The way we use is called vertex expansion.

► **Definition 6** (Expansion ratio).  $h(G) = \min \left\{ \frac{|N_G(U)|}{|U|} \mid U \subset V \text{ such that } 1 \leq |U| \leq \frac{1}{2}|V| \right\}$

The definition of expander graphs we use in this paper is as follows.

► **Definition 7** (Expander graphs, Definition 3.1.8 in [31]). *A family  $(\Gamma_i)_{i \in \mathbb{N}}$  of finite non-empty connected graphs  $\Gamma_i = (V_i, E_i)$  is an expander family, if there exist constants  $d \geq 1$  and  $h > 0$ , independent of  $i$ , such that:*

- (1) *The number of vertices  $|V_i|$  “tends to infinity”, in the sense that for any  $N \geq 1$ , there are only finitely many  $i \in \mathbb{N}$  such that  $\Gamma_i$  has at most  $N$  vertices.*
- (2) *For each  $i \in \mathbb{N}$ , we have  $\max_{v \in V_i} \deg(v) \leq d$ , i.e., the maximum degree of the graphs is bounded independently of  $i$ .*
- (3) *For each  $i \in \mathbb{N}$ , the expansion constant satisfies  $h(\Gamma_i) \geq h > 0$ , i.e., it is bounded away from 0 by a constant independent of  $i$ .*

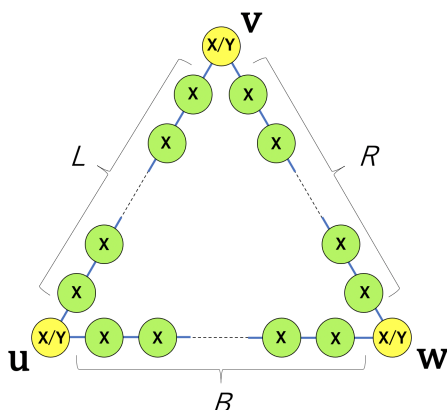
$h$  and  $d$  specify the family of expander graphs. The existence of expander graphs is by no means obvious, but it can be shown using probabilistic approaches or concrete constructions of such graphs [31]. In this paper, an expander graph denotes a graph with sufficiently large  $i$  of an expander family.

We want to prove the robustness of expander graphs to arbitrary vertex removals in terms of the size of grid minor, which is Lemma 8. We provide its proof in Appendix A.

► **Lemma 8.** *Let  $G = (V, E)$  be a expander graph. If we take a sufficiently small constant  $\epsilon > 0$ , the graph has a connected component  $C$  which contains a  $\Omega(|V|^{\frac{1}{4}}) \times \Omega(|V|^{\frac{1}{4}})$  grid as a minor after up to an  $\epsilon$  fraction of  $V$  are adversarially removed.*

#### 4 Quantum nonlocality of a triangle graph state

In specific circumstances, local classical circuits cannot simulate measurement outcomes of entangled quantum states. This is called *quantum nonlocality*. In this section, we explain it occurs in a triangle graph state, as first shown in [7]. We will use this property to show the hardness of classical circuits to solve the relation problem in Section 5.



■ **Figure 1** We consider an even cycle  $\Gamma$  and three vertices  $u, v, w$  such that all pairwise distances are even.  $L, R, B$  are the region between the three vertices. Each qubit of the corresponding graph state is measured by the  $X$  or  $Y$  basis.

First, we note about properties of a graph state in a triangle shape. Let  $\Gamma$  be a triangle such that the distance between three vertices  $u, v, w$  are all even and  $|\psi_\Gamma\rangle$  be a graph state for  $\Gamma$  as in Equation (1). We define  $L, R, B$  as the vertices between  $u$  and  $v$ ,  $v$  and  $w$ ,  $u$  and  $w$ , and  $M$  as a total number of vertices in  $\Gamma$ . Also define  $L_{odd}, L_{even}, R_{odd}, R_{even}, B_{odd}, B_{even}$  as vertices of  $L, R, B$  which have odd and even distance from  $u, v, w$  respectively. The three bits  $x = x_u x_v x_w$  decide the measurement basis of  $u, v, w$  (the  $X$  basis if  $x_i = 0$  and the  $Y$  basis if  $x_i = 1$ ) and the other vertices are measured in the  $X$  basis. We denote  $\tau(x)$  possible measurement outcomes of all qubits of  $|\psi_\Gamma\rangle$  for input  $x$ , i.e.,  $\tau(x) = \{z \in \{0, 1\}^M : \langle z | H^{\otimes M} S_u^{x_u} S_v^{x_v} S_w^{x_w} | \psi_\Gamma \rangle \neq 0\}$ . We consider a relationship between input  $x$  and output  $z \in \tau(x)$ . Define the following summations:

$$z_L = \bigoplus_{i \in L_{odd}} z_i \quad z_R = \bigoplus_{i \in R_{odd}} z_i \quad z_B = \bigoplus_{i \in B_{odd}} z_i \quad z_E = \bigoplus_{i \in \{u, v, w\} \cup R_{even} \cup L_{even} \cup B_{even}} z_i.$$

▷ **Claim 9** (Claim 3 in [14]). Let  $x = x_u x_v x_w \in \{0, 1\}^3$  and suppose  $z \in \tau(x)$ . Then  $z_R \oplus z_B \oplus z_L = 0$ . Moreover, if  $x_u \oplus x_v \oplus x_w = 0$  then

$$\begin{aligned} (x_u x_v x_w = 000) \quad z_E = 0, & \quad (x_u x_v x_w = 110) \quad z_E \oplus z_L = 1, \\ (x_u x_v x_w = 101) \quad z_E \oplus z_B = 1, & \quad (x_u x_v x_w = 011) \quad z_E \oplus z_R = 1. \end{aligned}$$

The following lemma is about quantum nonlocality in graph states of triangles and similar to Lemma 3 in Section 4.1 of [14]. It shows that when we assume a classical circuit has a kind of locality, the classical circuit cannot satisfy the relation of Claim 9 as inputs and outputs.

► **Lemma 10** ([7, 14, 26]). Consider a classical circuit which takes as an input a bit string  $x = x_u x_v x_w \in \{0, 1\}^3$  and a random string  $r$ , and outputs  $z \in \{0, 1\}^M$  which are corresponding to vertices of  $\Gamma$ . Let us assume output bits in  $L$  depend on  $r$  and at most one geometrically near input bit, which is either  $x_u$  or  $x_v$ . Similarly, we assume output bits in  $R$  and  $B$  depend on  $r$  and at most one geometrically near input bit. Then, the classical circuit  $C$  cannot output  $z \in \tau(x)$  with high probability.

## 5 Proof of separation of depth between quantum and classical circuits to solve Graph State Sampling problem

In this section, we define Graph State Sampling problem and prove a separation of depth between quantum circuits and classical circuits. We consider an almost all induced subgraph  $S$  of a graph  $G \times K_2$  such that  $G$  is an expander graph. Then, we prove  $\rho(G \times K_2)$  can be solved on all inputs by a constant-depth quantum circuit on  $\Theta(|G \times K_2|)$  qubits, but  $\Omega(\log |S|)$  depth is required for any classical probabilistic circuits to solve  $\rho(S)$  on all inputs.

### 5.1 Definition of Graph State Sampling problem $\rho(G)$

In this subsection, for any graph  $G$ , we define Graph State Sampling problem  $\rho(G)$ .

The relation is defined as a subset of  $\{0, 1\}^{|V|+|E|} \times \{0, 1\}^{|V|}$  and thus consists of pairs  $(x, z)$ , where  $x \in \{0, 1\}^{|V|+|E|}$  represents the input and  $z \in \{0, 1\}^{|V|}$  represents the output. Each bit of  $x$  corresponds to a vertex or an edge. The string  $x$  decides the quantum graph state and the measurement bases: a  $CZ$  gate corresponding to edge  $e$  is applied if  $x_e = 1$ , and the qubit corresponding to a vertex  $v$  is measured in the  $X$  basis if  $x_v = 0$ , or in the  $Y$  basis if  $x_v = 1$ . The quantum state  $|\psi_x\rangle$  for each  $x$  before the measurement in the computational basis is thus:

$$|\psi_x\rangle = H^{\otimes |V|} \prod_{x_v=1} S_v \left( \prod_{x_e=1} CZ_e \right) H^{\otimes |V|} |0^{|V|}\rangle.$$

The output  $z \in \{0, 1\}^{|V|}$  of the relation is any possible outcome of the measurement of this quantum state (note that there are possibly several measurement outcomes  $z$  for each  $x$ ). Since the probability of measurement results of each binary string  $z$  is  $|\langle z | \psi_x \rangle|^2$ , the definition of  $\rho(G)$  is as follows.

► **Definition 11.** *Given a graph  $G$ ,*

$$\rho(G) = \{(x, z) | x \in \{0, 1\}^{|V|+|E|} \text{ and } z \in \{0, 1\}^{|V|} \text{ such that } |\langle z | \psi_x \rangle|^2 > 0\}.$$

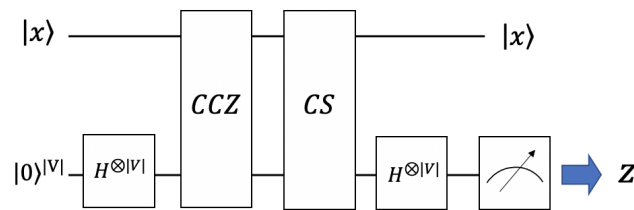
### 5.2 Constant-depth quantum circuits to solve Graph State Sampling problem

The following is easily shown by Lemma 5.

► **Lemma 12.** *When the maximum degree of graph  $G = (V, E)$  is bounded by a constant,  $\rho(G)$  on all inputs can be solved with certainty by a constant-depth quantum circuit on  $\Theta(|G|)$  qubits composed of one- and two-qubit gates.*

**Proof.** The initial state  $|x\rangle \otimes |0^{|V|}\rangle$  is prepared on  $|E| + 2|V| = \Theta(|G|)$  qubits. We apply  $H^{\otimes |V|}$  to the last  $|V|$  qubits and then controlled- $CZ$  ( $CCZ$ ) gates and controlled- $S$  ( $CS$ ) gates that apply  $CZ_e$  if  $x_e = 1$  and  $S_v$  if  $x_v = 1$ . Since  $G$  is edge colorable with a constant number from Lemma 5 and  $CCZ$  gates corresponding to edges assigned the same color can be applied simultaneously (since they act on disjoint sets of qubits), the total depth of  $CCZ$  gates can be bounded by a constant.  $CS$  gates can also be applied in constant depth. We finally apply  $H^{\otimes |V|}$  to the last  $|V|$  qubits and measure these qubits in the computational basis, which gives a string  $z$  such that  $(x, z) \in \rho(G)$ .

The total depth of this circuit can be bounded by a constant. (Note that each  $CCZ$  and  $CS$  gate can be implemented in constant depth using our elementary gates [5].) We refer to Figure 2 for an illustration. ◀



■ **Figure 2** Constant-depth quantum circuit to solve  $\rho(G)$ .

### 5.3 Hardness to solve Graph State Sampling problem with shallow classical circuits

In this subsection, we prove Theorem 3, and especially the classical hardness. The impossibility argument in Section 4 assumed classical circuits had a kind of geometrical locality. The lower bounds in this section, however, do not require any geometrical locality of shallow classical circuits. The proofs are similar to the proofs of the results of Section 4.2 in [14], with the notable exception of Claim 16 and the discussion afterwards (in particular, the definition of boxes), which are specifically tailored for the expander graphs we consider.

#### 5.3.1 Good and bad vertices

To begin with, we define “good” and “bad” vertices. In a shallow classical circuit, most input bits are not correlated with many output bits. We call a vertex “bad” if the corresponding input bit are correlated with many output bits. Here is the formal definition.

► **Definition 13.** *Given a graph  $G = (V, E)$ , we consider  $\rho(G)$  and a classical probabilistic circuit  $C$  for it. Then, a vertex  $v \in V$  is good if  $L_C(x_v) = O(|V|^{\frac{1}{16}})$  and bad if  $v$  is not good.*

The following claim is similar to Claim 5 in [14].

▷ **Claim 14.** Let  $G = (V, E)$  be a graph such that the maximum degree is bounded by a constant and  $C$  be a classical probabilistic circuit for  $\rho(G)$ . Suppose the fan-in is bounded by a constant  $K$  and the depth  $d$  is less than  $\frac{\log |V|}{32 \log K}$ , the number of bad vertices is  $o(|V|)$ .

*Proof.* Since the number of correlated input bits increases by at most  $K$  times when the depth increases by 1,

$$|L_C(z_i)| \leq K^d < |V|^{\frac{1}{32}} \quad \text{for all } v \in V. \tag{2}$$

Let us consider a bipartite graph whose vertices are respective bits of  $x$  and  $z$ , and an edge is spanned if and only if  $x_i$  and  $z_j$  are correlated. Since the maximum degree of  $G$  is bounded by a constant, we have  $|x| = |V| + |E| = \Theta(|V|)$ . From Equation (2) and considering edges spanned from  $z$ , the total number of edges is limited by  $|V| \cdot |V|^{\frac{1}{32}}$ . Since bad vertices are correlated with  $\Omega(|V|^{\frac{1}{16}})$  output bits, the total number of  $x_v$  such that  $v$  is bad is  $O(|V|^{\frac{31}{32}})$  and this means the number of bad vertices is  $o(|V|)$ . ◁

#### 5.3.2 Proof of Theorem 3

Before the proof, we rewrite Theorem 3 using the notations we defined. The reason we consider the graph product  $G \times K_2$  is to take a cycle which has even length for using Lemma 10.

► **Theorem 15.** *There exist constants  $\alpha > 0$  and  $\epsilon > 0$  such that the following holds for all sufficiently large expander graphs  $G$ :*

- (i)  $\rho(G \times K_2)$  can be solved on all inputs with certainty by a constant-depth quantum circuit on  $\Theta(|G \times K_2|)$  qubits composed of one- and two-qubit gates;
- (ii) for any induced subgraph  $S$  such that  $|S| \geq (1 - \epsilon)|G \times K_2|$ , no bounded-fanin classical probabilistic circuits whose depth is less than  $\alpha \log(|S|)$  can solve  $\rho(S)$  on all inputs.

**Proof of Theorem 15.** We can take  $|G| = |V|$  arbitrary large from the property of expander graphs. Then  $|G \times K_2| = 2|G|$  and  $|S| \geq (1 - \epsilon)|G \times K_2|$  are also sufficiently large. Let us introduce some convenient notations. Given a vertex  $u \in G$ , we denote  $u'$  and  $u''$  the two corresponding vertices in  $G \times K_2$  (with no special order). Given a vertex  $v \in G \times K_2$ , we denote  $\bar{v}$  the other vertex in  $G \times K_2$  associated to the same vertex in  $G$ .

First, we prove Theorem 15 (i). We consider the Graph State Sampling problem  $\rho(G \times K_2)$ . Since the maximum degree of  $G$  is bounded by a constant  $d$ , the maximum degree of  $G \times K_2$  is bounded by  $d + 1$ . From Lemma 12,  $\rho(G \times K_2)$  can be solved with a constant-depth quantum circuit.

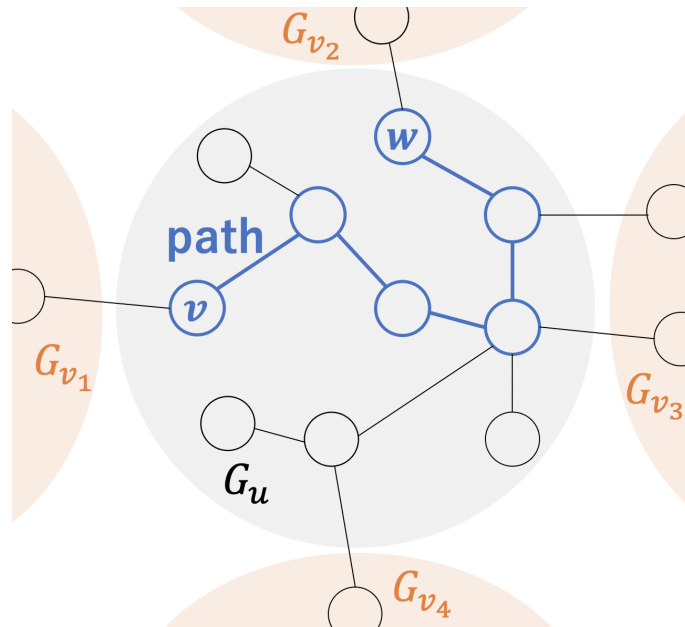
Next, we will prove Theorem 15 (ii), the hardness to solve  $\rho(S)$  on all inputs with shallow classical circuits. Let  $C$  be a classical probabilistic circuit to solve  $\rho(S)$  on all inputs and  $K$  be the bounded fan-in of  $C$ . In order to reach a contradiction, we assume the depth of  $C$  is less than  $\frac{\log |S|}{32 \log K}$ . Then the number of bad vertices is small, which enables us to prove the following claim.

▷ **Claim 16.**  $S$  contains an induced subgraph  $G' \times K_2$  such that all vertices are good and  $G'$  contains a  $\Omega(|V|^{\frac{1}{4}}) \times \Omega(|V|^{\frac{1}{4}})$  grid as a minor.

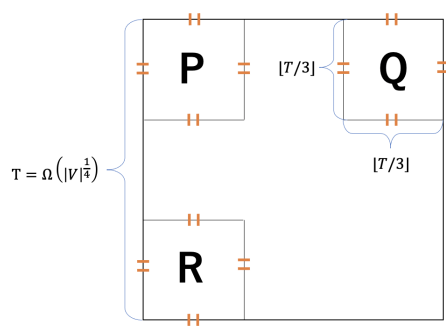
*Proof.* From Claim 14, we know that  $S$  contains  $o(|S|)$  bad vertices. Let us remove all these bad vertices, and write  $S_{good}$  the remaining set. We further remove all vertices  $u \in S_{good}$  such that  $\bar{u} \notin S_{good}$ . The remaining set of vertices induces a graph  $H \times K_2$ , for an induced subgraph  $H$  of  $G$  such that  $|H| \geq (1 - 2\epsilon - o(1))|G|$ . From Lemma 8, when  $\epsilon$  is taken small enough, the graph  $H \times K_2$  has a connected component  $G' \times K_2$ , where  $G'$  contains a  $\Omega(|V|^{\frac{1}{4}}) \times \Omega(|V|^{\frac{1}{4}})$  grid as a minor. ◁

Remember the definition of a graph minor (Definition 2 in Section 2.1). Each connected subgraph  $G_u$  in  $G'$  forming the grid (except connected subgraphs on the corners of the grid) is adjacent to the four connected subgraphs  $G_{v_1}, G_{v_2}, G_{v_3}, G_{v_4}$  where  $\{u, v_1\}, \{u, v_2\}, \{u, v_3\}$  and  $\{u, v_4\}$  are edges of the grid. For each  $G_u$ , we arbitrarily select two of these four components. Assume for instance that we selected  $G_{v_1}$  and  $G_{v_2}$ . We choose arbitrarily one vertex  $v$  in  $G_u$  adjacent to a vertex of  $G_{v_1}$ , and one vertex  $w$  in  $G_u$  adjacent to a vertex of  $G_{v_2}$ . We then arbitrarily choose one path inside  $G_u$  that connects  $v$  and  $w$  (such a path necessarily exists). We refer to Figure 3 for an illustration.

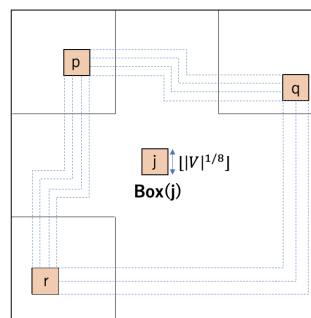
We denote  $T$  the length of one side of the grid ( $T = \Omega(|V|^{\frac{1}{4}})$ ). From each connected subgraph forming the  $T \times T$  grid of  $G'$ , we choose one vertex such that it is on a path selected in the way above (this condition of the vertices is required when adding missing segments inside boxes for Claim 18). For such a vertex  $j$ , we define  $\text{Box}(j) \subseteq G' \times K_2$  as a 2D grid of connected subgraphs of size  $\lfloor |V|^{\frac{1}{8}} \rfloor \times \lfloor |V|^{\frac{1}{8}} \rfloor$  centered at the connected subgraph which  $j'$  and  $j''$  belong to. We choose grid-shaped regions  $P, Q, R \subseteq G' \times K_2$  as shown in Figure 4.  $P$  is a upper-left region of the graph product of  $K_2$  and a  $\lfloor T/3 \rfloor \times \lfloor T/3 \rfloor$  grid of connected subgraphs cut from  $G' \times K_2$ .  $Q$  is a upper-right region and  $R$  is a bottom-left region. The following claim is similar to Claim 6 in [14].



■ **Figure 3** The path in  $G_u$  described with blue line connects  $G_{v_1}$  and  $G_{v_2}$ .



■ **Figure 4** Definition of the regions  $P, Q, R$  of  $G' \times K_2$ .



■ **Figure 5** Definition of  $\text{Box}(j)$  and a possible choice of  $p, q, r$ .



## 74:12 Quantum Advantage with Shallow Circuits Under Arbitrary Corruption

▷ **Claim 17.** For all large enough  $|V|$ , we can choose a triple of vertices  $p, q, r \in V$  such that  $p', p'' \in P, q', q'' \in Q, r', r'' \in R$  and

$$\text{Box}(p) \subseteq P, \text{Box}(q) \subseteq Q, \text{Box}(r) \subseteq R, \quad (3)$$

$$L_C(x_{p'} \cup x_{p''}) \cap \text{Box}(q) = \emptyset, L_C(x_{p'} \cup x_{p''}) \cap \text{Box}(r) = \emptyset, \quad (4)$$

$$L_C(x_{q'} \cup x_{q''}) \cap \text{Box}(p) = \emptyset, L_C(x_{q'} \cup x_{q''}) \cap \text{Box}(r) = \emptyset, \quad (5)$$

$$L_C(x_{r'} \cup x_{r''}) \cap \text{Box}(p) = \emptyset, L_C(x_{r'} \cup x_{r''}) \cap \text{Box}(q) = \emptyset. \quad (6)$$

*Proof.* One connected subgraph in the grid minor can belong to at most  $|V|^{\frac{1}{8}} \times |V|^{\frac{1}{8}} = |V|^{\frac{1}{4}}$  boxes. Since the vertices in  $G' \times K_2$  are good, a given lightcone  $L_C(x_{u'} \cup x_{u''})$  can intersect with at most  $|V|^{\frac{1}{4}} \times |L_C(x_{u'} \cup x_{u''})| = |V|^{\frac{1}{4}} \times O(|S|^{\frac{1}{16}}) = O(|V|^{\frac{5}{16}})$  boxes. The number of possibilities to choose a box in  $Q$  is  $\Omega(|V|^{\frac{1}{4}}) \times \Omega(|V|^{\frac{1}{4}}) = \Omega(|V|^{\frac{1}{2}})$ . Thus if we pick boxes uniformly at random then

$$\Pr[L_C(x_{p'} \cup x_{p''}) \cap \text{Box}(q) = \emptyset] \leq O\left(\frac{|V|^{\frac{5}{16}}}{|V|^{\frac{1}{2}}}\right) < \frac{1}{6} \quad (7)$$

for large enough  $|V|$ . A similar bound applies to the five others that appear in the three equations (4, 5, 6). By the union bound, there exists at least one choice of  $p, q, r$  that satisfies all the four equations (3, 4, 5, 6). ◁

Below we consider a cycle  $\Gamma$  that is a subgraph of  $G' \times K_2$ . The following claim is similar to Claim 7 in [14].

▷ **Claim 18.** The following holds for all sufficiently large  $|V|$ . Fix some triple of vertices  $p, q, r$  satisfying the four equations (3, 4, 5, 6). Then there exists an even length cycle  $\Gamma$  containing  $p', p'', q', q'', r', r''$  such that the lightcones  $L_C(x_{p'} \cup x_{p''}), L_C(x_{q'} \cup x_{q''}), L_C(x_{r'} \cup x_{r''})$  contain no vertices of  $\Gamma$  lying outside of  $\text{Box}(p) \cup \text{Box}(q) \cup \text{Box}(r)$ .

*Proof.* Since the size of connected subgraphs of each box is  $\lfloor |V|^{\frac{1}{8}} \rfloor \times \lfloor |V|^{\frac{1}{8}} \rfloor$ , we can choose  $\lfloor |V|^{\frac{1}{8}} \rfloor$  pairwise vertex disjoint paths  $\gamma$  that connect any pair of boxes  $\text{Box}(p), \text{Box}(q), \text{Box}(r)$  (in each connected subgraph, we can always find a path which connects adjacent connected subgraphs). We refer to Figure 5 for an illustration. Let  $\gamma(a, b)$  be a path connecting  $\text{Box}(a)$  and  $\text{Box}(b)$ , where  $a \neq b \in \{p, q, r\}$ . Any triple of paths  $\gamma(p, q), \gamma(q, r), \gamma(p, r)$  can be completed to a cycle  $\Gamma$  by adding the missing segments of the cycle inside the boxes  $\text{Box}(p), \text{Box}(q), \text{Box}(r)$  because  $p, q, r$  are defined to take such a path inside each box. Since all vertices are good in  $G' \times K_2$  and each connected subgraph belongs to at most one path  $\gamma$ , we infer that  $L_C(x_{p'} \cup x_{p''})$  intersects with at most  $2 \cdot O(|S|^{\frac{1}{16}}) = O(|V|^{\frac{1}{16}})$  paths  $\gamma$ . Thus if we pick the path  $\gamma(p, q)$  uniformly at random among all  $\lfloor |V|^{\frac{1}{8}} \rfloor$  possible choices then

$$\Pr[L_C(x_{p'} \cup x_{p''}) \cap \gamma(p, q) \neq \emptyset] \leq O\left(\frac{|V|^{\frac{1}{16}}}{|V|^{\frac{1}{8}}}\right) < \frac{1}{9} \quad (8)$$

for enough large  $|V|$ . The same bound applies to eight remaining combinations of lightcones  $L_C(x_{p'} \cup x_{p''}), L_C(x_{q'} \cup x_{q''}), L_C(x_{r'} \cup x_{r''})$  and paths  $\gamma(p, q), \gamma(p, r)$ , and  $\gamma(q, r)$ . By the union bound, there exists at least one triple of paths  $\gamma(p, q), \gamma(q, r), \gamma(p, r)$  that do not intersect with  $L_C(x_{p'} \cup x_{p''}), L_C(x_{q'} \cup x_{q''}), L_C(x_{r'} \cup x_{r''})$ . When we choose  $v'$  and  $v''$  consecutively, any cycle in  $G' \times K_2$  has an even length. Since we can take the cycle in the way above, the cycle has a even length and is the desired cycle  $\Gamma$ . ◁

Let  $p, q, r$  and  $\Gamma$  be chosen as described in Claim 18. Let  $M$  be the even number of vertices of  $\Gamma$ . When we choose  $p', p'', q', q'', r', r''$  properly, the distances between  $p', q', r'$  are all even. Consider the subset of instances where

$$x_e = \begin{cases} 1 & \text{if } e \text{ is an edge of } \Gamma \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad x_v = 0 \text{ if } (v \in V \setminus \{p', q', r'\})$$

There are  $2^3 = 8$  such instances corresponding to choices of input bits  $x_{p'}, x_{q'}, x_{r'} \in \{0, 1\}$ . Let us fix inputs  $x$  of the circuit  $C$  except  $\{x_{p'}, x_{q'}, x_{r'}\}$  and consider only output bits  $z_j$  with  $j \in \Gamma$ . By the way of fixing, we obtain a classical circuit  $D$  which takes a three-bit string  $x_{p'}x_{q'}x_{r'} \in \{0, 1\}^3$  and a random string  $r$  as input and output  $z_\Gamma \in \{0, 1\}^M$ . For any input bit  $x_i \in \{x_{p'}, x_{q'}, x_{r'}\}$  we have  $L_D(x_i) \subseteq L_C(x_i)$  since any pair of input and output variables which are correlated in  $D$  are also correlated in  $C$ , by definition. Our assumption that  $C$  can solve  $\rho(S)$  on all inputs implies that  $D$  can output  $z_\Gamma \in \tau(x_{p'}x_{q'}x_{r'})$ . From Lemma 10, at least one output bit  $z_j$  such that  $j \in \Gamma$  and  $j \notin \{p', q', r'\}$  depends on the two geometrically near inputs from  $x_{p'}, x_{q'}, x_{r'}$ . By  $L_D(x_i) \subseteq L_C(x_i)$ , the same is true for the input-output dependency of  $C$ . From Claim 9, for each  $x_i$ ,  $L_C(x_i)$  only intersects with  $z_j$  such that  $j \in \text{Box}(i)$  and there is a contradiction. Therefore, the depth of any classical probabilistic circuit that has bounded fan-in  $K$  and solves  $\rho(S)$  on all inputs is not less than  $\frac{\log |S|}{32 \log K}$ . This concludes the proof of Theorem 3. ◀

---

## References

- 1 Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the 43rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2011)*, pages 333–342, 2011.
- 2 Scott Aaronson and Alex Arkhipov. Bosonsampling is far from uniform. *Quantum Information & Computation*, 14(15–16):1383–1423, 2014.
- 3 Scott Aaronson and Lijie Chen. Complexity-Theoretic Foundations of Quantum Supremacy Experiments. In *Proceedings of 32nd Computational Complexity Conference (CCC 2017)*, volume 79 of *LIPICs*, pages 22:1–22:67, 2017.
- 4 Andris Ambainis. Understanding quantum algorithm via query complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3265–3285, 2019.
- 5 Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013.
- 6 Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.
- 7 Jonathan Barrett, Carlton M. Caves, Bryan Eastin, Matthew B. Elliott, and Stefano Pironio. Modeling pauli measurements on graph states with nearest-neighbor classical communication. *Physical Review A*, 75(1):012103, 2007.
- 8 Ya. M. Barzdin. On the realization of networks in three-dimensional space. In *Selected Works of A. N. Kolmogorov: Volume III: Information Theory and the Theory of Algorithms*, pages 194–202. Springer, 1993.
- 9 Adam Bene Watts, Robin Kothari, Luke Schaeffer, and Avishay Tal. Exponential separation between shallow quantum circuits and unbounded fan-in shallow classical circuits. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 515–526, 2019.
- 10 Jan Benhelm, Gerhard Kirchmair, Christian F. Roos, and Rainer Blatt. Towards fault-tolerant quantum computing with trapped ions. *Nature Physics*, 4(6):463–466, 2008.

- 11 Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.
- 12 Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.
- 13 Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. "Quantum Supremacy" and the Complexity of Random Circuit Sampling. In *Proceedings of 10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *LIPICs*, pages 15:1–15:2, 2019.
- 14 Sergey Bravyi, David Gosset, and Robert König. Quantum advantage with shallow circuits. *Science*, 362(6412):308–311, 2018.
- 15 Sergey Bravyi, David Gosset, Robert König, and Marco Tomamichel. Quantum advantage with noisy shallow circuits. *Nature Physics*, 16(10):1040–1045, 2020.
- 16 Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2126):459–472, 2011.
- 17 Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Average-case complexity versus approximate simulation of commuting quantum computations. *Physical Review Letters*, 117(8):080501, 2016.
- 18 Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum*, 1:8, 2017.
- 19 Michael Capalbo, Omer Reingold, Salil Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2002)*, pages 659–668, 2002.
- 20 Julia Chuzhoy and Rachit Nimavat. Large minors in expanders. *arXiv*, 2019. [arXiv:1901.09349](https://arxiv.org/abs/1901.09349).
- 21 Matthew Coudron, Jalex Stark, and Thomas Vidick. Trading locality for time: certifiable randomness from low-depth circuits. *Communications in Mathematical Physics*, 382(1):49–86, 2021. Also presented at QIP 2019.
- 22 Ronald De Wolf. Quantum communication and complexity. *Theoretical Computer Science*, 287(1):337–353, 2002.
- 23 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2000.
- 24 Andreas Wallraff et al. Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics. *Nature*, 431(7005):162–167, 2004.
- 25 Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7):467–488, 1982.
- 26 François Le Gall. Average-Case Quantum Advantage with Shallow Circuits. In *Proceedings of 34th Computational Complexity Conference (CCC 2019)*, volume 137 of *LIPICs*, pages 21:1–21:20, 2019.
- 27 Daniel Grier, Nathan Ju, and Luke Schaeffer. Interactive quantum advantage with noisy, shallow clifford circuits. *arXiv*, 2021. [arXiv:2102.06833](https://arxiv.org/abs/2102.06833).
- 28 Daniel Grier and Luke Schaeffer. Interactive shallow clifford circuits: quantum advantage against  $NC^1$  and beyond. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, pages 875–888, 2020.
- 29 Misha Gromov and Larry Guth. Generalizations of the kolmogorov–barzdin embedding estimates. *Duke Mathematical Journal*, 161(13):2549–2603, 2012.
- 30 Marc Hein, Jens Eisert, and Hans J. Briegel. Multiparty entanglement in graph states. *Physical Review A*, 69(6):062311, 2004.
- 31 Emmanuel Kowalski. *An introduction to expander graphs*. Société Mathématique de France, 2019.
- 32 Michael Krivelevich. Expanders – how to find them, and what to find in them. *Surveys in Combinatorics*, pages 115–142, 2019.

- 33 Michael Krivelevich and Benjamin Sudakov. Minors in expanding graphs. *Geometric and Functional Analysis*, 19(1):294–331, 2009.
- 34 Hoi-Kwong Lo. Classical-communication cost in distributed quantum-information processing: a generalization of quantum-communication complexity. *Physical Review A*, 62(1):012313, 2000.
- 35 N. David Mermin. Extreme quantum entanglement in a superposition of macroscopically distinct states. *Physical Review Letters*, 65(15):1838, 1990.
- 36 Yasunobu Nakamura, Chii Dong Chen, and Jaw Shen Tsai. Spectroscopy of energy-level splitting between two macroscopic quantum states of charge coherently superposed by josephson coupling. *Physical Review Letters*, 79(12):2328, 1997.
- 37 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
- 38 Jeremy L O’Brien. Optical quantum computing. *Science*, 318(5856):1567–1570, 2007.
- 39 Asher Peres. Incompatible results of quantum measurements. *Physics Letters A*, 151(3-4):107–108, 1990.
- 40 John Preskill. Quantum computing and the entanglement frontier. *Bulletin of the American Physical Society*, 58, 2013.
- 41 John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018.
- 42 Ran Raz and Avishay Tal. Oracle separation of BQP and PH. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 13–23, 2019.
- 43 Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
- 44 Barbara M. Terhal and David P. DiVincenzo. Adaptive quantum computation, constant depth quantum circuits and arthur-merlin games. *Quantum Information & Computation*, 4(2):134–145, 2004.
- 45 Jian-Qiang You and Franco Nori. Atomic physics and quantum optics using superconducting circuits. *Nature*, 474(7353):589–597, 2011.
- 46 Han-Sen Zhong et al. Quantum computational advantage using photons. *Science*, 370:1460–1463, 2020.

## A Proof of Lemma 8

First, we introduce a notation. The range in Definition 6,  $1 \leq |U| \leq \frac{1}{2}|V|$ , may be a little arbitrary. In terms of the range where the expansion ratio is considered, we define more general expander graphs as follows.

► **Definition 19** (Definition 2.2 in [32]). *Let  $G = (V, E)$  be a graph, let  $I$  be a set of positive integers. The graph  $G$  is an  $I$ -expander if a positive constant  $h$  exists such that  $N_G(U) \geq h|U|$  for every vertex subset  $U \subset V$  satisfying  $|U| \in I$ .*

Note that this definition does not limit the maximum degree of graphs. When the graph  $G$  is an expander graph (as defined as Definition 7),  $G$  is also a  $\left[1, \frac{|V|}{2}\right]$ -expander with bounded degree.

Then, the following claim shows an expander graph still has a large connected component and it can be described using the notation of Definition 19 when a small fraction of vertices are adversarially removed.

▷ **Claim 20.** Let  $G = (V, E)$  be an expander graph. If we take a sufficiently small constant  $\epsilon > 0$ , the graph has a connected component  $C$  which has more than  $\frac{|V|}{2}$  vertices and is a  $\left[\frac{|C|}{3}, \frac{2|C|}{3}\right]$ -expander after up to an  $\epsilon$  fraction of the vertices are adversarially removed.

## 74:16 Quantum Advantage with Shallow Circuits Under Arbitrary Corruption

Proof. Let  $h$  be the expansion ratio and  $d$  be the maximum degree of the graph  $G$ . Let  $V' \subset V$  be an arbitrary subset of vertices such that  $|V'| \leq \epsilon|V|$ . Let  $C_1, \dots, C_m$  be the connected components of the left graph after  $|V'|$  vertices are adversarially removed.

To reach a contradiction, we assume every connected component is equal to or smaller than half of  $|V|$ , i.e., for all  $i$ ,  $|C_i| \leq \frac{|V|}{2}$ . From  $|V \setminus V'| + |V'| = |V|$ ,  $|V \setminus V'| = |V| - |V'| \geq (1 - \epsilon)|V|$ . Each connected component  $C_i$  has its neighbor  $N_G(C_i)$  such that  $|N_G(C_i)| \geq h|C_i|$  since  $|C_i| \leq \frac{|V|}{2}$ . A vertex can be a neighbor of at most  $d$  connected components at the same time. Therefore, by the summation of neighbors of all connected components  $C_i$ ,

$$|N_G(V \setminus V')| \geq \frac{h|V \setminus V'|}{d} \geq \frac{h}{d}(1 - \epsilon)|V|.$$

In terms of the total number of vertices,  $|V \setminus V'| + |N_G(V \setminus V')| \leq |V|$ . Then,

$$\epsilon|V| \geq |V'| = |V| - |V \setminus V'| \geq |N_G(V \setminus V')| \geq \frac{h}{d}(1 - \epsilon)|V|.$$

Thus,  $\epsilon \geq \frac{\frac{h}{d}}{1 + \frac{h}{d}}$  and this contradicts  $\epsilon$  is sufficiently small. Therefore we can pick a connected component  $C$  such that  $|C| > \frac{|V|}{2}$  from  $C_1, \dots, C_m$ .

In the graph  $C$ , we consider an arbitrary subset of vertices  $W$ , which satisfies  $\frac{|C|}{3} \leq W \leq \frac{2|C|}{3}$ . From the expander property of  $G$ ,  $|N_G(W)| \geq \frac{h|C|}{3} > \frac{h|V|}{6}$ . Since the number of removed vertices is up to  $\epsilon|V|$ ,  $|N_C(W)| > (\frac{h}{6} - \epsilon)|V|$ . If we take  $\epsilon$  smaller than  $\frac{h}{6}$ ,  $C$  is a  $\left[\frac{|C|}{3}, \frac{2|C|}{3}\right]$ -expander.  $\triangleleft$

The next claim is to show there is a relation between  $I$ -expanders of Definition 5.

▷ **Claim 21** (Lemma 2.4 in [32]). Let graph  $G = (V, E)$  be a  $\left[\frac{|V|}{3}, \frac{2|V|}{3}\right]$ -expander. Then there is a vertex subset  $Z \subset V$  such that  $|Z| < \frac{|V|}{3}$  and the graph  $G' = G[V \setminus Z]$  is a  $\left[1, \frac{|G'|}{2}\right]$ -expander.

The most significant result of minors of expander graphs is Claim 22 below. It is known that this bound ( $O(\frac{|V|}{\log(|V|)})$ ) is tight especially in terms of the size of grid minors.

▷ **Claim 22** (Corollary 8.3 in [32] and Theorem 1.1 in [20]). Let graph  $G = (V, E)$  be a  $\left[1, \frac{|V|}{2}\right]$ -expander. For any graph  $H$  with  $O(\frac{|V|}{\log(|V|)})$  vertices and edges,  $G$  contains  $H$  as a minor.

Finally, using the above claims, we can prove Lemma 8.

**Proof of Lemma 8.** From Claim 20, after the removal, the left graph has a connected component  $C$  which is a  $\left[\frac{|C|}{3}, \frac{2|C|}{3}\right]$ -expander. Using Claim 21,  $C$  contains an induced subgraph  $C'$  such that  $C'$  is a  $\left[1, \frac{|C'|}{2}\right]$ -expander and  $|C'| > \frac{2|C|}{3} > \frac{|V|}{3}$ . When  $n$  is sufficiently large,  $\frac{n}{\log(n)} \gg n^{\frac{1}{2}}$ . Therefore, from Claim 22,  $C$  contains a  $\Omega(|V|^{\frac{1}{4}}) \times \Omega(|V|^{\frac{1}{4}})$  grid as a minor since the maximum degree of grid graphs is 4, which is a constant, and the number of vertices and edges of a  $\Omega(|V|^{\frac{1}{4}}) \times \Omega(|V|^{\frac{1}{4}})$  grid is  $\Omega(|V|^{\frac{1}{2}})$ .  $\blacktriangleleft$